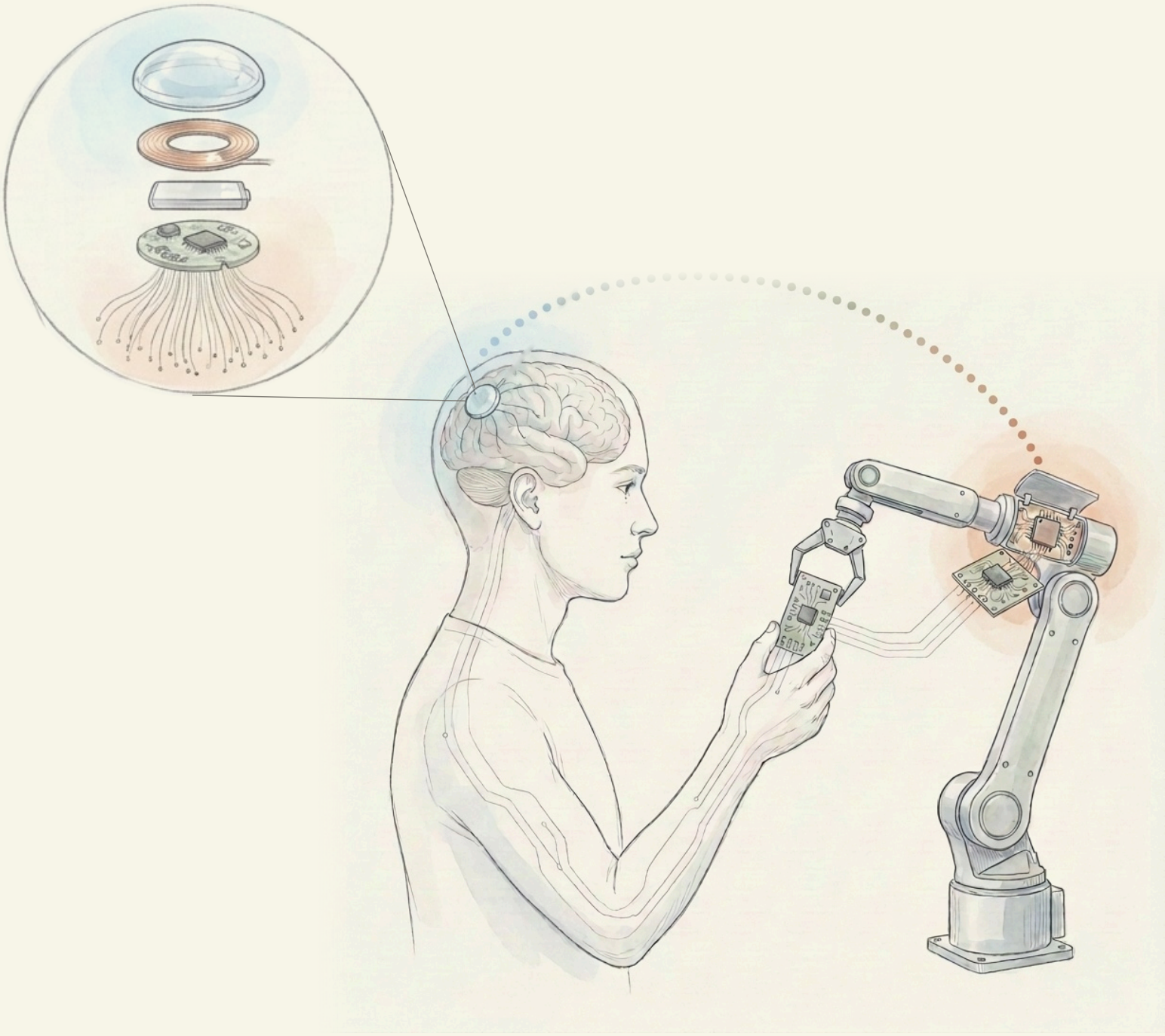
Introduction to

Electrical Engineering

Building Intelligent Hardware

FIRST EDITION

YASSER KHAN



YASSER KHAN

INTRODUCTION TO ELECTRICAL ENGINEERING

BUILDING INTELLIGENT HARDWARE

FIRST EDITION

Introduction to Electrical Engineering: Building Intelligent Hardware
First Edition

Copyright © 2026 Yasser Khan. All rights reserved.

No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author, except for brief quotations in reviews and scholarly works.

ISBN 979-8-9826941-1-9

The author has made every effort to make the information in this book accurate, but it is provided as is, without warranty of any kind. The author is not liable for any loss, damage, or injury that results from its use. The laboratory exercises use low-voltage hardware powered from a computer's USB port; follow the instructions of your instructor and the documentation of each part.

Arduino, Python, Jupyter, Neuralink, and the other product and company names mentioned in this book are trademarks of their respective owners. They are used only to identify products, with no endorsement implied.

Preface

Electronics has moved from the desk into everything around us. A watch measures a pulse with light, a car watches the lane with a camera, a hearing aid separates a voice from the noise of a restaurant, and an implant the size of a coin listens to neurons and turns their activity into the motion of a cursor. These devices are *intelligent hardware*: they sense the physical world, decide something about it on the spot, and communicate the result, often from a battery and within a few milliseconds.

The need for such hardware keeps growing, and much of it comes from problems that software alone cannot reach. Health care wants measurements taken continuously on the body rather than once a year in a clinic. Environmental monitoring wants thousands of small sensors spread over a city or a field. Assistive devices want to respond in real time to a person who cannot use a keyboard. In each case the data are too large, too private, or too urgent to send everywhere, so part of the intelligence has to live next to the sensor, in the hardware itself.

Building intelligent hardware draws on several fields at once. A sensor turns a physical quantity into a small electrical signal. Circuits and semiconductor devices shape and amplify that signal. A converter turns it into numbers, a processor or a neural network extracts the information that matters, and a communication link carries the result away. A designer has to understand every link of this chain well enough to see how they constrain one another: a noisier sensor needs a better amplifier or a smarter algorithm; a slower radio needs more processing on the device; a smaller battery changes every one of those choices. The hardest problems sit at the joints between the links.

That kind of understanding does not come from reading alone. Hardware design is learned the way it is practiced: by building something, measuring it, finding that it does not quite behave as the equations promised, and working out why. The noise on a real signal, the resistor that loads a divider, the light in the room that corrupts an optical link, and the classifier that fails on a new background are all lessons that an experiment teaches in minutes and a lecture struggles

to teach at all. Experiential learning, the cycle of doing, observing, reflecting, and trying again, is not an extra added to the theory; it is how the theory becomes useful.

This book is written for that kind of learning. It grew out of EE 105 at the University of Southern California, a first course in electrical engineering built around one demonstration board and an Arduino microcontroller. Each chapter develops the ideas of one part of the sensing chain, with worked examples and problems, and then puts them to work in a laboratory on the board: reading a touch panel, tracing the curves of a diode and a transistor, measuring temperature and a heartbeat, classifying images with a small neural network, and sending messages between two boards on a beam of light. The labs run from Jupyter notebooks that talk to the board directly, so the time goes to the engineering rather than to the tools. The book ends with a chapter on the final project, where you choose a problem of your own and build a small piece of intelligent hardware to address it.

How the book follows the course

The course has six modules, and the chapter numbers are the module numbers (Figure P.1). Chapter 0 introduces electronic systems, the board, the Arduino, and Python. Chapters 1 and 2 provide the tools: circuit laws and semiconductor devices. Chapters 3 to 5 put them to work in sensing, machine learning, and communication. Chapter 6 is about the final project. Each module also has an online quiz on the course portal.

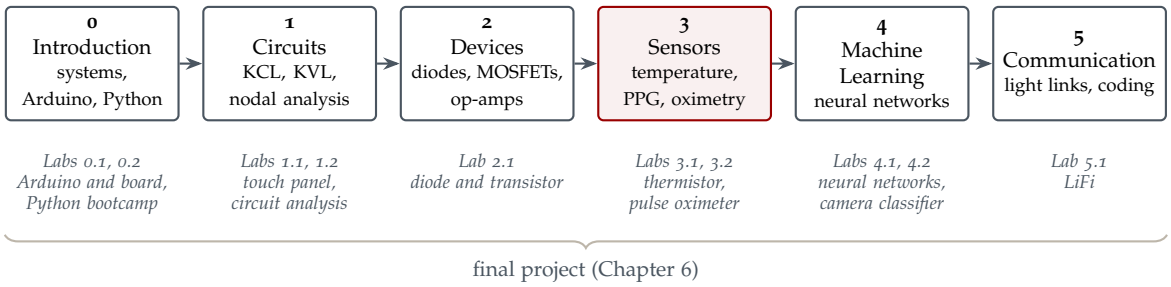


Figure P.1: The six modules of EE 105 and the labs that go with them. The sensors module is where the earlier material first comes together in one working instrument, and the final project draws on all of it.

How each chapter is organized

Every chapter opens with a short list of what you will be able to do by the end, then develops the ideas in the order of the lectures. Along the way you will find:

Worked examples

marked with a red rule in the left margin. Many of them are the

module's homework problems, solved step by step. Try each one before reading the solution.

Margin notes
in the wide right margin hold definitions, key equations, historical asides, and figure captions, so the main text can stay focused.

Laboratory sections
near the end of Chapters 0 to 5. Each starts with a summary box and a box that shows exactly how to set the board's module selector and jumpers, then gives the procedure, the results you should expect, and what to do when something goes wrong. Chapter 0 explains how the lab kit is installed and how lab work is submitted.

Problems
at the end of each chapter, marked by difficulty: ○ for a direct application, ● for a multi-step problem, and ●● for a design or open-ended problem. Answers to all of them are in Appendix A.

What you need

You need algebra and a little calculus; the few ideas from linear algebra used in Chapters 1 and 4 are developed where they are needed. No programming experience is assumed: Chapter 0 starts Python from the beginning. For the labs you need the EE 105 demonstration board, an Arduino Nano 33 BLE Sense, a laptop running Windows or macOS, and the course lab kit, which installs everything else with one setup file.

A note on the writing

AI tools were used to improve the clarity and grammar of the text. The author takes full responsibility for the content of this book, including any errors.

Acknowledgments

I thank Rehan Kapadia, co-instructor of EE 105, and Atiyeh Abbasi Jalal, Vikas Addepalli, Shadman Shahid, and Walter Unglaub, who helped develop the course content that is adapted in this textbook. Rehan Kapadia and Atiyeh Abbasi Jalal developed the content of Chapter 4, Machine Learning, and Rehan Kapadia and Walter Unglaub developed the content of Chapter 5, Communications. I am also grateful to the teaching assistants, course mentors, and students of every

semester, who tested the labs and the demonstration board and found the problems first.

I dedicate this book to my wife, Sifat Muin, and our children, Inan Khan and Ilyn Khan.

Yasser Khan
Los Angeles

Contents

Preface 3

0 *Electrical Engineering and Electronic Systems* 11

0.1 A Brain-Computer Interface, Taken Apart 11
0.2 What Is Electrical Engineering? 18
0.3 Transistors and the Growth of Computing 18
0.4 Units, Scales, and Prefixes 21
0.5 Electronic Systems: Sense, Process, Communicate 24
0.6 Voltage, Current, and Resistance 26
0.7 Signals: Analog and Digital 28
0.8 The EE 105 Demonstration Board 34
0.9 The Arduino Microcontroller 40
0.10 Python for Engineers 44
0.11 Laboratory: Arduino and the Demo Board 46
0.12 Laboratory: Python Bootcamp 67
0.13 Problems 83

1 *Circuits* 92

1.1 Charge, Current, Voltage, and Resistance 93
1.2 Circuit Elements and Schematics 97
1.3 Kirchhoff's Laws 99
1.4 Series, Parallel, and Divider Circuits 103

1.5 *Measuring Circuits: Meters, Loading, and Fuses* 111
1.6 *Nodal Analysis* 114
1.7 *Resistive Touch Sensing* 119
1.8 *Equivalent Circuits: Thevenin and Norton* 125
1.9 *Capacitors and Inductors* 128
1.10 *Laboratory: Asymmetric Touch Panel* 130
1.11 *Laboratory: Circuit Analysis with Python* 142
1.12 *Problems* 157

2 *Semiconductor Devices* 165
2.1 *Tunable Resistance: Conductors, Insulators, and Semiconductors* 166
2.2 *Silicon, Electrons, and Holes* 168
2.3 *Energy Bands and the Band Gap* 174
2.4 *The pn Junction* 177
2.5 *The Diode* 180
2.6 *Diode Circuits and Diode Models* 184
2.7 *The MOSFET* 188
2.8 *CMOS Logic* 196
2.9 *Light-Emitting Diodes* 197
2.10 *Photodiodes, Solar Cells, and Phototransistors* 201
2.11 *Op-Amps: Turning Currents into Voltages* 205
2.12 *Laboratory: Diode and Transistor I-V Characteristics* 215
2.13 *Problems* 232

3 *Sensors* 239
3.1 *What Is a Sensor?* 240
3.2 *Characterizing a Sensor* 242
3.3 *Temperature Sensors and Resistive Interfaces* 245
3.4 *Analog-to-Digital Conversion* 255
3.5 *Op-Amp Circuits for Sensor Readout* 258
3.6 *Signals in Time and Frequency* 266

3.7 *Optical Biosensing and the Beer-Lambert Law* 269
3.8 *Photoplethysmography and Pulse Oximetry* 274
3.9 *Heart Rate from the PPG* 280
3.10 *Laboratory: Thermistor Temperature Sensor* 282
3.11 *Laboratory: Pulse Oximeter and PPG* 295
3.12 *Problems* 311

4 *Machine Learning* 318
4.1 *From Sensors to Decisions* 319
4.2 *Vectors, Matrices, and the Multiply-Accumulate* 321
4.3 *The Artificial Neuron* 323
4.4 *Perceptrons and Decision Boundaries* 326
4.5 *Multilayer Networks and the Forward Pass* 329
4.6 *Learning: Cost Functions and Gradient Descent* 333
4.7 *Backpropagation* 338
4.8 *Training, Testing, and Judging a Classifier* 341
4.9 *Hardware for Neural Networks* 346
4.10 *Laboratory: Neural Networks in Python* 353
4.11 *Laboratory: Camera Classifier* 368
4.12 *Problems* 384

5 *Communications* 392
5.1 *Communicating with Light* 393
5.2 *The Anatomy of a Communication System* 395
5.3 *Information, Bits, and Symbols* 397
5.4 *Putting Bits on Light: Modulation* 401
5.5 *The Optical Link* 403
5.6 *Noise* 410
5.7 *Decisions, Bit Errors, and Eye Diagrams* 416
5.8 *Framing, Synchronization, and Error Control* 423
5.9 *Sensor Fusion* 429
5.10 *Laboratory: LiFi, Sending Text over Light* 435
5.11 *Problems* 453

6 *The Final Project: Building Your Own Intelligent Hardware* 460

6.1 Finding a Good Project 461

6.2 From Idea to Specification 463

6.3 Carrying Out the Project 465

6.4 Testing and Judging the Result 467

6.5 Building Blocks and Ideas 468

6.6 Problems 469

A *Answers to Chapter Problems* 471

About the Author 491

0
Electrical Engineering and Electronic Systems

In this chapter you will learn to

- Take a brain-computer interface apart into sensing, amplification, conversion, processing, communication, and power, and name the chapter of this book that teaches each part.
- Estimate data rates, power budgets, and battery life from a few stated numbers.
- Trace the growth of computing from the first transistors to modern chips, and use Moore's law to estimate growth rates.
- Work fluently with SI units, prefixes, binary and hexadecimal numbers, and orders of magnitude.
- Explain how an analog voltage becomes a digital number (sampling, bits, ADCs) and how a number becomes a voltage again (DACs, PWM).
- Find your way around the EE 105 demo board: its blocks, the module selector, the jumpers, and the firmware commands.
- Write, upload, and debug Arduino sketches that use digital outputs, analog inputs, PWM, and the serial port.
- Use Python, NumPy, and Matplotlib in a Jupyter notebook to organize and analyze data.

0.1 A Brain-Computer Interface, Taken Apart

Imagine a person who cannot move their hands steering a cursor across a computer screen only by intending to move it. That is what a *brain-computer interface* (BCI) does: it records the electrical activity of neurons and turns it into commands for a machine. The company Neuralink builds one, and its public descriptions give enough

numbers to take the whole system apart. This book starts there for two reasons. Every part of the implant is a topic you will study in a later chapter, and the kit on your desk (the demo board, the Arduino, and the lab notebooks) is a small-scale version of the same kind of intelligent hardware.

The implant

According to Neuralink's public descriptions, the clinical implant, called the Link (N1), is a sealed housing of titanium and glass about 23 mm across and 8 mm thick, roughly the size of a small stack of coins. It replaces a disk of skull, so it sits in the bone under the scalp (Figure 0.1). From it run 64 flexible polymer threads, each only about 5 µm thick, with 16 electrodes on every thread: $64 \times 16 = 1024$ electrodes in all. The threads are far too thin and soft to push in by hand. A surgical robot (R1) inserts them into the cortex, the outer layer of the brain, while avoiding the blood vessels on its surface.

Inside the housing, a custom chip amplifies and digitizes the signals of all 1024 channels. A Bluetooth radio sends the data to a phone or computer, where software decodes the movement the person intends, for example to move a cursor. A rechargeable battery powers the implant. It is charged without wires, through the scalp, by *inductive charging*: a coil in an external charger makes a changing magnetic field, and that field induces a current in a second coil inside the implant.

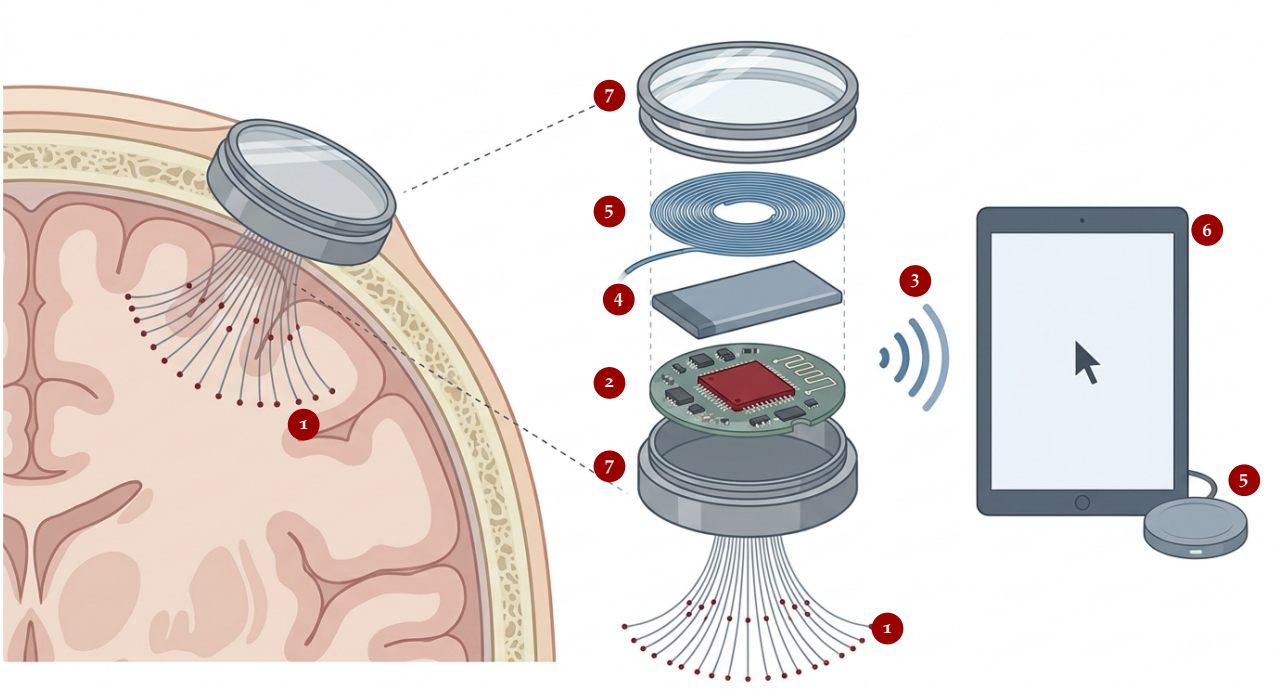
An earlier research system is described in more technical detail in a 2019 paper by Elon Musk and Neuralink.[1] Its custom chip, an *application-specific integrated circuit* (ASIC), served 256 channels, each with its own amplifier. The amplifiers added an input-referred noise of only about 5.9 µV rms. "Input-referred" means that the noise is quoted as if it were a voltage at the electrode, before amplification, so that it can be compared directly with the signal; "rms" (root mean square) is the standard measure of the typical size of a voltage that fluctuates. An analog-to-digital converter (ADC) on the same chip turned each channel into 10-bit numbers about 19 300 times per second, and the whole chip used about 6 mW. Spike detection, the step that finds the brief pulses that neurons fire, was done in software, and the research prototype sent its data out through a USB-C cable. The wireless version came later.

[1] E. Musk and Neuralink, "An integrated brain-machine interface platform with thousands of channels," *Journal of Medical Internet Research*, 2019.

The dissection

Follow a signal through the system (Figure 0.2). Each block is a topic of this book.

Electrodes. A neuron that fires produces a short voltage pulse called



1 Electrodes on 64 thin threads pick up the spikes of nearby neurons. *Sensors, Chapter 3.*

2 Circuit board. Its custom chip holds an amplifier and an ADC for every channel, made of transistors, and finds the spikes. *Circuits and Devices, Chapters 1 and 2; Machine learning, Chapter 4.*

3 Bluetooth radio link that carries the results out of the body. *Communication, Chapter 5.*

4 Battery. The power budget of every block sets its size. *Chapters 0 and 1.*

5 Charging coil, and the charger outside the head that feeds it through the scalp. *Chapters 0 and 1.*

6 Tablet or computer that decodes the intended movement and moves the cursor. *Machine learning, Chapter 4.*

7 Sealed housing of titanium and glass, set into the skull.

Figure 0.1: A wireless brain-computer interface. Left: cross section of the head with the coin-sized implant set into the skull and its thin threads reaching into the cortex. Center: exploded view of the implant, with the cap, the charging coil, the battery, the circuit board that carries the custom chip and the radio, and the threads. Right: a tablet receives the data over Bluetooth, and a charger outside the head recharges the battery through the scalp. The numbers match the key below the drawing, which names the chapter that teaches each part.

a *spike*, about a millisecond long. At an electrode near the neuron the spike is tiny, typically tens to a few hundred microvolts. The electrode and the tissue around it form an interface where the signal, carried by ions in salty fluid, is passed to the electrons of a metal, and the electrode must survive that environment for years. Turning a physical quantity into an electrical signal is the subject of Chapter 3 (Sensors).

Amplifiers. A 100 µV signal is too small to digitize directly, so each channel has its own amplifier, which multiplies the voltage by a fixed factor, its *gain*, while adding as little noise as possible. The amplifiers and the rest of this *analog front end* are built from the circuit ideas of Chapter 1 (Circuits) and the operational amplifiers (op-amps) of Chapters 2 and 3.

ADC. The amplified voltage is sampled and turned into numbers. Section 0.7 of this chapter explains sampling, bits, and step size, and Chapter 3 returns to ADCs in sensor systems. The chip that holds the amplifiers and the ADC is itself made of transistors, the devices of Chapter 2.

Spike detection and decoding. Most of the recording is background. Software finds the spikes, and a decoder learns which patterns of spikes go with which intended movement. Learning a mapping from data to decisions is machine learning, the subject of Chapter 4.

Radio. The Bluetooth link carries the results out of the body. How information rides on a carrier, and how fast a link can send it, are the topics of Chapter 5 (Communication).

Power. A battery feeds every block, and the energy it stores sets how long the implant runs between charges. The *power budget*, the sum of what every block uses, decides the battery size and how often it must be charged. Power and energy start in Section 0.6 of this chapter and continue in Chapter 1.

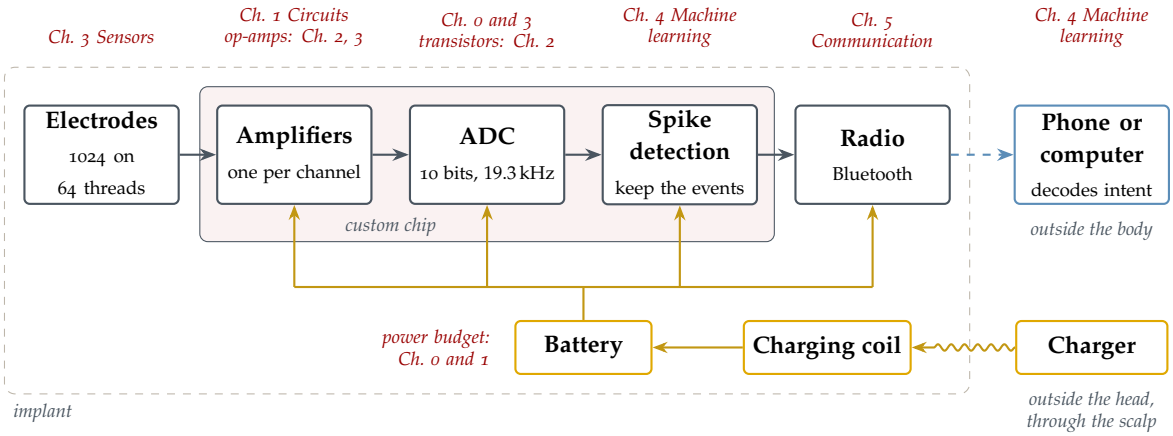


Figure 0.2: The brain-computer interface as a chain of blocks. Signals flow from the electrodes to the decoder; the battery, recharged through the scalp by inductive charging, powers every block inside the implant. The red tag above each block names the chapter that teaches it. The amplifiers, the ADC, and the spike detector share the custom chip.

Electrical engineering is not the whole story. The robot that in-

serts the threads needs mechanical and control engineering, the soft threads need materials science and chemistry, and the decoding software draws on computer science and neuroscience. Electrical engineering sits in the middle, connecting the physical world to the computing world. Some implants also work in the other direction: a digital-to-analog converter (DAC) turns numbers into current pulses that stimulate neurons, as in the deep brain stimulators already used in clinics.

The same system, on your desk

You will not build a brain implant in this course, but you will build systems with the same skeleton, using the kit of Table 0.1. The EE 105 demo board carries an Arduino microcontroller surrounded by small hardware blocks: a resistor network that acts as a touch panel, a diode and a transistor you can measure, light emitters and detectors for a pulse sensor and for optical communication, and a camera for machine vision. The Arduino digitizes their signals, and Python notebooks on your laptop process, plot, and learn from the data. Each block senses something, turns it into numbers, processes the numbers, and passes the result on: the same chain the implant follows.

In the implant	In the EE 105 kit	Ch.
electrodes on threads	photodiodes, optical pulse sensor, thermistor	3
an amplifier per channel	op-amp amplifiers on the board	1–3
10-bit ADC at 19.3 kHz	the Arduino's 10-bit ADC	0, 3
custom chip of transistors	diode and transistor of the devices block	2
spike detection, decoding	neural network that classifies camera images	4
Bluetooth radio	LED-to-photodiode light link	5
battery, inductive charging	USB supply, 3.3 V rail, power estimates	0, 1

Table 0.1: The implant and the EE 105 kit, block by block. The kit is a small-scale version of the same system.

The worked examples below use the published numbers, plus a few assumptions that are stated each time, to answer the questions an engineer asks first about such a system: how much data, how much power, and how long the battery lasts.

> **Worked Example 0.1** *How much data does a 1024-channel implant produce?*
>
> Suppose each of the 1024 channels is sampled 19 300 times per second with 10 bits per sample, as in the 2019 research chip.

(a) What is the raw data rate? (b) A Bluetooth Low Energy radio signals at 1 or 2 Mbit/s, and the useful data it delivers is lower; take 1 Mbit/s. By what factor must the implant shrink its data to fit the link?

Solution. (a) The raw rate is channels times samples per second times bits per sample:

$$1024 \times 19\,300\ 1/\mathrm{s} \times 10\ \mathrm{bits} = 1.976 \times 10^{8}\ \mathrm{bit/s} \approx 198\ \mathrm{Mbit/s}.$$

Dividing by 8 bits per byte, that is about 24.7 MB every second.

(b) $198/1 \approx 200$. The raw stream is about 200 times more than the link carries, and even at the full 2 Mbit/s signaling rate it is about 100 times more. Streaming everything is impossible, so the implant must shrink the data before the radio: detect spikes and send only when and where they occur, or compress the signal in other ways. Processing near the sensor also saves energy, because sending a bit by radio costs far more energy than a short computation on the chip. ■

Worked Example 0.2 *Chips and power for 1024 channels*

Suppose a 1024-channel implant is built from the 256-channel chips of the 2019 research system, each using 6 mW. (a) How many chips are needed? (b) How much power do they use together? (c) What is the power per channel, and how much energy does the chip spend on each sample of one channel?

Solution. (a) $1024/256 = 4$ chips.

(b) $4 \times 6\,\mathrm{mW} = 24\,\mathrm{mW}$.

(c) One chip serves 256 channels, so each channel gets $6\,\mathrm{mW}/256 = 23.4\,\mu\mathrm{W}$. Each channel takes 19 300 samples every second. Power is energy per second, so dividing the power by the number of samples per second gives the energy per sample:

$$E = \frac{P}{f_s} = \frac{23.4 \times 10^{-6}\,\mathrm{W}}{19\,300/\mathrm{s}} = 1.21 \times 10^{-9}\,\mathrm{J} = 1.21\,\mathrm{nJ}.$$

About a nanojoule amplifies and digitizes one sample. Numbers like this let engineers compare designs: a chip that needs half the energy per sample can record twice as many channels on the same battery. ■

Worked Example 0.3 *Battery life of an implant*

Assume the implant's battery stores 150 mAh at 3.7 V and that the whole implant uses an average of 34 mW: the four chips of

Worked Example 0.2 plus about 10 mW for processing and the radio. These are assumptions for the estimate, not published values. (a) How much energy does the battery store, in watt-hours and in joules? (b) How long does a full charge last? (c) If the charger delivers a net 0.1 W into the battery, how long does a full charge take?

SOLUTION. (a) A milliampere-hour is a charge (a current times a time). Multiplying by the battery voltage turns it into energy: $150\,\text{mAh} \times 3.7\,\text{V} = 555\,\text{mWh} = 0.555\,\text{Wh}$. One watt-hour is one watt for 3600 seconds, or 3600 J, so the battery holds $0.555 \times 3600 = 1998\,\text{J} \approx 2.0\,\text{kJ}$.

(b) Time is energy divided by power: $t = 0.555\,\text{Wh}/0.034\,\text{W} = 16.3\,\text{h}$, about two thirds of a day. The same answer comes from current: the implant draws $I = P/V = 34\,\text{mW}/3.7\,\text{V} = 9.19\,\text{mA}$, and $150\,\text{mAh}/9.19\,\text{mA} = 16.3\,\text{h}$.

(c) $0.555\,\text{Wh}/0.1\,\text{W} = 5.55\,\text{h}$. Every milliwatt saved stretches the time between charges: at 30 mW the same battery would last $0.555/0.030 = 18.5\,\text{h}$. ■

WORKED EXAMPLE 0.4 *Spike detection shrinks the data*

Suppose the implant detects spikes on its chip and, for each spike, sends one 32-bit message: 10 bits for the channel number and 22 bits for the time. Assume each channel sees an average of 10 spikes per second. (a) What data rate results? (b) By what factor is that smaller than the raw rate of Worked Example 0.1? (c) What is the largest average spike rate per channel that fits a 1 Mbit/s link?

SOLUTION. (a) Messages per second times bits per message:

$$1024 \times 10\ 1/\text{s} \times 32\ \text{bits} = 327\,680\ \text{bit/s} \approx 0.33\,\text{Mbit/s}.$$

Ten bits are just enough for the channel number, because $2^{10} = 1024$.

(b) $1.976 \times 10^8 / 327\,680 \approx 603$, so the data shrinks by a factor of about 600 and now fits the link with room to spare.

(c) The link carries 10^6 bits per second, and each spike costs 32 bits on one of 1024 channels: $10^6/(1024 \times 32) = 30.5$ spikes per second per channel. The price of this saving is that the decoder learns *when* each spike happened on each channel but no longer sees the shape of the waveform. ■

KEY IDEA An intelligent system senses, amplifies, digitizes, processes, communicates, and is powered. Each of these blocks is a chapter of this book.

0.2 What Is Electrical Engineering?

Electrical engineering is the discipline concerned with studying, designing, and applying equipment, devices, and systems that use electricity, electronics, and electromagnetism. The definition covers a power plant and a hearing aid, a cell tower and a camera chip. What ties them together is a shared set of physical laws (how charge moves and how fields behave) and a shared design habit: model each part simply, connect the parts, and check that the whole does what it should.

The field splits into subfields, each with its own tools and industry. Table 0.2 lists the ones you will meet in EE 105. Most real products draw on several at once, as the brain-computer interface showed.

Subfield	What it studies	Chapter
Circuits	voltage and current in networks	1
Devices	semiconductors, diodes, transistors, LEDs	2
Sensors	turning physical quantities into signals	3
Signals and systems	sampling and filtering signals	0, 3
Computing and ML	microcontrollers to neural networks	0, 4
Communications	moving information between places	5
Electromagnetics	fields, waves, antennas, light	2, 5
Power	generating, converting, storing energy	all

Table 0.2: Subfields of electrical engineering that appear in EE 105.

0.3 Transistors and the Growth of Computing

PHONES, LAPTOPS, DATA CENTERS, and the processors that train large AI models all depend on one device repeated billions of times: the transistor. The 1956 Nobel Prize in Physics went to William Shockley, John Bardeen, and Walter Brattain "for their researches on semiconductors and their discovery of the transistor effect." Their device replaced the vacuum tube and started a seventy-year run of shrinking electronics. The custom chip in the implant of Section 0.1 is one product of that run.

The transistor as a switch

A transistor has three terminals (Figure 0.3). In the most common type, the metal-oxide-semiconductor field-effect transistor (MOSFET), two heavily doped regions called the *source* and *drain* sit in a body of silicon. Between them, separated from the silicon by a thin insulating oxide, is the *gate*. With no voltage on the gate, almost no current flows from drain to source: the switch is off and only a small leakage current I_{off} remains. When the gate voltage rises above a threshold, a

thin conducting channel forms under the oxide and a large current I_{on} flows. The gate controls the channel like a hand on a faucet: a small effort at the gate controls a large flow between the other two terminals.



Figure 0.3: Left: cross section of an n-channel MOSFET. A positive gate voltage pulls electrons into a thin channel (dashed) that connects source and drain. Right: drain current with the gate off and on (shapes only, not to scale). A good switch has a large I_{on} and a tiny I_{off}.

For computing we want three things from this switch: a clean on and off (large I_{on}, tiny I_{off}), switching as fast as possible, and switching with as little energy as possible, so that billions of transistors can run at once without overheating the chip. Almost every advance in chip technology serves one of these goals. Chapter 2 explains the physics.

From one transistor to billions

A modern processor packs an enormous number of these switches into a few square centimeters. The AMD Instinct MI300 accelerator (2023), built for training AI models, contains about 150 billion transistors; its computing dies are made in a 5 nm process. Chips of this generation pack more than 100 million transistors into each square millimeter. At that density, a square the size of a human cell, 30 µm on a side, holds about a hundred thousand transistors. Parts of the chip are stacked in three dimensions, so that computing and memory sit on top of one another. In a cross section of such a chip, the transistors (the *active devices*) form a thin sheet of silicon at the bottom, under a much taller stack of copper wiring that connects them into circuits.

The path to these numbers was gradual and remarkably steady (Figure 0.5). In 1971 Intel released the 4004, the first commercial microprocessor. It handled 4 bits at a time, had about 2300 transistors on a chip smaller than a square inch, used transistors about 10 µm long, and ran at about 750 kHz. Even so, it was more capable than room-sized computers of earlier decades that weighed 30 tons. By 1993 the first Pentium ran at 66 MHz, nearly 90 times faster. By 2012 processors ran at 1 to 3 GHz with several cores on one chip, and transistor features had shrunk from 10 µm to about 22 nm, roughly 450 times smaller in length (and some 200 000 times smaller in area). Supercomputers grew even faster: the fastest machine on the first list of the world's 500 most powerful computers (1993) did about 6×10^{10} floating-point operations per second, and the fastest in 2022 about 1.1×10^{18}, a factor of about 18 million in 29 years.

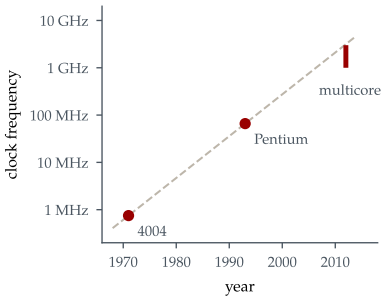


Figure 0.4: Processor clock frequencies on a log scale. The dashed line is the steady growth from the Intel 4004 (1971) to the Pentium (1993); by 2012 clock rates had settled at 1 to 3 GHz and chips added cores instead.

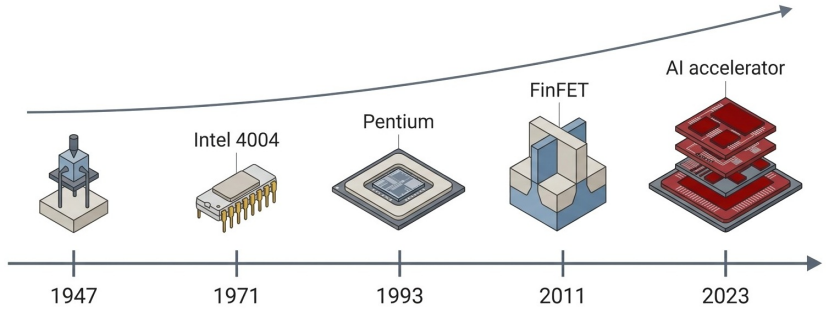


Figure 0.5: Milestones in electronics, from the first transistor (1947) to modern AI accelerators. Each step packed more, smaller, and faster switches onto a chip.

Moore's law

The steady growth has a name. In 1965 Gordon Moore, who later cofounded Intel, observed that the number of transistors on a chip was doubling about every year; in 1975 he revised the pace to about every two years. This trend, *Moore's law*, is an observation about an industry rather than a law of physics, but it held for about five decades. If a quantity doubles every T_2 years, then after t years it has grown by the factor

$$\frac{N(t)}{N(0)} = 2^{t/T_2}, \qquad \text{so} \qquad \text{number of doublings} = \log_2 \frac{N(t)}{N(0)}. \tag{0.1}$$

Clock speed did not keep rising forever (Figure 0.4). Faster switching dissipates more energy per second, and by the mid-2000s chips were near the limit of how much heat a package can remove. Designers kept shrinking transistors but spent the extra ones on more cores instead of higher clock rates. Shrinking itself also got harder: in a very short transistor the drain influences the channel almost as much as the gate does, and the switch leaks when it should be off. The fix has been to wrap the gate around more of the channel. In a *FinFET* (used since about 2011) the channel is a thin vertical fin and the gate covers three sides; in a *nanoribbon* (gate-all-around) transistor the channel is a stack of thin ribbons, each fully surrounded by the gate. More gate contact means a sharper turn-on and less leakage.

Worked Example 0.5 *Moore's law from the 4004 to the MI300*

The Intel 4004 (1971) had about 2300 transistors, and the AMD MI300 (2023) has about 150 billion. (a) How many doublings separate them? (b) What doubling time does that imply? (c) Starting from 2300 transistors and doubling every two years, what transistor count does Eq. (0.1) predict for 2023?

Solution. (a) The growth factor is $150 \times 10^9 / 2300 = 6.5 \times 10^7$. The number of doublings is its base-2 logarithm: $\log_2(6.5 \times 10^7) =$

$\ln(6.5 \times 10^7) / \ln 2 = 25.96$, about 26 doublings.

(b) From 1971 to 2023 is 52 years, so the doubling time is $52/26 = 2.0$ years, just what Moore predicted in 1975.

(c) $t/T_2 = 52/2 = 26$, so $N = 2300 \times 2^{26} = 2300 \times 6.71 \times 10^7 = 1.54 \times 10^{11}$, about 154 billion. The prediction made from a 1975 trend lands within a few percent of a 2023 chip. ■

WORKED EXAMPLE 0.6 *The Apollo Guidance Computer versus a laptop*

The Apollo Guidance Computer that flew on Apollo 11 had 2048 words of erasable memory (RAM), and each word held 16 bits. A modern laptop has 64 000 000 000 bits of RAM. (a) How many bits of RAM did Apollo have? (b) How many times more RAM does the laptop have? (c) The Arduino Nano 33 BLE Sense you will use has 256 KB of SRAM, where 1 KB here means 1024 bytes. How does it compare to Apollo?

SOLUTION. (a) Bits are words times bits per word: $2048 \times 16 = 32\,768$ bits. That is 4096 bytes, or 4 KB.

(b) The ratio is

$$\frac{64\,000\,000\,000}{32\,768} \approx 1.95 \times 10^6,$$

so the laptop has about two million times more RAM.

(c) The Nano's SRAM holds $256 \times 1024 \times 8 = 2\,097\,152$ bits. Dividing by Apollo's 32 768 bits gives exactly 64. The small board on your desk has 64 times the working memory of the computer that guided the first Moon landing. ■

0.4 *Units, Scales, and Prefixes*

The numbers in the last two sections span an enormous range, from 5 µm threads and nanometer transistors to centimeter chips, and from gigahertz clocks to a heartbeat once a second. The tool for handling such ranges is the International System of Units (SI) with its prefixes.

Quantities you will use constantly

Table 0.3 lists the electrical quantities that appear in almost every chapter. Current is the rate of charge flow (one ampere is one coulomb per second), voltage is the energy given to each coulomb (one volt is one joule per coulomb), resistance is the ratio of voltage to current, and power is the rate of energy transfer. Section 0.6 gives a physical picture of each.

Quantity	Symbol	Unit	In base terms
Charge	Q	coulomb (C)	A s
Current	I	ampere (A)	C/s
Voltage	V	volt (V)	J/C
Resistance	R	ohm (Ω)	V/A
Power	P	watt (W)	J/s = V A
Energy	E	joule (J)	W s
Frequency	f	hertz (Hz)	s^{-1}
Time	t	second (s)	

Table 0.3: Electrical quantities and their SI units.

Prefixes and engineering notation

Writing 0.000 005 m is clumsy and error-prone, so we attach a prefix that stands for a power of ten (Table 0.4): 5 µm means 5×10^{-6} m. Engineers write numbers in *engineering notation*, with an exponent that is a multiple of three (4.7 kΩ, 15 mA, 64 MHz), which keeps the leading number between 1 and 999. A prefix on a unit that is squared or cubed is squared or cubed with it: $1\,\text{mm}^2 = (10^{-3}\ \text{m})^2 = 10^{-6}\ \text{m}^2$.

Prefix	Symbol	Factor	Example from this chapter
tera	T	10^{12}	terabyte disk drives
giga	G	10^{9}	3 GHz processor clock (2012)
mega	M	10^{6}	64 MHz Nano 33 clock; 1 MB flash
kilo	k	10^{3}	19.3 kHz neural sampling; 4.7 kΩ
		10^{0}	3.3 V logic supply
milli	m	10^{-3}	6 mW recording chip; 3.22 mV ADC step
micro	μ	10^{-6}	5 µm thread; 5.9 µV noise
nano	n	10^{-9}	5 nm process; 15.6 ns clock period
pico	p	10^{-12}	picofarad capacitors

Table 0.4: SI prefixes used in electrical engineering. Memory sizes are the one common exception to powers of ten: there, 1 KB usually means $2^{10} = 1024$ bytes.

When a quantity spans many orders of magnitude, engineers plot it on a logarithmic scale, where equal steps along the axis are equal factors. Figure 0.6 puts lengths and frequencies from this chapter on such scales; the 5 nm process size and a 150 mm silicon wafer differ by a factor of 3×10^{7}.

Worked Example 0.7 *Converting with prefixes*

An output pin of the Nano 33 BLE Sense can supply at most 15 mA at 3.3 V. (a) Express this current in amperes and in microamperes. (b) How many electrons pass a point each second at this current? The charge of one electron is 1.602×10^{-19} C. (c) What power does the pin deliver, in milliwatts?

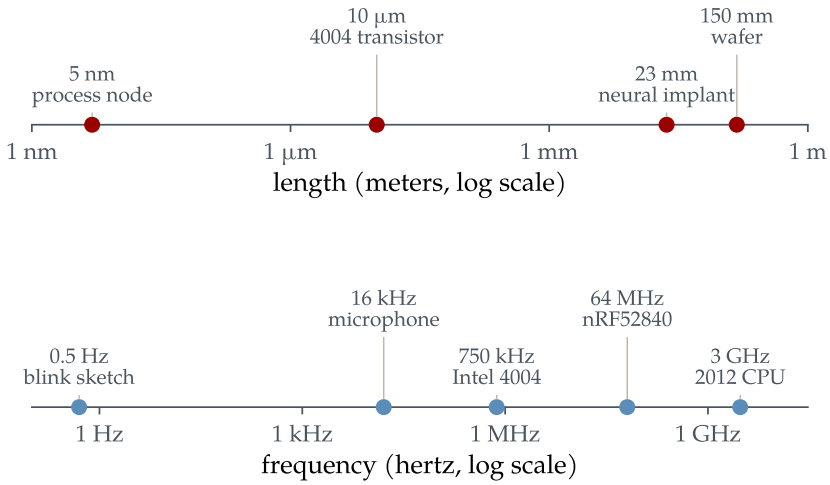


Figure 0.6: Lengths (top) and frequencies (bottom) from this chapter on logarithmic scales. Each labeled tick is a factor of 1000 larger than the one to its left.

Solution. (a) Milli means 10^{-3}, so $15\,\text{mA} = 15 \times 10^{-3}\,\text{A} = 0.015\,\text{A}$. Micro means 10^{-6}, so $0.015\,\text{A} = 0.015/10^{-6} = 15\,000\,\mu\text{A}$.

(b) One ampere is one coulomb per second. The number of electrons per second is the current divided by the charge per electron:

$$\frac{0.015\,\text{C/s}}{1.602 \times 10^{-19}\,\text{C}} \approx 9.36 \times 10^{16} \text{ electrons per second.}$$

(c) $P = VI = 3.3\,\text{V} \times 0.015\,\text{A} = 0.0495\,\text{W} = 49.5\,\text{mW}$. ■

Worked Example 0.8 *Sizing up the implant*

Use the public dimensions of the implant: a disk 23 mm across and 8 mm thick, and threads about 5 µm thick. (a) Write the thread thickness in meters and in nanometers. (b) How many thread thicknesses fit across the implant? (c) Treating the implant as a cylinder, find its volume in cubic millimeters, cubic centimeters, and milliliters.

Solution. (a) $5\,\mu\text{m} = 5 \times 10^{-6}$ m $= 0.000\,005$ m. One micrometer is 1000 nanometers, so the thickness is 5000 nm.

(b) Put both lengths in the same unit before dividing: $23\,\text{mm} = 23 \times 10^{-3}\,\text{m}$, so $23 \times 10^{-3}/5 \times 10^{-6} = 4600$ thread thicknesses.

(c) The radius is 11.5 mm, so

$$V = \pi r^2 h = \pi \times (11.5\,\text{mm})^2 \times 8\,\text{mm} = 3324\,\text{mm}^3.$$

A centimeter is 10 millimeters, so $1\,\text{cm}^3 = (10\,\text{mm})^3 = 1000\,\text{mm}^3$, not $10\,\text{mm}^3$. The volume is $3.32\,\text{cm}^3$, and since one milliliter is

one cubic centimeter, about 3.3 mL: about two thirds of a teaspoon of space for the battery, the chip, the radio, and the charging coil together. ■

0.5 *Electronic Systems: Sense, Process, Communicate*

ANY ENGINEERED ELECTRONIC SYSTEM can be drawn as a short chain (Figure 0.7). Something happens in the real world: a neuron fires, a finger presses a screen, blood pulses through a fingertip, light reflects off a handwritten digit. A *sensor* turns that event into an electrical signal. A *processing* stage, made of analog circuits, a microcontroller, or a computer running software, turns the signal into useful information. A *communication* link carries the information to wherever it is needed, and a *power* source keeps every block running. The output is information a person or another machine can act on: the intended cursor movement, which button was pressed, the heart rate, the digit on the page.

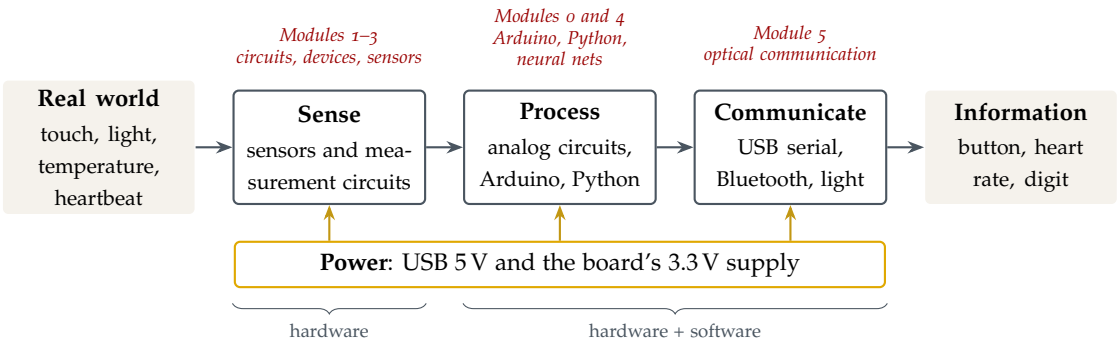


Figure 0.7: The engineered system: sensors turn real-world events into electrical signals, processing extracts information, communication delivers it, and power feeds every block. The course modules are shown above the block they mainly develop.

In EE 105 the sensing side is hardware you will study and build (resistor networks, diodes, transistors, photodiodes, a thermistor, and their measurement circuits), all connected to an Arduino. Processing is a mix of code on the Arduino and Python on your laptop. The brain-computer interface of Section 0.1 follows the same chain, and so does a wearable sensor patch (Figure 0.8).

The chain also has a biological parallel. A neuron collects signals on its dendrites, combines them in the cell body, and sends an output down its axon. A circuit model replaces these with resistors and an amplifier, and the artificial neuron of machine learning keeps only the arithmetic: weighted inputs, a sum, and an activation function. Chapter 4 builds on this last picture.

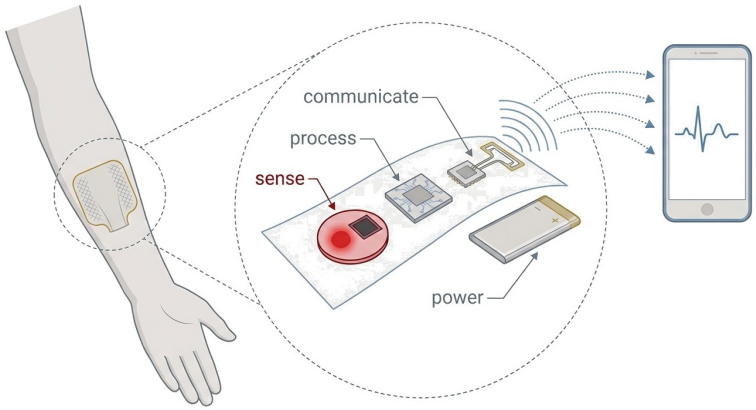


Figure 0.8: A soft wearable sensor patch seen as an electronic system. Sensors touch the skin, a small processor turns their signals into numbers and decisions, a radio sends results to a phone, and a thin battery powers everything. The EE 105 board has the same four parts.

Two examples from the course

A pulse oximeter clips onto a fingertip and reports heart rate and blood oxygen. Red and infrared light shine into the finger and a photodetector measures the light that comes back out (sensing). Driver and read circuits set the light levels and turn the detector's tiny current into a voltage; a microcontroller digitizes and processes it (processing) and transmits it (communication). Software extracts the heart rate and the oxygen level from the resulting waveform, the photoplethysmogram (PPG). Chapter 3 builds this system on the demo board.

A resistive touchscreen finds where a finger presses. The EE 105 version is a 3-by-3 grid of resistors with a push button at each node (Figure 0.9). One corner sits at 3.3 V and the opposite corner at ground, so each node has its own voltage. Pressing a button connects that node to an Arduino analog input, which measures the voltage and looks up the button. The sensor is just resistors and switches; the intelligence is in the measurement. Chapter 1 shows how to calculate the node voltages, and Worked Example 0.14 reads them with the ADC.

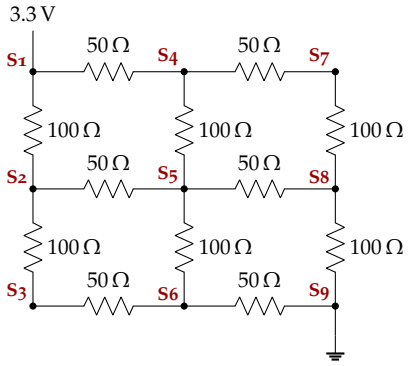


Figure 0.9: The touch network of the demo board's circuits block (Network 1). Horizontal resistors are 50 Ω and vertical ones 100 Ω. Button S*k* connects its node to analog pin A0, so each button produces a different voltage.

How the course modules map onto the system

EE 105 has six modules, and this book has one chapter per module, with the chapter number equal to the module number (Table 0.5). This chapter (Module 0) sets up the tools. Modules 1 and 2 give you the circuit and device ideas behind any sensor, Module 3 combines them into sensors (a thermistor thermometer and a pulse oximeter), Module 4 adds neural networks that classify camera images, and Module 5 closes the chain with communication by light. Each chapter has explanation, worked examples, laboratories on the demo board, and problems.

Module	Topic	Demo board hardware	Block
0	Introduction	Arduino Nano 33 BLE Sense	process
1	Circuits	touch network, resistor grid, divider header	sense
2	Devices	diode and transistor, DAC sweeps	sense
3	Sensors	thermistor, LED and photodiode pair 1, optical pulse sensor	sense
4	ML	OV7675 camera	process
5	Comm.	LED and photodiode pair 2	communicate

Table 0.5: EE 105 modules (and chapters), the demo board hardware each one uses, and the system block it develops.

KEY IDEA Sense, process, communicate, and power: if you can name these four blocks in a device, you can start to understand how it works.

0.6 *Voltage, Current, and Resistance*

Every circuit is built from a few quantities. Chapter 1 develops them carefully and shows how to analyze circuits with them; here is the short version, enough to read the rest of this chapter.

Voltage is pressure, current is flow

Voltage is the push that a source such as a battery gives to charges. Think of water pressure: a pump raises water to high pressure, the water flows through pipes doing work, and returns at low pressure. A battery raises charges to a higher energy at its positive terminal; they flow through the circuit, light a lamp, and return to the negative terminal. The volt honors Alessandro Volta (1745–1827), inventor of the voltaic pile, the first battery; older books call voltage *electromotive force*.

The pressure picture also explains why voltage is always measured *between two points*: what drives the flow is a difference. When we say a node is "at 3.3 V," we mean 3.3 V higher than a reference point we have agreed to call ground (0 V). Two circuits that share no connection have no common reference, so a voltage "between" them has no meaning.

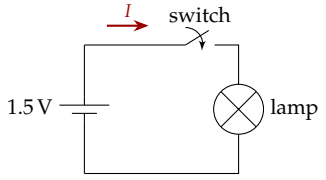


Figure 0.10: A flashlight: a 1.5 V cell, a switch, and a lamp. Conventional current I leaves the positive terminal.

Current is the rate at which charge flows past a point. One ampere means one coulomb, about 6.24×10^{18} electrons, passing each second. Benjamin Franklin guessed that the moving charge was positive and defined current as flowing from $+$ to $-$ through the circuit. J. J. Thomson later discovered the electron, and in metal wires the moving charges turned out to be these negative electrons, which drift the other way. Engineers kept Franklin's *conventional current* (Figure 0.10), and every equation in this book uses it.

Resistance and Ohm's law

Resistance measures how strongly a component opposes current. For a uniform wire it depends on the material (through its resistivity ρ), the length L, and the cross-sectional area A:

$$R = \rho \frac{L}{A}. \tag{0.2}$$

A longer wire resists more and a thicker one less, like a long, narrow pipe versus a short, wide one; most metals also resist more when hot.

For many materials, the current through a component is proportional to the voltage across it. This is *Ohm's law*:

$$V = IR. \tag{0.3}$$

Plot current against voltage for a resistor and you get a straight line through the origin with slope $1/R$. Components that obey Eq. (0.3) are called ohmic. Diodes and transistors are not, and that is what makes them interesting (Chapter 2). The power a resistor turns into heat, and more generally the power delivered to any component with voltage V across it and current I through it, is

$$P = VI = I^2R = \frac{V^2}{R}, \tag{0.4}$$

where the last two forms hold for a resistor. Energy is power times time, $E = Pt$.

KEY EQUATION
$V = IR, \quad P = VI$
Ohm's law and electrical power

Resistors come as fixed parts with color-coded bands, variable potentiometers, and sensing types such as thermistors (temperature) and photoresistors (light) that reappear in Chapter 3.

> **WORKED EXAMPLE 0.9** *Power and energy from a USB port*
>
> A USB port supplies 5 V and, in its basic mode, up to 0.5 A. (a) What is the most power it can deliver? (b) How much energy does it deliver in one hour at that power, in joules and in watt-hours? (c) How much charge flows in that hour, in coulombs and in milliampere-hours?

SOLUTION. (a) $P = VI = 5\,\text{V} \times 0.5\,\text{A} = 2.5\,\text{W}$.

(b) $E = Pt = 2.5\,\text{W} \times 3600\,\text{s} = 9000\,\text{J} = 9\,\text{kJ}$. In watt-hours, $2.5\,\text{W} \times 1\,\text{h} = 2.5\,\text{Wh}$; check: $9000/3600 = 2.5$.

(c) $Q = It = 0.5\,\text{A} \times 3600\,\text{s} = 1800\,\text{C}$, or $500\,\text{mA} \times 1\,\text{h} = 500\,\text{mAh}$. Battery capacities are quoted in milliampere-hours because it is a convenient charge unit: $1\,\text{mAh} = 3.6\,\text{C}$. ■

WORKED EXAMPLE 0.10 *Ohm's law as a unit converter*

The devices block of the demo board measures a transistor's current by passing it through a $1\,\text{k}\Omega$ sense resistor and reading the voltage across that resistor on pin A3 (Section 0.8). (a) Show that the reading in volts equals the current in milliamperes. (b) A3 reads $0.75\,\text{V}$. What is the current, and how much power does the sense resistor turn into heat?

SOLUTION. (a) From Eq. (0.3), $V = IR = I \times 1\,\text{k}\Omega$. Milli and kilo cancel: $1\,\text{mA} \times 1\,\text{k}\Omega = 10^{-3}\,\text{A} \times 10^{3}\,\Omega = 1\,\text{V}$. So $1\,\text{V}$ on A3 means $1\,\text{mA}$, $0.5\,\text{V}$ means $0.5\,\text{mA}$, and so on.

(b) $I = 0.75\,\text{mA}$. The power is $P = VI = 0.75\,\text{V} \times 0.75\,\text{mA} = 0.5625\,\text{mW}$, about half a milliwatt. Note the units again: volts times milliamperes gives milliwatts. ■

From idea to circuit board

Engineers build a new circuit in stages. A *breadboard*, a block of spring clips that grips component leads, allows changes in seconds but makes loose, noisy contacts. A *protoboard* has copper-plated holes on a 0.1 in grid where parts are soldered (hold the iron like a pen, heat the pad and lead together, feed solder into the joint, and wear safety glasses). A *printed circuit board* (PCB) replaces wires with etched copper traces: you draw a schematic in a tool such as KiCad, lay out the board, and send the files to a fabricator. The EE 105 demo board went through exactly these steps.

0.7 *Signals: Analog and Digital*

AN ELECTRODE, A MICROPHONE, OR A PHOTODIODE produces a voltage that varies smoothly in time. Such a signal is *analog*: at every instant it can take any value in its range. A microcontroller, though, works only with numbers made of bits. Bridging these two worlds is the job of the analog-to-digital converter (ADC) on the way in (Figure 0.11) and the digital-to-analog converter (DAC) on the way out. Every module of EE 105 crosses this bridge at least once, and so does

every channel of the brain implant.

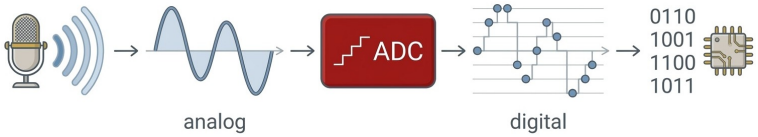


Figure 0.11: From sound to numbers. A smooth analog voltage from a microphone enters an ADC, which outputs a sequence of binary numbers that a processor can store and compute with.

Sampling and quantization

An ADC does two things (Figure 0.12). First, it *samples*: it measures the voltage at regular instants spaced T_s apart. The number of samples per second is the sampling rate $f_s = 1/T_s$. Second, it *quantizes*: it rounds each sample to one of a fixed set of levels and outputs the level's number as a binary code. An N-bit ADC has 2^N levels. If its input range runs from 0 V to a reference voltage V_{ref}, the spacing between levels, called the least significant bit (LSB) or the resolution, is

$$\Delta = \frac{V_{\text{ref}}}{2^N}. \tag{0.5}$$

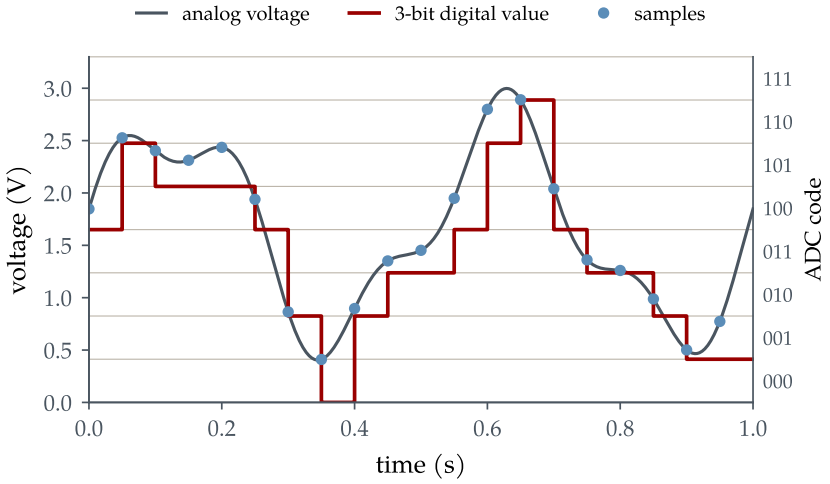


Figure 0.12: An analog voltage (gray) sampled at 20 Hz (dots) and quantized by a 3-bit ADC with $V_{\text{ref}} = 3.3$ V. With only $2^3 = 8$ levels, each 0.41 V apart, the digital version (red) is a coarse staircase. A 10-bit ADC has 1024 levels and follows the curve far more closely.

On the Arduino Nano 33 BLE Sense, `analogRead()` returns a 10-bit number from 0 to 1023 by default, and the reference is the board's 3.3 V supply. The step size is therefore $3.3\,\text{V}/1024 \approx 3.22\,\text{mV}$. To convert a reading back to volts, this chapter and the Lab 0.1 sketches use

$$V \approx \text{code} \times \frac{3.3\,\text{V}}{1023}, \tag{0.6}$$

which maps code 1023 to exactly 3.3 V. Dividing by 1024 instead, as the lab notebooks do, is equally valid; the two conventions differ by at most one step.

How fast must you sample? Fast enough to catch every wiggle you care about. The rule, made precise in Chapter 3, is that you need more than two samples per period of the fastest component in the signal, so the sampling rate must be more than twice the highest frequency you want to keep. A microphone sampled at 16 kHz can capture sounds below 8 kHz, enough for speech (Figure 0.13). A heartbeat at about 1 Hz needs only a few tens of samples per second.

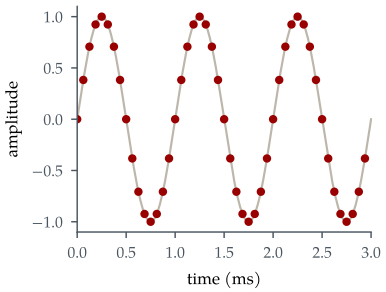


Figure 0.13: A 1 kHz tone sampled at 16 kHz: 16 samples per period, plenty to trace its shape.

Worked Example 0.11 *Sampling a neural spike*

The 2019 recording chip samples each channel at 19.3 kHz. (a) How much time separates two samples? (b) A spike lasts about 1 ms. How many samples fall within one spike? (c) What is the highest frequency this sampling rate can capture?

Solution. (a) $T_s = 1/f_s = 1/19\,300/\mathrm{s} = 5.18 \times 10^{-5}\,\mathrm{s} = 51.8\,\mu\mathrm{s}$.

(b) $1\,\mathrm{ms}/51.8\,\mu\mathrm{s} = 19.3$, so about 19 samples trace each spike: enough to see its shape as well as its timing.

(c) Less than $f_s/2 = 9.65\,\mathrm{kHz}$. The fast edges of a spike contain frequencies of a few kilohertz, which this rate captures. ■

Worked Example 0.12 *The ADC step in a neural recorder*

Suppose each recording channel amplifies the electrode voltage with a gain of 200 before a 10-bit ADC whose input range is 1.2 V. (These two numbers are assumptions for the example.) (a) What is the ADC step size? (b) What change in the electrode voltage does one step represent? Compare it with the amplifier noise of 5.9 µV rms. (c) What is the largest electrode signal (peak to peak) that fits in the ADC range?

Solution. (a) From Eq. (0.5), $\Delta = 1.2\,\mathrm{V}/2^{10} = 1.2\,\mathrm{V}/1024 = 1.17\,\mathrm{mV}$.

(b) The amplifier multiplied the electrode voltage by 200, so one step at the ADC corresponds to $1.17\,\mathrm{mV}/200 = 5.86\,\mu\mathrm{V}$ at the electrode. That is about the size of the noise. More bits would only describe the noise in finer detail, and fewer bits would start to lose real signal, so 10 bits is a sensible match.

(c) $1.2\,\mathrm{V}/200 = 6\,\mathrm{mV}$, far more than the tens to hundreds of microvolts of a spike, which leaves room for slow drifts at the electrode. ■

Bits, bytes, and binary numbers

A bit is a binary digit, 0 or 1; eight bits make a byte ($2^8 = 256$ values). Each extra bit doubles the number of values, so N bits can count from 0 to $2^N - 1$. Binary place values are powers of two: $101_2 = 1 \times 4 + 0 \times$

$2 + 1 \times 1 = 5$. A 10-bit ADC result or a 16-bit audio sample takes two bytes, and a grayscale pixel takes one. Long binary numbers are hard to read, so engineers often write them in *hexadecimal* (base 16), where each digit stands for exactly four bits: the digits are 0–9 and A–F (for 10–15), and the prefix `0x` marks the number as hex.

Many modern sensors contain their own ADC and deliver numbers rather than voltages. The proximity sensor on the Nano 33 BLE Sense (the APDS9960) shines infrared light, measures the reflection with a photodiode, digitizes it internally, and sends the Arduino a number. The optical pulse sensor on the demo board (the MAX86916) does the same with red and infrared light. Both talk to the Arduino over I^2C, a two-wire digital bus described in Section 0.8, where each chip has an address written in hex.

Worked Example 0.13 *Binary and hexadecimal on the demo board*

(a) The DAC chip on the demo board answers at I^2C address `0x10` and the optical sensor at `0x57`. Write both in decimal and in binary. (b) Write the largest 10-bit ADC code in binary and in hex. (c) How many bits does it take to give each of 1024 channels its own number?

Solution. (a) Each hex digit is worth a power of 16. `0x10` $= 1 \times 16 + 0 = 16$, and in binary each hex digit becomes four bits: `1` $\to 0001$, `0` $\to 0000$, so $16 = 0001\,0000_2$. Similarly `0x57` $= 5 \times 16 + 7 = 87$, and `5` $\to 0101$, `7` $\to 0111$ give $87 = 0101\,0111_2$. (Check: $64 + 16 + 4 + 2 + 1 = 87$.)

(b) The largest 10-bit code is $2^{10} - 1 = 1023 = 11\,1111\,1111_2$. Grouping the bits by four from the right gives 11, 1111, 1111, which are 3, `F`, `F`: $1023 =$ `0x3FF`.

(c) N bits give 2^N different numbers. $2^{10} = 1024$, so 10 bits number channels 0 to 1023, which is why the spike messages of Worked Example 0.4 use a 10-bit channel field. ■

Worked Example 0.14 *From voltage to ADC code: reading the touch network*

When pressed, buttons S2, S5, and S6 of the touch network in Figure 0.9 put 1.946 V, 1.650 V, and 0.804 V on pin A0. (Chapter 1 shows how to calculate these.) (a) What 10-bit code does `analogRead(A0)` return for each? (b) A simple sketch decides that S2 was pressed if the code lies between 590 and 615, S5 between 500 and 520, and S6 between 240 and 260. Do the codes fall inside? (c) A reading of 604 arrives. What voltage is that?

Solution. (a) Invert Eq. (0.6): code $= V / 3.3\,\text{V} \times 1023$, rounded to

an integer.

$$\begin{aligned} &\text{S2:} \quad 1.946/3.3 \times 1023 = 603.3 \;\rightarrow\; 603, \\ &\text{S5:} \quad 1.650/3.3 \times 1023 = 511.5 \;\rightarrow\; 511 \text{ or } 512, \\ &\text{S6:} \quad 0.804/3.3 \times 1023 = 249.2 \;\rightarrow\; 249. \end{aligned}$$

S5 sits exactly halfway between two codes, so noise decides which one appears.

(b) Yes: $590 < 603 < 615$, $500 < 512 < 520$, and $240 < 249 < 260$. Each window is about ± 10 codes, or $\pm 32\,\text{mV}$, around its target. The margin absorbs resistor tolerance and noise.

(c) $604 \times 3.3\,\text{V}/1023 = 1.948\,\text{V}$, a reading of S2 just one step above the ideal. ■

From numbers back to voltages: DACs and PWM

To drive the world rather than measure it, a system must turn numbers into voltages. A digital-to-analog converter does this directly: an N-bit DAC accepts a code from 0 to $2^N - 1$ and outputs a proportional voltage. The demo board carries an AD5593R, a chip that combines a 12-bit DAC and an ADC and talks to the Arduino over I^2C. Its codes run from 0 to 4095, and the course code converts between code and voltage at 1.65 mV per code, so code 2000 gives 3.3 V. The output cannot exceed the 3.3 V supply, so codes above about 2000 simply saturate. The devices, sensors, and communication labs use its channels to sweep a diode, bias a transistor, and set the brightness of LEDs.

WORKED EXAMPLE 0.15 *Setting a DAC voltage*

A sketch applies 0.9 V to the diode with these lines:

```
float dac_voltage = 0.9;
int inc = dac_voltage / dac_conversion; // 0.00165 V per code
setDACVal(dac_diode, inc);
```

What code is sent, and what voltage actually appears?

SOLUTION. $0.9/0.00165 = 545.45$. Assigning the result to an `int` in C discards the fraction (it truncates, it does not round), so the code is 545. The output is $545 \times 1.65\,\text{mV} = 0.899\,\text{V}$, within one step (1.65 mV) of the target. For most experiments that error is far below the noise. ■

The Nano 33 BLE Sense itself has no true DAC. Its "analog output" is *pulse-width modulation* (PWM): a digital pin switches rapidly between 0 V and 3.3 V, and the fraction of each period it spends high, the *duty*

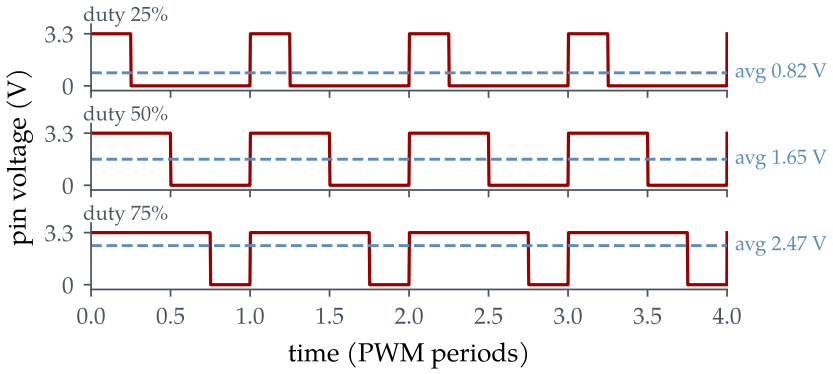


Figure 0.14: Pulse-width modulation. The pin switches between 0 V and 3.3 V; the fraction of time it spends high (the duty cycle) sets the average voltage (dashed).

cycle D, sets the average voltage (Figure 0.14):

$$\bar{V} = D \times 3.3\,\text{V}. \tag{0.7}$$

The function `analogWrite(pin, value)` takes a value from 0 (always low) to 255 (always high), so $D = \text{value}/255$. A slow load, such as an LED seen by your eye, responds only to the average; a fast circuit sees the pulses unless a capacitor filters them.

Worked Example 0.16 *Average voltage from PWM*

(a) What duty cycle and average voltage does `analogWrite(pin, 64)` give? (b) Which value gives an average closest to 1.65 V, and what average results?

Solution. (a) $D = 64/255 = 0.251$, or 25.1%, and $\bar{V} = 0.251 \times 3.3\,\text{V} = 0.828\,\text{V}$.

(b) Solve Eq. (0.7) for the value: $\text{value} = 255 \times 1.65/3.3 = 127.5$. The value must be an integer, and 127 and 128 are equally close: they give $127/255 \times 3.3 = 1.644\,\text{V}$ and $128/255 \times 3.3 = 1.656\,\text{V}$. With only 256 levels, PWM on this pin cannot hit 1.65 V exactly; its step is $3.3\,\text{V}/255 = 12.9\,\text{mV}$, about eight times coarser than the 1.65 mV step of the AD5593R DAC. ■

A one-bit signal: the PDM microphone

The microphone on the Nano 33 BLE Sense uses an unusual format called *pulse density modulation* (PDM). Instead of sending multi-bit samples, it sends a very fast stream of single bits. When the sound pressure is high, the stream contains mostly ones; when it is low, mostly zeros (Figure 0.15). The local density of ones tracks the sound. A hardware block in the microcontroller averages this bitstream and converts it to ordinary 16-bit samples, called pulse code modulation (PCM), at an

audio rate such as 16 kHz. Your code only ever sees the 16-bit samples. Worked Example 0.20 uses them to detect a clap.

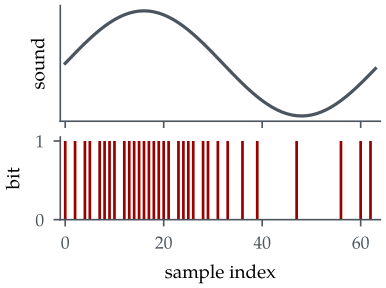


Figure 0.15: Pulse density modulation of one period of a sound wave. Each line is a 1 bit; ones crowd together where the sound is high.

0.8 The EE 105 Demonstration Board

THE DEMONSTRATION BOARD is one printed circuit board that holds the hardware for every lab of the course (Figure 0.16). The board in use is the black revision 1.1. An Arduino Nano 33 BLE Sense plugs into the top center. Around it sit four hardware blocks, outlined and labeled on the board, plus the switch and the jumpers that connect them to the Arduino.

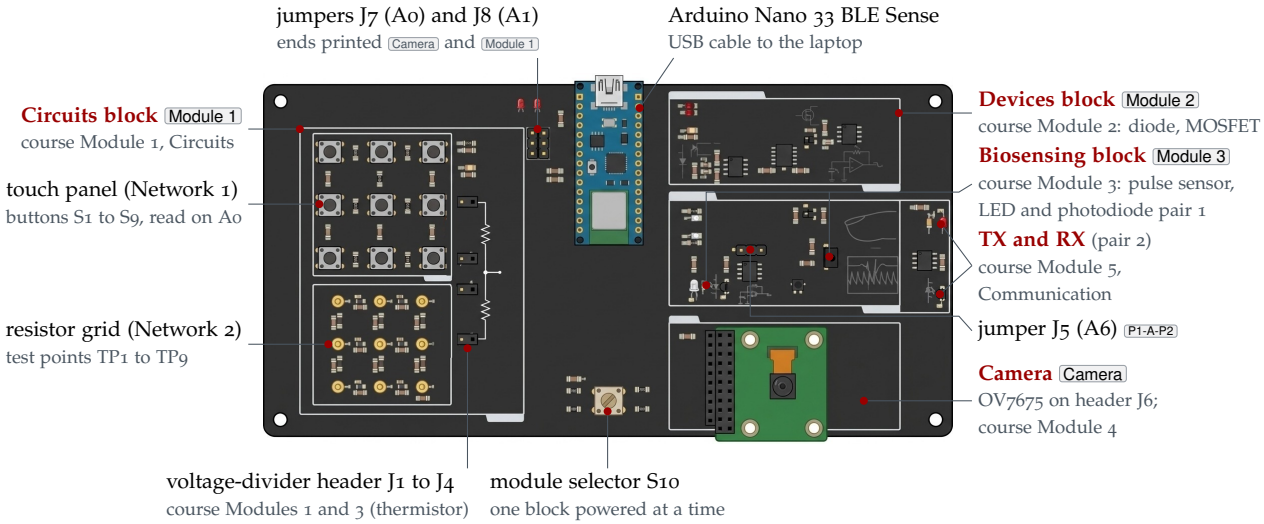


Figure 0.16: The EE 105 demonstration board, revision 1.1, drawn from a photograph. Gray plates show labels printed on the board. The circuits block (Module 1) fills the left half; the devices block (Module 2) is at the upper right; the biosensing block (Module 3) is at the right middle, with the communication LED and photodiode, TX and RX (Module 5), at the right edge beside it; the camera (Module 4) is at the lower right. The Arduino Nano 33 BLE Sense sits at the top center with jumpers J7 and J8 just to its left, and the module selector S10 is at the bottom center. J7 and J8 choose whether A0 and A1 read the camera or the touch panel and the voltage divider; J5, in the biosensing block, chooses which photodiode reaches A6.

Blocks, silkscreen labels, and course modules

The labels printed on the board (its *silkscreen*) use the course's module numbers: the circuits block is "Module 1," the devices block "Module 2," and the biosensing block "Module 3," the blocks of Modules 1, 2, and 3 of the course. The book also calls the blocks by their function. Table 0.6 lists them. When an instruction names a label printed on the board, the label is shown in a gray plate, like Module 1; the positions of the module selector carry the same names.

The module selector and the jumpers

The *module selector* S10, at the bottom center, is a small four-position rotary switch (the board prints "Rotate to select the module" under it). Its positions are printed around it: Camera at the top, Module 1 at

Silkscreen	Block and what it holds	Module (chapter)	S10 on
"Module 1"	**circuits block** (left half): touch network with buttons S1–S9 (Network 1); 3-by-3 grid of 100 Ω resistors with test points TP1–TP9 (Network 2); voltage-divider header J1–J4	1 Circuits; the divider header also serves the thermistor lab of Module 3	Module 1
"Module 2," "Diodes and transistors"	**devices block** (upper right): diode, n-channel MOSFET, two difference amplifiers	2 Devices	Module 2
"Module 3," "Biosensing"	**biosensing block** (right middle): LED and photodiode pair 1, MAX86916 optical pulse sensor, jumper J5	3 Sensors	Module 3
"TX," "RX"	narrow box beside the biosensing block, at the right edge of the board: LED (TX) and photodiode (RX) of pair 2, the communication pair	5 Communication	Module 3
"Camera"	OV7675 camera on header J6 (lower right)	4 Machine Learning	Camera
Arduino	Nano 33 BLE Sense (top center)	0 Introduction, and all	any

Table 0.6: Demo board blocks: the board's own silkscreen label, what each block holds, the course module (and chapter) that uses it, and the selector position that powers it.

the right, Module 2 at the bottom, and Module 3 at the left (Figure 0.17). Turn the slotted dial with a small flat screwdriver until it points at the label you need. The selector powers one block at a time: Module 1 the circuits block (including the divider header), Module 2 the devices block, Module 3 the biosensing block (sensors and communication), and Camera the camera. The three numbered blocks each have a power LED and a status LED (printed M1 Power, M1 Status, and so on); the camera block has one LED. The Arduino can tell which position is selected: the switch also connects a resistor ladder to analog pin A7. On paper, with the blocks that are off drawing no current, the ladder gives about 2.2 V, 1.65 V, 1.1 V, and 0.83 V for Module 1, Module 2, Module 3, and Camera (Worked Example 0.17), and close to 0 V if no position is selected. On a real board the blocks that are off load the ladder, so the readings come out lower and vary from board to board; one board read about 1.85, 0.9, 0.65, and 0.31 V. The firmware command `module` reports the reading and its best guess of the position. The first cell of every lab notebook checks that the reading fits the position the lab needs and prints a warning if it does not; the check never stops the lab.

Three *jumpers* choose what reaches some of the Arduino's pins (Figure 0.17). Each is a row of three pins with a removable cap that connects the middle pin to one of its neighbors. J7 and J8 are the two short upright headers side by side just left of the Arduino. The board prints Camera at their top end and Module 1 at their bottom end. J7 chooses what A0 reads: with its cap on the Module 1 end (pins 2 and

3) A0 reads the touch network; on the Camera end (pins 1 and 2), the camera. J8 does the same for A1: the Module 1 end connects the output of the voltage divider, the Camera end the camera. Both caps always sit on the same end, so you never need to tell the two headers apart. Their usual place is the Module 1 end; only the camera lab moves both to the Camera end, and you move them back afterwards. J5 is a short header lying on its side in the biosensing block, next to the printed label P1-A-P2. Its middle pin, A, leads to A6. With the cap on P1–A (pins 1 and 2), A6 reads the photodiode of pair 1 (sensors); with the cap on A–P2 (pins 2 and 3), it reads the RX photodiode of pair 2 (communication).

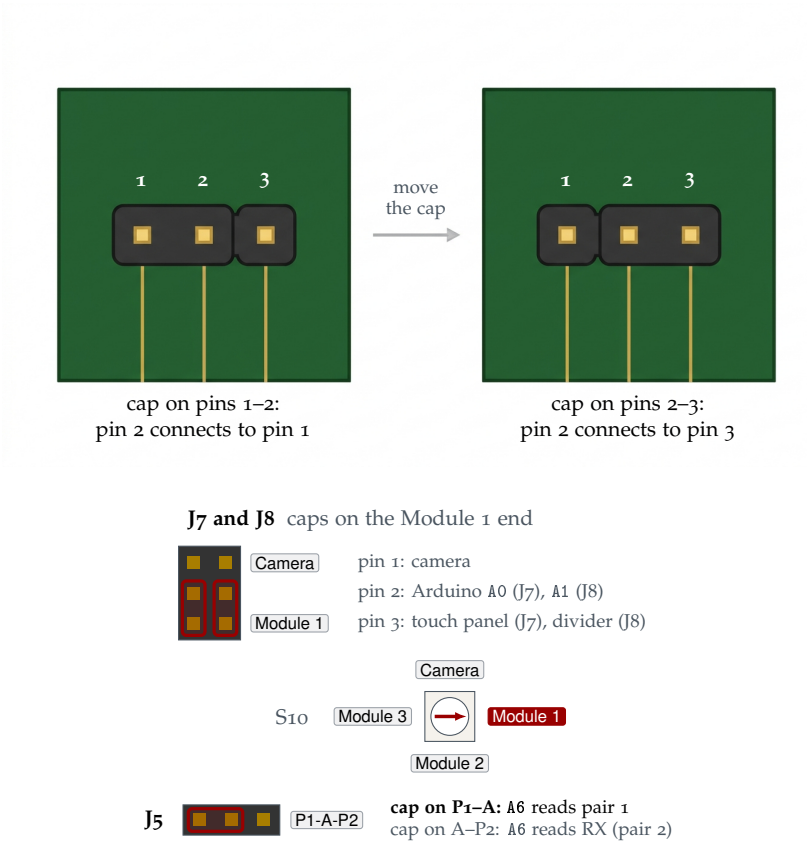


Figure 0.17: Top: a jumper is a row of three pins; the removable cap connects the middle pin to one of its neighbors. Below: the jumpers and the selector as they appear on the board. J7 and J8 are shown in their usual position, both caps on the Module 1 end (pins 2–3: touch panel on A0, divider on A1). The selector S10 is shown on Module 1. J5 is shown on P1–A, which sends the photodiode of pair 1 to A6; the receiving board of the communication lab uses A–P2.

What each block holds

The circuits block (left half) has three parts. Network 1 is the touch network of Figure 0.9: nine push buttons S1 to S9 on an asymmetric grid of 50 Ω and 100 Ω resistors, read on A0 through J7. Network 2 is a 3-by-3 grid of 100 Ω resistors with test points TP1 to TP9 for multimeter practice. The *voltage-divider header* is a column of four single-pin sockets to the right of the button grid, next to a printed divider draw-

ing labeled 3.3V, R1, A1, R2, and GND: J1 (top) carries 3.3 V, J2 and J3 are both the divider midpoint (connected to A1 through J8), and J4 (bottom) is ground. You plug one resistor (or a sensor) into J1–J2 as the top part, R1, and another into J3–J4 as the bottom part, R2. Chapter 1 analyzes this circuit, and the thermistor lab of Chapter 3 plugs a temperature sensor into it.

The devices block (upper right, "Diodes and transistors") holds a diode and an n-channel MOSFET (a DMN3404L). DAC channel 0 drives the diode through a 1 kΩ sense resistor, and a unity-gain difference amplifier (four 1 MΩ resistors and a TLV9102 op-amp) copies the voltage across that resistor to A2. The transistor's gate is driven by DAC channel 1 (with a pull-down resistor to ground), and its drain is fed from DAC channel 2 through another 1 kΩ sense resistor, with the source grounded. A second difference amplifier copies the voltage across this sense resistor to A3, so the reading on A3 in volts equals the drain current in milliamperes (Worked Example 0.10). Sweeping the DAC traces each device's current-voltage curve (Chapter 2).

The biosensing block (right middle, "Biosensing") has two LED and photodiode pairs and an optical sensor. The LED of pair 1 is driven by DAC channel 3 through a 10 Ω resistor; the LED of pair 2 by DAC channel 4. Each photodiode feeds a transimpedance amplifier (with 3 MΩ feedback), a circuit that turns the photodiode's tiny current into a voltage, and J5 picks which one reaches A6. Pair 1 (LED D3 and photodiode D4) sits side by side at the lower left of the block, the LED shining across onto the photodiode; pair 2 is TX and RX at the board edge. The MAX86916 measures red and infrared light reflected from a fingertip and sends the numbers over I^2C. Pair 1 and the MAX86916 belong to the sensors module (Chapter 3); pair 2 is the communication pair used by the light link of Chapter 5.

The camera (an OV7675) plugs into header J6 at the lower right and is used for image classification in Chapter 4. It shares A0 and A1 with the circuits block through J7 and J8.

The communication pair sits in a narrow box at the right edge of the board, beside the biosensing block: the LED of pair 2, printed TX, near the top, and its photodiode, printed RX, near the bottom. Both are side-view parts that face out over the board edge, so two boards talk to each other edge to edge (Chapter 5).

How the Arduino talks to the board

Figure 0.18 shows the connections. Simple voltages go straight to the Arduino's analog input pins, which digitize them with the 10-bit ADC over 0 to 3.3 V. The two smart chips share the I^2C bus: two wires, data (SDA, pin A4) and clock (SCL, pin A5), over which bytes are sent to an

address that names the chip (`0x10` for the AD5593R DAC, `0x57` for the MAX86916). Everything reaches your laptop through the USB cable, which also powers the board.

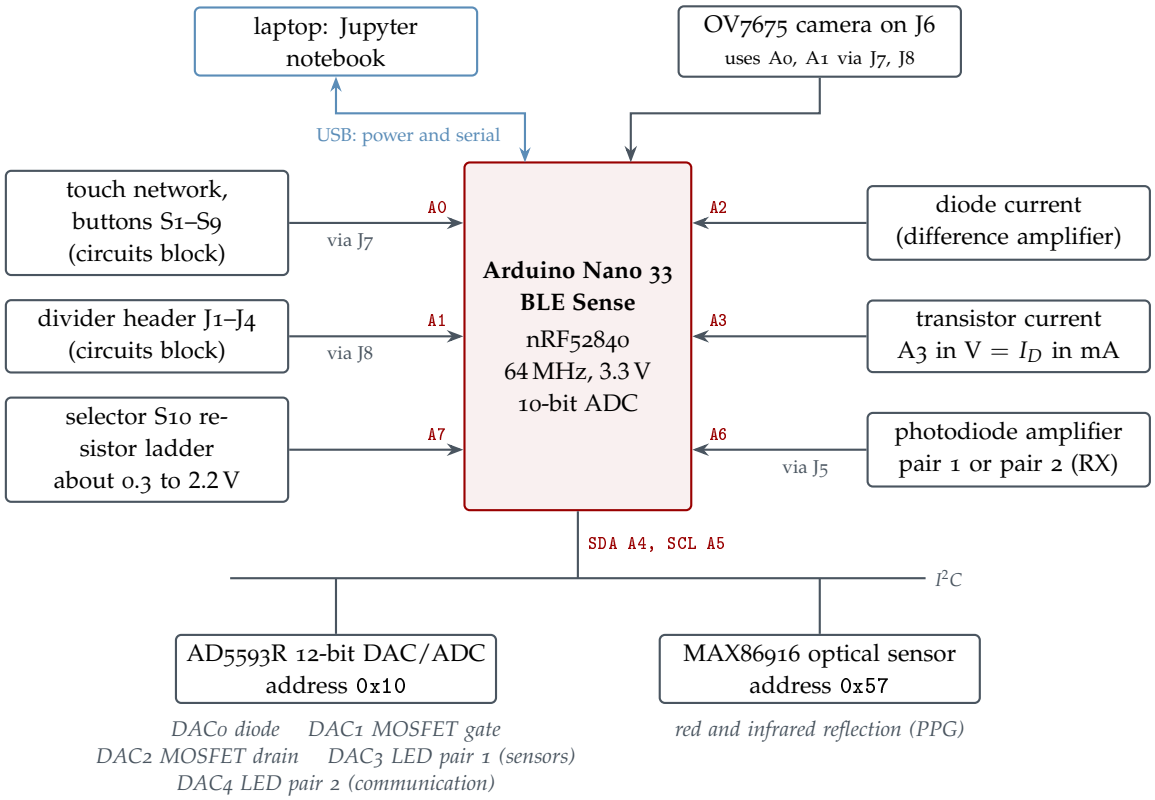


Figure 0.18: How the demo board connects to the Arduino. Analog voltages reach pins A0 to A7 (A0, A1, and A6 through jumpers); the DAC and the optical sensor share the I^2C bus; the laptop connects over USB.

The course firmware and its commands

One Arduino program, the course firmware, runs almost every hardware lab (the camera lab uploads a camera program of its own, and in Lab 0.1 you write your own sketches), and the lab notebooks upload it for you when the board needs it. Most of the time a notebook talks to the firmware in a compact protocol of its own. The firmware also understands short descriptive commands (Table 0.7), which you can send from a notebook cell, for example `board.monitor("circuits")`, or from any serial terminal at 115200 baud. Each command runs until it finishes or until you send `stop`. Turn the selector to the right position first; `module` tells you where it is.

Command	Selector	What the firmware does
`circuits`	Module 1	names the touch-panel button that is pressed and prints its node voltage
`divider`	Module 1	prints the divider output voltage on A1, a few times per second
`temperature`	Module 1	prints the thermistor temperature from the divider (Lab 3.1)
`diodes`	Module 2	sweeps the diode: DAC voltage, diode voltage, current
`transistors`	Module 2	MOSFET output curves, I_D versus V_{DS} for several V_{GS}
`transfer`	Module 2	MOSFET transfer curve, I_D versus V_{GS}
`light`	Module 3	turns on LED pair 1 and prints the photodiode reading (J5 on P1–A)
`sensors`	Module 3	streams red and infrared PPG readings from the MAX86916
`communications send` *text*	Module 3	sends the text as light pulses from the TX LED (pair 2)
`communications receive`	Module 3	decodes light pulses on the RX photodiode (J5 on A–P2) and prints the text
`module`	any	reports the selector position read on A7
`stop`	any	stops the running command
`help`	any	lists the commands

Table 0.7: Commands of the course firmware (115200 baud, one command per line), with the selector position each one needs.

WORKED EXAMPLE 0.17 *Which position is the selector in?*

On paper, with the blocks that are off drawing no current, the selector ladder puts about 2.2 V, 1.65 V, 1.1 V, or 0.83 V on A7 for the positions Module 1, Module 2, Module 3, and Camera. (a) What ADC code does each produce? (b) A simple decision rule uses thresholds halfway between neighboring voltages (and halfway to 0 V below Camera). Find the thresholds. (c) A7 reads code 520, then 270, then 100. Which positions are these?

SOLUTION. (a) With code $= V/3.3 \times 1023$: 2.2 V gives 682, 1.65 V gives 511.5 (511 or 512), 1.1 V gives 341, and 0.83 V gives 257.

(b) The midpoints are $(2.2 + 1.65)/2 = 1.925$ V, $(1.65 + 1.1)/2 = 1.375$ V, $(1.1 + 0.83)/2 = 0.965$ V, and $0.83/2 = 0.415$ V. Above 1.925 V is Module 1; between 1.375 V and 1.925 V, Module 2; between 0.965 V and 1.375 V, Module 3; between 0.415 V and 0.965 V, Camera; below 0.415 V, nothing selected.

(c) Code 520 is $520 \times 3.3/1023 = 1.677$ V: Module 2 (devices). Code 270 is 0.871 V: Camera. Code 100 is 0.323 V: no position selected. On paper the closest pair of levels, Module 3 and Camera, is 0.27 V (about 84 codes) apart, far more than the ADC noise. On a real board the blocks that are off load the ladder and pull every level down by an amount that varies from board to board, so a Module 1 reading can land near the paper value for Module 2. That is why the lab notebooks accept a wide range for each position and only print a warning when a reading does not fit. ■

0.9 The Arduino Microcontroller

ARDUINO is a family of open-source microcontroller boards (Uno, Leonardo, Mega, LilyPad, Nano, and others) with a simple programming environment, the Arduino IDE. A microcontroller is a small computer on one chip, with processor, memory, and input/output hardware, built to sense and control things. Arduino became a standard tool for rapid prototyping because the boards are cheap, a large community has written libraries for almost every sensor, and connecting hardware takes a few lines of code. It is a gentle entry point into *embedded systems*, the computers hidden inside products.

The Nano 33 BLE Sense

EE 105 uses the Arduino Nano 33 BLE Sense. Its processor, the Nordic nRF52840, is a 32-bit ARM Cortex-M4F running at 64 MHz, with 1 MB of flash memory for your program and 256 KB of SRAM for data. It has

a Bluetooth 5 radio, and it carries its own sensors: the APDS9960 (ambient light, color, proximity, and gestures), the LPS22HB (air pressure), a temperature and humidity sensor, an inertial measurement unit (acceleration, rotation, and magnetic field; this sensor's model and the temperature and humidity sensor's depend on the board revision), a PDM microphone, and an RGB LED. Table 0.8 compares it with the Arduino Uno that many people learn on first.

	Arduino Uno	Nano 33 BLE Sense
Processor	ATmega328P (8-bit)	nRF52840 (32-bit Cortex-M4F)
Clock	16 MHz	64 MHz
Flash / SRAM	32 KB / 2 KB	1 MB / 256 KB
Operating voltage	5 V	3.3 V
Current per I/O pin	20 mA	15 mA
Digital I/O pins	14	14
PWM	6 pins	all digital pins
Analog inputs	6	8
Analog output	PWM only	PWM only (no DAC)
Wireless	none	Bluetooth 5

Table 0.8: The Arduino Uno and the Nano 33 BLE Sense.

The most important line in that table for everyday work is the operating voltage. The Nano runs at 3.3 V. Its logic high is 3.3 V, its ADC reference is 3.3 V, and you must never apply more than 3.3 V to one of its pins. Many tutorials written for the 5 V Uno, including Arduino's own `ReadAnalogVoltage` example, use 5.0 in their voltage conversions; on this board replace it with 3.3, or every reading will be 52% too high ($5.0/3.3 = 1.52$).

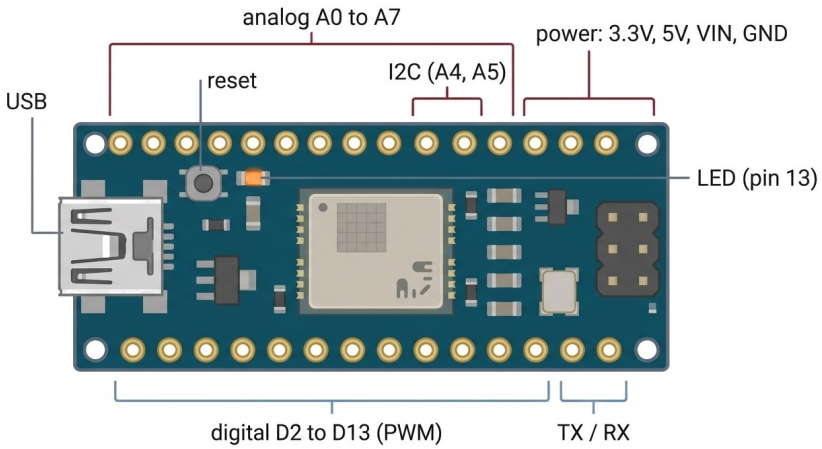


Figure 0.19: The Arduino Nano 33 BLE Sense and its pin groups: digital pins D2–D13 (all PWM capable), analog inputs A0–A7, power pins (3.3 V, 5 V, VIN, GND), the I^2C pins A4 (SDA) and A5 (SCL), serial pins TX and RX, the USB connector, the reset button, and the built-in LED on pin 13.

The pins run along both long edges (Figure 0.19). One side has digital pins D2–D13 plus RX and TX (D0 and D1); every digital pin can produce PWM, marked with a tilde (~). The other side has analog

inputs A0–A7 (A4 and A5 double as I^2C) and the power pins. The built-in LED is on pin 13.

The structure of a sketch

An Arduino program is called a *sketch*. It is written in C++ with a few simplifications, and it always has two functions (Figure 0.20). `setup()` runs once when the board powers up or resets; use it to configure pins and start the serial port. `loop()` then runs over and over, forever. Here is the classic first sketch, which blinks the built-in LED:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);     // pin 13 is an output
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // LED on (pin at 3.3 V)
  delay(1000);                      // wait 1000 ms
  digitalWrite(LED_BUILTIN, LOW);   // LED off (pin at 0 V)
  delay(1000);
}
```

The LED is on for one second and off for one second, so one full cycle takes 2 s and the blink frequency is $1/2\,\text{s} = 0.5\,\text{Hz}$. A common slip is to read `delay(1000)` as a 1 Hz blink. In general, with equal on and off times t_{half},

$$f = \frac{1}{2\,t_{\text{half}}}, \qquad t_{\text{half}} = \frac{1}{2f}. \tag{0.8}$$

Declaring `const int ledPin = 13;` and writing `ledPin` in place of the built-in name does exactly the same thing on this board. Giving pins names at the top of a sketch is a good habit: when the wiring changes, you edit one line.

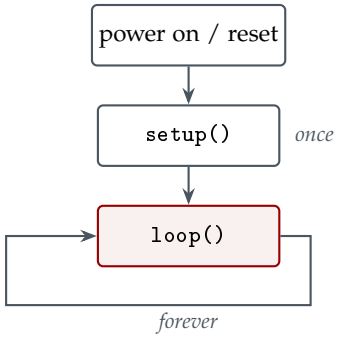


Figure 0.20: Every sketch runs `setup()` once, then repeats `loop()` until power is removed.

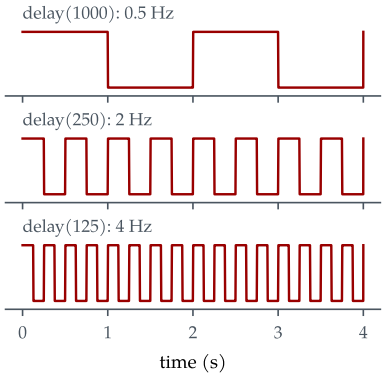


Figure 0.21: LED state for three values of the delay between toggles. Halving the delay doubles the frequency.

Digital outputs and inputs

A pin set with `pinMode(pin, OUTPUT)` is driven to 3.3 V by `digitalWrite(pin, HIGH)` and to 0 V by `LOW`; it can supply about 15 mA, enough for an LED with a series resistor but not a motor. As an `INPUT`, `digitalRead(pin)` returns `HIGH` or `LOW`. An unconnected input "floats" and reads at random, so buttons usually use `INPUT_PULLUP`, which ties the pin to 3.3 V through an internal resistor until the button pulls it to ground.

`delay(ms)` freezes the whole program, so the board cannot read a sensor while it waits. Instead, keep time with `millis()`, the milliseconds since start-up, and act when enough time has passed:

```
unsigned long lastToggle = 0;
bool ledOn = false;

```

```
void loop() {
  if (millis() - lastToggle >= 500) {   // 500 ms has passed
    lastToggle = millis();
    ledOn = !ledOn;
    digitalWrite(LED_BUILTIN, ledOn ? HIGH : LOW);
  }
  // ... the loop is free to do other work here ...
}
```

Analog input and PWM output

`analogRead(pin)` on A0–A7 returns the 10-bit ADC code from Section 0.7, from 0 for 0 V to 1023 for 3.3 V. Convert it with Eq. (0.6). `analogWrite(pin, value)` on a digital pin produces PWM with duty cycle value/255, so `analogWrite(LED_BUILTIN, 64)` lights the LED at about one quarter brightness (Worked Example 0.16). The helper `map(x, inLow, inHigh, outLow, outHigh)` rescales a number from one range to another with integer arithmetic, which is handy for turning a 10-bit reading into an 8-bit PWM value.

The serial port

The USB cable also carries a serial link, a stream of bytes in each direction. `Serial.begin(115200)` opens it at a speed in bits per second (baud) that both ends must share; the course firmware uses 115200. `Serial.print(x)` sends a value as text and `Serial.println(x)` adds a newline. In the other direction, `Serial.available()` counts waiting bytes and `Serial.readStringUntil('\n')` reads a line; the course firmware reads its commands this way.

The Arduino IDE has a *Serial Monitor*, which shows the text and lets you type commands back, and a *Serial Plotter*, which graphs numbers against time, one line per time step, with comma-separated values drawn as separate traces. In the EE 105 lab kit, notebook cells do both jobs (Section 0.11). Only one program can open the port at a time, so close any other program that is using the board before a notebook connects to it.

Libraries and the on-board sensors

A library is a package of code that hides the details of a chip. The on-board sensors use `Arduino_APDS9960`, `Arduino_LPS22HB`, and `PDM` (the last comes with the board package); the IMU library must match the board revision printed on your board. In the Arduino IDE you install a library from Sketch → Include Library → Manage Libraries; in the lab kit, one notebook call such as `ee105.install_library("Arduino_LPS22HB")` does it.

A few words of C

Arduino code is C++, and a few C rules trip up people coming from Python. Statements end with a semicolon and blocks use braces, not indentation. Variables have declared types (`int`, `float`, `bool`, `unsigned long`), and a function is declared with its return type and typed parameters, as in `int add(int a, int b) { return a + b; }`. Integer division drops the fraction (`5/2` is 2, and `3/1023` is 0). And `^` is not a power operator in C; use `pow(base, exponent)`.

Worked examples that put these pieces together with the board's sensors follow the Arduino lab (Section 0.11).

0.10 Python for Engineers

The Arduino is good at collecting data in real time, but its memory is small and it has no screen. For analysis, plotting, and machine learning you will use Python on your laptop, mainly with NumPy (arrays and linear algebra), Matplotlib (plots), and SymPy (symbolic math).

You will run Python in *Jupyter notebooks*, which mix code cells, their output, and formatted text in one document. You run one cell at a time and adjust, which suits lab work. Every EE 105 lab is one notebook (Section 0.11). Worked examples on everyday data (temperatures, step counts, prices), which let you focus on the Python, follow the Python lab (Section 0.12).

Variables and containers

Python variables need no type declaration: `x = 10` makes an integer, `y = 3.14` a float, `name = "Alice"` a string, and `is_on = True` a boolean. Four built-in containers hold collections of values, and each fits a different job.

A *list* (square brackets) is an ordered, changeable sequence, such as `temps = [21.5, 23.0, 19.8]`, indexed from zero. A *tuple* (parentheses) is an ordered sequence that cannot change after creation, such as a coordinate `(10, 20)`. A *dictionary* (braces and colons) stores values under meaningful keys, as in `sensor = {'temp': 24.5, 'battery': 0.25}`, read back with `sensor['temp']`. A *set* (braces, no colons) keeps only unique values and supports intersection (`&`), union (`|`), and difference (`-`).

Control flow and list comprehensions

Python marks blocks by indentation. Conditionals use `if`, `elif`, and `else`; loops use `while` and `for`, with `break` to exit early and `continue` to skip to the next pass. `range(start, stop, step)` stops *before* `stop`.

A *list comprehension* builds a list from a loop in one line. The pattern is `[expression for item in sequence if condition]`. For example, the squares of the even numbers from 0 to 10 are

```
squares = [i**2 for i in range(11) if i % 2 == 0]
# squares is [0, 4, 16, 36, 64, 100]
```

which replaces a four-line loop. Note that Python, unlike C, uses `**` for powers. A *dictionary comprehension* works the same way with a key and a value, as in `{d: len(d) for d in ['Mon', 'Tue']}`.

NumPy arrays

For vectors and matrices, use NumPy. Its central object is the *array*, made with `np.array([1, 2, 3])` or constructors such as `np.zeros`, `np.eye` (identity), `np.linspace(0, 9, 10)` (ten points from 0 to 9), and `np.arange(0, 10, 1)`. Arithmetic on arrays acts element by element: for lists, `[1, 2, 3] + [1, 2, 3]` joins them into six elements, while for arrays it gives `[2, 4, 6]`, and far faster than a loop. A comparison such as `a > 5` gives an array of `True` and `False`, a *mask*: `a[a > 5]` keeps the elements where the mask is true, and `(a > 5).sum()` counts them.

Slicing picks out parts of an array with `start:stop:step`. As with `range`, the start is included and the stop is excluded, and any part can be left out: `a[:10]` is the first ten elements, `a[-10:]` the last ten, `a[::2]` every other element. For a 2-D array, give one slice per axis, rows first: `A[0]` is the first row, `A[:, 0]` the first column, `A[:3, :3]` the top-left 3×3 block. The `.shape` attribute gives the dimensions, and `.reshape()` rearranges the same data into a new shape.

Other workhorses are `A.T` (transpose), `A @ B` or `np.dot` (matrix product, where order matters), `np.linalg.solve`, `np.vstack` and `np.hstack` (joining), `np.argmax` and `np.argmin` (position of extremes), and `np.argsort`. Most take an `axis` argument: `axis=0` works down the columns, `axis=1` across the rows.

Solving linear equations

Many problems end in a set of linear equations $A\mathbf{x} = \mathbf{b}$: prices from receipts, the amounts in a mixture, the two numbers of a calibration line, and, in Chapter 1, the currents in a circuit. `np.linalg.solve(A, b)` solves them in one line, faster and more accurately than computing

`np.linalg.inv(A) @ b`. A quick check is that `A @ x` reproduces `b`. SymPy solves the same equations symbolically when you want exact fractions. Worked Example 0.25, after the Python lab, solves one such system.

Plotting and reading the board

Matplotlib turns arrays into figures: `plt.plot(x, y)`, axis labels with quantity and unit via `plt.xlabel()` and `plt.ylabel()`, then `plt.show()` or `plt.savefig()`. Label every axis with the quantity and its unit, such as "time (s)" or "voltage (V)".

In the labs, Python talks to the board through the course toolkit, a small package that ships with the lab kit. `board = ee105.connect()` finds the board on the USB port and uploads the course firmware if needed, so you never pick a serial port by hand. `board.read(1, n=200, interval_ms=5)` asks for 200 readings of pin A1, 5 ms apart, and returns them as a table (a pandas DataFrame) with the time, the raw ADC code, and the voltage. `board.monitor("divider")` shows the text that a firmware command prints. Underneath, the toolkit uses the pySerial package to open the same serial port the Arduino IDE would use, and as with the IDE, only one program can have that port open at a time.

KEY IDEA The Arduino measures in real time; Python analyzes and plots. The serial port over USB is the link between them.

0.11 *Laboratory: Arduino and the Demo Board*

LAB 0.1 AT A GLANCE

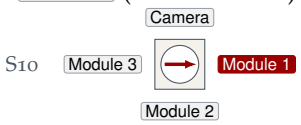
Goal: program the Nano 33 BLE Sense on the demo board from a notebook: blink the LED at chosen frequencies, print and plot sensor readings, read voltages from the circuits block, and set the LED brightness with PWM.
Hardware: EE 105 demo board with the Arduino Nano 33 BLE Sense, USB data cable, laptop.
Notebook: `lab0.1-arduino-board.ipynb` in the folder `lab0.1-arduino-board` of the lab kit. The `ee105` folder next to the lab folders holds the course toolkit; leave it where it is.
Before the lab: run the kit's setup file at home ("How the lab kit works" below; 15 to 30 minutes and an internet connection), and read Sections 0.7 and 0.9.
Time: about 2 hours.
You hand in: `lab0.1_submission.json` on the course portal; keep your notebook with your answers.

Key takeaways

- Equal on and off times give $f = 1/(2t_{\text{half}})$: two `delay(1000)` calls blink at 0.5 Hz, and 4 Hz needs halves of 125 ms.
- The 10-bit ADC maps 0 to 3.3 V onto codes 0 to 1023, so $V = \text{code} \times 3.3/1023$; the 5 V formula reads 1.52 times too high.
- One meter of height changes the air pressure by about 12 Pa; the sensor scatters by 1 to 5 Pa, so it resolves a meter.
- The nine buttons read within about 30 mV of nodal analysis; with no button pressed, A0 floats and wanders over a wide range.
- PWM sets an average: duty d of 255 gives $d/255 \times 3.3$ V on average, so S5's duty of 127 is half brightness.

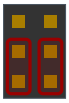
Set up the board for Lab 0.1

Selector. Turn S10 to Module 1 (circuits block):



Jumpers. Put the caps of J7 and J8, the two headers just left of the Arduino, on the Module 1 end (pins 2 and 3), so A0 reads the touch network:

J7 and J8 caps on the Module 1 end

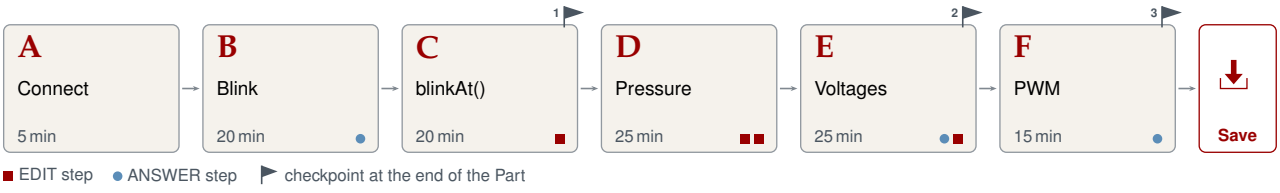


pin 1: camera
pin 2: Arduino `A0` (J7), `A1` (J8)
pin 3: touch panel (J7), divider (J8)

Plugged in. Nothing; leave the divider header J1 to J4 empty.

Check. The first notebook cell asks the firmware for the selector position and prints a line beginning with `[ok]` that says S10 is on `Module 1` (circuits block). If the reading does not fit Module 1, the cell prints a `[!!]` warning instead; check that S10 points at Module 1 and carry on.

Figure 0.22 shows where S10, J7 and J8, and the touch panel are on the board.



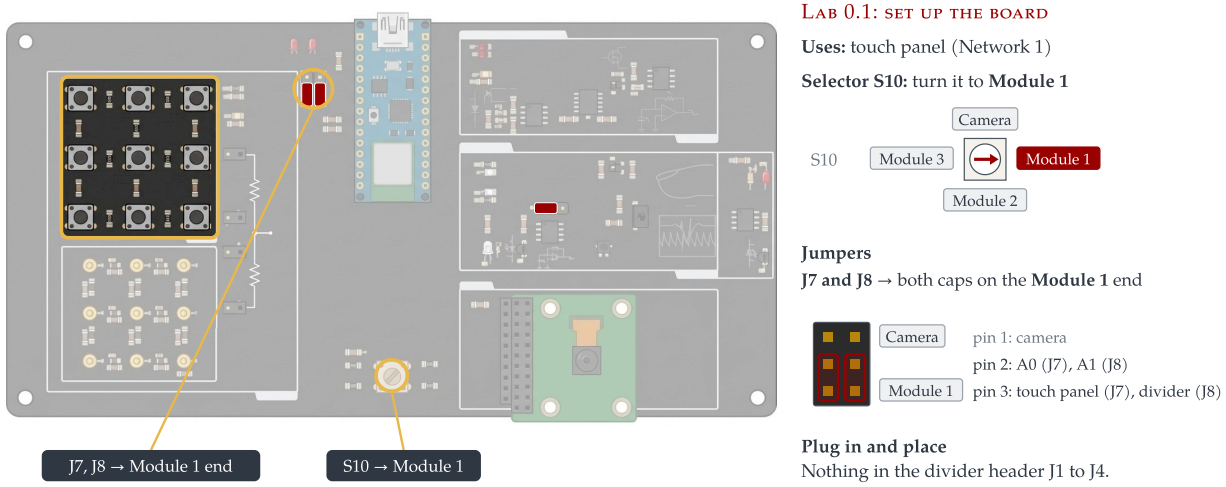
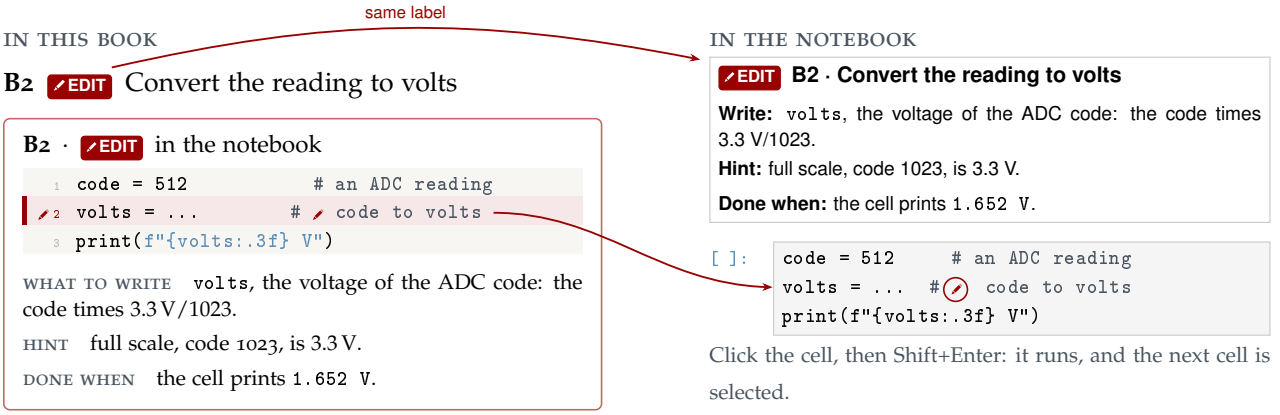


Figure 0.22: The board for Lab 0.1. The touch panel (highlighted) is the only block the lab reads. S10 is on Module 1, and the caps of J7 and J8 are on the Module 1 end; the divider header stays empty.

How the labs work

This book is the lab manual. Each lab is one section of it: how to set up the board, every step with its pictures, the numbers you should see, the questions, and a checklist at the end. The lab kit, one zip file ("How the lab kit works" below), holds the software: a notebook for each lab. Every code cell of a notebook is one step of the book, with the same label, and the cell above it shows the step's tag, a short line that says what the step asks, and the book's picture of the action where there is one. The notebook shows the essentials of each step; the book has the full story: the explanations, the theory, the expected numbers, the troubleshooting, and the questions. The book and the zip file are all you need. Keep the book open next to your laptop and work through both together.



The same step in this book (left) and in the notebook (right): the same label and badge, the same three lines in short, and the line to change marked with the writing hand. Where the book shows a picture tagged EXPECTED, the notebook shows it above the cell.

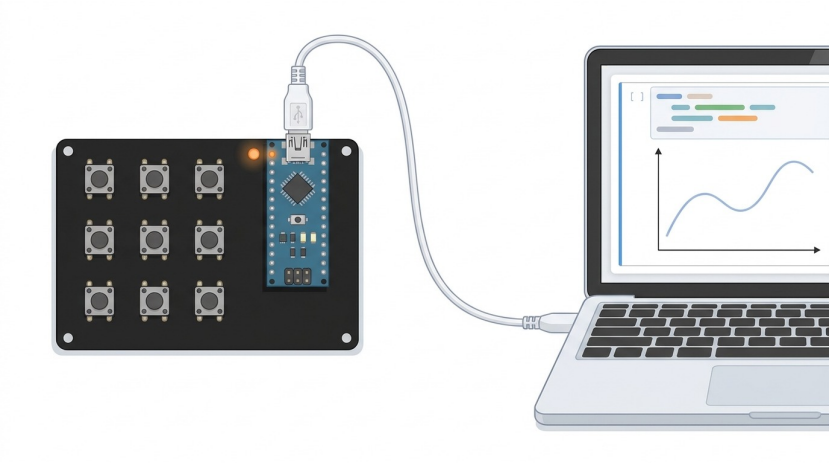


Figure 0.23: Lab 0.1 at the bench. One USB data cable powers the board and carries the data. The notebook uploads your sketch, the orange LED next to the USB connector blinks, and the plot cell draws what the board prints.

The label of a step, a Part letter and a number such as B2, stands at the start of the step in the book and at the top of the cell above its code in the notebook. Its tag says what the step asks of you:

- ▸ RUN — run the cell as it is.
- ◆ DO — do something at the bench first, such as plugging in a cable or resting a finger on a sensor, then run the cell. A picture in the margin, and in the notebook, shows the action.
- ✎ EDIT — change the code first. In the notebook, the lines to change carry the writing-hand sign at the end (this book draws it as ✎). The book prints the cell with those lines tinted, then says what to write, gives a hint, and says when you are done; the notebook repeats the three in short.
- ⊡ ANSWER — type a value into the cell, one you read off a plot or a meter; the book and the notebook name the variable, its unit, and where to find it.
- ✓ CHECK — a checkpoint: compare your results with the criterion before you go on, and in a lab session show them to the TA. The notebook repeats the criterion in a CHECK cell after the step that closes the checkpoint, and where it can check, it prints a line that starts with ✓.

To run a cell, click it and press Shift+Enter. Run the cells in order, from the top: a cell uses what the cells above it computed, so after a break or a restart, run them again from the top. A "You should see" box gives what a working board shows (the notebook quotes its first sentence); if yours differs, the Troubleshooting list of the lab helps. Each lab opens with its *Key takeaways*, and a picture tagged EXPECTED,

in the margin of this book and above the cell in the notebook, shows what a step's cell drew on the simulated board: compare its shape and ranges with yours, not the exact numbers. The lab map at the start of each lab shows its Parts, their times, the EDIT and ANSWER steps, and the checkpoints; the notebook opens with the same map, drawn with every step, and with the board setup picture and list. Steps that stream data from the board plot it live: the pulse oximeter marks your beats and shows your heart rate, the touch panel lights up the pressed pad, the thermometer has a thermometer bar, and the diode and transistor sweeps grow point by point.

Your work goes into the notebook. EDIT and ANSWER steps are done in their cells. The Analysis questions of a lab are answered in the Questions cells at the end of its notebook: double-click a cell to type, and press Shift+Enter to show it formatted again. The last step, Save (SAVE on the lab map), writes the file you hand in, `lab0.1_submission.json` for this lab, next to the notebook; you upload it on the course portal. Keep the notebook, with your answers in it.

Every lab ends with a box *Finish the lab*: a checklist with one line for each EDIT and ANSWER step, the recordings, the checkpoints, the questions, and the upload; the notebook ends with the same list under the heading *Finished?*. The lab is finished when every box is ticked.

Objectives

- Set up the lab kit, then compile and upload a sketch to the Nano 33 BLE Sense from a notebook.
- Control the blink frequency of the built-in LED and relate the delays to the frequency, $f = 1/(2t_{\text{half}})$.
- Print sensor data over the serial port, then record and plot it in Python.
- Read voltages from the circuits block and convert them correctly for a 3.3 V, 10-bit ADC.
- Drive the LED brightness with PWM.

How the lab kit works

Every lab in this book runs from one lab kit, a single zip file that holds a folder for each lab, the course toolkit, the board firmware, and the data sets. Unzip it once and keep its folders together. Then run its setup file: `setup-windows.bat` on Windows or `setup-mac.command` on a Mac (on a Mac, the first time, right-click the file, choose Open, and confirm with Open). The setup installs everything the labs need: a Python environment with JupyterLab and the scientific packages, the Arduino command-line tools, and the board package for the Nano 33 BLE Sense. It then starts JupyterLab in the kit folder. The first run takes 15 to 30 minutes and needs an internet connection and about 5 GB of free disk space, so do it at home with a good connection. It writes every step to `setup-log.txt` in the kit folder; if it stops, the last lines of that file say why, and running it again skips the steps already done. Starting the same file again later just opens JupyterLab with your labs, in a few seconds. Keep its window open while you work.

Each lab is one Jupyter notebook in its own folder, named after the lab (`lab0.1-arduino-board`, `lab0.2-python-bootcamp`, and so on). The notebook does the work (Figure 0.23): it connects to the board, uploads the course firmware when needed, runs the measurements, plots them, and holds your analysis and answers. You never open the Arduino IDE or choose a serial port. To practice without a board, set `SIMULATE = True` in the notebook's first code cell; the notebook then runs on a simulated board with realistic but made-up data.

The notebook's last cell saves your results next to the notebook in one file named after the lab: `lab0.1_submission.json` for Lab 0.1, `lab0.2_submission.json` for Lab 0.2, and so on. To submit a lab, log in to the course portal at `ee105.org/portal` with your student ID and portal PIN, open the lab, and upload that one file. It is graded automatically and the grade is recorded. You can upload again after a fix. For a lab that uses the board, a file from a simulated run counts as practice and earns no credit. The same portal holds the online quizzes, one per module; each draws its questions at random from a question bank, is timed, and is graded as soon as you submit it.

Background

Read Sections 0.7 and 0.9. In this lab you write your own Arduino sketches inside the notebook, the only lab of the course where you do. A cell that begins with `%%arduino` *name* is a sketch: running the cell saves it as `sketches/`*name*`/`*name*`.ino` in the lab folder, compiles it, and uploads it to the board. The option `--monitor 6` then shows what the board prints for 6 seconds, and `--plot 20` plots it for 20 seconds, which replaces the IDE's Serial Monitor and Serial Plotter.

No IDE needed. In this kit a notebook cell that begins with `%%arduino` *name* is an Arduino sketch. Running the cell compiles it, uploads it, and can show what the board prints. The Arduino IDE, its Serial Monitor, and its Serial Plotter are replaced by notebook cells.

Your sketch replaces the course firmware on the board; the next lab puts the firmware back by itself.

Every sketch has two functions (Figure 0.20): `setup()` runs once after power-up or reset, then `loop()` runs over and over. Equal on and off times t_{half} give a blink period $T = 2t_{\text{half}}$ and the frequency $f = 1/(2t_{\text{half}})$ of Eq. (0.8) (Figure 0.21). The built-in LED is on pin 13, next to the USB connector (Figure 0.19). The Nano's 10-bit ADC maps 0 to 3.3 V onto the codes 0 to 1023, so one step is $3.3\,\text{V}/1024 \approx 3.22\,\text{mV}$ and a reading converts as $V = \text{code} \times 3.3/1023$, Eq. (0.6). Many Arduino examples assume a 5 V board; on this board that conversion is wrong. `analogWrite(pin, duty)` produces PWM: the pin switches between 0 and 3.3 V quickly, and its average is $D \times 3.3\,\text{V}$ with the duty cycle $D = \text{duty}/255$, Eq. (0.7) (Figure 0.14).

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **B1**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT or ANSWER needs your input in the cell first ("How the labs work" above explains the tags).

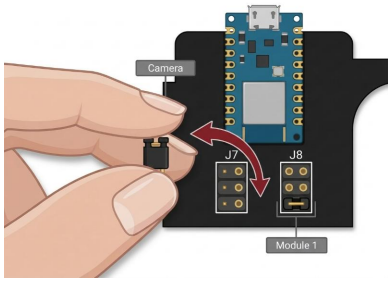
Part A: Connect

GOAL Connect the board and get every line of the first cell to start with `[ok]`. about 5 min

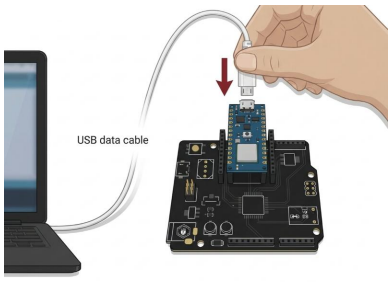
A1 DO Connect the board

Set up the board as in the box above: S10 on Module 1, and the caps of J7 and J8 (the two small headers just left of the Arduino) on the Module 1 end, pins 2 and 3. Plug the Nano into your laptop with a USB cable that carries data (a charge-only cable lights the board but cannot upload). Click the first code cell, A1, and press Shift+Enter. The cell checks the Arduino tools (the setup file installed them; if it did not, the cell downloads them now, about 200 MB), connects to the board, uploads the course firmware if the board needs it, and reads the selector position. On Windows, allow the driver installation if a dialog appears. No board today? `SIMULATE = True` in this cell runs the Python cells on a simulated board, but the simulator cannot run the sketches you write: their cells only save them.

YOU SHOULD SEE every line starts with `[ok]` (apart from progress lines while something installs or uploads): the tools, the board (firmware `ee105`), and S10 on `Module 1` (circuits block). A `[!!]` line about the selector is only a warning: if S10 points at Module 1, carry on.



A1 J7 and J8 are single rows of 3 pins. Put each cap on pins 2 and 3, the Module 1 end. (The picture draws the two headers two pins wide.)



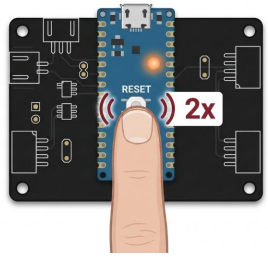
A1 Plug the Nano into your laptop with a USB *data* cable.

Part B: Blink

GOAL Blink the LED, time ten blinks, and compare the frequency with the prediction. about 20 min

B1 DO Upload Blink

B1 is the classic Blink sketch of Section 0.9. `%%arduino blink` saves the cell as `sketches/blink/blink.ino`, compiles it, and uploads it. Run it and watch the orange LED next to the USB connector. The first compile of the board package takes 1 to 3 minutes, later compiles about 20 s; do not restart a cell that is still compiling. If the upload fails, double-tap the white RESET button on the Nano (the orange LED starts to pulse slowly) and run B1 again.



B1 If an upload fails, double-tap the white RESET button: the orange LED pulses slowly. Then run the cell again.

YOU SHOULD SEE the orange LED on for 1 s, off for 1 s, over and over: 0.5 Hz, not 1 Hz.

B2 ANSWER Time ten blinks at three delays

B2 · ANSWER in the notebook

```
1  # delay in ms -> time for ten full blinks in seconds
2  ten_blinks_s = {1000: None,   # ✎ s, at delay(1000)
3                  250: None,    # ✎ s, at delay(250)
4                  125: None}    # ✎ s, at delay(125)
5
6  for t_half_ms, t10 in ten_blinks_s.items():
7      f_pred = 1 / (2 * t_half_ms / 1000)
8      f_meas = 10 / t10 if t10 else float("nan")
9      print(f"delay {t_half_ms:4d} ms: predicted {f_pred:.2f} Hz,"
10           f" measured {f_meas:.2f} Hz")
```

TYPE IN `ten_blinks_s`: the time for ten full blinks (on and off) at each delay, in place of each `None` on lines 2 to 4. Time the 1000 row first. Then change *both* delays of **B1** to 250, run B1 again, time ten blinks, and fill in the 250 row; then do the same with 125. Run B2 after each row: it prints the measured frequency $10/t$ next to the prediction $f = 1/(2t_{\text{half}})$, Eq. (0.8).

UNIT seconds.

WHERE TO FIND IT your phone's stopwatch. Start it as the LED turns on and stop it at the tenth turn-on after that. At `delay(1000)` one blink takes 2 s, so ten take about 20 s.

YOU SHOULD SEE ten blinks in about 20, 5, and 2.5 s: about 0.50, 1.98, and 3.85 Hz against the predicted 0.5, 2, and 4 Hz (Figure 0.21). At 4 Hz the measured frequency comes out a few percent low (Question 1).

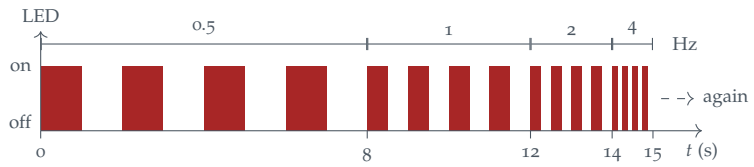


Figure 0.24: What `blinkAt()` should do (step C1). One pass of `loop()` blinks four times at each of 0.5, 1, 2, and 4 Hz; the LED is on for $t_{\text{half}} = 1/(2f)$ and off for as long. Four blinks take $4/f$, so the pass lasts $8+4+2+1 = 15$ s and then starts again.

Part C: `blinkAt()`: blink at any frequency

GOAL Write one function that blinks the LED at any frequency you ask for. about 20 min

C1 EDIT Write `blinkAt(float freqHz)`

C1 · EDIT in the notebook

```
1  %%arduino blink_at
2  const int ledPin = 13;
3
4  void blinkAt(float freqHz) {
5    // --> YOUR CODE HERE: the half period in ms, then
6    // --> LED on, wait, LED off, wait
7  }
8
9  void setup() {
10   pinMode(ledPin, OUTPUT);
11 }
12
13 void loop() {
14   float freqs[] = {0.5, 1.0, 2.0, 4.0};
15   for (int k = 0; k < 4; k++) {
16     for (int n = 0; n < 4; n++) blinkAt(freqs[k]);
17   }
18 }
```

WHAT TO WRITE the body of `blinkAt`, in place of the comment on lines 5 and 6: one blink, on and then off, at the frequency `freqHz`. Compute the half period in milliseconds, then turn the LED on, wait, turn it off, and wait. The sketch names its pin, `const int ledPin = 13;` (pin 13 is the same LED as `LED_BUILTIN`), and the given `loop()` blinks four times at each of 0.5, 1, 2, and 4 Hz.

HINT `delay()` takes milliseconds, so $t_{\text{half}} = 1000/(2f)$ ms by Eq. (0.8): 125 ms at 4 Hz, 1000 ms at 0.5 Hz. Keep it in an `unsigned long`, and use `digitalWrite(ledPin, HIGH)` and `LOW`.

DONE WHEN the LED blinks four times slowly, four times faster, four faster again, and four fastest, as in Figure 0.24: one pass of `loop()` takes $8+4+2+1 = 15$ s, then it repeats.

CHECK **Checkpoint 1** The LED blinks four times each at 0.5, 1, 2, and 4 Hz with your `blinkAt()`, a 15 s cycle. Your first reusable Arduino function.

Part D: Print and plot sensor readings

GOAL Read the pressure sensor, plot it, and find out whether it can sense one meter of height. about 25 min

The Nano 33 BLE Sense carries a barometric pressure sensor, the LPS22HB, on the board itself. Near sea level air pressure is about 101 kPa, and it falls by about 12 Pa (0.012 kPa) for every meter you climb.

D1 ▸ RUN Install the pressure-sensor library

D1 installs the sensor's Arduino library, `Arduino_LPS22HB`. It needs the internet and is needed once per laptop.

D2 ◆ DO Print the pressure

The sketch reads the sensor five times per second and prints each reading in kilopascals, one per line. `--monitor 6` uploads it and then shows what it prints for 6 seconds. The sketch waits for the notebook to open the serial port (`while (!Serial)`), which `--monitor` does. Run D2 and blow gently toward the board while it prints.

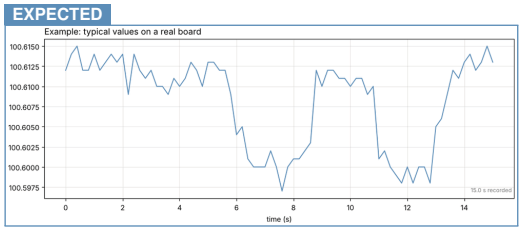
YOU SHOULD SEE five readings per second near 100 kPa (less at altitude), changing by a few hundredths of a kilopascal while you blow.

D3 ◆ DO Plot the pressure (15 s)

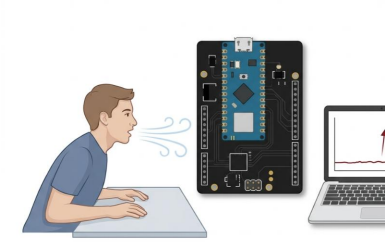
The sketch keeps running after D2 finishes. `ee105.serial_plot(15)` replaces the Serial Plotter: it plots every number the sketch prints for 15 seconds and returns the data as a table, `p`, with the time `t_s` in seconds since the plot started and one column per number. Run D3, keep the board still for the first 5 seconds, then hold it and move it up and down by about a meter.

YOU SHOULD SEE a flat trace for 5 s that scatters by only a pascal or two, then a step of about 0.012 kPa each time the board moves by a meter, down when you lift it (Figure 0.25b).

EXPECTED



D3 Typical values: still for 5 s, then lifted by a meter twice; each lift lowers the pressure by about 12 Pa.

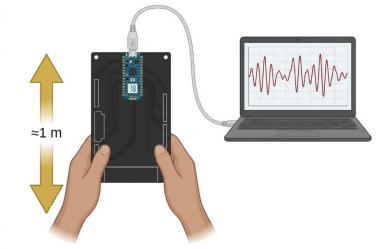


D2 Blow gently toward the board while D2 prints.

EXPECTED

```
...
[ok] Uploaded.
--- output for 6 s ---
100.613
100.611
100.612
100.614
100.611
100.612
100.634
100.652
100.641
100.619
100.613
...
```

D2 Typical output: one reading per line near 100 kPa, a few hundredths higher while you blow.



D3 Keep the board still for 5 s, then lift it about a meter and lower it again.

D4 EDIT Print the pressure and the time

D4 · EDIT in the notebook

```
%%arduino pressure_time --monitor 6
#include <Arduino_LPS22HB.h>

void setup() {
  Serial.begin(115200);
  while (!Serial) {}            // wait for the port
  if (!BARO.begin()) {
    Serial.println("Pressure sensor not found");
    while (1) {}
  }
}

void loop() {
  float kPa = BARO.readPressure();   // kilopascals
  Serial.println(kPa, 3);   // --> kPa, a comma, the time
  delay(200);
}
```

WHAT TO WRITE in place of line 15, the lines that print *two* numbers per line, separated by a comma: the pressure with three decimals, as now, and the time in seconds, `millis() / 1000.0`. The rest of the sketch is the one of D2.

HINT `Serial.print()` prints without ending the line, `Serial.println()` ends it. The plotter reads numbers separated by commas, one line per reading.

DONE WHEN the 6 s of output show lines such as `101.325,12.35`: the pressure and a time that grows by 0.20 from one line to the next.

EXPECTED

```
...
[ok] Uploaded.
--- output for 6 s ---
100.612,1.43
100.613,1.63
100.611,1.83
100.612,2.04
100.614,2.24
100.612,2.44
100.611,2.64
100.613,2.84
...
```

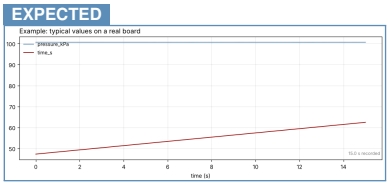
D4 Typical output: the pressure, a comma, and a time that grows by about 0.20 per line.

D5 DO Plot both numbers (15 s)

D5 plots the two numbers of your sketch for 15 s and keeps them in `p`, with the columns `pressure_kPa` and `time_s`. Keep the board still for the first 5 seconds of this recording too, then move it up and down as in D3: D6 and your results file use this table.

YOU SHOULD SEE two traces: the pressure near 100 kPa, and the time, a straight line that climbs by one second per second.

EXPECTED



D5 Typical values: the pressure flat near 100 kPa; the time climbs by one second per second.

D6 EDIT Noise while the board sits still

D6 · EDIT in the notebook

```
still = ...       # ✎ pressure of the first 5 s (kPa)
noise_pa = ...    # ✎ its standard deviation, in Pa
print(f"noise {noise_pa:.1f} Pa from {len(still)} readings")
verdict = "resolves" if noise_pa < 12 else "cannot resolve"
print(f"the sensor {verdict} 1 m of height (12 Pa)")
```

WHAT TO WRITE `still`, the pressure readings of the first 5 s of `p`, when the board sat still, and `noise_pa`, their standard deviation in pascals.

HINT the column `t_s` of `p` is the time since the plot started; a condition on it inside `p.loc[..., "pressure_kPa"]` picks rows. A column has a `.std()` method, and 1 kPa is 1000 Pa.

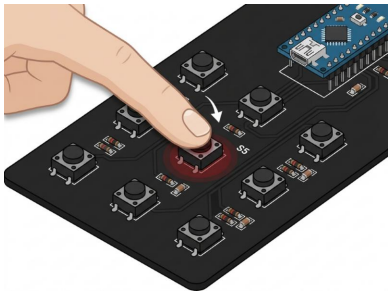
DONE WHEN the cell prints a noise of a few pascals (typically 1 to 5 Pa) from about

25 readings, and says whether the sensor resolves the 12 Pa of one meter (Question 2).

Part E: Read a voltage from the demo board

GOAL Fix the voltage conversion and measure all nine buttons of the touch panel. about 25 min

The selector is on Module 1, so the circuits block is powered, and J7 connects its touch network to pin A0 (Figure 0.9): each of the nine buttons S1 to S9 connects one node of a resistor grid to A0, and each node has its own voltage (Table 0.9). The ADC returns 0 to 1023 for 0 to 3.3 V. Arduino's own example `ReadAnalogVoltage` assumes a 5 V board.



E1 Hold the center button, S5, until the cell has finished.

E1 DO Hold S5 and read it with the 5 V conversion

E1 is `ReadAnalogVoltage` as written for a 5 V board, with `inPin` already set to A0. Hold the center button, S5, keep holding, and run E1: it uploads the sketch and prints for 4 s. Write down the voltage; **E3** asks for it.

YOU SHOULD SEE one voltage per line, steady while you hold S5: about 2.50 V.

E2 EDIT Fix the conversion for this 3.3 V board

E2 · EDIT in the notebook

```
1  %%arduino read_voltage --monitor 4
2  const int inPin = A0;   // touch network (J7, Module 1 end)
3
4  void setup() {
5    Serial.begin(115200);
6  }
7
8  void loop() {
9    int code = analogRead(inPin);              // 0 ... 1023
10   float volts = code * (5.0 / 1023.0);   // --> FIX: 3.3 V
11   Serial.println(volts, 3);
12   delay(100);
13 }
```

WHAT TO WRITE on line 10, the line marked `FIX`: this board's ADC reference in place of `5.0`. Then hold S5 and run E2.

HINT the Nano runs at 3.3 V, and so does its ADC (Section 0.9); full scale, code 1023, is the reference voltage.

DONE WHEN with S5 held, every line reads about 1.65 V, half of 3.3 V.

E3 ANSWER S5 before and after the fix

E3 · ANSWER in the notebook

```
s5_with_5v0 = None    # ✎ V, S5 in E1 (5.0 / 1023.0)
s5_with_3v3 = None    # ✎ V, S5 in E2 (3.3 / 1023.0)
if s5_with_5v0 and s5_with_3v3:
    ratio = s5_with_5v0 / s5_with_3v3
    print(f"S5 ratio {ratio:.2f} (5.0 / 3.3 = 1.52)")
```

TYPE IN `s5_with_5v0`, the S5 reading of **E1**, and `s5_with_3v3`, the S5 reading of **E2**, in place of `None`. The cell prints their ratio.

UNIT volts, as the sketches printed them.

WHERE TO FIND IT the output of E1 and E2.

YOU SHOULD SEE about 2.50 V and 1.65 V, a ratio of 1.52, which is 5.0/3.3 (Question 3).

E4 ▸ RUN Set up the button recorder

E4 defines `record_button(name)`: it records what the running sketch, the one of **E2**, prints for 2 s, averages the readings, and stores the result under `name` in the table `buttons`. Run it once.

E5 ◆ DO Record each button, S1 to S9

Hold S1, keep holding, and run E5; keep holding until it prints (2 s). Then change the name in the line marked ✎ to `"S2"`, hold S2, and run E5 again, and so on to S9. Running it again for a button replaces that button's value. (The picture of **E1** shows the grip.)

YOU SHOULD SEE one line per run: the button, its average voltage, and about 20 readings.

E6 ▸ RUN Compare with nodal analysis

E6 puts your nine voltages next to the values of nodal analysis (Chapter 1, Table 0.9) and their difference in millivolts.

YOU SHOULD SEE every button within about 30 mV of the table: S1 near 3.3 V (a few counts below 1023 is normal), S9 near 0, S5 near 1.65 V (Question 4).

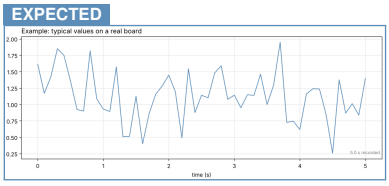
E7 ▸ RUN Ao with no button pressed (5 s)

Keep your hands off the buttons and run E7: it plots what the sketch of **E2** prints for 5 s.

YOU SHOULD SEE a trace that wanders over a wide range instead of a steady level: with no button pressed, nothing sets the voltage of Ao (Question 5).

Button	Voltage	Code
S1	3.300 V	1023
S2	1.946 V	603
S3	1.185 V	367
S4	2.496 V	774
S5	1.650 V	511 or 512
S6	0.804 V	249
S7	2.115 V	656
S8	1.354 V	420
S9	0.000 V	0

Table 0.9: Expected Ao voltage and ADC code for each touch-panel button (from the analysis in Chapter 1).



E7 Typical values: no steady level; Ao jumps around over a volt or more.

✓CHECK **Checkpoint 2** Your table has all nine button voltages, each within a few tens of millivolts of nodal analysis, and the idle plot of A0 wanders. You measured a resistor network with the Arduino and checked it against nodal analysis.

Part F: PWM: brightness follows the button voltage

GOAL Make the LED brightness follow the button you press.

about 15 min

`analogWrite(pin, duty)` switches the pin between 0 and 3.3 V quickly; the LED looks dimmer or brighter with the duty value, 0 to 255 (Figure 0.14).

F1 ◆DO Dim the LED with the buttons

The sketch below reads A0, turns the 10-bit code into an 8-bit duty value with `map()`, sets the LED brightness with it, and prints the code and the voltage on one line. `--plot 20` uploads it and plots both numbers for 20 s. Run F1 and press different buttons, one at a time, while it plots.

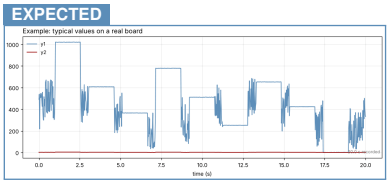
```
%%arduino pwm_led --plot 20
const int inPin = A0;          // touch network, Network 1

void setup() {
  Serial.begin(115200);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  int code = analogRead(inPin);           // 0 ... 1023
  float volts = code * (3.3 / 1023.0);    // 3.3 V reference
  int duty = map(code, 0, 1023, 0, 255);  // 10-bit -> 8-bit
  analogWrite(LED_BUILTIN, duty);         // brightness
  Serial.print(code);
  Serial.print(",");
  Serial.println(volts);                  // two traces
  delay(20);
}
```

YOU SHOULD SEE the LED brighter for buttons with a higher voltage (S1 full brightness, S9 off), and the code stepping with each button. The voltage steps too, but at most 3.3 V it looks flat on the code's scale.

EXPECTED



F1 Typical values, S1 to S9 in turn: the code (y1) steps, wild in between; the voltage (y2) is tiny on this scale.

F2 ◆ANSWER Code and duty value of S5

F2 · ◆ANSWER in the notebook

```
code_s5 = ...   # ✎ ADC code of S5 (0 to 1023)
duty_s5 = ...   # ✎ duty from map(), 0 to 255 (use //)
print(f"S5: code {code_s5}, duty {duty_s5} of 255,",
```

```
        f"{duty_s5 / 255:.0%} of full brightness")
```

TYPE IN `code_s5`, the ADC code of S5, and `duty_s5`, the duty value that `map()` gives for it.

UNIT code 0 to 1023; duty 0 to 255.

WHERE TO FIND IT the code trace of the **F1** plot while you held S5, or your S5 voltage of **E2** with Eq. (0.6). `map()` uses integer division, `//` in Python.

YOU SHOULD SEE code 511 or 512 and duty 127 of 255: 50% of full brightness (Question 6).

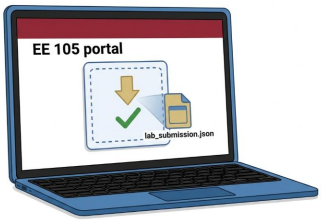
✓CHECK **Checkpoint 3** The LED brightness follows the button you press, and the plot shows both traces. You turned a voltage into a number and a number back into brightness: the ADC and PWM ends of every lab to come.

Save and upload

Save ▸ RUN Write `lab0.1_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab0.1_submission.json` in the lab folder: your blink times, the pressure recording of **D5** and its noise, the two S5 readings, the nine button voltages, the idle A0 recording, and `duty_s5`. Log in at `ee105.org/portal`, open Lab 0.1, and upload that file. If the cell stops because `p` or `idle` is missing, run that recording step and then Save again.

YOU SHOULD SEE `[ok] Saved lab0.1_submission.json`, and then the portal shows your grade. A run on the simulated board counts as practice and earns no credit.



Save Upload `lab0.1_submission.json` on the course portal under Lab 0.1.

Finish the lab

You programmed a microcontroller from a notebook: a blink at any frequency, a live pressure plot, a voltmeter for the touch panel, and an LED dimmer. The lab is done when every box is ticked.

- ☐ **B2** ANSWER `ten_blinks_s` for delays of 1000, 250, and 125 ms
- ☐ **C1** EDIT `blinkAt()`
- ☐ **D4** EDIT the sketch prints the pressure and the time
- ☐ **D5** DO a 15 s recording with the board still for the first 5 s
- ☐ **D6** EDIT `still`, `noise_pa`
- ☐ **E2** EDIT the 3.3 V conversion
- ☐ **E3** ANSWER `s5_with_5v0`, `s5_with_3v3`
- ☐ **E5** DO all nine buttons recorded
- ☐ **E7** RUN the idle Ao recording
- ☐ **F2** ANSWER `code_s5`, `duty_s5`
- ☐ CHECK Checkpoint 1: `blinkAt()` at 0.5, 1, 2, and 4 Hz
- ☐ CHECK Checkpoint 2: nine button voltages, and a wandering idle Ao
- ☐ CHECK Checkpoint 3: the LED follows the buttons
- ☐ Questions 1 to 6 answered in the notebook
- ☐ **Save** SAVE `lab0.1_submission.json` uploaded on the portal

Typical results

Quantity	Typical result on a real board
Blink (**B2**)	0.50, 1.98, and 3.85 Hz for delays of 1000, 250, and 125 ms
Pressure (**D3**)	about 100 kPa near sea level; a step of about 12 Pa per meter
Noise while still (**D6**)	1 to 5 Pa
S5 (**E1**, **E2**)	2.50 V with the 5.0 V conversion, 1.65 V with 3.3 V
Buttons (**E6**)	within about 30 mV of Table 0.9
No button (**E7**)	Ao wanders over a wide range
PWM, S5 (**F2**)	code 511 or 512, duty 127 of 255

Table 0.10: Expected results for Lab 0.1.

Table 0.10 lists the numbers. The blink frequencies come out near 0.5, 2, and 4 Hz for delays of 1000, 250, and 125 ms (Figure 0.25a); the stopwatch frequency usually comes out a few percent low at 4 Hz, because your reaction time matters more when the total is short. The pressure reads about 100 kPa near sea level (less at higher altitude). With the board still, the readings scatter by only a few pascals, and one meter of height changes the pressure by about 12 Pa (0.012 kPa), so the up-and-down motion is visible, especially after averaging (Figure 0.25b). With S5 held, the 5 V conversion reports about 2.50 V and the corrected one about 1.65 V: the wrong version reads $5.0/3.3 = 1.52$

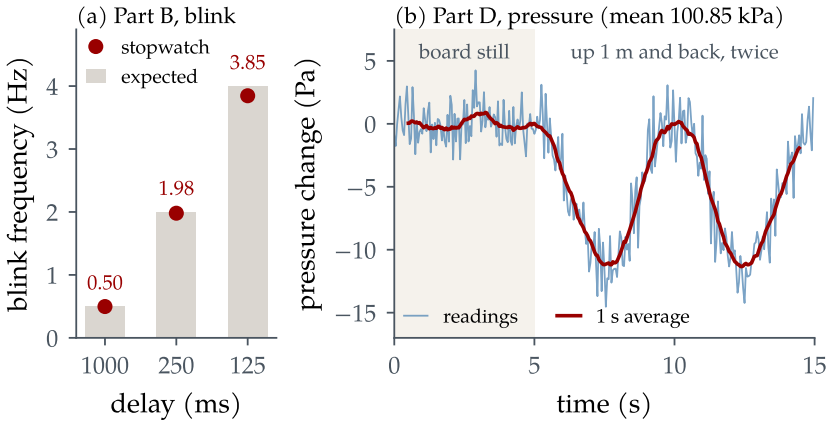


Figure 0.25: Typical Lab 0.1 results. (a) Blink frequency from ten blinks timed with a stopwatch (dots) against $f = 1/(2t_{\text{half}})$ (bars); at 4 Hz the measured frequency is about 4% low. (b) Pressure relative to its mean over the first 5 s. Look for a scatter of only a pascal or two while the board is still, and a dip of about 12 Pa each time you lift the board by a meter, clearest in the 1 s average.

times too high. The nine buttons agree with Table 0.9 to within about 30 mV; the middle buttons differ most, from the tolerance of the resistors. With no button pressed, A0 is connected to nothing and its reading wanders. In Part F, S5 gives a code near 512 and a PWM value of 127, half brightness: `map()` computes $512 \times 255/1023 = 127.6$ and drops the fraction.

Troubleshooting

- *No board found, or the upload fails.* Uploading is the step most likely to fail. Use a data cable (a charge-only cable powers the board but carries no data), plug directly into the laptop rather than a hub, and try another port. If an upload fails, double-tap the white RESET button on the Nano (the orange LED starts to pulse slowly) and run the cell again (**B1**).
- *Port busy.* Another program, such as an open Arduino IDE or a second notebook, is using the board. Close it and run the cell again.
- *The LED blinks at half the frequency, or looks always on* (**C1**). Each half of the blink takes $1000/(2f)$ ms, not the full period $1000/f$; and the LED needs the second `delay()` after it turns off.
- *The pressure sketch does not compile.* Run **D1** first; it needs the internet.
- *The pressure sketch prints nothing.* The sketch waits for the serial port to open (`while (!Serial)`); the `--monitor` option opens it. If you deleted that line, the first readings may be lost, which is harmless.
- *S5 reads* 2.50 V, *not* 1.65 V. The sketch still uses `5.0 / 1023.0`; fix line 10 of **E2** and run it again.
- *One button reads wrong* (**E5**). You let go too early: hold it for the whole 2 s and run E5 again for that button.

- *The **F1** plot shows one trace only.* The sketch is missing the comma between the code and the voltage.
- *Nothing happens with `SIMULATE = True`.* The simulator cannot run sketches you write; the `%%arduino` cells only save them. This lab needs a real board.
- *A0 reads the wrong values, or wanders even while a button is held.* Check that S10 is on Module 1 and that the caps of J7 and J8 are on the Module 1 end; on the Camera end, A0 is connected to the camera header instead.
- *The next lab does not find the course firmware.* After this lab the board runs your last sketch. The next notebook that calls `ee105.connect()` uploads the course firmware again by itself.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. What blink frequencies did you measure in Part B (**B2**)? Compare each with Eq. (0.8), and explain any shortfall at 4 Hz.
2. Estimate the standard deviation of the pressure while the board sat still (**D6**). Can the sensor resolve a one-meter change in height?
3. In Part E, what voltage did S5 give with the 5.0 V conversion (**E1**) and with 3.3 V (**E2**)? What is the ratio, and why?
4. Which button gives the highest voltage and which the lowest? Does S5, at the center of the grid, read close to half of 3.3 V? Which button differs most from Table 0.9 (**E6**)?
5. With no button pressed, what does A0 read (**E7**)? Explain the behavior using the idea of a floating input.
6. What PWM value corresponds to button S5 (**F2**)? What fraction of full brightness is that?

Worked examples: programming the Arduino

These examples practice the sketch structure, timing, serial output, and on-board sensors of Section 0.9. Each one runs on the Nano 33 BLE Sense by itself, with no other part of the demo board.

> WORKED EXAMPLE 0.18 *Sweeping the blink frequency with `pow()`*
>
> Step **C1** of the lab asks you to write `blinkAt(float freqHz)`, a function that blinks the LED once at a given frequency. Suppose

you have it. Use it with `pow()` to blink four times each at 0.5 Hz, 1 Hz, 2 Hz, and 4 Hz, repeating forever, without listing the four frequencies in the sketch. How long does one pass through all four frequencies take, and how many blinks does the LED make in a minute?

SOLUTION. From Eq. (0.8), one blink at frequency f lasts $1/f$ seconds: t_{half} on and t_{half} off. The four frequencies are powers of two, 2^k for $k = -1, 0, 1, 2$, so a loop over k produces them with `pow()` (`^` is not a power operator in C):

```
const int ledPin = 13;
// blinkAt(freqHz): your function from step C1 goes here

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {
  for (int k = -1; k <= 2; k++) {
    float f = pow(2, k);                  // 0.5, 1, 2, 4 Hz
    for (int n = 0; n < 4; n++) blinkAt(f);
  }
}
```

Four blinks take $4/f$: 8 s at 0.5 Hz, 4 s at 1 Hz, 2 s at 2 Hz, and 1 s at 4 Hz. One full pass takes $8 + 4 + 2 + 1 = 15$ s and makes 16 blinks, so in one minute the LED completes four passes, 64 blinks. The pass time depends only on the frequencies asked for, not on how `blinkAt` is written. ■

WORKED EXAMPLE 0.19 *Proximity alarm*

Use the APDS9960 proximity sensor on the Nano 33 BLE Sense to detect whether an object is within 3 cm of the board. By default the built-in LED blinks at 1 Hz (0.5 s on, 0.5 s off). Whenever an object is within 3 cm, it blinks at 4 Hz. Work through these checkpoints: (0) a 1 Hz blink as baseline, (1) print proximity readings to the serial port, (2) plot them as your hand moves, (3) find the reading at 3 cm, and (4) add the decision logic.

SOLUTION. *Checkpoint 0.* By Eq. (0.8), 1 Hz needs $t_{\text{half}} = 500$ ms (the blink sketch with `delay(500)`) and 4 Hz needs $1/(2 \times 4) = 125$ ms. Since `delay()` would block the sensor reads, the final program uses `millis()`.

Checkpoints 1 and 2. The `Arduino_APDS9960` library provides an object `APDS`. After `APDS.begin()`, `APDS.readProximity()` returns a reading from 0 to 255 whenever `APDS.proximityAvailable()` is true. The library documents 0 as closest and 255 as farthest, so the value *falls* as your hand approaches. One `Serial.println()` per

reading suits both a monitor and a plotter.

Checkpoint 3. Hold a hand at 3 cm (use a ruler) and note the reading. It depends on the object's reflectivity, the lighting, and the sensor, so you must measure it. As representative values, a run might read about 250 with nothing nearby and 150 at 3 cm; we use 150 below.

Checkpoint 4. An object is "within 3 cm" when the reading is below the threshold. The half period is chosen from that test, and the `millis()` timer toggles the LED:

```
#include <Arduino_APDS9960.h>

const int PROX_THRESHOLD = 150;   // reading at 3 cm
int proximity = 255;              // 0 = touching, 255 = far
unsigned long lastToggle = 0;
bool ledOn = false;

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  if (!APDS.begin()) {
    Serial.println("APDS9960 not found");
    while (1);                    // stop here
  }
}

void loop() {
  if (APDS.proximityAvailable()) {
    int p = APDS.readProximity();
    if (p >= 0) proximity = p;    // -1 signals a read error
    Serial.println(proximity);    // monitor and plotter
  }
  bool near = (proximity < PROX_THRESHOLD);
  unsigned long half = near ? 125 : 500;    // 4 Hz or 1 Hz
  if (millis() - lastToggle >= half) {
    lastToggle = millis();
    ledOn = !ledOn;
    digitalWrite(LED_BUILTIN, ledOn ? HIGH : LOW);
  }
}
```

If the reading flickers around the threshold, the blink rate will chatter; a simple fix is *hysteresis*: switch to fast blinking below 150 but back to slow only above, say, 170. ■

Worked Example 0.20 *Sound level indicator*

Use the PDM microphone to toggle the built-in LED on and off whenever the sound amplitude exceeds a threshold, such as a clap near the board. Follow the checkpoints: (1) configure the microphone with the `PDM` library and confirm data arrives, (2) read the 16-bit samples into a buffer and print them, (3) find a clap threshold by plotting, and (4) add the toggle logic.

SOLUTION. *Checkpoint 1.* The `PDM` library calls a function you register with `PDM.onReceive()` each time a block of samples is ready. Inside it, `PDM.available()` returns the number of bytes waiting; a nonzero value confirms that data is coming in. `PDM.begin(1, 16000)` starts one channel at 16 kHz.

Checkpoint 2. The hardware delivers 16-bit signed samples (type `short`, −32768 to 32767). `PDM.read(buffer, bytes)` copies them into an array; at 2 bytes per sample and 16 kHz, that is 32 000 bytes per second. Printed, the samples form a waveform centered near zero.

Checkpoint 3. Print one number per buffer, the peak absolute value. Representative values: a few hundred counts in a quiet room, many thousands for a nearby clap. Read yours from the plot and set the threshold between them (we use 10 000).

Checkpoint 4. Toggle the LED when the peak crosses the threshold, then ignore triggers for 300 ms so one clap, which spans several buffers, toggles only once.

```
#include <PDM.h>

short buf[512];                    // 16-bit PCM samples
volatile int nSamples = 0;
const int CLAP_THRESHOLD = 10000;  // from Checkpoint 3
bool ledOn = false;
unsigned long lastClap = 0;

void onPDMdata() {
  int nBytes = PDM.available();
  PDM.read(buf, nBytes);
  nSamples = nBytes / 2;           // 2 bytes per sample
}
void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  PDM.onReceive(onPDMdata);
  PDM.begin(1, 16000);             // 1 channel, 16 kHz
}
void loop() {
  if (nSamples == 0) return;
  int peak = 0;
  for (int i = 0; i < nSamples; i++)
    peak = max(peak, abs(buf[i]));
  nSamples = 0;
  Serial.println(peak);            // for the plotter
  if (peak > CLAP_THRESHOLD && millis() - lastClap > 300) {
    lastClap = millis();
    ledOn = !ledOn;
    digitalWrite(LED_BUILTIN, ledOn ? HIGH : LOW);
  }
}
```

The variable `nSamples` is marked `volatile` because it is changed inside the callback, outside the normal flow of `loop()`; the keyword

tells the compiler to reread it every time. ■

0.12 *Laboratory: Python Bootcamp*

LAB 0.2 AT A GLANCE

Goal: practice the Python and NumPy used all semester, pass the six bootcamp questions, and apply the same tools to linear equations and to arrays of measurements.
Hardware: laptop only.
Notebooks: three, all in the folder `lab0.2-python-bootcamp` of the lab kit (Figure 0.26): the lab notebook `lab0.2-python-bootcamp.ipynb` (Parts A, D, and E, and Save), `bootcamp-guide.ipynb` (Part B), and `bootcamp-questions.ipynb` with its autograder `autograder_v2.py` (Part C). The data files of Part E, `a1_readings.csv` and `levels.csv`, are in the same folder.
Before the lab: read Section 0.10. The kit's setup file of Lab 0.1 installed everything this lab needs.
Time: about 1 h 45 min.
You hand in: `lab0.2_submission.json` on the course portal; keep both notebooks you wrote in, with your answers.

KEY TAKEAWAYS

- Lists join, arrays compute: `[1, 2] + [1, 2]` has four elements, two arrays add element by element, and `@` is the matrix product.
- A slice `start:stop:step` includes the start and stops before the stop: `A[1::2, ::2]` keeps the odd rows and the even columns.
- `axis=0` works down the columns, `axis=1` across the rows: after `reshape(-1, 10)`, `axis=1` gives one mean per 50 ms window.
- Linear equations are $A\mathbf{x} = \mathbf{b}$: `np.linalg.solve(A, b)` solves them, and `A @ x - b` near 10^{-15} checks the answer.
- A 10-bit ADC on 3.3 V has steps of 3.22 mV: readings of a steady voltage fall on a few whole codes and scatter by about one step.

SET UP FOR LAB 0.2

No board. This lab runs on your laptop only; the demo board stays unplugged. The readings of Part E come from two data files in the lab folder, recorded once on the toolkit's simulated board.

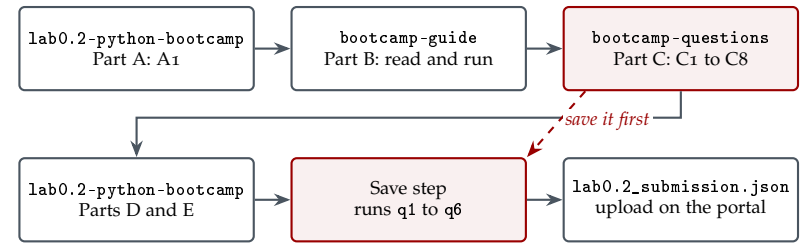
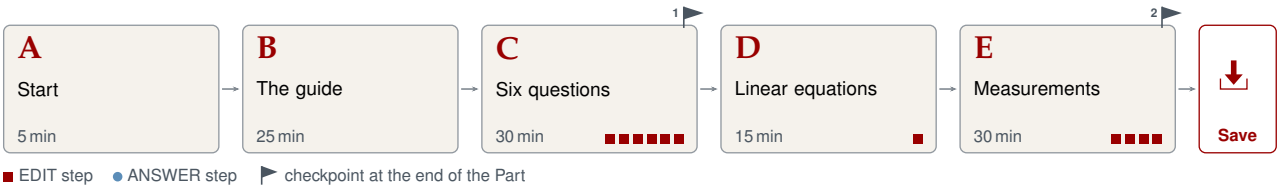


Figure 0.26: The three notebooks of Lab 0.2, in the order you use them. Parts B and C happen in their own notebooks; the lab notebook's Save step reads your six functions from the *saved* questions notebook and runs them on new inputs.



Objectives

- Use lists, dictionaries, comprehensions, and NumPy arrays, including slicing, masks, and `axis`.
- Pass the six bootcamp questions with the autograder.
- Solve a small linear system with `np.linalg.solve` and check the result.
- Summarize an array of measurements with its mean, standard deviation, and histogram.

Background

Read Section 0.10. The bootcamp *guide* notebook walks through Python and NumPy with runnable examples; the *questions* notebook has six short problems and an autograder. If you can answer all six, you have the Python you need for EE 105. Table 0.11 lists them.

The key takeaways above are the rules that cause most of the surprises; Figure 0.29 shows slicing with steps. In Part E the data are ADC readings (Figure 0.11): one step is $\Delta = 3.3\,\text{V}/1024 \approx 3.22\,\text{mV}$ (Eq. (0.5)), and every reading is a whole number of steps (Figure 0.12). The standard deviation of repeated readings says how far to trust one reading.

Q	step	Task
1	C2	find the NumPy function that solves $A\mathbf{x} = \mathbf{b}$
2	C3	return the entries of A in odd rows and even columns
3	C4	smallest of the row indexes of the column minimums
4	C5	stack rows 0, 2, 4, … of A below B
5	C6	element-wise product of $[N, N-1, \ldots, 1]$ and $[1, 2, \ldots, N]$
6	C7	normalize the rows of A and the columns of B by their sums

Table 0.11: The six bootcamp questions and their steps in `bootcamp-questions.ipynb`.

Procedure

Each step below is one code cell of a notebook, with the same label above it (**A1**, **C1**, …). Parts A, D, and E are in the lab notebook, Part B in the guide, and Part C in the questions notebook (Figure 0.26). Run the cells in order with Shift+Enter; a step tagged EDIT needs your code in the cell first (Section 0.11 explains the tags).

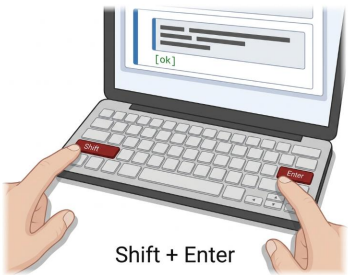
Part A: Start

GOAL Check that Python and the course packages load. about 5 min

A1 ▸ RUN Check Python and the course packages

Open the lab notebook, `lab0.2-python-bootcamp.ipynb`, click its first code cell, and press Shift+Enter. The cell loads NumPy, pandas, and Matplotlib and checks the course packages that the setup file installed in the `ee105` environment.

YOU SHOULD SEE one line per package, each beginning with `[ok]`. If one says `[!!]`, close JupyterLab and run the setup file again.



A1 Click the first code cell and press Shift+Enter.

EXPECTED

```
[ok] Python 3.13.18, ee105 toolkit 3.0
[ok] numpy 2.5.3
[ok] pandas 3.0.5
[ok] matplotlib 3.11.2
[ok] scipy 1.18.1
[ok] sympy 1.14.0
[ok] serial 3.5
```

A1 Every line starts with `[ok]`; your version numbers may differ.

Part B: The guide

GOAL Work through the bootcamp guide, predicting each output before you run it.

about 25 min

Part B has no cell in the lab notebook. Open `bootcamp-guide.ipynb` from the file list on the left of JupyterLab and keep the lab notebook open. Run its cells in order with Shift+Enter; before each example, predict what it prints, and change at least one value and run it again. Watch three things: `lst + lst` versus `arr + arr`, slicing with steps (Figure 0.29), and `@` versus element-wise `*`.

YOU SHOULD SEE outputs that match your predictions, or a surprise worth a second look. The guide ends by sending you to the questions notebook.

Part C: The questions

GOAL Pass all six bootcamp questions and see `Score: 6/6`.

about 30 min

Part C is in `bootcamp-questions.ipynb`, steps C1 to C8. Each question cell defines your function and runs its test, which prints `Question N Passed!` or stops with a message that says what went wrong; fix the function and run the cell again. Write down which questions failed at first and why (Question 1 at the end asks for it). The guide has an example of every function you need.

C1 ▸ RUN Load NumPy and the autograder

C1 loads NumPy and the autograder with `%run autograder_v2.py`. Run it again after a kernel restart.

YOU SHOULD SEE `Autograder loaded!`

C2 EDIT Question 1: the function that solves $A\mathbf{x} = \mathbf{b}$

C2 · EDIT in the notebook

```
1 func = None   # ✎ the NumPy function that solves A x = b
2
3 def q1(A, b):
4     return func(A, b)
5
6 test_q1(q1)
```

WHAT TO WRITE on line 1, in place of `None`, the NumPy function that solves a system of linear equations $A\mathbf{x} = \mathbf{b}$. Search the NumPy documentation for it.

HINT it is in `numpy.linalg`, and circuit analysis uses it all the time. Write the function's name without parentheses: `q1` calls it.

DONE WHEN the cell prints `Question 1 Passed!` The test solves a 2×2 system and the café system of Part D.

C3 EDIT Question 2: odd rows, even columns

C3 · EDIT in the notebook

```
1 def q2(A):
2     """Entries of A in odd rows and even columns
3     (rows and columns count from 0).
4     A = np.arange(1, 26).reshape(5, 5) gives
5     [[ 6,  8, 10],
6      [16, 18, 20]]
7     """
```

```
8       return None   # ✎ replace None with your answer
9
10  test_q2(q2)
```

WHAT TO WRITE on line 8, an array of every entry of `A` that is in an odd row and in an even column (rows and columns count from 0).

HINT slice each axis with `start::step`, `A[rows, columns]`; check that you did not swap the rows and the columns.

DONE WHEN the cell prints `Question 2 Passed!`

C4 ✎ EDIT Question 3: smallest row index of the column minimums

C4 · ✎ EDIT in the notebook

```
1   def q3(A):
2       """Row index of the minimum of each column of A,
3       then the smallest of those indexes.
4       [[5, 1, 3, 2], [1, 4, 2, 3], [3, 2, 1, 4]]: the
5       minimums are in rows [1, 0, 2, 0], so 0.
6       """
7       return None   # ✎ replace None with your answer
8
9   test_q3(q3)
```

WHAT TO WRITE on line 7, for an $M \times N$ array, the row index of the minimum of each column, and then the smallest of those indexes.

HINT `np.argmin` takes an `axis`; `axis=0` works down the columns.

DONE WHEN the cell prints `Question 3 Passed!`

C5 ✎ EDIT Question 4: every other row of A below B

C5 · ✎ EDIT in the notebook

```
1   def q4(A, B):
2       """The rows of B (P x N), then rows 0, 2, 4, ...
3       of A (M x N). A = [[1, 2, 3], [4, 5, 6], [7, 8, 9],
4       [10, 11, 12]] and B = [[1, 1, 1], [2, 2, 2]] give
5       [[1, 1, 1], [2, 2, 2], [1, 2, 3], [7, 8, 9]]
6       """
7       return None   # ✎ replace None with your answer
8
9   test_q4(q4)
```

WHAT TO WRITE on line 7, for an $M \times N$ array `A` and a $P \times N$ array `B`, the rows of `B` followed by rows 0, 2, 4, ... of `A`.

HINT *below* means more rows, not more columns: `np.vstack`, or `np.concatenate` along `axis=0`.

DONE WHEN the cell prints `Question 4 Passed!`

C6 EDIT Question 5: element-wise product of two ranges

C6 · EDIT in the notebook

```
1  def q5(N):
2      """[N*1, (N-1)*2, ..., 1*N], the element-wise
3      product of u = [N, N-1, ..., 1] and
4      v = [1, 2, ..., N]. N = 5 gives [5, 8, 9, 8, 5].
5      """
6      u = None    # ✎ [N, N-1, ..., 1]
7      v = None    # ✎ [1, 2, ..., N]
8      return None    # ✎ their element-wise product
9
10 test_q5(q5)
```

WHAT TO WRITE lines 6 to 8: the vectors $u = [N, N-1, \ldots, 1]$ and $v = [1, 2, \ldots, N]$, and their element-wise product $[N \cdot 1, (N-1) \cdot 2, \ldots, 1 \cdot N]$.

HINT `np.arange` stops *before* its stop value, and a negative step counts down.

DONE WHEN the cell prints `Question 5 Passed!`

C7 EDIT Question 6: normalize rows and columns

C7 · EDIT in the notebook

```
1  def q6(A, B):
2      """A with each row divided by its sum, and B with
3      each column divided by its sum.
4      A = [[1, 2, 3], [4, 5, 6]], B = [[1, 2], [3, 4]]
5      give [[1/6, 2/6, 3/6], [4/15, 5/15, 6/15]] and
6      [[1/4, 2/6], [3/4, 4/6]].
7      """
8      return None, None    # ✎ (A normalized, B normalized)
9
10 test_q6(q6)
```

WHAT TO WRITE on line 8, two arrays: `A` with its rows normalized (each row divided by its sum) and `B` with its columns normalized (each column divided by its sum).

HINT `A.sum(axis=1)` has one value per row; with `keepdims=True` it stays a column, so the division broadcasts across each row.

DONE WHEN the cell prints `Question 6 Passed!`

C8 ▸ RUN Run all six tests

C8 runs every test once more; a failed test does not stop the others. Then save the questions notebook (Ctrl+S, or Cmd+S on a Mac): the Save step of the lab notebook reads your six functions from the saved file.

YOU SHOULD SEE six lines `Question` *N* `Passed!` and `Score: 6/6`.

EXPECTED

```
Question 1 Passed!
Question 2 Passed!
Question 3 Passed!
Question 4 Passed!
Question 5 Passed!
Question 6 Passed!
Score: 6/6
```

C8 One `Passed!` line per question, then `Score: 6/6`.

✓CHECK **Checkpoint 1** The questions notebook prints `Score: 6/6`. Six for six: you have the NumPy that every lab of the semester uses.

Part D: Linear equations

GOAL Solve a linear system with `np.linalg.solve` and check the answer.

about 15 min

Back in the lab notebook. Question 1 asked for the NumPy function that solves $A\mathbf{x} = \mathbf{b}$, and many problems have that form. Worked Example 0.25 works one out: three orders at a café,

$$\begin{aligned} 2c + b + j &= 13.25 \\ c + 2b &= 8.00 \\ c + b + 2j &= 13.75 \end{aligned} \quad \Longrightarrow \quad \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} c \\ b \\ j \end{bmatrix} = \begin{bmatrix} 13.25 \\ 8.00 \\ 13.75 \end{bmatrix},$$

where c, b, and j are the prices of a coffee, a bagel, and a juice in dollars. Each row of the matrix is one order, each column one item.

D1 ▸ RUN Solve the café system

D1 solves the café system with `np.linalg.solve` and checks the answer by multiplying back: `A @ x` must reproduce `b`.

YOU SHOULD SEE coffee 3.50, bagel 2.25, juice 4.00 dollars, and `A @ x - b` of about 10^{-15} or exactly zero: rounding error of floating-point arithmetic.

EXPECTED

```
coffee, bagel, juice: [3.5  2.25 4.  ]
A @ x - b = [0.00000000e+00
0.00000000e+00 1.77635684e-15]
```

D1 Three prices, then three residuals that are zero or about 10^{-15}.

D2 EDIT Prices at the school store

D2 · EDIT in the notebook

```
A = ...   # 3 x 3: a row per order; pen, notebook, folder
b = ...   # the three totals (dollars)
x = ...   # solve A x = b

print("pen, notebook, folder (dollars):", np.round(x, 2))
print(f"largest |A @ x - b| = {np.abs(A @ x - b).max():.1e}")
```

WHAT TO WRITE `A`, `b`, and `x` for three orders at a school store: 3 pens, 2 notebooks, and 1 folder cost 8.40 dollars; 1 pen and 4 notebooks cost 3.80 dollars; 2 pens, 1 notebook, and 2 folders cost 10.05 dollars. `A` is 3×3, one row per order, with the columns in the order pen, notebook, folder; `b` holds the three totals; `x` solves $A\mathbf{x} = \mathbf{b}$.

HINT **D1** has the same shape; follow its pattern. An item that an order does not contain gets a 0.

DONE WHEN the cell prints three prices and a largest $|A\mathbf{x} - \mathbf{b}|$ of about 10^{-15} or less: the prices reproduce all three totals (Question 2).

Part E: Arrays of measurements

GOAL Summarize ADC readings and find nine voltage levels in a noisy recording. about 30 min

E1 ▸ RUN Load `a1_readings.csv`

`a1_readings.csv` holds 200 readings of one analog input of the board's Arduino while the voltage on it was steady, taken 5 ms apart (made with the toolkit's simulated board). `pd.read_csv` loads it as a table, a pandas DataFrame named `a1`, with the time `t_s`, the raw ADC code `raw` (0 to 1023), and `volts`. E1 shows its first five rows.

YOU SHOULD SEE five rows with codes near 478 and voltages near 1.54 V.

EXPECTED

	t_s	raw	volts
0	0.000	479	1.54365
1	0.005	479	1.54365
2	0.010	478	1.54043
3	0.015	478	1.54043
4	0.020	478	1.54043

E1 Five rows: the time `t_s`, a code near 478, and about 1.54 V.

E2 EDIT Mean and standard deviation

E2 · EDIT in the notebook

```
v_mean = ...    # ✎ mean of a1["volts"], in V
v_std = ...     # ✎ standard deviation of a1["volts"], in V

step = hw.ADC_V_PER_COUNT                  # one ADC step (V)
codes, counts = np.unique(a1["raw"], return_counts=True)
print(f"mean {v_mean:.4f} V, std {v_std * 1e3:.2f} mV")
print(f"one ADC step is {step * 1e3:.2f} mV:",
      f"the std is {v_std / step:.2f} steps")
print(f"{len(codes)} different ADC codes: {codes}")
```

WHAT TO WRITE `v_mean` and `v_std`, the mean and the standard deviation of the 200 voltages `a1["volts"]`, in volts. The cell then compares the standard deviation with one ADC step, $\Delta = 3.3\,\text{V}/1024 = 3.22\,\text{mV}$ (`hw.ADC_V_PER_COUNT`), and counts the distinct ADC codes with `np.unique`.

HINT `np.mean` and `np.std` take a column of the table as they take an array.

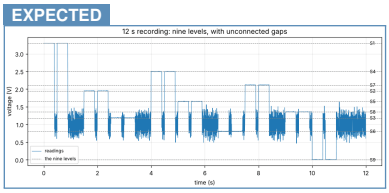
DONE WHEN the cell prints a mean of 1.5391 V, a standard deviation of 3.08 mV (0.96 steps), and five different codes, 476 to 480 (Question 4).

E3 ▸ RUN Plot the readings, then load `levels.csv`

E3 plots the readings with their mean, a band of plus and minus one standard deviation, and a histogram of the raw codes. Then it loads `levels.csv`: 2400 readings (12 s, 5 ms apart) of an input switched between nine fixed voltages, about 1 s each, with unconnected gaps in between. They are the nine buttons of the board's touch panel (Chapter 1); here they are nine numbers in `hw.TOUCH_NODE_V`. E3 makes the lists `names` (S1 to S9) and `levels` and plots the recording with the nine levels dashed.

YOU SHOULD SEE a histogram with one bar per code and gaps between the bars (Figure 0.28a): every reading is a whole number of ADC steps (Question 4). Then flat stretches on the dashed levels, and gaps where

EXPECTED



E3 The second of two figures: flat stretches on the nine dashed levels, wild swings in the gaps.

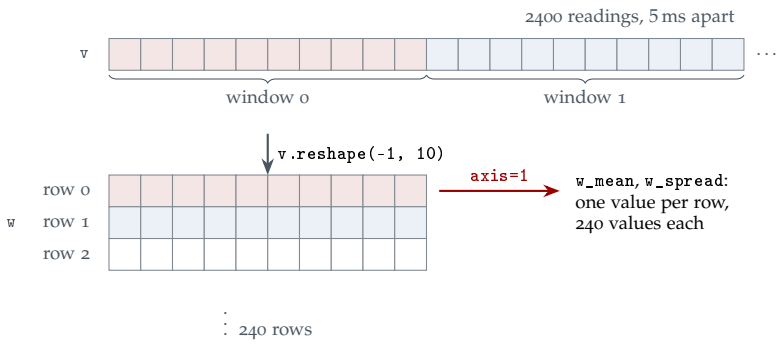


Figure 0.27: Cutting a recording into windows (step E5). `reshape(-1, 10)` turns the 2400 readings into 240 rows of 10, one row per 50 ms window; `axis=1` computes along each row.

the unconnected input swings around (on a real board mostly 60 Hz pickup from the room's wiring, which the simulated recording imitates). The method below trusts only *steady* stretches.

E4 EDIT Write `nearest_level(v)`

E4 · EDIT in the notebook

```
1  TOL_V = 0.05                  # accept a match within 50 mV
2
3  def nearest_level(v):
4      """Name of the closest level; None if none is within
           TOL_V."""
5      # ✎ the name of the closest level, or None
6
7  print(nearest_level(1.66), nearest_level(3.29),
8        nearest_level(1.0))     # expect S5 S1 None
```

WHAT TO WRITE the body of `nearest_level(v)`, in place of line 5: return the name of the level closest to `v`, or `None` if even the closest one is more than `TOL_V` (50 mV) away.

HINT the distances are `np.abs(levels - v)`; `np.argmin` of them is the position of the closest level in `levels` and in `names`.

DONE WHEN the cell prints `S5 S1 None`.

E5 EDIT Window means and spreads

E5 · EDIT in the notebook

```
1  WIN = 10            # readings per window (50 ms)
2  STEADY_V = 0.04     # steady: spread below 40 mV
3
4  v = lv["volts"].to_numpy()
5  w = v[: len(v) // WIN * WIN].reshape(-1, WIN)   # rows
6  w_mean = ...        # ✎ mean of each row, with axis=1
7  w_spread = ...      # ✎ largest minus smallest of each row
8  steady = w_spread < STEADY_V
9  labels = [nearest_level(m) if s else None
10           for m, s in zip(w_mean, steady)]
11 matched = sum(x is not None for x in labels)
12 print(f"{len(w)} windows, {steady.sum()} steady,",
```

```
        f"{matched} matched")
```

WHAT TO WRITE `w_mean` and `w_spread`: the mean of each row of `w` and its spread, the largest minus the smallest reading of the row. Line 5 has already cut the recording into 50 ms windows of `WIN` = 10 readings with `reshape(-1, WIN)`, one row per window (Figure 0.27); `reshape` needs the number of readings to be a multiple of `WIN`, so it first drops the readings left over at the end. A window is *steady* when its spread is below `STEADY_V` (40 mV, about 12 ADC steps), and only steady windows get a level name.

HINT `axis=1` works along each row; `.max(axis=1)` and `.min(axis=1)` give the largest and smallest of each row.

DONE WHEN the cell prints `240 windows, 144 steady, 144 matched`.

E6 EDIT Count the windows of each level

E6 · EDIT in the notebook

```
counts = ...   # ✎ {level name: windows labeled with it}

found = [name for name in names if counts[name] >= 2]
table = pd.DataFrame({"level_V": levels}, index=names)
lvl = levels / hw.ADC_V_PER_COUNT
table["level_code"] = np.round(lvl).clip(0, 1023).astype(int)
table["steady_windows"] = [counts[n] for n in names]
table["seconds"] = table["steady_windows"] * WIN * 0.005
print(f"found {len(found)} of 9 levels: {found}")
if len(found) == 9:
    print("✓ All nine levels found. With the prices of",
          "Part D checked, that is Checkpoint 2.")
table
```

WHAT TO WRITE `counts`, a dictionary with one entry per name in `names`: the number of windows that E5 labeled with that name. A level counts as *found* if it has at least two windows (100 ms).

HINT a dictionary comprehension over `names` with `labels.count(name)` does it in one line.

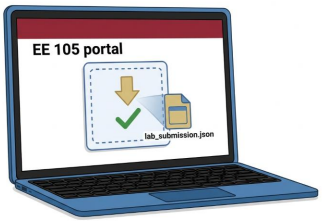
DONE WHEN the cell prints `found 9 of 9 levels` and the ✓ line, and the table shows 16 steady windows (0.8 s) for each level.

✓CHECK **Checkpoint 2** Part D prints the three prices with `A @ x - b` near zero, and Part E finds all nine levels. You solved a linear system and turned a noisy recording into nine named levels.

Save and upload

Save ▸ RUN Write `lab0.2_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the lab notebook (double-click a cell to edit it; Shift+Enter shows it formatted again), and save `bootcamp-questions.ipynb` (Ctrl+S, or Cmd+S on a Mac). Then run the Save cell. It reads your six functions from the saved questions notebook, tests them, runs



Save Upload `lab0.2_submission.json` on the course portal under Lab 0.2.

them on new inputs, and writes the answers with your Part D and E results to `lab0.2_submission.json` in the lab folder. Log in at `ee105.org/portal`, open Lab 0.2, and upload that file. The portal checks the answers, so a function that passes its test only by accident shows up there.

YOU SHOULD SEE `Autograder loaded!`, the six test lines, `Score: 6/6`, and `[ok] Saved lab0.2_submission.json`; then the portal shows your grade.

FINISH THE LAB

You drilled the Python of the course: six NumPy functions, a linear system, and a window method that finds nine levels in a noisy recording. The lab is done when every box is ticked.

☐	**C2**	EDIT	Question 1, `q1`
☐	**C3**	EDIT	Question 2, `q2`
☐	**C4**	EDIT	Question 3, `q3`
☐	**C5**	EDIT	Question 4, `q4`
☐	**C6**	EDIT	Question 5, `q5`
☐	**C7**	EDIT	Question 6, `q6`
☐			`bootcamp-questions.ipynb` saved
☐	**D2**	EDIT	`A`, `b`, `x` of the school store
☐	**E2**	EDIT	`v_mean`, `v_std`
☐	**E4**	EDIT	`nearest_level`
☐	**E5**	EDIT	`w_mean`, `w_spread`
☐	**E6**	EDIT	`counts`
☐		CHECK	Checkpoint 1: `Score: 6/6`
☐		CHECK	Checkpoint 2: the store prices check out; all nine levels found
☐			Questions 1 to 5 answered in the lab notebook
☐	**Save**	SAVE	`lab0.2_submission.json` uploaded on the portal

Typical results

With correct functions the autograder prints `Score: 6/6`. The usual first bugs: Question 2 with rows and columns swapped (`A[::2, 1::2]` instead of `A[1::2, ::2]`); Question 4 stacking side by side (`np.hstack`) instead of below (`np.vstack`); Question 5 with `np.arange(1, N)`, which stops one short; and Question 6 dividing by `A.sum(axis=1)` without `keepdims=True`, which broadcasts the row sums the wrong way. In Part E, `a1_readings.csv` (recorded on the simulated board) scatters by just under one ADC step over five codes (Figure 0.28a); a steady voltage on a real board usually shows one to

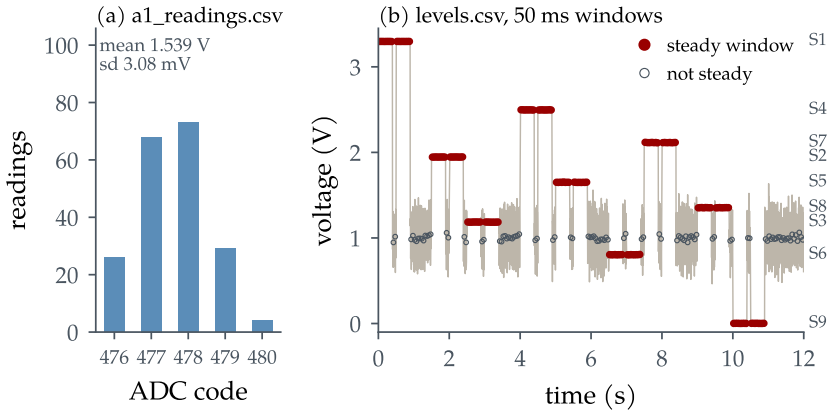


Figure 0.28: Part E with the two data files of the lab. (a) The 200 readings of `a1_readings.csv` fall on only five ADC codes, 476 to 480; the bars stand apart because no reading can land between two codes. (b) The 240 windows of `levels.csv`: 144 are steady (dots on the nine levels, 16 per level); the others fall in the unconnected gaps, where the input wanders around 1 V, or straddle a change of level.

three steps (3 to 10 mV). The window method finds all nine levels of `levels.csv`, 16 steady windows (0.8 s) each (Figure 0.28b).

Troubleshooting

- *`%run autograder_v2.py` is not found, or `NameError` for `test_q1`* (**C1**). Open the questions notebook from the lab folder, so that the autograder file is next to it, and run C1 again after every kernel restart.
- *A test passes alone but `test_all` fails.* You probably changed a function after testing it. Run every question cell again, then **C8**.
- *Shapes do not match.* Print `.shape` of every array in the failing line. Most errors in Questions 4 and 6 are a missing `axis` or `keepdims`.
- *An integer result looks wrong.* `np.arange` and slices stop *before* the stop value, and integer arrays stay integer until you divide.
- *A package is missing* (**A1** prints `[!!]`). Close JupyterLab and run the kit's setup file again.
- *The results file has an old answer although you fixed the function.* The Save step reads the *saved* questions notebook: save it, then run Save again.

Analysis questions

Answer in the Questions cells at the end of the lab notebook.

1. Report your autograder result (**C8**). For any question that failed at first, explain the bug you found.
2. Which NumPy function solves $A\mathbf{x} = \mathbf{b}$? Use it to re-solve Worked Example 0.25 and confirm the prices (**D1**).

3. Explain, with a two-element example of each, the difference between `lst + lst` and `arr + arr`, and between `A * B` and `A @ B`.
4. In Part E, how does the standard deviation compare with one ADC step, and how many distinct codes appeared (**E2**)? Why does the histogram have gaps between bars (**E3**)?
5. Why does `reshape(-1, 10)` need the number of readings to be a multiple of 10, and how did the notebook make sure it was (**E5**)?

Worked examples: Python

These examples practice the Python of Section 0.10 on everyday data (temperatures, step counts, prices), so that you can focus on the language rather than the physics.

Worked Example 0.21 *Filtering sensor readings with a comprehension*

A home weather network reports, for each sensor, its last temperature reading in degrees Celsius and its battery level as a fraction of full. Write a function `readings_with_good_battery` that takes a dictionary of sensors and returns a sorted list of the temperatures from all sensors whose battery is at least 0.5 (half full). Use a list comprehension, no explicit `for` loop. Check it on this input, for which the expected output is `[18.0, 31.0]`:

```
sensors = {'kitchen': {'temp': 24.5, 'battery': 0.25},
           'garage':  {'temp': 31.0, 'battery': 1.0},
           'attic':   {'temp': 18.0, 'battery': 0.5}}
```

Solution. Each value of the outer dictionary is itself a dictionary with keys `'temp'` and `'battery'`. Loop over the inner dictionaries with `.values()`, keep those with a battery of at least 0.5, collect their temperatures, and sort:

```
def readings_with_good_battery(sensors):
    return sorted([s['temp'] for s in sensors.values()
                   if s['battery'] >= 0.5])
```

Tracing the example: the kitchen sensor (0.25) is dropped; the garage sensor (1.0) is kept; the attic sensor is at exactly 0.5, and since the test is $\geq$ it is kept. The comprehension produces `[31.0, 18.0]` in dictionary order, and `sorted()` turns it into `[18.0, 31.0]`, the expected output. Using > instead of >= would wrongly drop the attic. ■

Worked Example 0.22 *A week of step counts*

A fitness tracker recorded these step counts from Monday to Sunday:

```
steps = np.array([8200, 10450, 6300, 12010, 9800, 4300, 7500])
```

Using NumPy without loops, find (a) the total and the daily mean, (b) the day with the most steps, (c) how many days reached at least 8000 steps, and which counts those were, and (d) the distance walked in the week, in kilometers, at 0.75 m per step.

Solution.

```
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
total, mean = steps.sum(), steps.mean()   # 58560, 8365.7
best = days[np.argmax(steps)]             # 'Thu'
good = steps >= 8000                      # mask of 7 booleans
print(good.sum(), steps[good])  # 4 [ 8200 10450 12010  9800]
km = steps.sum() * 0.75 / 1000            # 43.92
```

(a) The total is 58 560 steps and the mean $58\,560/7 = 8365.7$ steps per day. (b) `np.argmax` returns index 3, the largest count (12 010), which is Thursday. (c) The mask is true on four days, with counts 8200, 10 450, 12 010, and 9800. (d) $58\,560 \times 0.75\,\text{m} = 43\,920\,\text{m} = 43.92\,\text{km}$. ■

Worked Example 0.23 *Sorting columns by one row*

Write a function `create_and_sort_array(n)` that creates a $2 \times n$ NumPy array of random integers from 1 to 50 inclusive and returns it with its columns reordered so that the second row (index 1) is in ascending order. Both rows must move together. Check the logic on `[[23, 45, 12, 38], [17, 8, 41, 29]]`.

Solution. Two details matter. First, `np.random.randint(low, high)` excludes `high`, so integers 1 to 50 need `randint(1, 51)`. Second, sorting the second row alone with `np.sort` would break the pairing between the rows. Instead, `np.argsort` returns the column *order* that would sort the row, and indexing every row with that order moves whole columns:

```
import numpy as np

def create_and_sort_array(n):
    arr = np.random.randint(1, 51, size=(2, n))  # 51 excluded
    order = np.argsort(arr[1])     # column order from row 1
    sorted_array = arr[:, order]   # move whole columns
    return sorted_array
```

For the test array, row 1 is $[17, 8, 41, 29]$. Its smallest entry (8) is at index 1, then 17 at index 0, then 29 at index 3, then 41 at index 2, so `argsort` gives `[1, 0, 3, 2]`. Taking columns in that order

gives `[[45, 23, 38, 12], [8, 17, 29, 41]]`: the second row is ascending and each top entry stayed above its original partner. The same trick sorts any table by one of its rows, for example runners' bib numbers by their finishing times. ■

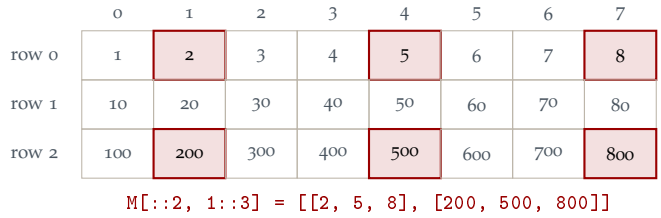


Figure 0.29: Slicing the 3×8 decade table of Worked Example 0.24. `::2` keeps rows 0 and 2; `1::3` keeps columns 1, 4, and 7.

Worked Example 0.24 *Slicing a table of decades*

Engineering notation groups numbers by decades (factors of ten). Write a function `decade_table_slice(rows, cols)` that builds a `rows`×`cols` array whose first row is $[1, 2, 3, \ldots]$, second row $[10, 20, 30, \ldots]$, third row $[100, 200, 300, \ldots]$, and so on. It then keeps every second row starting from row 0 and, from those, every third column starting from column 1, and returns the result. For `rows = 3, cols = 8` the answer should be `[[2, 5, 8], [200, 500, 800]]`.

Solution. Entry (i, j) of the table is $10^i \times (j+1)$: a power of ten for the row times a multiple for the column. That is an outer product of the column vector $[1, 10, 100, \ldots]^T$ and the row vector $[1, 2, \ldots, \text{cols}]$. NumPy builds it without loops by *broadcasting*: give the first vector shape $(\text{rows}, 1)$ with `[:, None]` and multiply; NumPy stretches both to $(\text{rows}, \text{cols})$.

```
import numpy as np

def decade_table_slice(rows, cols):
    decades = 10 ** np.arange(rows)        # [1, 10, 100, ...]
    multiples = np.arange(1, cols + 1)     # [1, 2, ..., cols]
    table = decades[:, None] * multiples   # (rows, cols)
    return table[::2, 1::3]  # rows 0, 2, ...; cols 1, 4, ...
```

The slice `::2` on the row axis means "start at 0, step 2," and `1::3` on the column axis means "start at 1, step 3" (Figure 0.29). For a 3×8 table this keeps rows 0 and 2 and columns 1, 4, and 7, giving `[[2, 5, 8], [200, 500, 800]]` as required. `np.outer(decades, multiples)` builds the same table.

One caution about size. NumPy's default integers are usually 64-bit, with a largest value of about 9.2×10^{18}. A row index of 19

would need 10^{19}, which overflows silently into a wrong, negative number. With at most 9 rows the largest entry is 9×10^8 for 9 columns, far from the limit. ■

WORKED EXAMPLE 0.25 *Prices from receipts*

Three receipts from a campus café read: 2 coffees, 1 bagel, and 1 juice for \$13.25; 1 coffee and 2 bagels for \$8.00; 1 coffee, 1 bagel, and 2 juices for \$13.75. Find the price of each item.

SOLUTION. Call the prices c, b, and j. Each receipt is one linear equation:

$$\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 0 \\ 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} c \\ b \\ j \end{bmatrix} = \begin{bmatrix} 13.25 \\ 8.00 \\ 13.75 \end{bmatrix}.$$

Each row of the matrix holds the quantities on one receipt, and each column belongs to one item.

```
import numpy as np
A = np.array([[2, 1, 1],
              [1, 2, 0],
              [1, 1, 2]])
b = np.array([13.25, 8.00, 13.75])
x = np.linalg.solve(A, b)
print(x)          # [3.5  2.25 4.  ]
print(A @ x)      # [13.25  8.   13.75], the receipts again
```

By hand: the second receipt gives $c = 8 - 2b$. Put that into the first: $2(8 - 2b) + b + j = 13.25$, so $j = 3b - 2.75$. Put both into the third: $(8 - 2b) + b + 2(3b - 2.75) = 13.75$, which simplifies to $5b + 2.5 = 13.75$, so $b = 2.25$. Then $c = 8 - 4.5 = 3.50$ and $j = 6.75 - 2.75 = 4.00$. A coffee costs \$3.50, a bagel \$2.25, and a juice \$4.00, as NumPy found. ■

Summary

- A brain-computer interface such as Neuralink's implant is a complete intelligent system: electrodes (Chapter 3), amplifiers (Chapters 1 to 3), an ADC built from transistors (Chapters 0, 2, 3), spike detection and decoding (Chapter 4), a Bluetooth radio (Chapter 5), and a battery with a power budget (Chapters 0 and 1). Its raw data, 1024×10 bits $\times$ 19.3 kHz $\approx$ 198 Mbit/s, far exceeds a Bluetooth Low Energy link, so data must be reduced on the implant.
- Electrical engineering studies and builds devices and systems that use electricity, electronics, and electromagnetism; most products draw on several subfields at once.
- The transistor is a voltage-controlled switch; we want a large I_{on}, a

tiny I_{off}, fast switching, and low energy. Transistor counts doubled about every two years (Moore's law), from about 2300 in the Intel 4004 (1971) to about 150 billion in 2023. Clock speeds stalled in the 2000s; FinFETs and nanoribbon transistors keep the shrinking going.

- SI prefixes step by factors of 1000; prefixes on squared or cubed units are squared or cubed too; log scales show quantities that span many orders of magnitude.
- Every electronic system can be drawn as sense $\rightarrow$ process $\rightarrow$ communicate, fed by power. The EE 105 modules develop these blocks in turn.
- Voltage is a difference between two points (pressure); current is the rate of charge flow, $1\,\text{A} = 1\,\text{C/s}$; for a resistor $V = IR$; power is $P = VI$ and energy $E = Pt$.
- An N-bit ADC with reference V_{ref} has step $\Delta = V_{\text{ref}}/2^N$, and the sampling rate must exceed twice the highest frequency of interest. On the Nano 33 BLE Sense, `analogRead` returns 0–1023 with $\Delta \approx 3.22\,\text{mV}$; convert with $V = \text{code} \times 3.3/1023$. PWM gives an average voltage $D \times 3.3\,\text{V}$; the AD5593R DAC on the demo board gives true analog outputs at 1.65 mV per code.
- The demo board has a circuits block, a devices block, a biosensing block (sensors and communication), and a camera. Selector S10 (positions printed Module 1, Module 2, Module 3, and Camera) powers one at a time; jumpers J7, J8, and J5 route A0, A1, and A6. The firmware commands include `circuits`, `module`, and `stop`.
- A sketch has `setup()` (once) and `loop()` (forever). Equal on and off times t_{half} give a blink frequency $1/(2t_{\text{half}})$. Use `millis()` rather than `delay()` when the board must do several things at once.
- Python lists, dictionaries, and comprehensions organize data; NumPy arrays add element-wise arithmetic, masks, slicing, and `np.linalg.solve`; each lab is one notebook whose last cell saves the file you upload to the course portal.

0.13 Problems

0.1 ° **The implant by the numbers.** Use the public figures for the Neuralink implant: 1024 electrodes on 64 threads, threads about 5 µm thick, a disk 23 mm across and 8 mm thick, and 1024 channels each sampled at 19.3 kHz.

(a) How many electrodes are on each thread?

(b) Write the thread thickness in nanometers and in meters, and the implant's thickness in micrometers.

(c) How many samples per second does the implant take in total, over all channels?

0.2 ◦ **Map the implant.** For each part of the brain-computer interface, name its block (sense, process, communicate, or power) and the chapter of this book that teaches it: (a) the electrodes; (b) the amplifiers; (c) the ADC; (d) the spike detector; (e) the Bluetooth radio; (f) the decoder on the phone; (g) the battery and charging coil. Then name the part of the EE 105 kit that plays the same role as (a), (e), and (f).

0.3 • **A day of raw data.** The implant of Worked Example 0.1 produces 197.6 Mbit/s of raw data.

(a) How many bytes would one day of raw data fill? Give the answer in terabytes (1 TB $= 10^{12}$ bytes).

(b) How long would a 1 Mbit/s link need to send just one second of raw data?

(c) What is the raw data rate of a single 256-channel chip? Would even one chip's raw data fit through the link?

0.4 • **Events or counts?** An implant with 1024 channels sends each detected spike as a 24-bit message (10-bit channel number, 14-bit time). Each channel fires an average of 15 spikes per second.

(a) What data rate results, and what fraction of a 1 Mbit/s link does it use?

(b) What is the largest average spike rate per channel that fits the link?

(c) A different design sends no times at all: every 25 ms it sends, for each channel, a 4-bit count of the spikes in that interval. What data rate does that need?

(d) What information does the counting design lose, and what is the most spikes per second it can report on one channel?

0.5 • **Power budget.** An implant uses four 256-channel recording chips at 6 mW each, plus an assumed 10 mW for processing and the radio. Its battery stores 200 mAh at 3.7 V.

(a) What is the total power, and what average current does the implant draw from the battery?

(b) How much energy does the battery hold, in watt-hours and joules, and how long does a full charge last?

(c) Suppose the radio needs 10 nJ for every bit it sends. What power would streaming the raw 197.6 Mbit/s take, and how long would the battery last? What power does sending 0.33 Mbit/s of spike messages take?

0.6 •• **Design: a 4096-channel implant.** A future implant has 4096 channels with the same ADC (10 bits, 19.3 kHz) and the same 256-channel, 6 mW chips.

(a) What is the raw data rate? How many chips are needed, and how much power do they use?

(b) How many bits does a channel number need now?

(c) Each spike is sent as a 32-bit message, and each channel averages 20 spikes per second. Does the data fit a 1 Mbit/s link? What is the largest average spike rate that fits?

(d) Propose two changes that would make the design fit the link, and say what each one costs.

0.7 ◦ **Which block?** For each part of the EE 105 system, say whether it belongs mainly to sensing, processing, communication, or power, and explain in one sentence: (a) the APDS9960 proximity sensor; (b) the nRF52840 processor; (c) the Bluetooth radio; (d) the USB cable; (e) the OV7675 camera; (f) the AD5593R DAC driving LED pair 2 in the communication lab; (g) a Jupyter notebook that computes heart rate from a PPG.

0.8 • **Moore's law.** Assume transistor counts double every two years, starting from 2300 transistors in 1971.

(a) What count does this predict for 2023? Compare with the 150 billion of the MI300.

(b) Starting from 150 billion transistors in 2023 and keeping the same pace, in what year would a chip reach one trillion (10^{12}) transistors?

(c) A different technology improves by a factor of 1000 every 20 years. What is its doubling time?

0.9 • **Extrapolating clock speeds.** Use the numbers of Section 0.3: 750 kHz in 1971 and 66 MHz in 1993.

(a) Assuming steady exponential growth, by what factor did clock speed grow each year? How many years did it take to double?

(b) Predict the clock speed in 2012 and compare with the 1 to 3 GHz quoted in the text.

(c) Predict the clock speed in 2024. Why do real processors not run that fast?

0.10 ◦ **Prefix practice.** Convert (a) 0.0047 MΩ to kilohms; (b) 2200 pF to microfarads; (c) 15 mA to microamperes; (d) 5 nm to micrometers; (e) 230 million transistors per mm^2 to transistors per μm^2.

0.11 ◦ **Clock periods.** (a) What is the clock period of the nRF52840 at 64 MHz? (b) Of the Arduino Uno at 16 MHz? (c) The microphone delivers samples at 16 kHz. How many nRF52840 clock cycles pass between two samples?

0.12 ◦ **Battery units.** A phone battery is rated 2000 mAh at 3.7 V.

(a) How much charge does it hold, in coulombs?

(b) How much energy does it hold, in watt-hours and in kilojoules?

(c) If a full charge lasts 10 h, what are the average current and the average power?

0.13 ◦ **Ohm's law as a relation.** A resistor has 2.0 V across it and carries 4.0 mA.

(a) What is its resistance, in ohms and in kilohms?

(b) What current would the same resistor carry with 3.3 V across it?

(c) How much power does it dissipate at 3.3 V?

(d) Show from the units in Table 0.3 that V^2/R is measured in watts.

0.14 • **Voltage, current, and resistance in words.** Answer in one or two sentences each.

(a) Why is a voltage always given between two points? What does "this node is at 3.3 V" mean?

(b) In a copper wire, which way do the electrons move compared with the conventional current?

(c) A wire is replaced by one of the same material that is twice as long and has twice the diameter. By what factor does its resistance change? Use Eq. (0.2).

(d) Why can't a microcontroller pin rated 15 mA drive a motor that needs 500 mA, even though both run at a few volts?

0.15 • **Memory budget.** The Nano 33 BLE Sense has 256 KB of SRAM

(1 KB = 1024 bytes). Each 10-bit or 16-bit sample is stored in 2 bytes.

(a) How many samples fit in the whole SRAM?

(b) How many seconds of 16 kHz microphone audio is that?

(c) The implant's raw stream is 24.7 MB/s (1 MB = 10^6 bytes). How long would it take to fill 256 KB?

(d) How many 2-byte samples would fit in the 4 KB of the Apollo Guidance Computer, and how long would they last at 16 kHz?

0.16 • **ADC arithmetic.** The Nano 33 BLE Sense ADC has 10 bits and a 3.3 V reference.

(a) What is the step size Δ in millivolts?

(b) What code does a 1.20 V input produce? (Use code $= V/3.3 \times 1023$, rounded.)

(c) What voltage corresponds to code 700?

(d) If the ADC is set to 12 bits, what is the new step size?

0.17 • **How many bits?** You want to resolve changes of 1 mV in a signal that spans 0 to 3.3 V. What is the minimum number of ADC bits, and what step size does that ADC actually give?

0.18 •• **Choosing the ADC resolution.** Return to the recording channel of Worked Example 0.12: gain 200, ADC range 1.2 V, amplifier noise 5.9 µV rms, 1024 channels at 19.3 kHz.

(a) With a 12-bit ADC instead of 10 bits, what is the step referred to the electrode, and what is the raw data rate?

(b) With an 8-bit ADC, what is the step referred to the electrode? How many times the noise is it?

(c) Which of 8, 10, and 12 bits would you choose, and why? Consider resolution, noise, and data rate.

0.19 ◦ **Binary and hexadecimal.**

(a) Convert $1011\,0110_2$ to decimal and to hex.

(b) Convert `0x3FF` and `0x57` to decimal.

(c) Convert 300 to binary and to hex.

(d) How many bits are needed to number 64 threads? 1024 channels? 4096 channels?

0.20 • **How fast to sample?** Use the rule that the sampling rate must be more than twice the highest frequency of interest.

(a) Speech content of interest extends to 4 kHz. What is the minimum sampling rate?

(b) A PPG has useful content below 10 Hz. What is the minimum sampling rate? At 25 Hz, how many samples are recorded in 10 s?

(c) For the 19.3 kHz neural ADC, what is the time between samples and the highest frequency it can capture?

(d) How many bytes per second does a 16 kHz microphone produce with 16-bit samples?

0.21 • **DAC codes.** A sketch sets the AD5593R output at 1.65 mV per code and truncates fractional codes, as in Worked Example 0.15. (a) What codes produce 1.0 V and 2.5 V, and what voltages actually result? (b) What is one code step as a percentage of 3.3 V?

0.22 • **PWM.** `analogWrite(pin, value)` on the Nano 33 BLE Sense switches between 0 V and 3.3 V with duty cycle value/255.

(a) What value gives an average of 1.0 V as closely as possible, and what average results?

(b) What are the duty cycle and the average voltage for `value = 191`?

(c) Why does an LED driven this way look steadily dim rather than flickering?

0.23 • **Reading the selector.** Use the selector levels and thresholds of Worked Example 0.17.

(a) A7 reads codes 690, 350, 240, and 60 at four different times. Convert each to volts and name the selector position.

(b) What is the smallest difference between two neighboring selector levels, in volts and in ADC codes?

(c) Why does the firmware compare with thresholds halfway between levels rather than testing for exact codes?

0.24 ◦ **Blink timing.** (a) A sketch turns the LED on for 250 ms and off for 750 ms. What are the blink frequency and the duty cycle? (b) What equal delays give a 2.5 Hz blink?

0.25 • **Reading Arduino code.** Consider this loop:

```
void loop() {
  int code = analogRead(A0);
  int duty = map(code, 0, 1023, 0, 255);
  analogWrite(LED_BUILTIN, duty);
  Serial.println(duty);
```

```
  delay(50);
}
```

(a) `analogRead(A0)` returns 512. `map` computes $(\text{code}-0)\times(255-0)/(1023-0)+0$ in integer arithmetic. What does it print?

(b) What average voltage appears on the LED pin?

(c) At most how many lines per second does the loop print?

(d) A student writes `float volts = code * (3 / 1023);` and always gets 0. Why, and how do you fix it?

0.26 •• **Serial data budget.** A hardware UART sends each byte as 10 bits (a start bit, 8 data bits, and a stop bit).

(a) At 115200 baud, how many bytes per second can it carry?

(b) A sketch prints readings such as `1.65` followed by a carriage return and newline (6 bytes per line). How many lines per second fit?

(c) A three-axis accelerometer produces 16-bit readings on each axis at 100 Hz, sent as raw binary. What byte rate is needed, and does it fit in a 9600-baud link?

(d) A three-axis gyroscope with the same format is added. Does 9600 baud still suffice? If not, what is the lowest standard rate among 9600, 19200, 38400, and 115200 that works?

0.27 • **Converting a batch of readings in Python.** An array of ADC codes is `codes = np.array([12, 512, 1023, 767, 249, 603])`. Using NumPy without loops, write expressions that (a) convert all codes to volts with Eq. (0.6), (b) find the index of the largest voltage, (c) compute the mean voltage, and (d) count how many readings exceed 2.0 V. Give the numerical results.

0.28 • **A button classifier in Python.** A simple sketch identifies touch-panel buttons from the A0 code with these open intervals: S1 (1000, 1020), S2 (590, 615), S3 (360, 375), S4 (755, 775), S5 (500, 520), S6 (240, 260), S7 (635, 660), S8 (400, 425), S9 (0, 200).

(a) Write a Python function `classify(code)` that returns the button name, or `None` if the code is in no interval. Store the intervals in a dictionary.

(b) Use a list comprehension to classify the codes `[604, 250, 1010, 418, 655, 100, 530]`.

(c) Convert the S5 interval to volts. How far are its edges below and above the ideal 1.65 V?

(d) Button S1 puts exactly 3.3 V on A0, and S9 puts 0 V. Using Table 0.9, will this classifier ever report S1? Will it reliably report S9? Suggest fixes.

0.29 ° **Temperatures with comprehensions.** Five days of noon temperatures are `temps_c = [18.5, 21.0, 25.5, 30.0, 16.0]` (Monday to Friday).

(a) Write a list comprehension that converts them to Fahrenheit ($F = 9C/5 + 32$) and give the result.

(b) Write a comprehension that keeps only the Fahrenheit values above 70 °F.

(c) Write a dictionary comprehension that maps each day name to its Celsius temperature, and use it to count the days warmer than 20 °C.

0.30 • **Sound levels with NumPy.** A sound meter logged twelve readings in decibels, one every 5 seconds: `levels = np.array([42, 45, 44, 60, 72, 68, 50, 47, 46, 43, 41, 44])`.

(a) Reshape the array into four rows of three readings (15 seconds each) and compute the mean of each row.

(b) Which 15-second interval was loudest? Use `np.argmax`.

(c) Use a mask to list the readings above 55 dB.

(d) What do `levels[::3]` and `levels[-4:]` return?

(e) Give one line of NumPy that returns the three loudest readings, largest first.

0.31 •• **Calibrating a temperature sensor.** A temperature sensor's ADC code is a straight-line function of temperature, code $= aT + b$. In a calibration run it reads code 412 at 20.0 °C and code 537 at 45.0 °C.

(a) Write the two readings as a matrix equation for the unknowns a and b, and solve it with `np.linalg.solve`. Give a (with units) and b.

(b) What temperature does code 450 mean?

(c) Convert the readings `np.array([430, 455, 470])` to temperatures in one NumPy expression.

(d) What temperature change does one ADC code represent?

0.32 • **Mixing solutions.** A lab has a 10% salt solution and a 40% salt solution.

(a) How much of each must be mixed to make 600 mL of a 20% solution? Write the two conditions (total volume, total salt) as $A\mathbf{x} = \mathbf{b}$ and solve it.

(b) Check your answer with `A @ x`.

(c) Solve the same system for a 45% target. What does the answer tell you?

0.33 •• **Timing a light link.** A simple LiFi transmitter (Chapter 5 explains LiFi) sends each character as a frame of 10 time slots of 3 ms each: one start slot (LED off), 8 data bits, and one stop slot (LED on). It sends a 30-character message and then waits 1000 ms before repeating.

(a) How long does one character take?

(b) How long does one repetition (message plus pause) take?

(c) What is the data rate in bits per second while a character is being sent, counting only the 8 data bits?

(d) The receiver treats a photodiode reading above code 200 (on A6) as light. What voltage is that?

(e) What would the data rate in (c) become with 1 ms slots?

0.34 •• **Design: a wearable pulse patch.** A skin patch measures a PPG with a red and an infrared LED and a photodiode, as in Chapter 3.

(a) Draw a block diagram of the patch and label the sensing, processing, communication, and power blocks.

(b) A resting heart rate is 72 beats per minute. What is that frequency in hertz? If the PPG is sampled at 25 Hz, how many samples fall in one beat, and how much time separates samples?

(c) With 16 bits per sample on two channels (red and infrared), what raw data rate does the patch produce, in megabytes per day?

(d) Give one advantage of streaming raw samples to a phone and one of computing heart rate on the patch.

1
Circuits

In this chapter you will learn to

- Define charge, current, voltage, resistance, and power, and compute a resistance from its material and shape with $R = \rho L / A$.
- Describe the ideal circuit elements (wire, open circuit, resistor, voltage source, current source) by their current-voltage relationships.
- Find the nodes of any schematic and write Kirchhoff's current and voltage laws for it.
- Reduce series and parallel resistor networks, and use voltage dividers, current dividers, and the Wheatstone bridge.
- Measure voltage, current, and resistance correctly, estimate the error a meter introduces, and explain how a fuse protects a circuit.
- Solve a resistor network by nodal analysis, by hand and as a matrix equation $\mathbf{G}\mathbf{v} = \mathbf{i}$ in Python.
- Replace a two-terminal network by its Thevenin or Norton equivalent, and describe what capacitors and inductors are and how they store energy.
- Predict, and then measure, the node voltages of the demo board's resistive touch panel.

Press any of the nine buttons on the circuits block of the EE 105 demo board and the Arduino tells you which one you pressed. There is no keypad controller on the board and no special chip. There are twelve small resistors arranged in a square grid, a 3.3 V supply, and a single analog input pin. Each button connects that pin to a different corner of the grid, and every corner sits at a different voltage. Read

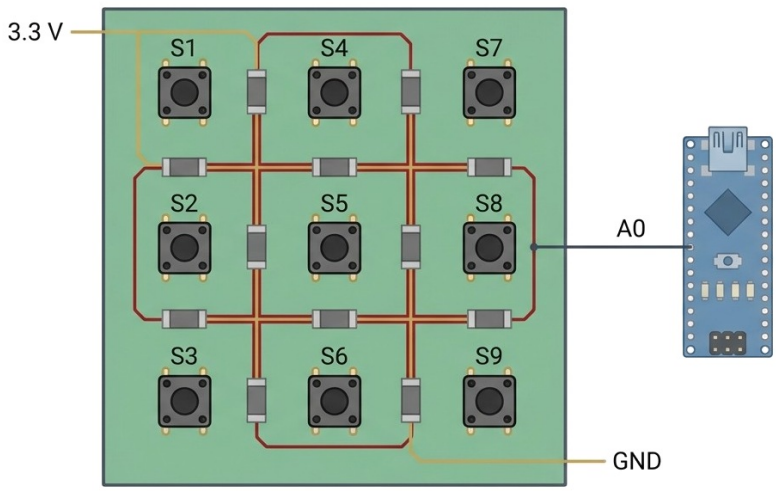


Figure 1.1: The touch panel on the circuits block of the EE 105 demo board. Nine push buttons, S1 to S9, sit at the corners of a resistor grid. Pressing a button connects that corner to the Arduino's analog pin A0, which measures its voltage.

the voltage and you know the button (Figure 1.1).

Predicting those nine voltages before you power the board is the goal of this chapter. On the way you will meet the quantities every electrical engineer works with (charge, current, voltage, resistance, and power), the two conservation laws that tie them together, and a systematic recipe, nodal analysis, that turns any resistor network into a set of linear equations a computer can solve in a millisecond. The same ideas explain the resistive touch screens in ATMs and car dashboards, the meters you will use on the bench, and the fuse that keeps a short circuit from turning into a fire. Chapter 2 then adds the first nonlinear elements, diodes and transistors, and Chapter 3 uses everything here to read real sensors.

1.1 *Charge, Current, Voltage, and Resistance*

Everything electrical starts with charge. An electron carries a charge of $-q$, where $q = 1.602 \times 10^{-19}\,\text{C}$ and the coulomb (C) is the unit of charge. A proton carries $+q$. Ordinary matter is neutral because it holds the same number of each, but in a metal some electrons are only loosely bound and can drift from atom to atom. A circuit is a closed path that lets those mobile charges move and do something useful along the way: light an LED, heat a filament, or change the reading of an analog-to-digital converter.

Current is the rate at which charge passes through a surface, such as the cross section of a wire. If an amount of charge ΔQ crosses in a time Δt, the average current is $I = \Delta Q / \Delta t$. When the flow changes

smoothly in time we take the limit and write

$$i(t) = \frac{dQ}{dt}. \tag{1.1}$$

The unit is the ampere: one ampere (A) is one coulomb per second, which is about 6.24×10^{18} electrons every second. The currents in this course are much smaller, typically milliamperes (mA) on the demo board and microamperes (µA) in a photodiode. Running the definition backward, the charge that passes in a time interval is the current multiplied by the time, $\Delta Q = I\,\Delta t$, for a steady current. A milliampere flowing for one second moves one millicoulomb.

A current has a direction, shown by an arrow on the schematic. By a convention that predates the discovery of the electron, the arrow points the way *positive* charge would move.[1] If your analysis returns a negative current, the charge simply flows opposite to the arrow you drew. You never need to guess directions correctly in advance.

[1] Benjamin Franklin picked the sign of "positive" charge in the 1740s, about 150 years before J. J. Thomson identified the electron. In a metal wire the electrons actually drift opposite to the conventional current arrow. Circuit analysis does not care, because a negative charge moving left carries the same current as a positive charge moving right.

Voltage is the energy per unit charge needed to move a charge between two points. If moving a charge Q from point b to point a takes energy W, the voltage of a relative to b is

$$V_{ab} = V_a - V_b = \frac{W}{Q}, \tag{1.2}$$

measured in volts (V), where one volt is one joule per coulomb. Voltage is always a *difference* between two points. When we talk about "the voltage of a node" we mean its voltage relative to a chosen reference point called ground, which we define to be 0 V.

A useful picture is water in pipes (Figure 1.2). A pump lifts water to a tank; the height of the water plays the role of voltage. The water then flows back down through the pipes, and the flow rate plays the role of current. A narrow section of pipe resists the flow and causes the pressure to drop across it, just as a resistor causes a voltage drop. The pump is the battery: it does work on the water to raise it, and the water gives that energy back as it falls. Like every analogy this one breaks down eventually, but it gets the signs right, and it makes Kirchhoff's voltage law (Section 1.3) feel obvious: walk around any closed loop and you end at the height you started from.

Resistance measures how strongly an element opposes current. For many materials (metals, carbon films, doped semiconductors at fixed temperature) the current is proportional to the voltage across the element, and the constant of proportionality defines the resistance R, measured in ohms (Ω). This proportionality is *Ohm's law*:

$$V = IR \qquad \Longleftrightarrow \qquad I = \frac{V}{R} = GV. \tag{1.3}$$

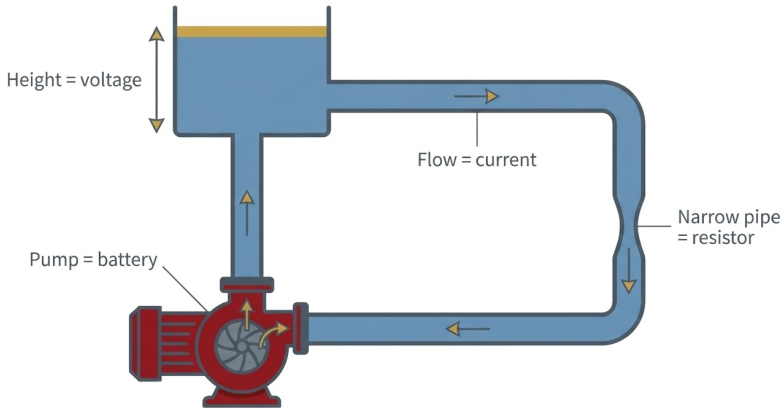


Figure 1.2: The water analogy. The pump (battery) lifts water to a higher level (voltage); water flows back down (current) through a narrow pipe (resistor), losing height as it goes. Around the full loop the net change in height is zero.

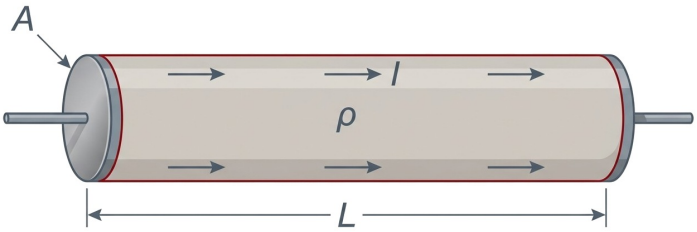


Figure 1.3: A resistor is a piece of material with resistivity ρ, length L along the current, and cross-sectional area A. Its resistance is $R = \rho L/A$: doubling the length doubles R, doubling the area halves it.

The reciprocal $G = 1/R$ is the conductance, measured in siemens (S). Conductance is handy when elements share a voltage, as you will see with parallel resistors and nodal analysis.

Resistance depends on what the element is made of and on its shape. Picture a uniform bar of length L and cross-sectional area A (Figure 1.3). A longer bar is like several short bars placed end to end, so its resistance grows in proportion to L. A fatter bar is like several thin bars side by side, each carrying part of the current, so its resistance falls in proportion to A. The material enters through its resistivity ρ, in ohm-meters:

$$R = \rho \frac{L}{A}. \tag{1.4}$$

Table 1.1 lists typical values. They span more than twenty orders of magnitude, which is why copper makes wires, glass makes insulators, and silicon, sitting in the middle, makes transistors (Chapter 2).

Power. A charge Q that falls through a voltage V gives up energy QV. If charge flows at a rate $I = dQ/dt$, energy is delivered at the rate

$$P = VI, \tag{1.5}$$

in watts (W), one joule per second. Combining with Ohm's law gives two other forms for a resistor, $P = I^2R = V^2/R$. A resistor turns all of this power into heat, so every real resistor has a power rating; the small surface-mount parts on the demo board are rated around 0.1 W,

KEY EQUATION

$V = IR$

Ohm's law: the voltage across a resistor is its current times its resistance.

Material	ρ (Ω m)
Silver	1.6×10^{-8}
Copper	1.7×10^{-8}
Aluminum	2.8×10^{-8}
Nichrome	1.1×10^{-6}
ITO film	$\sim 10^{-6}$
Seawater	~ 0.2
Silicon (pure)	$\sim 2 \times 10^{3}$
Glass	10^{10}–10^{14}

Table 1.1: Approximate resistivities near 20 °C. ITO (indium tin oxide) is the transparent conductor in touch screens.

and a common through-hole resistor is rated 0.25 W. Energy is power accumulated over time, $W = Pt$, which is why batteries are rated in watt-hours. A milliamp-hour rating gives charge instead; multiply it by the battery voltage to get energy.

Worked Example 1.1 *Counting charge*

(a) A USB cable carries a steady 20 mA into a circuit for one minute. How much charge flows, and how many electrons is that? (b) A sensor draws 2 mA for 3 s and then 5 mA for 2 s. How much charge does it take in those 5 s, and what is its average current?

Solution. (a) For a steady current, $\Delta Q = I\,\Delta t = (0.020\,\text{A})(60\,\text{s}) = 1.2\,\text{C}$. Each electron carries $1.602 \times 10^{-19}\,\text{C}$, so the number of electrons is

$$N = \frac{1.2\,\text{C}}{1.602 \times 10^{-19}\,\text{C}} = 7.49 \times 10^{18}.$$

(b) Add the charge of the two intervals. Since one milliampere for one second is one millicoulomb,

$$Q = (2\,\text{mA})(3\,\text{s}) + (5\,\text{mA})(2\,\text{s}) = 6 + 10 = 16\,\text{mC}.$$

The average current is the total charge divided by the total time: 16 mC in 5 s is 3.2 mA. That is not the plain average of 2 and 5 mA (3.5 mA), because the two currents last for different times. ■

Worked Example 1.2 *Resistivity of a cylindrical conductor*

A cylindrical conductor has length $L = 10\,\text{cm}$, radius $r = 1\,\text{mm}$, and resistance $R = 0.54\,\text{m}\Omega$. Which resistivity matches it: (a) 1.7×10^{-5}, (b) 1.7×10^{-6}, (c) 1.7×10^{-8}, or (d) $5.4 \times 10^{-9}\ \Omega\,\text{m}$?

Solution. Solve Equation (1.4) for the resistivity, $\rho = RA/L$, and keep everything in SI units. The cross-sectional area is

$$A = \pi r^2 = \pi\,(1 \times 10^{-3}\,\text{m})^2 = 3.14 \times 10^{-6}\ \text{m}^2.$$

Then

$$\rho = \frac{(0.54 \times 10^{-3}\,\Omega)(3.14 \times 10^{-6}\,\text{m}^2)}{0.10\ \text{m}} = 1.70 \times 10^{-8}\ \Omega\,\text{m}.$$

The answer is (c). Table 1.1 tells you the conductor is copper. A quick sanity check: a 10 cm piece of thick copper wire should have a resistance far below one ohm, and 0.54 mΩ is. ■

WORKED EXAMPLE 1.3 *Scaling a block*

By what factor does the resistance of a rectangular block change if you double every dimension? (a) 1, (b) 2, (c) $\tfrac{1}{2}$, (d) $\tfrac{1}{4}$.

SOLUTION. Let the block have length L along the current and a cross section of width w and thickness t, so $R = \rho L/(wt)$. Doubling all three dimensions gives

$$R' = \rho \frac{2L}{(2w)(2t)} = \frac{2}{4}\rho\frac{L}{wt} = \frac{R}{2}.$$

The answer is (c): the resistance halves. The length doubles R but the area quadruples, and area wins. This is one reason power cables are thick and why the tiny wires inside a chip have surprisingly large resistance. ■

WORKED EXAMPLE 1.4 *Ohm's law and a power rating*

A $10\,\Omega$ resistor is connected across a 5 V supply. Find the current and the power it dissipates. Can you use a 0.25 W resistor?

SOLUTION. Ohm's law gives $I = V/R = 5\,\text{V}/10\,\Omega = 0.5\,\text{A}$. The power is $P = VI = (5\,\text{V})(0.5\,\text{A}) = 2.5\,\text{W}$; as a check, $V^2/R = 25/10 = 2.5\,\text{W}$ and $I^2R = 0.25 \times 10 = 2.5\,\text{W}$. That is ten times the rating of a 0.25 W part, which would overheat within seconds. You need a power resistor rated well above 2.5 W (a 5 W part leaves a comfortable margin), or a larger resistance if the circuit allows it. ■

1.2 Circuit Elements and Schematics

A SCHEMATIC IS A MAP, not a picture. It shows which terminals are connected to which, and it says nothing about where parts sit on the board or how long the wires are. Each symbol stands for an *ideal element* defined entirely by the relationship it enforces between the current through it and the voltage across it. Figure 1.4 collects the five you need for this chapter.

Wire (short circuit). An ideal wire has zero voltage across it, whatever current it carries. The current is set by the rest of the circuit. Placing a wire directly across an element is called *shorting* it: the element's voltage is forced to zero, so a shorted resistor carries no current.

Open circuit. The dual of the wire: zero current, and any voltage the rest of the circuit cares to put across it. An open switch, a broken trace, and a blown fuse are open circuits. A resistor with $R \to \infty$ behaves

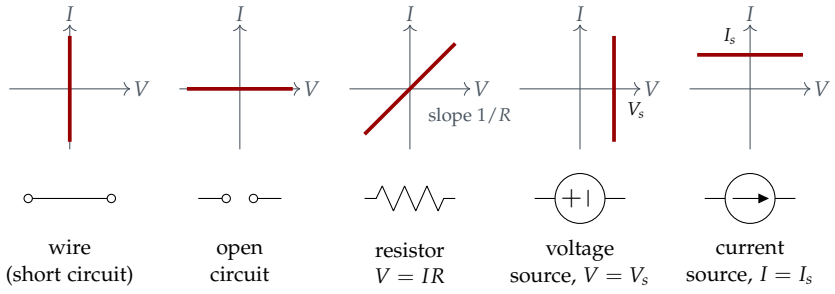


Figure 1.4: The five ideal elements and their current-voltage (I-V) relationships. A wire forces $V = 0$ and allows any current; an open circuit forces $I = 0$ and allows any voltage. A voltage source fixes V, a current source fixes I, and a resistor ties them together with Ohm's law.

like an open circuit, and one with $R \to 0$ behaves like a wire.

Resistor. Ohm's law, a straight line through the origin with slope $1/R$ on the I-V plot.

Ideal voltage source. Holds the voltage between its $+$ and $-$ terminals at V_s no matter what current flows; the current is decided by what you connect. A battery or a bench supply approximates one. Because an ideal source would happily deliver infinite current into a short, a real source is modeled as an ideal source in series with a small internal resistance R_s, which limits the current when the output is shorted.

Ideal current source. Pushes the current I_s in the direction of its arrow no matter what voltage appears across it. The light-driven current of a photodiode (Chapters 2 and 3) is well described this way. A real current source has a large resistance in parallel with the ideal one.

Switches and push buttons toggle between the two extremes: open (no current) and closed (a wire). Each button on the demo board is a normally open push button that closes only while you press it.

Ground and the passive sign convention. A ground symbol marks the reference node whose voltage we call 0 V. When you label an element, draw the current arrow so that it *enters the $+$ terminal* of the element's voltage. With this passive sign convention, $P = VI$ is the power the element absorbs. A resistor always comes out with $P > 0$. A source that is delivering energy comes out negative, because its current leaves through the $+$ terminal.

WORKED EXAMPLE 1.5 *Quantities, units, and symbols*

Define voltage, current, resistance, capacitance, and inductance, give their units, and draw their circuit symbols.

SOLUTION. Figure 1.5 collects the units and symbols.

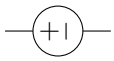
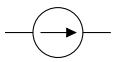
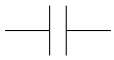
QUANTITY	UNIT	SYMBOL
voltage V	volt (V) $= \mathrm{J/C}$	
current I	ampere (A) $= \mathrm{C/s}$	
resistance R	ohm (Ω) $= \mathrm{V/A}$	
capacitance C	farad (F) $= \mathrm{C/V}$	
inductance L	henry (H) $= \mathrm{V{\cdot}s/A}$	

Figure 1.5: The five basic electrical quantities, their units, and the circuit symbol of the element associated with each.

- *Voltage* is the energy per unit charge between two points, $V = W/Q$, in volts (V). Symbol: a source with $+$ and $-$ marks, or the pair of $+$ and $-$ labels written across any element.
- *Current* is the rate of flow of charge, $I = dQ/dt$, in amperes (A). Symbol: an arrow on a wire; a current source is a circle with an arrow.
- *Resistance* is the ratio of voltage to current for an element that obeys Ohm's law, $R = V/I$, in ohms (Ω). Symbol: the zigzag.
- *Capacitance* is the charge stored per volt, $C = Q/V$, in farads (F). Symbol: two parallel plates. (The coulomb is the unit of *charge*; a farad is one coulomb per volt.)
- *Inductance* is the voltage produced per unit rate of change of current, $L = v/(di/dt)$, in henries (H). Symbol: a coil.

Capacitors and inductors are introduced in Section 1.9. ■

1.3 *Kirchhoff's Laws*

Two conservation laws are enough to analyze any circuit made of the elements above. Before stating them we need a precise idea of a node.

Nodes. A *node* is a set of points joined by ideal wires. Since a wire has no voltage across it, every point in a node is at the same voltage, no matter how the wires are drawn. A schematic often draws one node as several separate-looking lines, and finding the nodes is the first step of every analysis. A simple recipe is to take a highlighter: start at the $+$ terminal of the source and color every wire you can reach without passing through an element. That colored set is one node. Pick an uncolored wire, use a new color, and repeat until every wire is colored.

Figure 1.6 shows why the recipe matters. At first glance the circuit looks like it has many junctions. After coloring, there are only three

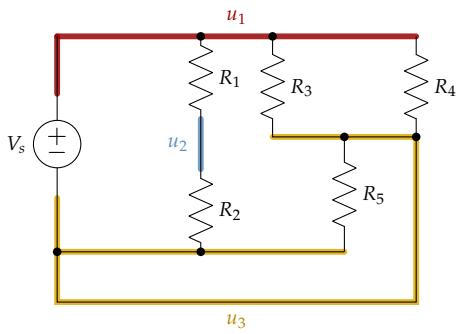


Figure 1.6: Finding nodes. The cardinal wires form node u_1, the short blue segment is node u_2, and every gold wire belongs to node u_3, even though it is drawn as two separate horizontal lines. The circuit has only three nodes. Both ends of R_5 land on u_3, so R_5 is shorted and carries no current; R_3 and R_4 are both connected between u_1 and u_3, directly across the source.

nodes. Resistors R_3 and R_4 connect u_1 to u_3, so each has the full source voltage across it. The chain R_1, R_2 also spans u_1 to u_3 with u_2 in the middle. And R_5 connects u_3 to itself: it is shorted and plays no role at all. Redrawing a circuit after coloring its nodes is often the fastest way to see what it really does.

Kirchhoff's current law (KCL). Charge cannot pile up in a wire, and it cannot vanish. Whatever current flows into a node must flow out:

$$\sum_{\text{into node}} I = \sum_{\text{out of node}} I \qquad \text{or} \qquad \sum_{\text{node}} I_k = 0, \tag{1.6}$$

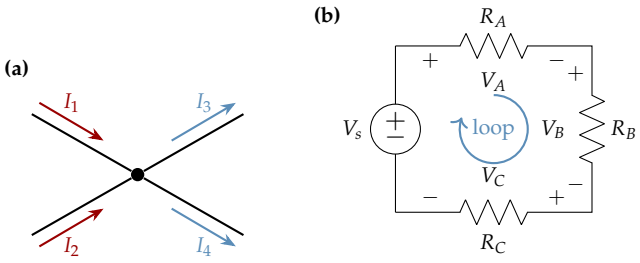
where in the second form currents leaving are counted positive and currents entering negative (either sign choice works if you use it consistently). For the node in Figure 1.7a, $I_1 + I_2 = I_3 + I_4$. KCL is conservation of charge.

Kirchhoff's voltage law (KVL). The voltages around any closed loop add to zero:

$$\sum_{\text{loop}} V_k = 0. \tag{1.7}$$

In the water picture, a hiker who walks a closed loop ends at the elevation where she started; every climb is paid back by a descent. To apply KVL, walk around the loop in either direction. When you enter an element at its $+$ terminal, add its voltage (a drop); when you enter at its $-$ terminal, subtract it (a rise). For the loop in Figure 1.7b, walking clockwise from the bottom left gives $-V_s + V_A + V_B + V_C = 0$, or $V_s = V_A + V_B + V_C$: the source voltage is shared among the three resistors. KVL is conservation of energy. A handy consequence: once every node voltage is known, the voltage across any element is simply the difference of the node voltages at its two ends, and KVL holds automatically.

KEY IDEA KCL: what flows into a node flows out. KVL: the voltages around a closed loop sum to zero.



$I_1 + I_2 = I_3 + I_4$ $\qquad$ $-V_s + V_A + V_B + V_C = 0$

Figure 1.7: (a) KCL at a node: the currents in equal the currents out. (b) KVL around a loop, walked clockwise: each resistor is entered at its + terminal (a drop) and the source at its − terminal (a rise).

Worked Example 1.6 *Stacked batteries*

Three 3 V cells are stacked as in Figure 1.8, with the bottom of the stack grounded. Find the node voltages V_A, V_B, and V_C.

Solution. No current flows (there is no closed path through any resistor), so each cell simply adds its voltage to the node below it. Apply KVL from ground upward, adding +3 V each time you pass from a − terminal to a + terminal:

$$V_C = 0 + 3 = 3\,\text{V}, \qquad V_B = V_C + 3 = 6\,\text{V}, \qquad V_A = V_B + 3 = 9\,\text{V}.$$

This is how a 9 V battery is built inside: six 1.5 V cells in series. If one cell were flipped, its contribution would subtract instead (with the middle cell flipped, $V_B = 0$ V and $V_A = 3$ V), so trace the polarity of every cell rather than adding blindly. ■

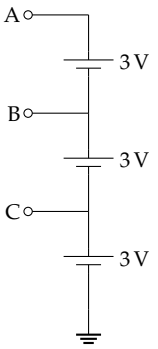


Figure 1.8: Three 3 V cells stacked in series. The long plate of each cell is its + terminal.

Worked Example 1.7 *KCL at a single node*

In Figure 1.9, a 10 V source feeds node N through 5 Ω, and N returns to ground through 10 Ω and 5 Ω. Find the node voltage V_N and the three currents, and check KCL.

Solution. Write each current with Ohm's law in terms of the unknown V_N. The current in a resistor equals the voltage at its starting end minus the voltage at its finishing end, divided by R:

$$I_1 = \frac{10 - V_N}{5}, \qquad I_2 = \frac{V_N}{10}, \qquad I_3 = \frac{V_N}{5}.$$

KCL at N says $I_1 = I_2 + I_3$. Multiply the whole equation by 10:

$$2(10 - V_N) = V_N + 2V_N \implies 20 = 5V_N \implies V_N = 4\,\text{V}.$$

Then $I_1 = 6/5 = 1.2$ A, $I_2 = 0.4$ A, and $I_3 = 0.8$ A. Check: $0.4 + 0.8 = 1.2$, so KCL holds. Notice what just happened: one KCL

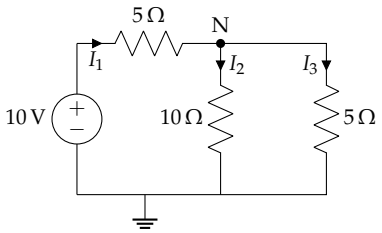


Figure 1.9: A single node N fed through 5 Ω and drained through two resistors to ground.

equation with one unknown voltage solved the whole circuit. That is the core of nodal analysis (Section 1.6). ■

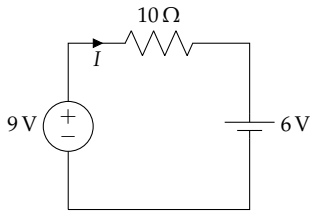


Figure 1.10: A 9 V supply charging a 6 V battery through 10 Ω. The battery's + terminal (long plate) is at the top.

Worked Example 1.8 *Who supplies and who absorbs?*

In Figure 1.10, a 9 V supply charges a 6 V battery through a 10 Ω resistor. Find the current and the power absorbed by each of the three elements.

Solution. One loop, one current. Walk clockwise from the bottom left: up through the supply (a rise of 9 V), across the resistor (a drop $10I$), and down through the battery, entering its + terminal (a drop of 6 V). KVL gives

$$-9 + 10I + 6 = 0 \quad \Longrightarrow \quad I = \frac{9-6}{10} = 0.3\,\text{A}.$$

Now apply the passive sign convention to each element. The resistor absorbs $I^2R = (0.3)^2(10) = 0.9\,\text{W}$. The current *enters* the battery's + terminal, so the battery absorbs $(6)(0.3) = 1.8\,\text{W}$: it is being charged. The current *leaves* the supply's + terminal, so the supply absorbs $-(9)(0.3) = -2.7\,\text{W}$; in other words it delivers 2.7 W. The total is $0.9 + 1.8 - 2.7 = 0$. In every circuit the powers absorbed add to zero, so the power delivered by the sources equals the power taken by everything else. ■

Worked Example 1.9 *From meter readings to every current*

In Figure 1.11, the source holds node a at 6 V and a voltmeter reads $V_b = 4.00\,\text{V}$ and $V_c = 3.00\,\text{V}$. Find the current in every resistor, the unknown R_5, and the power the source delivers.

Solution. Every resistor voltage is a difference of two node voltages (KVL). Divide by the resistance to get each current (Ohm's law), in mA with kΩ:

$$I_1 = \frac{6-4}{1} = 2\,\text{mA (a to b)}, \quad I_2 = \frac{6-3}{2} = 1.5\,\text{mA (a to c)},$$
$$I_3 = \frac{4-3}{1} = 1\,\text{mA (b to c)}, \quad I_4 = \frac{4}{4} = 1\,\text{mA (b to ground)}.$$

KCL at b checks the readings: 2 mA in, $1 + 1 = 2\,\text{mA}$ out. KCL at c gives the current we could not compute directly: in through R_2 and R_3, $1.5 + 1 = 2.5\,\text{mA}$, so R_5 carries 2.5 mA to ground and

$$R_5 = \frac{3\,\text{V}}{2.5\,\text{mA}} = 1.2\,\text{k}\Omega.$$

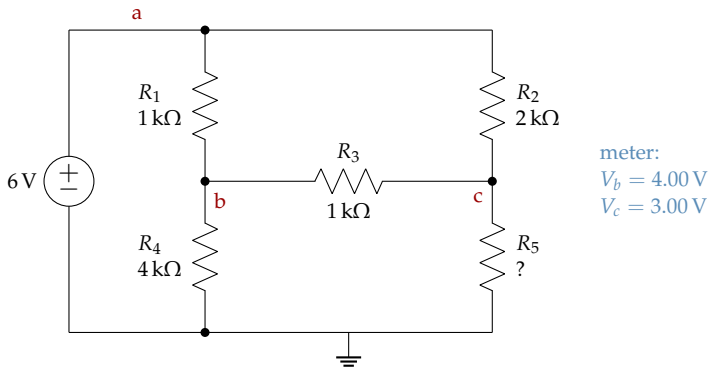


Figure 1.11: A network that cannot be reduced by series and parallel rules: R_3 bridges the two dividers. A voltmeter (black probe on ground) reads the two node voltages shown; the value of R_5 is unknown.

The source supplies $I_1 + I_2 = 3.5\,\text{mA}$ (KCL at a) and delivers $(6\,\text{V})(3.5\,\text{mA}) = 21\,\text{mW}$. As a check, the resistors absorb $4 + 4.5 + 1 + 4 + 7.5 = 21\,\text{mW}$ (V^2/R for each). Measuring node voltages and applying KCL and Ohm's law is exactly how you will debug circuits on the bench. ■

1.4 Series, Parallel, and Divider Circuits

Two patterns appear so often that they deserve their own rules. Elements are in *series* when they form a chain with nothing else attached at the joints between them; KCL then forces the same current through each. Elements are in *parallel* when they connect between the same two nodes; by definition they then share the same voltage.

Series resistors. Let a current I flow through $R_1, R_2, \ldots, R_n$ in series. Each resistor drops IR_k, and by KVL the total voltage is the sum, $V = I(R_1 + R_2 + \cdots + R_n)$. The chain therefore behaves like a single resistor

$$R_{\text{series}} = R_1 + R_2 + \cdots + R_n. \tag{1.8}$$

The voltage divider. Two resistors in series across a source V_s carry $I = V_s/(R_1 + R_2)$, so the voltage across R_2 is

$$V_2 = IR_2 = V_s \frac{R_2}{R_1 + R_2}, \qquad V_1 = V_s \frac{R_1}{R_1 + R_2}. \tag{1.9}$$

The source voltage splits in proportion to the resistances: the larger resistor takes the larger share. The divider is the single most common circuit in this book. It scales a 12 V signal down to an ADC's 3.3 V range, it reads a thermistor or a light-dependent resistor (Chapter 3), and the demo board's touch panel is nothing more than a cleverly arranged set of dividers. The circuits block even has a socket header for building one, which you use in Lab 1.2.

KEY EQUATION

$V_2 = V_s \dfrac{R_2}{R_1 + R_2}$

Voltage divider: the voltage across R_2 in a series pair.

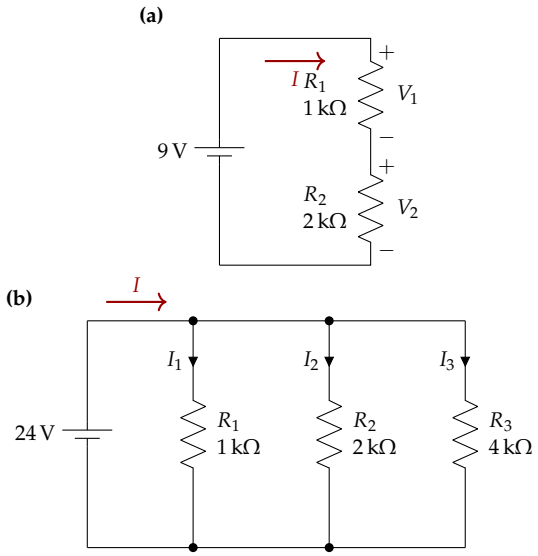


Figure 1.12: (a) Series: one current, $I = 9\,\text{V}/3\,\text{k}\Omega = 3\,\text{mA}$, divides the voltage into $V_1 = 3\,\text{V}$ and $V_2 = 6\,\text{V}$. (b) Parallel: one voltage, 24 V, drives $I_1 = 24\,\text{mA}$, $I_2 = 12\,\text{mA}$, and $I_3 = 6\,\text{mA}$, a total of 42 mA.

In Figure 1.12a, $R_1 + R_2 = 3\,\text{k}\Omega$ and $I = 3\,\text{mA}$. The divider rule gives $V_1 = 9 \times \frac{1}{3} = 3\,\text{V}$ and $V_2 = 9 \times \frac{2}{3} = 6\,\text{V}$, which add back to 9 V as KVL demands. The resistors dissipate $I^2R_1 = 9\,\text{mW}$ and $I^2R_2 = 18\,\text{mW}$; together that is exactly the $VI = (9\,\text{V})(3\,\text{mA}) = 27\,\text{mW}$ the battery supplies.

Parallel resistors. Now let $R_1, \ldots, R_n$ share a voltage V. Each carries V/R_k, and KCL adds the branch currents: $I = V(1/R_1 + 1/R_2 + \cdots + 1/R_n)$. The combination acts as one resistor with

$$\frac{1}{R_{\text{par}}} = \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}, \qquad G_{\text{par}} = G_1 + \cdots + G_n. \tag{1.10}$$

In series the resistances add; in parallel the conductances add. For just two resistors, Equation (1.10) becomes the "product over sum" rule $R_1 \parallel R_2 = R_1R_2/(R_1 + R_2)$. Adding a branch in parallel always gives the current another path, so the equivalent resistance is always smaller than the smallest branch. Two equal resistors in parallel give half of one.

In Figure 1.12b the three conductances are 1, 0.5, and 0.25 mS, which add to 1.75 mS, so $R_{\text{par}} = 571\,\Omega$ and the battery supplies $24/571 = 42\,\text{mA}$. The $1\,\text{k}\Omega$ resistor takes the most current and dissipates the most power, $(24\,\text{V})^2/1\,\text{k}\Omega = 0.58\,\text{W}$, versus 0.29 W and 0.14 W for the other two.

The current divider. When a current I_{in} enters two parallel resistors (Figure 1.13), the shared voltage is $V = I_{\text{in}}R_1R_2/(R_1 + R_2)$, and the

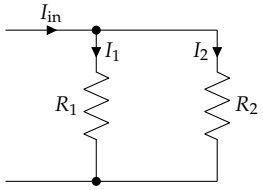


Figure 1.13: A current divider: I_{in} splits between two parallel branches, and the smaller resistor takes the larger share.

branch currents are

$$I_1 = I_{\text{in}} \frac{R_2}{R_1 + R_2}, \qquad I_2 = I_{\text{in}} \frac{R_1}{R_1 + R_2}. \tag{1.11}$$

Watch the subscripts: each branch current has the *other* resistor on top. For example, 16 mA entering 1 kΩ in parallel with 3 kΩ splits into $16 \times \frac{3}{4} = 12$ mA through the 1 kΩ resistor and 4 mA through the 3 kΩ resistor.

Reducing a network. Many circuits can be solved by collapsing series and parallel groups into single resistors until one resistor remains, finding the source current, and then expanding back out with Ohm's law and the divider rules. The examples below follow that pattern. When a network cannot be reduced to one resistor by the series and parallel rules (Figure 1.11 and the touch panel grid are two), you need the general method of Section 1.6.

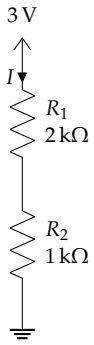


Figure 1.14: A two-resistor divider from a 3 V rail.

Worked Example 1.10 *A divider from a supply rail*

In Figure 1.14, $R_1 = 2\,\text{k}\Omega$ and $R_2 = 1\,\text{k}\Omega$ hang from a 3 V rail to ground. Find V_{R1}, I_{R1}, V_{R2}, and I_{R2}.

Solution. The rail symbol means "this node is held at 3 V relative to ground by a source that is not drawn." The two resistors are in series, so one current flows through both:

$$I = \frac{3\,\text{V}}{2\,\text{k}\Omega + 1\,\text{k}\Omega} = 1\,\text{mA} = I_{R1} = I_{R2}.$$

Ohm's law then gives $V_{R1} = (1\,\text{mA})(2\,\text{k}\Omega) = 2\,\text{V}$ and $V_{R2} = (1\,\text{mA})(1\,\text{k}\Omega) = 1\,\text{V}$. Check with the divider rule, $V_{R2} = 3 \times \frac{1}{3} = 1\,\text{V}$, and with KVL, $2 + 1 = 3$. The node between the resistors sits at 1 V. ■

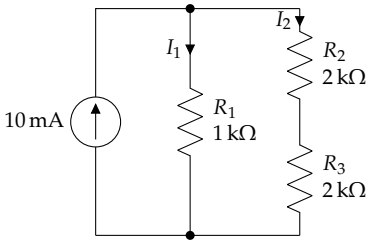


Figure 1.15: A current source feeding two parallel branches.

Worked Example 1.11 *A current source and a current divider*

The 10 mA current source of Figure 1.15 feeds $R_1 = 1\,\text{k}\Omega$ in parallel with the series pair $R_2 = R_3 = 2\,\text{k}\Omega$. Find the branch currents, the voltage across the current source, and the voltage across R_3.

Solution. The right branch is 4 kΩ in total. The current divider rule, with the other branch on top, gives

$$I_1 = 10 \times \frac{4}{1+4} = 8\,\text{mA}, \qquad I_2 = 10 \times \frac{1}{1+4} = 2\,\text{mA}.$$

They add to 10 mA, as KCL requires. The voltage across the parallel combination, and so across the current source, is $V = I_1 R_1 = (8\,\text{mA})(1\,\text{k}\Omega) = 8\,\text{V}$; check: $(10\,\text{mA})(1 \parallel 4\ \text{k}\Omega) = (10)(0.8) = 8\,\text{V}$.

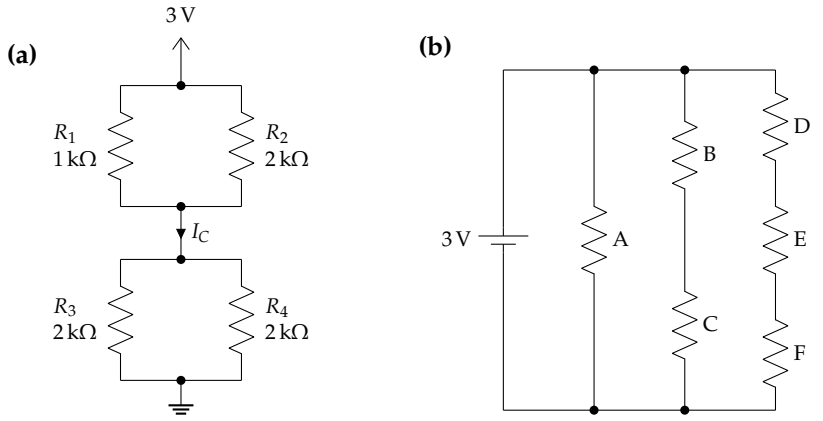


Figure 1.16: (a) Two parallel pairs in series, joined by the wire that carries I_C. (b) Six 1 kΩ resistors in three parallel branches across a 3 V battery.

The two equal resistors on the right split that 8 V evenly, so R_3 has 4 V. Notice that nobody told you the source voltage: a current source takes whatever voltage the circuit demands. ■

Worked Example 1.12 *Parallel pairs in series*

In Figure 1.16a, $R_1 = 1\,\text{k}\Omega$ and $R_2 = 2\,\text{k}\Omega$ are in parallel, as are $R_3 = R_4 = 2\,\text{k}\Omega$, and the two pairs are in series from a 3 V rail to ground. Find the voltage across and current through each resistor, and the current I_C in the connecting wire.

Solution. Collapse each pair with the product-over-sum rule:

$$R_{12} = \frac{(1)(2)}{1+2} = 0.667\,\text{k}\Omega, \qquad R_{34} = \frac{(2)(2)}{2+2} = 1\,\text{k}\Omega.$$

The pairs are in series, $R_{\text{tot}} = 1.667\,\text{k}\Omega$, so the rail supplies $I = 3/1.667 = 1.8\,\text{mA}$. Every bit of that current must pass through the connecting wire, so this is also I_C. The divider rule (or Ohm's law on each pair) gives the two voltages:

$$V_{R1} = V_{R2} = (1.8\,\text{mA})(0.667\,\text{k}\Omega) = 1.2\,\text{V},$$
$$V_{R3} = V_{R4} = (1.8\,\text{mA})(1\,\text{k}\Omega) = 1.8\,\text{V}.$$

They add to 3 V. Now Ohm's law on each resistor: $I_{R1} = 1.2/1 = 1.2\,\text{mA}$, $I_{R2} = 1.2/2 = 0.6\,\text{mA}$, and $I_{R3} = I_{R4} = 1.8/2 = 0.9\,\text{mA}$. KCL checks both junctions: $1.2 + 0.6 = 1.8$ and $0.9 + 0.9 = 1.8$. ■

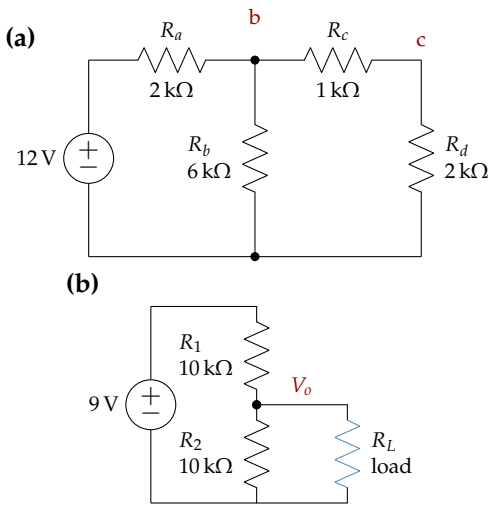


Figure 1.17: (a) A ladder network, reduced from the far end. (b) A divider whose output drives a load R_L.

Worked Example 1.13 *Three branches*

Six 1 kΩ resistors are arranged in three branches across a 3 V battery (Figure 1.16b): A alone; B and C in series; D, E, and F in series. Find the voltage across and the current through each resistor.

Solution. Each branch connects directly across the battery, so each branch has the full 3 V across it, regardless of what the other branches contain. Treat them one at a time.

Branch A: $V_A = 3\,\text{V}$, $I_A = 3/1 = 3\,\text{mA}$.

Branch B-C: two equal resistors in series split the voltage evenly, $V_B = V_C = 1.5\,\text{V}$, and $I_B = I_C = 3/2 = 1.5\,\text{mA}$.

Branch D-E-F: three equal resistors, $V_D = V_E = V_F = 1\,\text{V}$ and $I_D = I_E = I_F = 3/3 = 1\,\text{mA}$.

The battery supplies $3 + 1.5 + 1 = 5.5\,\text{mA}$, so the whole network looks like $3/5.5 = 545\,\Omega$, which is less than the smallest branch (1 kΩ), as it must be for parallel branches. ■

Worked Example 1.14 *Reducing a ladder*

Find the source current, the voltages at nodes b and c, and the current in every resistor of the ladder in Figure 1.17a.

Solution. Start at the end farthest from the source and work back. R_c and R_d are in series: $1 + 2 = 3\,\text{k}\Omega$. That branch is in parallel with R_b: $6 \parallel 3 = 18/9 = 2\,\text{k}\Omega$. Add R_a in series: the source sees $R_{\text{eq}} = 2 + 2 = 4\,\text{k}\Omega$, and supplies $I = 12/4 = 3\,\text{mA}$.

Now expand back out. R_a drops $(3)(2) = 6\,\text{V}$, so $V_b = 12 - 6 = 6\,\text{V}$. At b the current splits: R_b takes $6/6 = 1\,\text{mA}$ and the right

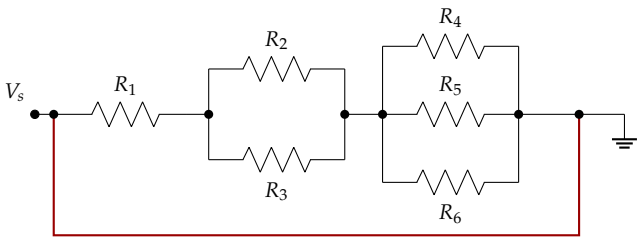


Figure 1.18: The circuit as drawn. The terminal V_s feeds R_1, then $R_2 \parallel R_3$, then $R_4 \parallel R_5 \parallel R_6$ to ground. The bottom wire (cardinal) also connects the V_s terminal straight to the grounded end.

branch takes $6/3 = 2\,\text{mA}$ (the two add to 3 mA). The divider on the right gives $V_c = 6 \times \frac{2}{3} = 4\,\text{V}$, and R_d carries $4/2 = 2\,\text{mA}$. Power check: the source delivers $(12)(3) = 36\,\text{mW}$, and the resistors absorb I^2R: $18 + 6 + 4 + 8 = 36\,\text{mW}$. ■

Worked Example 1.15 *Read the wires first*

Find the voltages across R_1 through R_6 in the network of Figure 1.18.

Solution. Start by finding nodes, as in Figure 1.6. The bottom wire runs from the V_s terminal to the right-hand end of the chain, which is grounded. That wire puts the V_s terminal and ground in the *same node*. The whole chain is shorted: both of its ends are at the same voltage, so by KVL the voltages of its three sections add to zero, and since no resistor in the chain can have a current without a voltage, no current flows anywhere in it. As drawn,

$$V_{R1} = V_{R2} = V_{R3} = V_{R4} = V_{R5} = V_{R6} = 0\,\text{V}.$$

If an ideal source really held that terminal at $V_s \neq 0$, the short would demand infinite current from it; a real supply would hit its current limit or blow its fuse (Section 1.5). No resistor values are given, which is a hint that the answer does not depend on them.

Now suppose the bottom wire were instead the return of a source, so that V_s appears across the chain. Then the chain is three groups in series, $R_{\text{eq}} = R_1 + (R_2 \parallel R_3) + (R_4 \parallel R_5 \parallel R_6)$, the current is $I = V_s/R_{\text{eq}}$, and the divider rule gives

$$V_{R1} = V_s \frac{R_1}{R_{\text{eq}}}, \qquad V_{R2} = V_{R3} = V_s \frac{R_2 \parallel R_3}{R_{\text{eq}}},$$
$$V_{R4} = V_{R5} = V_{R6} = V_s \frac{R_4 \parallel R_5 \parallel R_6}{R_{\text{eq}}}.$$

With representative values $V_s = 9\,\text{V}$, $R_1 = 1\,\text{k}\Omega$, $R_2 = 3\,\text{k}\Omega$, $R_3 = 6\,\text{k}\Omega$, $R_4 = 3\,\text{k}\Omega$, and $R_5 = R_6 = 6\,\text{k}\Omega$: $R_2 \parallel R_3 = 2\,\text{k}\Omega$, $R_4 \parallel R_5 \parallel R_6 = 1.5\,\text{k}\Omega$, $R_{\text{eq}} = 4.5\,\text{k}\Omega$, $I = 2\,\text{mA}$, and the voltages are 2 V, 4 V, and 3 V, adding to 9 V. ■

Worked Example 1.16 *A divider under load*

The divider of Figure 1.17b uses two 10 kΩ resistors on a 9 V source. Find the output V_o with no load, with a 10 kΩ load, and with a 100 kΩ load.

Solution. With nothing connected, Equation (1.9) gives $V_o = 9 \times \frac{10}{20} = 4.5\,\text{V}$. A load sits in parallel with R_2, so the bottom of the divider becomes $R_2 \parallel R_L$. With $R_L = 10\,\text{k}\Omega$, $R_2 \parallel R_L = 5\,\text{k}\Omega$ and

$$V_o = 9 \times \frac{5}{10+5} = 3.00\,\text{V},$$

a third lower than the unloaded value. With $R_L = 100\,\text{k}\Omega$, $R_2 \parallel R_L = 9.09\,\text{k}\Omega$ and $V_o = 9 \times 9.09/19.09 = 4.29\,\text{V}$, about 5% low. The divider rule holds only when the output current is negligible, which needs a load much larger than the divider resistors. Section 1.8 turns this into a one-line rule. ■

Worked Example 1.17 *The bridge*

Derive $V_x - V_y$ for the circuit of Figure 1.19.

Solution. The terminals V_x and V_y are open, so no current leaves the bridge through them. The left side is then a plain divider, R_1 over R_2, and the right side another, R_3 over R_4, both across V_s. By Equation (1.9),

$$V_x = V_s \frac{R_2}{R_1 + R_2}, \qquad V_y = V_s \frac{R_4}{R_3 + R_4},$$

so

$$V_x - V_y = V_s \left(\frac{R_2}{R_1 + R_2} - \frac{R_4}{R_3 + R_4} \right) = V_s \, \frac{R_2 R_3 - R_1 R_4}{(R_1 + R_2)(R_3 + R_4)}.$$

The output is zero, and the bridge is *balanced*, when $R_2 R_3 = R_1 R_4$, that is, when $R_1/R_2 = R_3/R_4$. For example, with $V_s = 5\,\text{V}$, $R_1 = R_4 = 1\,\text{k}\Omega$, and $R_2 = R_3 = 2\,\text{k}\Omega$, the output is $5\,(\frac{2}{3} - \frac{1}{3}) = 1.67\,\text{V}$; with all four equal it is 0 V. Because the bridge subtracts away the large common voltage of the two dividers, it can reveal a tiny change in one resistor, which is how strain gauges and many other resistive sensors are read (Chapter 3). ■

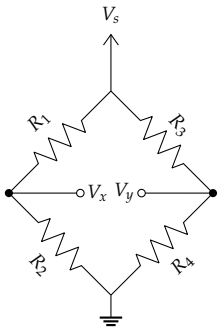


Figure 1.19: A bridge: two dividers side by side. The output is the difference $V_x - V_y$.

The Wheatstone bridge. Turn the balance condition around and the bridge becomes an instrument. Put the unknown resistor in the R_4 position, two fixed "ratio" resistors in R_1 and R_2, and an adjustable, calibrated resistor (a decade box) in R_3. Adjust R_3 until a meter across

the outputs reads zero. Then

$$R_4 = R_3 \frac{R_2}{R_1}. \tag{1.12}$$

This arrangement is the *Wheatstone bridge*. At balance no current flows between the outputs, so neither the meter's resistance nor the exact supply voltage affects the answer; only the three known resistors do.

Worked Example 1.18 *Measuring a resistor with a Wheatstone bridge*

In the bridge of Figure 1.19, $R_1 = 1.00\,\text{k}\Omega$, $R_2 = 10.0\,\text{k}\Omega$, and $V_s = 5\,\text{V}$. The unknown is R_4. The bridge balances with the decade box R_3 set to $472\,\Omega$. (a) Find R_4. (b) If R_4 then increases by 1%, what is $V_x - V_y$?

Solution. (a) By Equation (1.12), $R_4 = 472\,\Omega \times 10.0/1.00 = 4.72\,\text{k}\Omega$. The ratio arms let a decade box that only reaches a few kilohms measure a resistor ten times larger.

(b) Now $R_4 = 4.7672\,\text{k}\Omega$. The left divider is unchanged, and the right one shifts slightly:

$$V_x = 5 \times \frac{10}{11} = 4.5455\,\text{V}, \qquad V_y = 5 \times \frac{4.7672}{0.472 + 4.7672} = 4.5495\,\text{V}.$$

So $V_x - V_y = -4.1\,\text{mV}$. A 1% change in a resistor shows up as a few millivolts on a background of zero, which a difference amplifier can then enlarge. Chapter 2 introduces the op-amp that does this, and Chapter 3 reads sensor bridges that way. ■

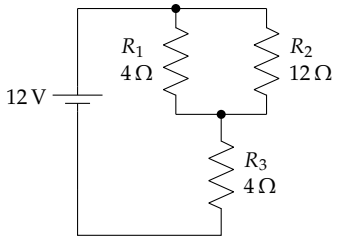


Figure 1.20: A power bookkeeping exercise: $R_1 \parallel R_2$ in series with R_3.

Worked Example 1.19 *Where does the power go?*

A 12 V battery drives $R_1 = 4\,\Omega$ in parallel with $R_2 = 12\,\Omega$, in series with $R_3 = 4\,\Omega$ (Figure 1.20). Find the power delivered by the battery and the power in each resistor.

Solution. $R_1 \parallel R_2 = 48/16 = 3\,\Omega$, so $R_{\text{eq}} = 3 + 4 = 7\,\Omega$ and $I = 12/7 = 1.714\,\text{A}$. The battery delivers $P = VI = 12 \times 12/7 = 20.57\,\text{W}$.

The full current flows through R_3: $P_3 = I^2 R_3 = (12/7)^2 \times 4 = 11.76\,\text{W}$, and $V_3 = 48/7 = 6.86\,\text{V}$. The parallel pair has the rest, $V_{12} = 12 - 6.86 = 5.14\,\text{V}$, so $P_1 = V_{12}^2/R_1 = 6.61\,\text{W}$ and $P_2 = V_{12}^2/R_2 = 2.20\,\text{W}$.

Check: $11.76 + 6.61 + 2.20 = 20.57\,\text{W}$. Power is conserved, which is a consequence of KCL and KVL together and a free test of any solution. ■

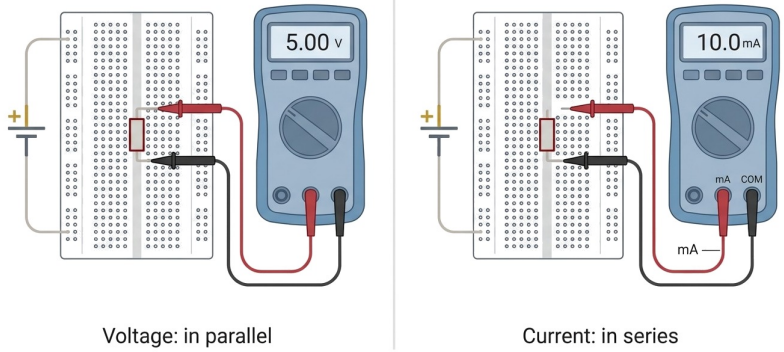


Figure 1.21: Using a multimeter on a breadboard. To read voltage, touch the probes across the resistor (in parallel). To read current, open the circuit and let the current pass through the meter (in series), with the red lead moved to the current jack.

1.5 *Measuring Circuits: Meters, Loading, and Fuses*

A METER IS PART OF THE CIRCUIT the moment you connect it. The digital multimeter on your bench measures voltage, current, and resistance, and each function has its own correct way to connect and its own way to disturb what it measures (Figures 1.21 and 1.22).

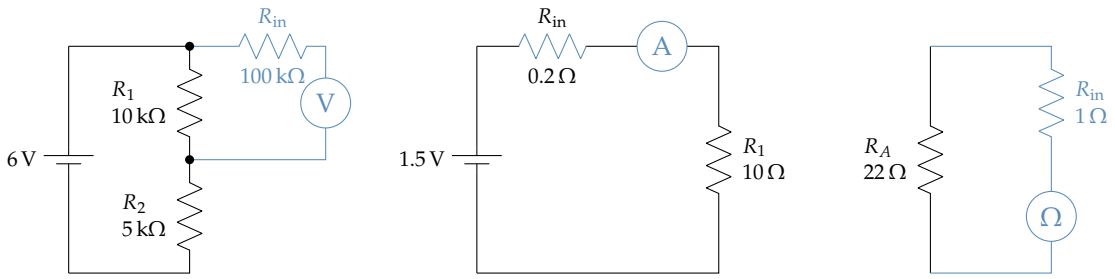
Voltmeter: in parallel. Voltage is a difference between two nodes, so you measure it by touching the two probes to those two nodes. Nothing in the circuit has to be disconnected. An ideal voltmeter draws no current, so it has infinite internal resistance R_{in}. A real one has a large but finite R_{in}, typically $10\,\text{M}\Omega$ for a bench multimeter.

Ammeter: in series. Current is a flow *through* a wire, so to measure it you must break the circuit and route the current through the meter. An ideal ammeter has zero resistance, so it adds no voltage drop. A real one has a small R_{in}, a fraction of an ohm, in its internal current-sensing shunt.

Never connect an ammeter in parallel with an element, or across a supply. Its near-zero resistance shorts whatever it touches, and the large current that results blows the meter's internal fuse (or worse).

Ohmmeter: on an isolated element. An ohmmeter pushes a small known current through the element and measures the resulting voltage. It must be the only source driving the element, so remove the resistor from the circuit (or at least power the circuit off and lift one end). Otherwise you measure the resistor in parallel with everything else connected to it. The resistance of the probe leads, a fraction of an ohm, adds to the reading, which matters only for small resistors.

Loading. Because a real meter has finite (or nonzero) resistance, connecting it changes the circuit. This disturbance is called loading. The three cases of Figure 1.22 are worked out in the next example. The rule



measuring voltage
true 4.00 V, reads 3.87 V

measuring current
true 150 mA, reads 147 mA

measuring resistance
true 22 Ω, reads 23 Ω

Figure 1.22: How each meter connects, and how its internal resistance (blue) causes an error. Left: a voltmeter with $R_{in} = 100\,k\Omega$ loads the $10\,k\Omega$ resistor it measures. Center: an ammeter adds $0.2\,\Omega$ in series with the $10\,\Omega$ load. Right: $1\,\Omega$ of lead and contact resistance adds to a $22\,\Omega$ resistor.

of thumb that falls out is simple: a voltmeter needs R_{in} much larger than the resistance of the circuit it touches, and an ammeter needs R_{in} much smaller than the resistance in series with it.

Worked Example 1.20 *How much does the meter disturb the circuit?*

Find the true and measured values for the three cases of Figure 1.22.

Solution. *Voltmeter.* Without the meter, R_1 gets $6 \times \frac{10}{10+5} = 4.00\,V$ by the divider rule. With the meter connected, R_1 is in parallel with $100\,k\Omega$: $R_1 \parallel R_{in} = (10)(100)/110 = 9.09\,k\Omega$. The divider now gives $6 \times \frac{9.09}{9.09+5} = 3.87\,V$, an error of -3.2%. With a $10\,M\Omega$ bench meter the reading is $3.999\,V$, and the error is negligible (Figure 1.23).

Ammeter. Without the meter, $I = 1.5/10 = 150\,mA$. With it, $I = 1.5/(10 + 0.2) = 147\,mA$, an error of -2.0%.

Ohmmeter. The meter sees $22 + 1 = 23\,\Omega$, an error of $+4.5\%$. Good meters let you short the probes together and subtract the lead resistance ("relative" or "zero" mode). ■

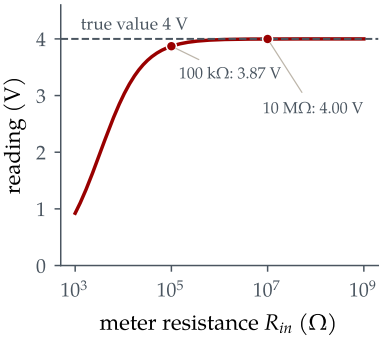


Figure 1.23: Voltmeter reading across the $10\,k\Omega$ resistor of Figure 1.22 versus the meter's input resistance. The error fades once R_{in} is a few hundred times the circuit's $3.33\,k\Omega$ Thevenin resistance (Section 1.8).

Measuring current with a voltmeter. A meter's current range is really a small, accurate *shunt* resistor followed by a voltmeter: the current is the shunt voltage divided by the shunt resistance. You can build the same thing into a circuit, and many boards do. The devices block of the demo board (Chapter 2) measures the diode and transistor currents this way, with a $1\,k\Omega$ sense resistor, so that 1 V across it means 1 mA.

Worked Example 1.21 *A shunt resistor as an ammeter*

In Figure 1.24a, a $0.1\,\Omega$ shunt is placed in series with a $20\,\Omega$ load on a 5 V supply, and a voltmeter reads the shunt voltage. Find the reading, the current it implies, the error the shunt causes, and the

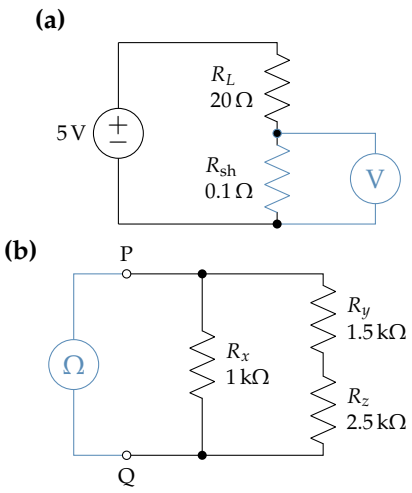


Figure 1.24: (a) A 0.1 Ω shunt and a voltmeter used as an ammeter. (b) An ohmmeter on R_x while R_x is still wired into its circuit (power off).

power in the shunt.

SOLUTION. The shunt is in series with the load, so $I = 5/(20 + 0.1) = 248.8\,\text{mA}$. The voltmeter reads $IR_{sh} = (0.2488\,\text{A})(0.1\,\Omega) = 24.88\,\text{mV}$, and dividing by $0.1\,\Omega$ recovers 248.8 mA exactly; the measurement of the *present* current is correct. The shunt did change that current, though: without it, $I = 5/20 = 250\,\text{mA}$, so the error is -0.50%. The shunt dissipates $I^2R = (0.2488)^2(0.1) = 6.2\,\text{mW}$. A smaller shunt disturbs the circuit less but gives a smaller voltage to read, which is the trade-off every current measurement makes. ■

WORKED EXAMPLE 1.22 *Measuring a resistor that is still in the circuit*

In Figure 1.24b, a student measures $R_x = 1\,\text{k}\Omega$ with an ohmmeter between P and Q, but R_y and R_z are still connected across the same two nodes. What does the meter read, and how do you fix it?

SOLUTION. The ohmmeter drives its test current into node P and out of node Q, and every path between those nodes carries part of it. The meter therefore reads $R_x \parallel (R_y + R_z) = 1 \parallel 4 = 0.8\,\text{k}\Omega = 800\,\Omega$, a -20% error. Lift one end of R_x (or remove it) so that it is the only path, and the meter reads 1 kΩ. If the circuit were also powered, the circuit's own currents would add to the meter's test current and the reading would mean nothing; always measure resistance with the power off. ■

Fuses. A short circuit is the most dangerous mistake on a bench. With a wire across the load, the only thing limiting the current is the small internal resistance of the supply and the wiring, so the current can be enormous, and the wires and the source heat up quickly ($P = I^2R$).

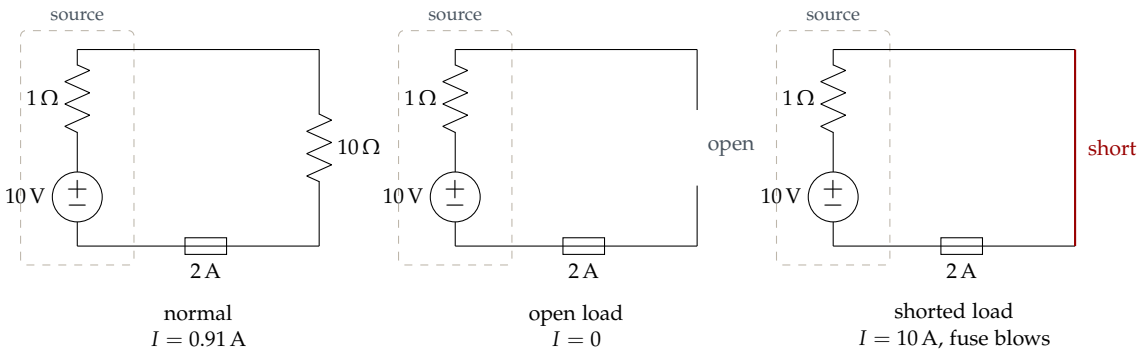


Figure 1.25: A 10 V source with 1 Ω internal resistance, protected by a 2 A fuse. Normal load: 0.91 A, well under the rating. Open load: no current. Shorted load: only the internal 1 Ω remains, the current tries to reach 10 A, and the fuse opens.

A *fuse* is a deliberately weak link: a thin metal strip in series with the circuit that melts when its current exceeds a rating, turning itself into an open circuit before anything else is damaged (Figure 1.25). A fuse is always in series with what it protects, and it is rated in amperes. Replace a blown fuse with one of the same rating, never a larger one, and find the short first.

Worked Example 1.23 *When does the fuse blow?*

A 12 V supply feeds three 10 Ω heaters in parallel through a 5 A fuse. Find the supply current (a) normally, (b) when one heater burns out open, and (c) when one heater shorts.

Solution. (a) Three 10 Ω resistors in parallel make $10/3 = 3.33\,\Omega$, so the current is $12/3.33 = 3.6$ A. The fuse holds.

(b) An open heater simply disappears from the circuit. Two in parallel make 5 Ω and draw 2.4 A. The fuse holds and the other heaters keep working, which is one reason household loads are wired in parallel.

(c) A shorted heater puts 0 Ω across the supply, and the parallel combination becomes 0 Ω. The current is limited only by the wiring and the supply's internal resistance and quickly exceeds 5 A. The fuse blows, opening the circuit and protecting the supply and wires. ■

1.6 Nodal Analysis

Series and parallel reduction runs out of steam quickly. The touch panel grid of Figure 1.32 cannot be reduced to one resistor: apart from the two resistor pairs at the corners S3 and S7, no resistors are in series or in parallel, so there is nothing more to collapse. Nodal analysis is the general method. It uses the node voltages as unknowns, writes KCL at each node, and expresses every current with Ohm's

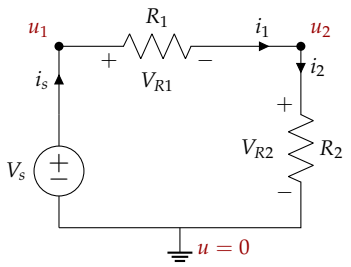


Figure 1.26: The nodal-analysis bookkeeping on a two-resistor circuit: a reference node ($u = 0$, ground), labeled node voltages u_1 and u_2, a current label on every element, and $+/-$ voltage labels that follow the passive sign convention.

law. The result is one linear equation per unknown node, which is exactly the kind of problem computers solve best. It takes seven steps. Figure 1.26 applies them to the simplest possible circuit so you can see each one.

1. *Pick a reference node* and mark it with a ground symbol. Any node works; the negative terminal of the main source is the usual choice. All other voltages are measured from it.
2. *Label every other node* with a voltage $u_1, u_2, \ldots$ (find the nodes with the highlighter method of Section 1.3).
3. *Label a current through every element,* with an arrow. The direction is your choice; a wrong guess just produces a negative number.
4. *Mark $+$ and $-$ on every element* so the current enters the $+$ terminal (passive sign convention). For a resistor, Ohm's law then reads $V_R = iR$ with no sign worries.
5. *Reduce the unknowns.* A voltage source fixes a node voltage outright: in Figure 1.26, $u_1 = V_s$. Elements in series share one current: $i_s = i_1 = i_2$. A current source fixes its branch current.
6. *Write the equations.* (a) KCL at every node whose voltage is still unknown. (b) The I-V law of every element, written with node voltages: the voltage across a resistor from node a to node b is $u_a - u_b$, so its current from a to b is $(u_a - u_b)/R$.
7. *Solve* the linear system, by substitution for small circuits or with a computer for large ones.

For Figure 1.26, steps 5 and 6 leave two equations, $V_s - u_2 = i_1 R_1$ and $u_2 - 0 = i_1 R_2$. Adding them eliminates u_2, giving $i_1 = V_s/(R_1 + R_2)$, and then $u_2 = V_s R_2/(R_1 + R_2)$: the voltage divider again, now derived instead of assumed.

The compact form. In practice you can merge steps 3 through 6. At each unknown node n, add up the currents *leaving* through every resistor, $(u_n - u_m)/R_{nm}$, where m is the node at the far end, and set the

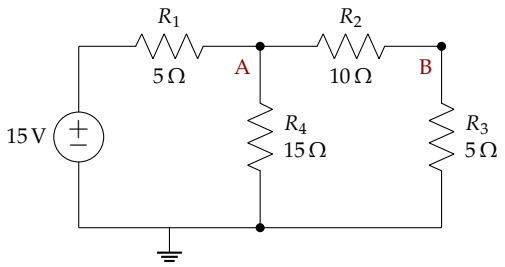


Figure 1.27: A two-node circuit: nodes A and B are unknown, the top of the source is held at 15 V.

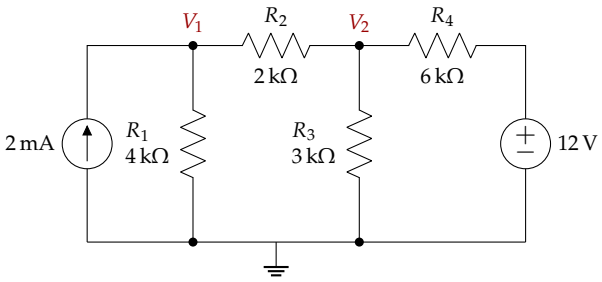


Figure 1.28: Nodal analysis with a current source: 2 mA is pushed into node 1, and a 12 V source feeds node 2 through R_4.

sum equal to the current pushed into the node by current sources (zero if there are none):

$$\sum_m \frac{u_n - u_m}{R_{nm}} = I_{\text{in},n}. \tag{1.13}$$

Every term has the same form, so you never have to think about directions. If the far-end node is held by a voltage source, its voltage is a known number.

Worked Example 1.24 *Two unknown nodes*

Find V_A and V_B in Figure 1.27, and the current in every resistor.

Solution. Ground is the bottom wire and the source fixes the top-left node at 15 V, so only V_A and V_B are unknown. Apply Equation (1.13) at each, with no current sources:

$$\text{node A:}\quad \frac{V_A - 15}{5} + \frac{V_A}{15} + \frac{V_A - V_B}{10} = 0,$$

$$\text{node B:}\quad \frac{V_B - V_A}{10} + \frac{V_B}{5} = 0.$$

The second equation gives $V_A = 3V_B$. Multiply the first by 30: $6V_A - 90 + 2V_A + 3V_A - 3V_B = 0$, so $11V_A - 3V_B = 90$. Substituting, $33V_B - 3V_B = 90$, hence $V_B = 3$ V and $V_A = 9$ V.

Currents: through R_1, $(15-9)/5 = 1.2$ A; through R_4, $9/15 = 0.6$ A; through R_2, $(9-3)/10 = 0.6$ A; through R_3, $3/5 = 0.6$ A. KCL at A: $1.2 = 0.6 + 0.6$. At B: $0.6 = 0.6$. ■

Worked Example 1.25 *Nodal analysis with a current source*

Find V_1 and V_2 in Figure 1.28, the current in each resistor, and the power delivered by each source.

Solution. Work in volts, milliamperes, and kilohms. The current source pushes 2 mA *into* node 1, so that is the right-hand side of Equation (1.13) there. Node 2 has no current source; the 12 V source enters through R_4 as a known far-end voltage:

$$\begin{aligned} \text{node 1:} \quad & \frac{V_1}{4} + \frac{V_1 - V_2}{2} = 2, \\ \text{node 2:} \quad & \frac{V_2 - V_1}{2} + \frac{V_2}{3} + \frac{V_2 - 12}{6} = 0. \end{aligned}$$

Multiply the first by 4 and the second by 6: $3V_1 - 2V_2 = 8$ and $-3V_1 + 6V_2 = 12$. Adding them gives $4V_2 = 20$, so $V_2 = 5\,\text{V}$, and then $V_1 = (8 + 10)/3 = 6\,\text{V}$.

Currents: R_1 carries $6/4 = 1.5\,\text{mA}$ down, R_2 carries $(6 - 5)/2 = 0.5\,\text{mA}$ from node 1 to node 2, R_3 carries $5/3 = 1.67\,\text{mA}$ down, and R_4 carries $(12 - 5)/6 = 1.17\,\text{mA}$ from the source into node 2. KCL at node 2: $0.5 + 1.17 = 1.67$.

Powers: the voltage across the current source is V_1, so it delivers $(6)(2) = 12\,\text{mW}$, and the voltage source delivers $(12)(1.17) = 14\,\text{mW}$. The resistors absorb V^2/R: $9 + 0.5 + 8.33 + 8.17 = 26\,\text{mW}$, which matches. ■

The matrix form $\mathbf{Gv} = \mathbf{i}$. Collect the terms of Equation (1.13) by unknown. For the two-node example of Figure 1.27,

$$\underbrace{\begin{bmatrix} \frac{1}{5} + \frac{1}{15} + \frac{1}{10} & -\frac{1}{10} \\ -\frac{1}{10} & \frac{1}{10} + \frac{1}{5} \end{bmatrix}}_{\mathbf{G}} \underbrace{\begin{bmatrix} V_A \\ V_B \end{bmatrix}}_{\mathbf{v}} = \underbrace{\begin{bmatrix} 15/5 \\ 0 \end{bmatrix}}_{\mathbf{i}}.$$

The pattern is general, and it lets you write $\mathbf{G}$ by inspection, without writing KCL at all:

- the diagonal entry G_{nn} is the sum of the conductances of all resistors touching node n;
- the off-diagonal entry G_{nm} is minus the conductance of the resistor joining nodes n and m (zero if none), so $\mathbf{G}$ is symmetric;
- the entry i_n is the current fed into node n: from current sources, plus V_k/R for every resistor that connects n to a node held at a known voltage V_k.

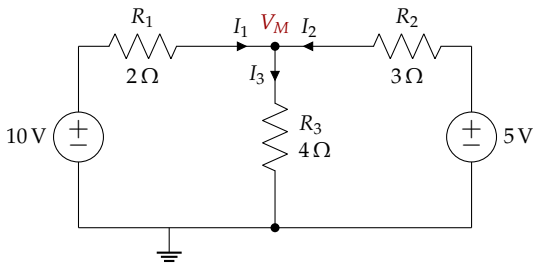


Figure 1.29: Two sources feeding one node through R_1 and R_2.

For the circuit with the current source (Figure 1.28), in mS and mA, the rules give

$$\mathbf{G} = \begin{bmatrix} \frac{1}{4}+\frac{1}{2} & -\frac{1}{2} \\ -\frac{1}{2} & \frac{1}{2}+\frac{1}{3}+\frac{1}{6} \end{bmatrix}, \qquad \mathbf{i} = \begin{bmatrix} 2 \\ 12/6 \end{bmatrix}:$$

the current source and the source-fed resistor both land on the right-hand side. Either way, $\mathbf{v} = \mathbf{G}^{-1}\mathbf{i}$, and in Python that is one line (you used `numpy` arrays in Chapter 0). For the two-node example:

```
import numpy as np
G = np.array([[1/5 + 1/10 + 1/15, -1/10],    # node A
              [-1/10,        1/10 + 1/5]])   # node B
i = np.array([15/5, 0.0])     # current fed into A
print(np.linalg.solve(G, i))  # -> [9. 3.]
```

This is how circuit simulators such as SPICE work inside: they build $\mathbf{G}$ from the netlist, one resistor at a time, and solve. Lab 1.2 has you do the same for the touch panel.[2]

KEY IDEA Nodal analysis turns any resistor network into $\mathbf{Gv} = \mathbf{i}$: conductances on the diagonal, minus the shared conductance off it, source currents on the right.

[2] An alternative, mesh analysis, assigns a current to each loop and writes KVL around it. It gives the same answers. Nodal analysis is usually easier for circuits with many parallel paths, like the touch panel grid, and it is the method computers use.

WORKED EXAMPLE 1.26 *Two sources, one unknown*

Find the node voltage V_M and the three currents in Figure 1.29.

SOLUTION. Both sources have their $-$ terminal on ground, so the left end of R_1 is at 10 V and the right end of R_2 is at 5 V. Only V_M is unknown. KCL at M, with all currents written as leaving:

$$\frac{V_M - 10}{2} + \frac{V_M}{4} + \frac{V_M - 5}{3} = 0.$$

Multiply by 12: $6V_M - 60 + 3V_M + 4V_M - 20 = 0$, so $13V_M = 80$ and $V_M = 6.15\,\text{V}$. Then $I_1 = (10 - 6.15)/2 = 1.92\,\text{A}$ into M, $I_2 = (5 - 6.15)/3 = -0.385\,\text{A}$ into M, and $I_3 = 6.15/4 = 1.54\,\text{A}$ down through R_3. The negative I_2 means the current in R_2 actually flows from M into the $+$ terminal of the 5 V source, which therefore absorbs power: if it were a rechargeable battery, the 10 V source would be charging it. Check KCL: $1.92 - 0.385 = 1.54$. ■

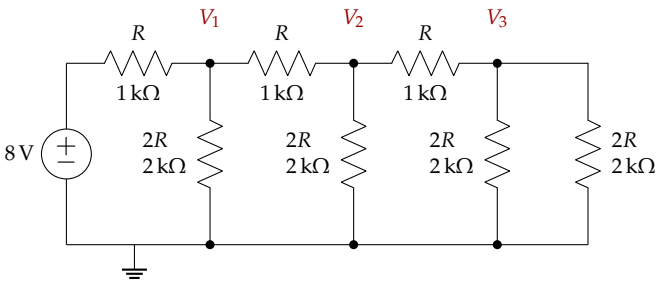


Figure 1.30: An R-2R ladder with $R = 1\,\text{k}\Omega$, the resistor network inside many digital-to-analog converters.

Worked Example 1.27 *Three nodes by inspection, solved in Python*

Write $\mathbf{G}$ and $\mathbf{i}$ for the R-2R ladder of Figure 1.30 by inspection, solve them with `numpy`, and check the answer by series-parallel reduction.

Solution. Use kΩ, so conductances are in mS and currents in mA. Node 1 touches the 1 kΩ from the source, a 2 kΩ to ground, and the 1 kΩ to node 2: $G_{11} = 1 + 0.5 + 1 = 2.5$. Node 2 touches the same three kinds of resistor: $G_{22} = 2.5$. Node 3 touches the 1 kΩ from node 2 and two 2 kΩ resistors to ground: $G_{33} = 1 + 0.5 + 0.5 = 2$. The 1 kΩ resistors between neighbors give $G_{12} = G_{23} = -1$, and nodes 1 and 3 share no resistor, so $G_{13} = 0$. Only node 1 connects to a known voltage, through 1 kΩ: $i_1 = 8/1 = 8\,\text{mA}$.

```
import numpy as np
G = np.array([[ 2.5, -1.0,  0.0],   # node 1 (mS)
              [-1.0,  2.5, -1.0],   # node 2
              [ 0.0, -1.0,  2.0]])  # node 3
i = np.array([8.0, 0.0, 0.0])       # mA fed into each node
print(np.linalg.solve(G, i))        # -> [4. 2. 1.]
```

So $V_1 = 4\,\text{V}$, $V_2 = 2\,\text{V}$, and $V_3 = 1\,\text{V}$. Check by reducing from the right end: the two 2 kΩ resistors at node 3 make 1 kΩ; adding the series 1 kΩ gives 2 kΩ, which in parallel with the 2 kΩ at node 2 is again 1 kΩ; the same happens at node 1. So every node looks into 1 kΩ, each series 1 kΩ resistor forms a one-to-one divider with it, and the voltage halves at every step: $8 \to 4 \to 2 \to 1$. That halving is why the ladder is useful for converting a binary number into a voltage. ■

1.7 *Resistive Touch Sensing*

A resistive touch screen is a voltage divider whose tap you move with your finger. It is built from two transparent plastic or glass sheets, each coated with a thin resistive film (usually ITO), held apart by tiny insulating spacer dots (Figure 1.31). To read the x position,

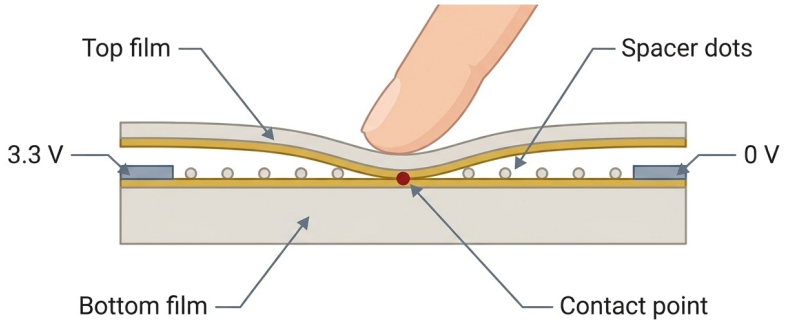


Figure 1.31: Inside a resistive touch screen. Two films coated with a resistive layer are separated by insulating spacer dots. A finger press brings them into contact; the voltage picked up at the contact point is proportional to the position along the driven film.

the controller applies a voltage, say 3.3 V, across the left and right edges of one sheet. Current flows uniformly through that film, so the voltage falls linearly from 3.3 V to 0 V across its width W. A finger press pushes the sheets into contact at one point, and the other sheet, which carries no current of its own, acts as a voltmeter probe: it picks up the voltage of the first sheet at the contact point,

$$V_{\text{touch}} = V_s \, \frac{x}{W}, \tag{1.14}$$

where x is measured from the grounded edge. The film on either side of the contact forms the two resistors of a divider with $R_2/(R_1 + R_2) = x/W$, by Equation (1.4). The controller then swaps the roles of the sheets to read y. An ADC converts each voltage to a number, and two numbers give a position.

The demo board version. The circuits block replaces the continuous film by a discrete 3×3 grid of resistors (Figure 1.32). Corner S1 is tied to the 3.3 V rail, corner S9 to ground, and current spreads through the grid from one to the other. Each of the nine nodes has a push button that connects it to a shared sense line, which jumper J7 routes to the Arduino's analog pin A0. The board carries two versions of the grid. In *Network 2* all twelve resistors are 100 Ω, a symmetric panel whose nodes are brought out to test points TP1 to TP9 for multimeter practice. In *Network 1*, the one wired to the buttons, the six horizontal resistors ($R_3, R_4, R_5, R_8, R_9, R_{10}$) are 50 Ω and the six vertical ones are 100 Ω: the asymmetric touch network. As the examples below show, only the asymmetric one gives every button its own voltage.

Worked Example 1.28 *What does the touchpad do?*

Describe the function of the touchpad used in class. What are its sensing and processing components?

Solution. The touchpad turns the position of a press into a voltage and the voltage into a location. It is a complete sensing system

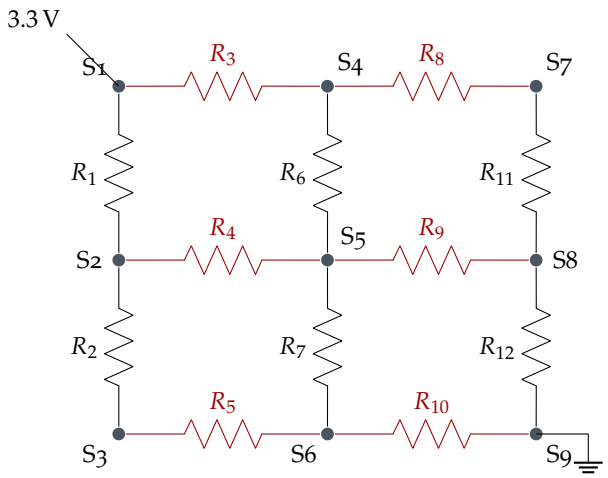


Figure 1.32: The touch panel grid of the circuits block. S1 is held at 3.3 V and S9 is grounded. Vertical resistors (black) are 100 Ω. Horizontal resistors (cardinal) are 50 Ω in Network 1 (asymmetric) and 100 Ω in Network 2 (symmetric). Resistor R_k carries current I_k, taken positive toward ground (downward or rightward).

in the sense of Chapter 0.

Sensor. The resistor grid of Figure 1.32, powered from 3.3 V, sets a different voltage at each of its nine nodes. The push buttons S1 to S9 are the touch-sensitive elements: pressing one connects that node to the shared sense line. (In a commercial screen, the two resistive films of Figure 1.31 play the role of the grid and the finger plays the role of the button.)

Processing. The sense line goes to the analog pin A0 of the Arduino Nano 33 BLE Sense. Its analog-to-digital converter measures the voltage and reports a number between 0 and 1023 (10 bits, 3.3 V reference). Software compares that number with the nine expected levels and reports the nearest button. A laptop running Python can also solve the grid with nodal analysis to predict the levels ahead of time (Lab 1.2). ■

Worked Example 1.29 *The symmetric grid (Network 2)*

All twelve resistors in Figure 1.32 are 100 Ω. Find the voltages S1 to S9 and the currents I_1 to I_{12}.

Solution. There are seven unknown node voltages, but symmetry cuts the work. The grid looks the same if you reflect it across the diagonal through S1, S5, and S9, so mirror-image nodes have equal voltages: $S_2 = S_4$, $S_3 = S_7$, $S_6 = S_8$. It also looks the same if you rotate it by 180° and swap 3.3 V with ground, so voltages pair up to add to 3.3 V: $S_8 = 3.3 - S_2$, $S_7 = 3.3 - S_3$, and $S_5 = 3.3 - S_5$, which gives $S_5 = 1.65\,\text{V}$. Combining the two, $S_3 = S_7 = 3.3 - S_3$, so $S_3 = 1.65\,\text{V}$ as well.

Only S_2 is left. KCL at S2 (neighbors S1, S3, S5, all through $100\,\Omega$):

$$\frac{S_2 - 3.3}{100} + \frac{S_2 - 1.65}{100} + \frac{S_2 - 1.65}{100} = 0,$$

so $3S_2 = 6.6$ and $S_2 = 2.2\,\text{V}$. So $S_1 = 3.3$, $S_2 = S_4 = 2.2$, $S_3 = S_5 = S_7 = 1.65$, $S_6 = S_8 = 1.1$, and $S_9 = 0$ volts. Ohm's law on each resistor gives the currents: $I_1 = I_3 = I_{10} = I_{12} = (1.1\,\text{V})/(100\,\Omega) = 11\,\text{mA}$ along the four edges that touch S1 or S9, and 5.5 mA in the other eight. The rail supplies $I_1 + I_3 = 22\,\text{mA}$, so the grid looks like $3.3/0.022 = 150\,\Omega$ between S1 and S9. ■

Worked Example 1.30 *The asymmetric grid (Network 1)*

Now the horizontal resistors are $50\,\Omega$ and the vertical ones $100\,\Omega$. Find S1 to S9 and I_1 to I_{12}.

Solution. The diagonal mirror symmetry is gone (horizontal and vertical resistors now differ), but the 180° rotation still maps the grid onto itself, so $S_5 = 1.65\,\text{V}$, $S_6 = 3.3 - S_4$, $S_7 = 3.3 - S_3$, and $S_8 = 3.3 - S_2$. Three unknowns remain. Write Equation (1.13) at S2, S3, and S4 and multiply each by $100\,\Omega$:

$$\begin{aligned}
&\text{S2:} \quad (S_2 - 3.3) + (S_2 - S_3) + 2(S_2 - 1.65) = 0,\\
&\text{S3:} \quad (S_3 - S_2) + 2\big(S_3 - (3.3 - S_4)\big) = 0,\\
&\text{S4:} \quad 2(S_4 - 3.3) + (S_4 - 1.65) + 2\big(S_4 - (3.3 - S_3)\big) = 0.
\end{aligned}$$

The S2 equation has three terms because S2 touches S1 and S3 through $100\,\Omega$ and S5 through $50\,\Omega$ (hence the factor 2). Collecting terms,

$$4S_2 - S_3 = 6.6, \qquad -S_2 + 3S_3 + 2S_4 = 6.6, \qquad 2S_3 + 5S_4 = 14.85.$$

Solving (by substitution, or with `numpy`) gives $S_2 = 1.946\,\text{V}$, $S_3 = 1.185\,\text{V}$, and $S_4 = 2.496\,\text{V}$, and therefore

S1	S2	S3	S4	S5	S6	S7	S8	S9
3.300	1.946	1.185	2.496	1.650	0.804	2.115	1.354	0.000

in volts. The currents follow from $(u_a - u_b)/R$ for each resistor, in milliamperes: $I_1 = I_{12} = 13.54$, $I_2 = I_5 = I_8 = I_{11} = 7.62$, $I_3 = I_{10} = 16.08$, $I_4 = I_9 = 5.92$, and $I_6 = I_7 = 8.46$. The rail now supplies $I_1 + I_3 = 29.6\,\text{mA}$ (an equivalent resistance of $111\,\Omega$). Check KCL at S5: in through R_4 and R_6, $5.92 + 8.46 = 14.38$; out through R_7 and R_9, $8.46 + 5.92 = 14.38$. ■

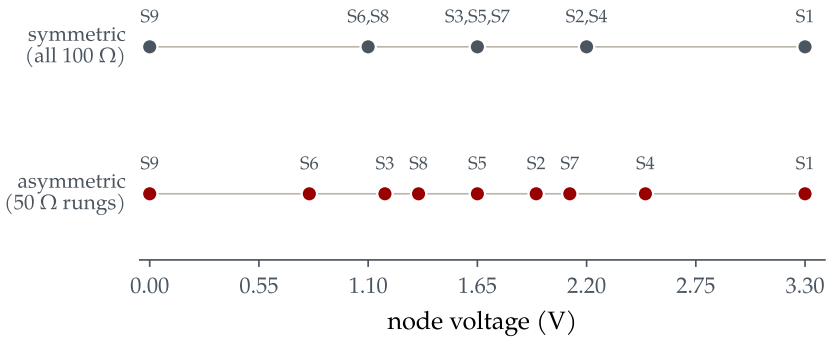


Figure 1.33: The nine button voltages for the symmetric and asymmetric grids. The symmetric grid produces only five distinct levels, so buttons S2 and S4 (and S3, S5, S7, and S6, S8) cannot be told apart. The asymmetric grid separates all nine; its closest pairs, S3 and S8 and also S2 and S7, are 0.169 V apart.

Worked Example 1.31 *Which grid makes a touchpad?*

What is the difference between the two grids, and which should you use for the touchpad?

Solution. Plot the two sets of voltages on one axis (Figure 1.33). In the symmetric grid, every button that is a mirror image of another across the S1-S9 diagonal has the same voltage, and the whole anti-diagonal (S3, S5, S7) sits at 1.65 V. Only five distinct voltages exist for nine buttons, so the Arduino cannot tell S2 from S4 or S3 from S7. Making the horizontal and vertical resistors different breaks that mirror symmetry and spreads the nine voltages apart. Use the asymmetric grid (Network 1).

The smallest gap in Network 1 is between S8 (1.354 V) and S3 (1.185 V), and by the 180° symmetry also between S2 and S7: 0.169 V, about 52 counts of a 10-bit ADC with a 3.3 V reference. A decoder that puts its decision threshold halfway between neighboring levels leaves each of these buttons 85 mV (26 counts) of room, which has to absorb resistor tolerance and ADC noise. Problem 1.29 asks you to find the rung resistance that makes the smallest gap as large as possible. ■

Nodes as weighted averages. Solve the KCL equation of a node with no current source, $\sum_m (u_n - u_m)/R_{nm} = 0$, for u_n:

$$u_n = \frac{\sum_m u_m/R_{nm}}{\sum_m 1/R_{nm}}. \tag{1.15}$$

Every inner node voltage is a conductance-weighted average of its neighbors, so it always lies between the lowest and highest of them, and a neighbor connected through a smaller resistor pulls harder. Lab 1.1 uses this formula to check measured voltages against each other.

WORKED EXAMPLE 1.32 *A node from its neighbors*

Use Equation (1.15) to find S2 and S5 of Network 1 from the voltages of their neighbors in the table above.

SOLUTION. S2 connects to S1 (3.3 V) and S3 (1.185 V) through 100 Ω and to S5 (1.650 V) through 50 Ω:

$$S_2 = \frac{3.3/100 + 1.185/100 + 1.650/50}{1/100 + 1/100 + 1/50} = \frac{0.07785}{0.04} = 1.946\,\text{V}.$$

S5 connects to S2 and S8 through 50 Ω and to S4 and S6 through 100 Ω:

$$S_5 = \frac{(1.946 + 1.354)/50 + (2.496 + 0.804)/100}{2/50 + 2/100} = \frac{0.066 + 0.033}{0.06} = 1.650\,\text{V}.$$

Both match the table. The S2 result sits closer to S5 than a plain average of the three neighbors (2.045 V) would, because the 50 Ω link to S5 counts twice. ■

WORKED EXAMPLE 1.33 *Two buttons at once*

What does A0 read if you press S2 and S8 of Network 1 together?

SOLUTION. Pressing both buttons connects S2 and S8 to the sense line, and so to each other: they become one node. Use the 180° symmetry. Rotating the grid and swapping 3.3 V with ground maps the grid onto itself, sends S2 to S8 and S8 to S2, and turns every voltage V into $3.3 - V$. The joined node maps onto itself, so its voltage V_J satisfies $V_J = 3.3 - V_J$, which gives $V_J = 1.65\,\text{V}$. A nodal solve with the two nodes merged (six unknowns) gives the same 1.650 V. A decoder reports S5, a button nobody pressed. A resistive panel cannot report two touches at once, which is one reason phones use capacitive screens instead. ■

From a count to a button. The Arduino's ADC turns the A0 voltage into an integer N from 0 to 1023. Chapter 0 converts it with

$$V = N\,\frac{3.3\,\text{V}}{1023}, \tag{1.16}$$

so one count is about 3.2 mV. (The lab notebooks divide by 1024 instead; the two conventions differ by less than one count.) To name the button, sort the nine levels, put a threshold halfway between each pair of neighbors (Figure 1.34), and report the level whose range contains the reading. That is the same as picking the nearest level.

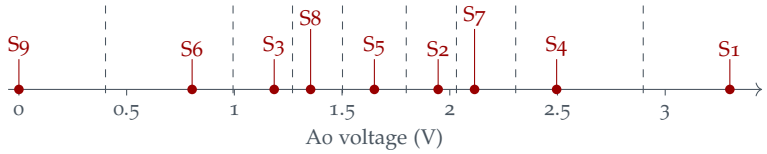


Figure 1.34: The nine Network 1 levels on the Ao voltage axis, with decision thresholds (dashed) halfway between neighbors: 0.40, 0.99, 1.27, 1.50, 1.80, 2.03, 2.31, and 2.90 V. Each button owns the voltages between its two thresholds. In Lab 1.1 you build the same picture from your measured levels.

Worked Example 1.34 *From an ADC reading to a button*

The ADC reports (a) 430 counts and (b) 600 counts. Which buttons are pressed, and how far is each reading from its expected level?

Solution. (a) By Equation (1.16), $V = 430 \times 3.3/1023 = 1.387\,\text{V}$. The thresholds on either side are 1.269 V (between S3 and S8) and 1.502 V (between S8 and S5), so the reading belongs to S8. It sits $1.387 - 1.354 = 33\,\text{mV}$, about 10 counts, above the S8 level, well inside the 85 mV margin.

(b) $V = 600 \times 3.3/1023 = 1.935\,\text{V}$, between the thresholds 1.798 V and 2.031 V: button S2, 11 mV below its level. ■

1.8 Equivalent Circuits: Thevenin and Norton

From the outside, any network of resistors and sources with two terminals behaves like something much simpler. Thevenin's theorem says that any linear two-terminal network can be replaced by one voltage source V_{th} in series with one resistor R_{th}. Norton's theorem says it can equally be replaced by one current source I_N in parallel with one resistor $R_N = R_{\text{th}}$. "Behaves like" means exactly that: connect any load to the terminals, and it sees the same voltage and current in all three versions. The equivalent is a black box with the same terminal behavior, which is what you want when the only question is what happens to the load.

To find the equivalent of a network seen from terminals A and B:

- V_{th} is the *open-circuit* voltage V_{AB} with nothing connected.
- I_N is the *short-circuit* current that flows through a wire placed from A to B.
- R_{th} is the resistance seen looking back into A and B with every independent source turned off: voltage sources replaced by wires (0 V), current sources by open circuits (0 A). Equivalently, $R_{\text{th}} = V_{\text{th}}/I_N$.

The Thevenin and Norton forms convert into each other with $V_{\text{th}} = I_N R_{\text{th}}$. That conversion is called a source transformation, and it works on any voltage source with a series resistor inside a larger circuit.

KEY EQUATION

$$R_{\text{th}} = \frac{V_{\text{th}}}{I_N}$$

Thevenin resistance from the open-circuit voltage and short-circuit current.

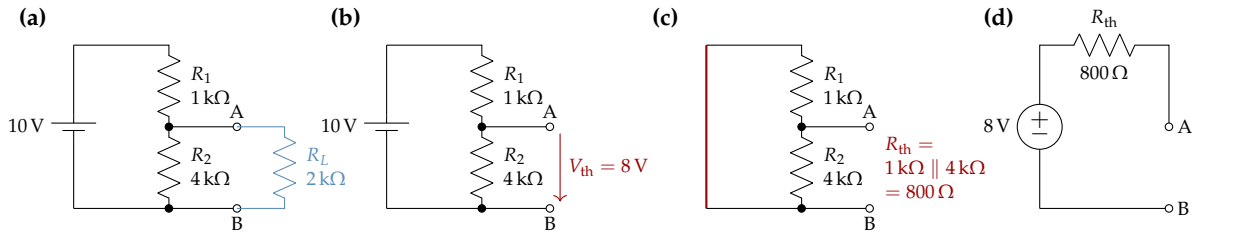


Figure 1.35: Finding a Thevenin equivalent. (a) The real circuit with a load R_L on terminals A-B. (b) Remove the load: the open-circuit voltage is the divider output, 8 V. (c) Replace the battery by a wire: looking into A-B you see $R_1 \parallel R_2 = 800\,\Omega$. (d) The equivalent: 8 V in series with 800 Ω.

Worked Example 1.35 *Thevenin and Norton equivalents*

For the circuit of Figure 1.35a, find the Thevenin and Norton equivalents at A-B, and use them to find the load voltage and current for $R_L = 2\,\text{k}\Omega$.

Solution. *Open-circuit voltage.* With the load removed, R_1 and R_2 form a divider: $V_{\text{th}} = 10 \times \frac{4}{1+4} = 8\,\text{V}$.

Thevenin resistance. Turn the 10 V source off (replace it by a wire). From A-B, R_1 now connects A to B through the wire, so R_1 and R_2 are in parallel: $R_{\text{th}} = (1)(4)/(1+4) = 0.8\,\text{k}\Omega = 800\,\Omega$.

Norton current. Short A to B. The short bypasses R_2, so the battery drives 10 V across R_1 alone: $I_N = 10/1 = 10\,\text{mA}$. Check: $V_{\text{th}}/R_{\text{th}} = 8/0.8 = 10\,\text{mA}$, and $R_N = 800\,\Omega$.

The load. With the equivalent, the load forms a simple divider with R_{th}:

$$V_L = V_{\text{th}}\frac{R_L}{R_{\text{th}} + R_L} = 8 \times \frac{2}{2.8} = 5.71\,\text{V}, \qquad I_L = \frac{V_L}{R_L} = 2.86\,\text{mA}.$$

Solving the original circuit by nodal analysis gives the same answer: $(V_L - 10)/1 + V_L/4 + V_L/2 = 0$ in milliamperes and kilohms, so $1.75V_L = 10$ and $V_L = 5.71\,\text{V}$. The advantage of the equivalent is that trying a different load takes one line instead of a new analysis. ■

Loading, in one line. The equivalent explains every loading effect in this chapter. A load R_L on a source with Thevenin resistance R_{th} receives $V_L = V_{\text{th}}R_L/(R_{\text{th}} + R_L)$, so it sees the full open-circuit voltage only when $R_L \gg R_{\text{th}}$; the fractional error is about R_{th}/R_L when that ratio is small. The divider under load (two 10 kΩ resistors on 9 V) is $V_{\text{th}} = 4.5\,\text{V}$ behind $R_{\text{th}} = 5\,\text{k}\Omega$, so a 10 kΩ load gets $4.5 \times 10/15 = 3\,\text{V}$, as found earlier. Seen from the voltmeter of Figure 1.22, the 6 V, 10 kΩ, 5 kΩ circuit is $V_{\text{th}} = 4\,\text{V}$ behind $R_{\text{th}} = 10\,\text{k}\Omega \parallel 5\,\text{k}\Omega = 3.33\,\text{k}\Omega$, and the meter's R_{in} plays the role of R_L.

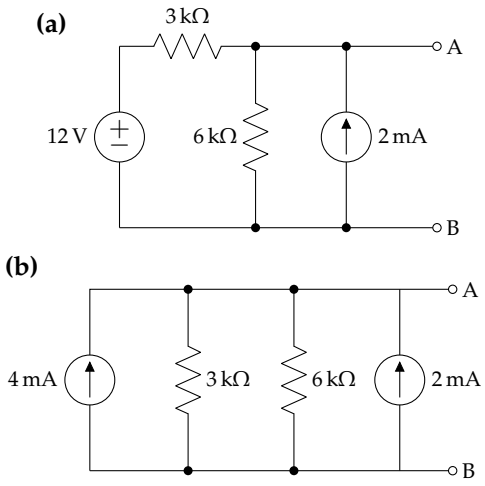


Figure 1.36: (a) A network with two sources. (b) The same network after the 12 V source and its 3 kΩ series resistor are replaced by a 4 mA source in parallel with 3 kΩ.

Worked Example 1.36 *A source transformation*

Find the Thevenin and Norton equivalents of Figure 1.36a at terminals A-B.

Solution. The 12 V source in series with 3 kΩ is a Thevenin form, so replace it by its Norton form: $12/3 = 4$ mA in parallel with 3 kΩ (Figure 1.36b). Now everything sits between the same two nodes. The two current sources both push current up into A, so they add to 6 mA, and the two resistors combine to $3 \parallel 6 = 2$ kΩ. The Norton equivalent is $I_N = 6$ mA in parallel with 2 kΩ, and the Thevenin equivalent is $V_{\text{th}} = (6)(2) = 12$ V in series with $R_{\text{th}} = 2$ kΩ.

Check with nodal analysis on the original circuit, with A open: $(V_A - 12)/3 + V_A/6 = 2$, so $2V_A - 24 + V_A = 12$ and $V_A = 12$ V. Check R_{th} by turning the sources off: the 12 V source becomes a wire and the 2 mA source an open circuit, leaving 3 kΩ in parallel with 6 kΩ, which is 2 kΩ. ■

Worked Example 1.37 *Measuring a Thevenin equivalent*

A sensor module has two output terminals and no schematic. A 10 MΩ voltmeter reads 3.00 V across them. With a 1.00 kΩ resistor connected across the terminals, the reading drops to 2.40 V. Find the Thevenin and Norton equivalents, and predict the output with a 470 Ω load.

Solution. The voltmeter draws almost nothing, so the first reading is the open-circuit voltage: $V_{\text{th}} = 3.00$ V. With the load, the module and the resistor form a divider:

$$2.40 = 3.00 \times \frac{1000}{R_{\text{th}} + 1000} \quad \Longrightarrow \quad R_{\text{th}} + 1000 = \frac{3.00}{2.40} \times 1000 = 1250,$$

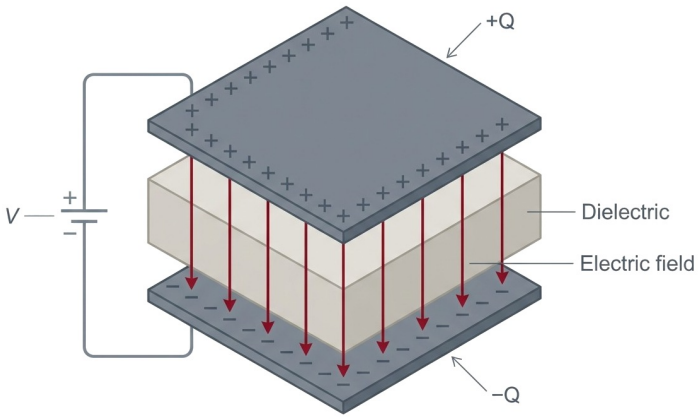


Figure 1.37: A parallel-plate capacitor. A voltage across two conducting plates separated by an insulator (the dielectric) piles up $+Q$ on one plate and $-Q$ on the other, and the electric field between them stores energy.

so $R_{\text{th}} = 250\,\Omega$. The Norton current is $V_{\text{th}}/R_{\text{th}} = 3.00/250 = 12\,\text{mA}$. With $470\,\Omega$, $V_L = 3.00 \times 470/(250 + 470) = 1.96\,\text{V}$. Two voltage readings were enough to predict the behavior with any load, without opening the box. (Measuring the short-circuit current directly also works, but shorting an unknown output is risky.) ■

1.9 *Capacitors and Inductors*

RESISTORS ONLY DISSIPATE ENERGY. Capacitors and inductors store it, a capacitor in an electric field and an inductor in a magnetic field, and they give it back later. That memory makes circuit behavior depend on time, which is the basis of timing circuits, filters, and power supplies. This section only introduces the two elements; the circuits in the rest of this chapter are purely resistive.

Capacitors. Two conductors separated by an insulator form a capacitor (Figure 1.37). Apply a voltage v and charge $+Q$ collects on one plate and $-Q$ on the other, in proportion to the voltage:

$$Q = Cv, \qquad i = C\frac{dv}{dt}, \qquad E_C = \tfrac{1}{2}Cv^2. \tag{1.17}$$

The capacitance C, in farads, grows with plate area and shrinks with plate spacing. Practical values run from picofarads to millifarads. The middle relation follows from differentiating $Q = Cv$ and using Equation (1.1). It has two important consequences. When the voltage is constant, $i = 0$: in DC steady state *a capacitor is an open circuit*. And because a sudden jump in v would require infinite current, the voltage on a capacitor cannot change instantaneously. The last relation says the stored energy grows with the square of the voltage.

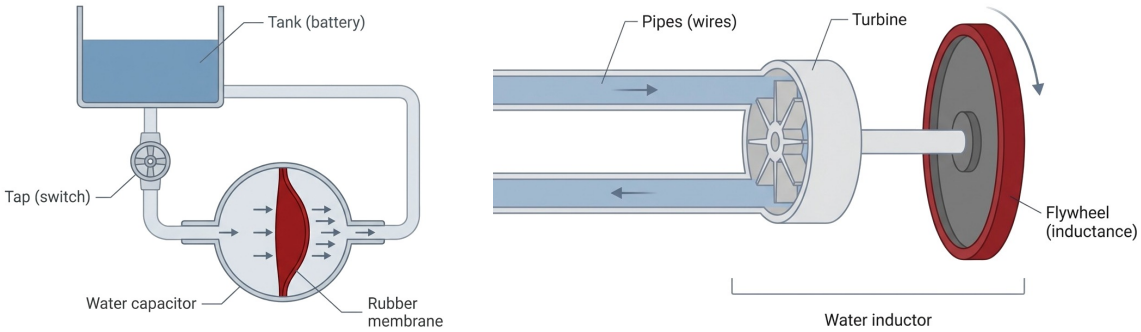


Figure 1.38: Water analogies. Left: a capacitor is a chamber divided by a rubber membrane; flow in one side bulges the membrane and pushes water out the other, and the stretched membrane stores energy. Right: an inductor is a turbine coupled to a heavy flywheel; the flywheel resists any change in the flow, and once spinning it keeps the water moving.

In the water analogy (Figure 1.38, left) a capacitor is a chamber with a rubber membrane across the middle. Water pushed in one side bulges the membrane and pushes the same amount out the other side, so current appears to flow "through" even though no water crosses the membrane. As the membrane stretches it pushes back harder, until its pressure matches the pump and the flow stops: the capacitor is charged, and the stretched membrane holds the stored energy.

Inductors. A coil of wire carrying current i produces a magnetic field, and a changing field induces a voltage that opposes the change:

$$v = L\,\frac{di}{dt}, \qquad E_L = \tfrac{1}{2}Li^2, \tag{1.18}$$

where the inductance L is in henries. Inductors are the dual of capacitors: with constant current $v = 0$, so in DC steady state *an inductor is a short circuit*, and the current through an inductor cannot change instantaneously. Their water analogue (Figure 1.38, right) is a turbine in the pipe connected to a heavy flywheel: it resists being spun up and, once spinning, keeps the water moving even after the pump stops pushing. That is why opening a switch in series with an inductor can produce a large voltage spike: the inductor tries to keep its current flowing.

Where you find them. Look closely at almost any circuit board, the Arduino on the demo board included, and you will see small ceramic capacitors placed right next to the power pins of the chips. These *decoupling* capacitors act as tiny local reservoirs: when a chip suddenly draws a burst of current, the nearby capacitor supplies it before the supply voltage can sag. Larger capacitors smooth the output of voltage regulators. Switching regulators, such as the step-down converter on the Arduino Nano 33 BLE Sense that makes 3.3 V from the 5 V USB supply, store energy in an inductor during each switching cycle and release it to the load in between. Coils also appear in wireless chargers and NFC tags, and the capacitive touch screen of a phone senses your finger as a small change in capacitance. Combined with a resistor, a

capacitor sets how fast a voltage can change, which is how the filters of Chapter 3 smooth noisy sensor signals.

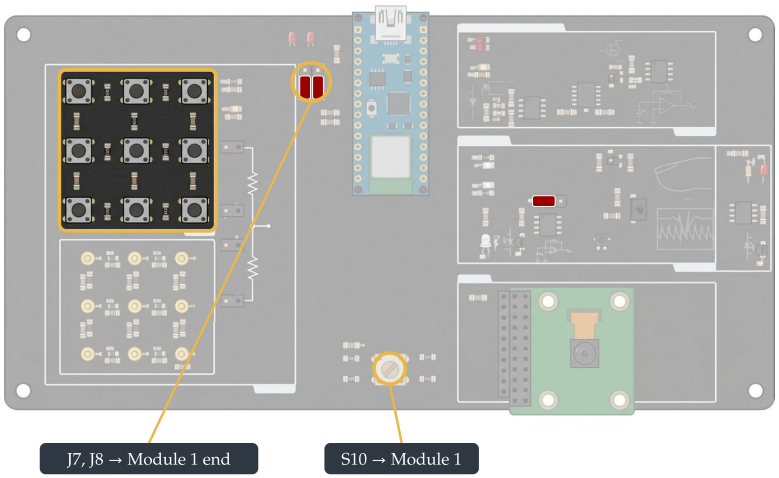
1.10 *Laboratory: Asymmetric Touch Panel*

LAB 1.1 AT A GLANCE

Goal. Measure the nine node voltages of the touch panel (Network 1), check them with nodal analysis, and build a decoder that names the pressed button from its voltage and recognizes "no button pressed."
Hardware. EE 105 demo board with the Arduino Nano 33 BLE Sense, USB data cable, laptop with the lab kit. A multimeter is useful for spot checks.
Notebook. `lab1.1-touch-panel.ipynb` in the folder `lab1.1-touch-panel` of the lab kit.
Before the lab. Read Section 1.7. With Equation (1.15), compute the voltage of S2 from $S_1 = 3.3\,\text{V}$, $S_3 = 1.185\,\text{V}$, and $S_5 = 1.650\,\text{V}$; step C1 asks for it.
Time. About 100 minutes.
You hand in. `lab1.1_submission.json` on the course portal; keep your notebook with your answers.

KEY TAKEAWAYS

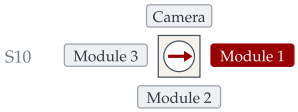
- Unequal resistors (100 Ω down, 50 Ω across) give each of the nine buttons its own voltage; an all-equal grid gives five levels.
- One count of the 10-bit ADC is $3.3\,\text{V}/1024 = 3.22\,\text{mV}$. Most buttons read within about 10 counts (30 mV) of nodal analysis.
- KCL makes every inner node the conductance-weighted average of its neighbors; your measured nodes obey it within about 10 mV.
- A decoder puts its thresholds halfway between levels. The closest pair is 169 mV (52 counts) apart: about 85 mV of margin each.
- With no button pressed, A0 floats and swings by hundreds of millivolts at 60 Hz; a held button stays within about 20 mV.



LAB 1.1: SET UP THE BOARD

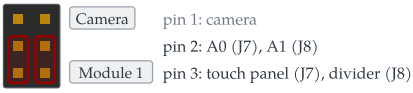
Uses: touch panel (Network 1)

Selector S10: turn it to **Module 1**



Jumpers

J7 and J8 → both caps on the **Module 1** end



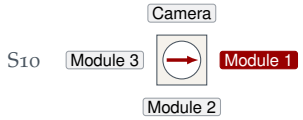
Plug in and place

Nothing in the divider header J1 to J4.

Figure 1.39: The board for Lab 1.1. Only the touch panel (Network 1, highlighted) is used. S10 is on Module 1 and the caps of J7 and J8 are on the Module 1 end, so A0 reads the button that is pressed.

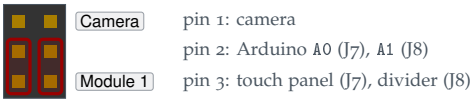
SET UP THE BOARD FOR LAB 1.1

Selector. Turn S10 to Module 1, which powers the circuits block (see Figure 0.16):



Jumpers J7 and J8. Put both caps on the Module 1 end (pins 2 and 3), so the Arduino's A0 reads the touch network:

J7 and J8 caps on the Module 1 end



Plugged in. Nothing. The divider header (J1 to J4) stays empty; this lab does not read A1. The camera can stay on J6.

Check. The first notebook cell asks the firmware for the selector position (the `module` command) and prints an `[ok]` line saying that S10 is on `Module 1` (circuits block). If the reading does not fit, it prints a `[!!]` warning instead; check that S10 points at Module 1 and carry on.

Figure 1.39 shows the touch panel, J7 and J8, and S10 on the board, and Figure 1.40 shows the lab in action.

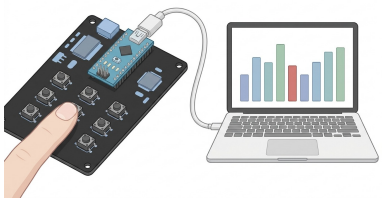
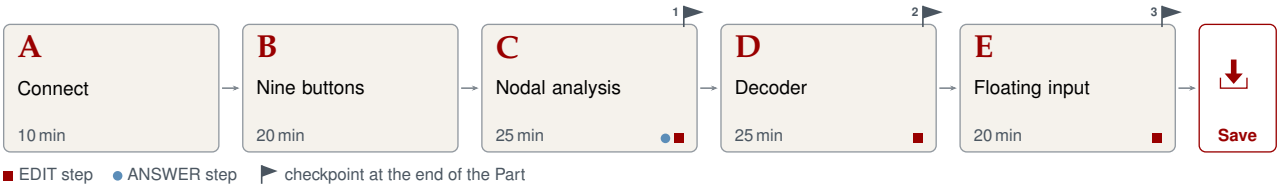


Figure 1.40: Lab 1.1 in action. A finger holds down the middle button, S5, and the notebook draws the measured voltage of each of the nine buttons; the bar of the pressed button stands out.



Objectives

- Measure the nine node voltages of the touch panel with the Arduino's 10-bit ADC.
- Check them against nodal analysis (Section 1.7) and against KCL, with a function of your own.
- Build a nearest-level decoder, and add a rule that recognizes a floating input.

Background

Network 1 is the asymmetric grid of Figure 1.32: 100 Ω vertical resistors, 50 Ω horizontal resistors, S1 on the 3.3 V rail and S9 on ground. Pressing button S*k* connects node S*k* to the shared sense line, and J7 passes that line to A0. Its nodal-analysis voltages are in the asymmetric-grid example of Section 1.7, and Equation (1.15) predicts any inner node from its neighbors.

The notebook reads A0 through the course firmware, so there is no sketch to edit and no serial port to pick. It converts ADC counts to volts with $3.3\,\text{V}/1024 = 3.22\,\text{mV}$ per count, which gives the predicted counts in Table 1.2. (Equation (1.16) would give one count fewer for S2, S3, and S4; either way, the agreement you are looking for is a few counts.)

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **B1**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT or ANSWER needs your input in the cell first (Section 0.11 explains the tags).

Part A: Connect to the board

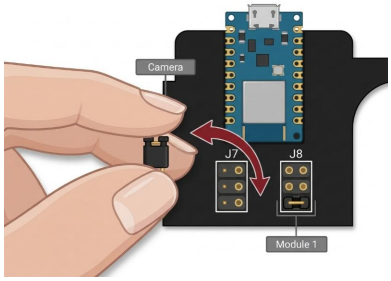
GOAL Connect the board and check that S10 reads Module 1.

about 10 min

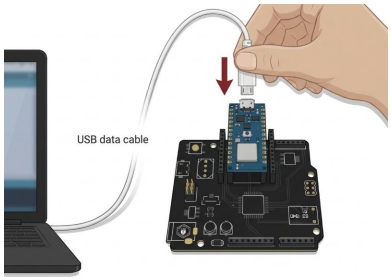
A1 ◆DO Connect the board

Set up the board as in the box above: S10 on Module 1, and both caps of J7 and J8 (the two small headers just left of the Arduino) on the Module 1 end, pins 2 and 3. Plug the Arduino into your laptop with a USB data cable, click the first code cell, A1, and press Shift+Enter. The cell finds the board, uploads the course firmware if the board needs it (for example after your own sketches of Lab 0.1), and checks the selector. No board at hand? Set `SIMULATE = True` in this cell to practice on a simulated board.

YOU SHOULD SEE two lines that begin with `[ok]`: the board is connected (firmware `ee105`), and S10 is on `Module 1` (circuits block). A `[!!]` line about the selector is only a warning: if S10 points at Module 1, carry on.



A1 J7 and J8 are single rows of 3 pins. Put each cap on pins 2 and 3, the Module 1 end. (The picture draws the two headers two pins wide.)



A1 Plug the Nano into your laptop with a USB *data* cable.

Part B: Measure the nine buttons

GOAL Measure the voltage of every button of the touch panel.

about 20 min

Before you start, predict: which button should read 3.3 V, which 0 V, and which half of 3.3 V?

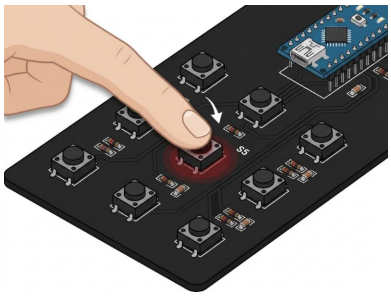
B1 ◆DO Measure the nine buttons

The loop asks for one button at a time, S1 to S9. When the box under the cell names a button, hold that button down, click the box, and press Enter; keep holding until its voltage prints, and the next button follows. `measure(name)` takes 200 readings of A0 over 40 ms (`board.read` with `n=200` and `interval_ms=0.2`) and keeps them and their average:

```
measured = {}        # button name -> volts
readings = {}        # button name -> the 200 raw ADC codes

def measure(name):
    """Hold button `name`: stores its readings and mean."""
    if board.simulated:
        board.sim.press(name)   # simulator only: hold the button
    else:
        input(f"Hold {name} down, then press Enter. "
              "Keep holding until its voltage prints. ")
    d = board.read(hw.PIN_TOUCH, n=200, interval_ms=0.2)
    readings[name] = d["raw"].to_numpy()
    measured[name] = float(d["volts"].mean())
    print(f"{name}: {measured[name]:.3f} V")

```



B1 Hold the button the box names, press Enter, and keep holding until its voltage prints.

```
for name in hw.TOUCH_NODE_V:      # "S1", "S2", ..., "S9"
    measure(name)
if board.simulated:
    board.sim.release()
```

YOU SHOULD SEE nine lines, `S1` to `S9`, each with a voltage within about 30 mV of Table 1.2: S1 near 3.30 V, S9 near 0, S5 near 1.65 V, and the others between 0.80 and 2.50 V, every one different.

B2 DO Redo one button (only if needed)

If one reading looks wrong, for example because you let go too early, redo just that button: in B2, remove the first # of the line marked ✎, put in the name of that button, hold it down, and run B2. Skip B2 if all nine look right.

Part C: Compare with nodal analysis

GOAL Check your nine voltages against KCL and the textbook's nodal analysis. about 25 min

KCL at a node n says that the currents leaving through its resistors add to zero, $\sum_m (u_n - u_m)/R_{nm} = 0$. Solved for u_n, that is Equation (1.15): every inner node voltage is a conductance-weighted average of its neighbors.

C1 ANSWER KCL at S2

C1 · ANSWER in the notebook

```
s2 = ...    # ✎ KCL at S2, solved for S2 (volts)
print(f"S2 from KCL: {s2:.3f} V")
print(f"textbook:    {hw.TOUCH_NODE_V['S2']:.3f} V")
```

TYPE IN `s2`, the voltage that KCL at S2 gives with the textbook values $S_1 = 3.3$ V, $S_3 = 1.185$ V, and $S_5 = 1.650$ V. S2 connects to S1 and S3 through 100 Ω and to S5 through 50 Ω. Type the number from your "Before the lab" calculation, or the expression.

UNIT volts.

WHERE TO FIND IT Equation (1.15) and the grid of Figure 1.32: the sum of u/R over the sum of $1/R$.

YOU SHOULD SEE `S2 from KCL: 1.946 V`, the textbook value printed under it.

C2 EDIT Write `kcl_voltage(node, V)`

C2 · EDIT in the notebook

```
# Network 1: (node, node) -> ohms.
# Vertical resistors 100 Ohm, horizontal resistors 50 Ohm.
R = {("S1", "S2"): 100, ("S2", "S3"): 100, ("S4", "S5"): 100,
```

```
    ("S5", "S6"): 100, ("S7", "S8"): 100, ("S8", "S9"): 100,
    ("S1", "S4"): 50, ("S4", "S7"): 50, ("S2", "S5"): 50,
    ("S5", "S8"): 50, ("S3", "S6"): 50, ("S6", "S9"): 50}

def kcl_voltage(node, V):
    """Voltage at node that makes its currents add to zero."""
    # ✎ for each resistor (a, b) in R that touches node:
    # ✎   add 1/r to one sum and V[other end]/r to another
    # ✎ then return the ratio of the two sums

for node in ["S2", "S3", "S4", "S5", "S6", "S7", "S8"]:
    kcl = kcl_voltage(node, measured)
    print(f"{node}: measured {measured[node]:.3f} V,"
          f" KCL from its neighbors {kcl:.3f} V")
```

WHAT TO WRITE the body of `kcl_voltage`, in place of the comments on lines 10 to 12: the voltage that KCL predicts at `node` from the voltages `V` of its neighbors, Equation (1.15). The dictionary `R` lists the twelve resistors of Network 1, each as `(node, node): ohms`. The loop at the end applies your function to your *measured* voltages.

HINT `for (a, b), r in R.items():` walks through the resistors; `node in (a, b)` tells whether one touches `node`, and its other end is `b` if `node == a`, else `a`. Keep two sums, then divide.

DONE WHEN each KCL prediction lands within about 10 mV of the measured value of that node (Question 1).

C3 ▸ RUN Compare with nodal analysis

C3 puts your measurements next to the nodal analysis of Section 1.7, in volts and in 10-bit counts (one count is 3.3 V/1024 = 3.22 mV), with the difference in millivolts and in percent, and saves the table as `touch_panel_table.csv`. Its plot draws each measured level (dot) on top of its nodal-analysis value (square), sorted by voltage.

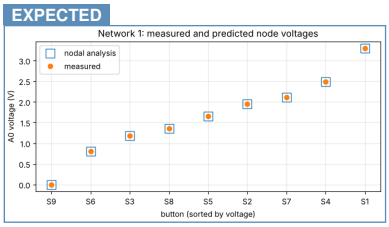
YOU SHOULD SEE most buttons within about 10 counts (30 mV) of the table; a middle button is often off by a few tens of millivolts (up to about 40 mV) because of 5% resistor tolerance. S1 usually reads a few counts below 1023 and S9 a few counts above 0 (Question 2).

✓CHECK **Checkpoint 1** All nine buttons agree with nodal analysis to within a few tens of millivolts, and the KCL predictions of C2 land close to your measurements. Your board obeys Kirchhoff's current law, and you checked it with a function of your own.

button	volts	counts
S1	3.300	1023
S2	1.946	604
S3	1.185	368
S4	2.496	775
S5	1.650	512
S6	0.804	249
S7	2.115	656
S8	1.354	420
S9	0.000	0

Table 1.2: Predicted A0 readings for Network 1, as the Lab 1.1 notebook tabulates them (3.3 V/1024 per count, S1 limited to 1023).

EXPECTED



C3 Each measured dot sits on its nodal-analysis square, sorted from S9 (0 V) to S1 (3.3 V).

Part D: Build a button decoder

GOAL Turn one voltage into a button name, and run it live.

about 25 min

A touch panel controller has to answer "which button?" from one voltage. The rule: pick the button whose level is *nearest* to the reading. That is the same as sorting the nine levels and putting a decision threshold halfway between each pair of neighbors (Figure 1.34): a reading between two thresholds belongs to the level between them. Use your measured levels rather than the predicted ones: they include the real resistor values of your board.

D1 EDIT Thresholds and `decode(v)`

D1 · EDIT in the notebook

```
1  levels = pd.Series(measured).sort_values()   # lowest first
2
3  thresholds = ...   # ✎ 8 voltages, halfway between neighbors
4
5  def decode(v):
6      """Name of the button whose level is nearest to v."""
7      # ✎ return a button name (from levels.index)
8
9  print("thresholds (V):", np.round(thresholds, 3))
10 gaps = np.diff(levels.values)               # neighbor gaps (V)
11 k = int(np.argmin(gaps))
12 lo_name, hi_name = levels.index[k], levels.index[k + 1]
13 print(f"closest pair: {lo_name} and {hi_name},",
14       f"{1000 * gaps[k]:.0f} mV",
15       f"({gaps[k] / hw.ADC_V_PER_COUNT:.0f} counts) apart;",
16       f"margin {500 * gaps[k]:.0f} mV each")
17 book = [decode(v) for v in hw.TOUCH_NODE_V.values()]
18 print("decoding the textbook voltages:", book)
19 mine = [decode(v) for v in measured.values()]
20 print("decoding your voltages:", mine)
```

WHAT TO WRITE on line 3, `thresholds`: the eight voltages halfway between neighboring levels. On line 7, the body of `decode(v)`: the name of the button whose range holds `v`. `levels` (line 1) is a pandas Series of your nine measured voltages, lowest first, with the button names as its index.

HINT `levels.values[:-1]` and `levels.values[1:]` line up each pair of neighbors. `np.searchsorted(thresholds, v)` counts the thresholds below `v`, which is the position of the button in `levels` (Figure 1.41); `levels.index` holds the names.

DONE WHEN the thresholds come out near 0.40, 0.99, 1.27, 1.50, 1.80, 2.03, 2.31, and 2.90 V; the closest pair, S3 and S8 or S2 and S7, is about 169 mV (52 counts) apart with a margin of about 85 mV each; and both decoded lists read S1 to S9 in order (Question 3).

D2 ▸ RUN Draw the levels and thresholds

D2 draws your nine levels on the voltage axis with your eight thresholds between them.

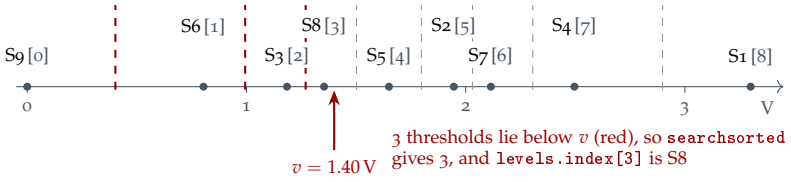


Figure 1.41: How `decode(v)` finds the button (step D1), drawn with the textbook levels. The dashed lines are the eight thresholds; the numbers in brackets are the positions in `levels`. A reading of 1.40 V has three thresholds below it, so it belongs to `levels.index[3]`, S8.

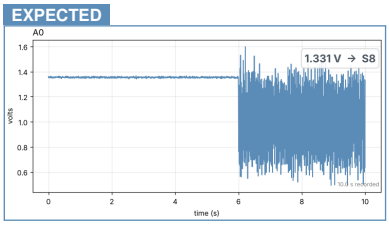
YOU SHOULD SEE a picture like Figure 1.34, built from your own levels.

D3 DO Run the decoder live (10 s)

D3 streams A0 for 10 s, one reading every 5 ms, and writes the voltage and the decoded button in the plot title; the bar on the right shows the voltage and lights up the pressed pad. Press different buttons while it runs, as in **B1**, and watch what the title says *after you let go*.

YOU SHOULD SEE the voltage and the name of the button you hold. After you let go, the title keeps naming buttons nobody pressed (Figure 1.42a): the floating input of Part E.

EXPECTED



D3 Flat while a button is held (here S8). After the release at 6 s, A0 swings by hundreds of millivolts and the title still names a button.

CHECK **Checkpoint 2** The live decoder names each of the nine buttons correctly. You built a working touch panel decoder: one analog pin, nine buttons.

D4 DO The firmware's own decoder (optional, 10 s)

The course firmware has a decoder of its own, the command `circuits` (Table 0.7). D4 sends it with `board.monitor` and stops it after 10 s; press a few buttons while it runs. Five times per second the firmware prints the button it detects and the node voltage, or `no button (A0 is floating)`. Compare its answers with your decoder's: it uses fixed thresholds from the textbook levels and a 20 ms spread test like the one you build next.

EXPECTED

```
...
S2  1.946 V
S2  1.944 V
S2  1.945 V
no button  (A0 is floating)
no button  (A0 is floating)
no button  (A0 is floating)
no button  (A0 is floating)
no button  (A0 is floating)
S7  2.115 V
S7  2.113 V
...
```

D4 Five lines a second: the button and its node voltage, or `no button` while A0 floats.

Part E: No button pressed: the floating input

GOAL Teach the decoder to say "no press" when nothing is pressed.

about 20 min

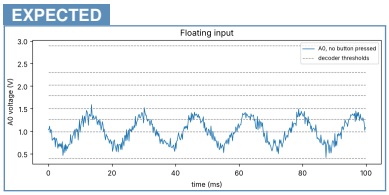
With no button pressed, nothing connects A0 to the grid. The pin *floats*: its voltage is set by stray charge and by pickup from the 60 Hz power wiring in the room.

E1 ▸ RUN A0 with no button pressed (0.1 s)

Keep your hands off the buttons and run E1. It records 0.1 s of A0 (500 readings), plots them with your decoder thresholds, and prints what `decode` makes of them.

> YOU SHOULD SEE a swing of hundreds of millivolts, often with a 60 Hz pattern (a period of about 17 ms), crossing several thresholds, and a decoder that reports buttons nobody pressed (Figure 1.42b). Yours may be larger, smaller, or less regular, and it changes when your hand comes near the board (Question 5).

EXPECTED



E1 A 60 Hz wave (period about 17 ms), hundreds of millivolts deep, across several thresholds. A real pin may swing more or less.

One reading cannot tell a floating input from a press, but a few readings can: a held button gives a *steady* voltage (only ADC noise, a few millivolts), while a floating input swings by hundreds of millivolts within one 60 Hz period (16.7 ms). So look at the *spread* (largest minus smallest) of the readings of the last 20 ms, and report "no press" when it is too large.

E2 ◆ DO Measure the 20 ms spread

Run E2. When the box asks, hold S5 down and press Enter; when it asks again, let go of all buttons and press Enter. Each time the cell takes 10 readings 2 ms apart and prints their spread.

> YOU SHOULD SEE two spreads in millivolts: under about 20 mV with S5 held, and hundreds of millivolts with no press.

E3 ✎ EDIT `SPREAD_LIMIT` and `decode_steady(v, spread)`

E3 · ✎ EDIT in the notebook

```
1  SPREAD_LIMIT = ...   # ✎ volts, between the two spreads of E2
2
3  def decode_steady(v, spread):
4      """A button name, or "no press" if not steady."""
5      # ✎ YOUR CODE HERE: a button name, or "no press"
6
7  print(f"SPREAD_LIMIT = {1000 * SPREAD_LIMIT:.0f} mV")
```

WHAT TO WRITE on line 1, `SPREAD_LIMIT` in volts, between the two spreads of **E2**. On line 5, the body of `decode_steady`: `"no press"` when `spread` is larger than `SPREAD_LIMIT`, and otherwise the button that `decode(v)` names.

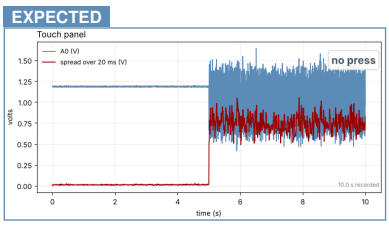
HINT a good choice comes from your data: half of the smallest gap between two levels, the decoder's own margin printed by **D1**. A reading that wobbles by more than that could cross a threshold anyway. `gaps` from D1 holds the gaps.

DONE WHEN the cell prints a limit between the two spreads of E2, near 85 mV.

E4 DO Run the improved decoder live (10 s)

E4 reads A0 every 2 ms, adds the spread of the last 10 readings (20 ms) to each sample, and plots both for 10 s, with `decode_steady` in the title and the pressed pad lit on the bar at the right. Press and release a few buttons while it runs.

YOU SHOULD SEE the name of the button while you hold one, and `no press` whenever your hands are off.



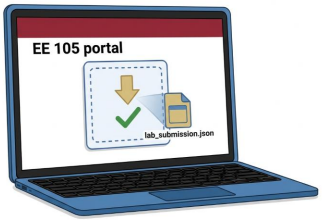
E4 The spread (red) stays near 0 while a button is held and jumps to hundreds of millivolts after the release: `no press`.

CHECK **Checkpoint 3** The improved decoder gives the right name for every button and "no press" when nothing is pressed. Your decoder tells a press from a floating pin, as a real touch controller must.

Save and upload

Save RUN Write `lab1.1_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab1.1_submission.json` in the lab folder: your measurements, KCL results, and decoder settings. Log in at `ee105.org/portal`, open Lab 1.1, and upload that file. If you redo a measurement later, run Save again and upload again. Keep the table `touch_panel_table.csv` of C3 in the folder: Lab 1.2 uses it.



Save Upload `lab1.1_submission.json` on the course portal under Lab 1.1.

YOU SHOULD SEE `[ok] Saved lab1.1_submission.json`, and then the portal shows your grade. A run on the simulated board counts as practice and earns no credit.

FINISH THE LAB

You turned a grid of resistors into a working touch panel controller: nine measured levels, a KCL check, a live decoder, and a test for a floating input. The lab is done when every box is ticked.

- ☐ **B1** DO — all nine buttons measured (redone with **B2** if needed)
- ☐ **C1** ANSWER — `s2`
- ☐ **C2** EDIT — `kcl_voltage`
- ☐ **D1** EDIT — `thresholds` and `decode`
- ☐ **E2** DO — the two 20 ms spreads
- ☐ **E3** EDIT — `SPREAD_LIMIT` and `decode_steady`
- ☐ CHECK — Checkpoint 1: nine voltages match nodal analysis and KCL
- ☐ CHECK — Checkpoint 2: the live decoder names all nine

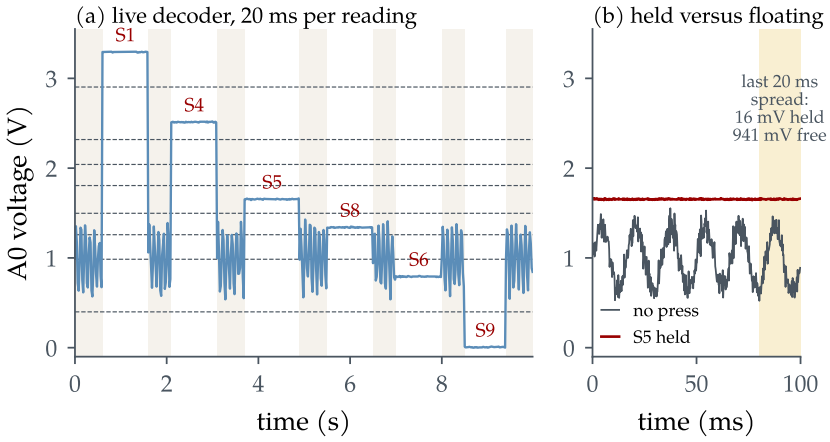


Figure 1.42: Typical Lab 1.1 readings, drawn with the touch-panel model of the simulated board (a real floating input may swing more or less). (a) Ten seconds of the live decoder: six buttons held in turn, nothing pressed in between (shaded). The dashed lines are the thresholds halfway between the measured levels. A held button gives a flat line inside its own range; the floating input swings across several ranges, so the decoder names buttons nobody pressed. (b) A tenth of a second at 0.2 ms per reading. Over the last 20 ms (shaded) the reading moves by about 16 mV with S5 held and by most of a volt, with a 60 Hz pattern, with nothing pressed.

		buttons
☐	✓CHECK	Checkpoint 3: right names, and "no press" when hands are off
☐		Questions 1 to 6 answered in the notebook
☐ **Save**	⭳SAVE	`lab1.1_submission.json` uploaded on the portal

Typical results

- Most buttons within about 10 counts (30 mV) of Table 1.2 (**C3**); a middle button is often off by a few tens of millivolts (typically up to about 40 mV) because of 5% resistor tolerance.
- The KCL predictions of **C2** within about 10 mV of the measured node, even where the node itself is off from the table, because KCL holds for the resistor values your board really has.
- Thresholds near 0.40, 0.99, 1.27, 1.50, 1.80, 2.03, 2.31, and 2.90 V (**D1**). The closest pairs, S3 and S8 and also S2 and S7, are about 169 mV (52 counts) apart, so each has about 85 mV of margin.
- With no button pressed, A0 wandering by hundreds of millivolts, often with a 60 Hz pattern; the decoder names buttons nobody pressed (Figure 1.42). The 20 ms spread with a button held is under about 20 mV (**E2**), so a `SPREAD_LIMIT` near 85 mV separates the two cases.
- S2 and S8 pressed together read about 1.65 V, decoded as S5.

Troubleshooting

- *"No Arduino found."* Use a USB data cable (some cables carry power only) and try another port.
- *The cell waits and nothing happens* (**B1**, **E2**). It waits for you: hold the button, click the box under the cell, and press Enter.
- *Every button reads about the same wandering voltage.* A0 is not connected to the touch network: check that the caps of J7 and J8 are on the Module 1 end and that S10 is on Module 1.
- *One button is far off.* You let go before the 200 readings were done. Redo it with **B2**.
- *"Port busy."* Another program has the board open: close other notebooks that use it, or any serial terminal.
- *The floating input looks quiet.* A real floating pin can drift slowly or sit near a rail. Run **E2** again a few times, and say so in Question 5.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. Which button differs most from nodal analysis (**C3**)? Give the difference in millivolts and in ADC counts. Do your measurements pass the KCL check of **C2**?
2. The first board sketch decoded the buttons with fixed windows: S1 from 1000 to 1020 counts and S9 from 1 to 199 counts, although the ideal readings are 1023 and 0. What do your S1 and S9 readings say about the rail and ground voltages the ADC sees? What does your decoder do differently?
3. Which two buttons are closest in voltage (**D1**)? How many counts apart are they, and how far can either drift before your decoder confuses them? Compare that margin with the noise.
4. Predict S2 and S4 for the symmetric Network 2 and explain why that panel cannot be decoded.
5. What did A0 read with no button pressed (**E1**), and why? Why does the spread rule use a 20 ms window rather than 2 ms?
6. Press S2 and S8 at the same time while **D3** runs. Compare the reading with the two-buttons example of Section 1.7. Then predict what S4 and S6 pressed together should read, using the same symmetry argument, and test it.

1.11 *Laboratory: Circuit Analysis with Python*

Lab 1.2 at a glance

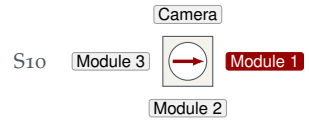
Goal. Write KCL and Ohm's law for both touch-panel networks, solve them exactly with SymPy, check the solution as $\mathbf{G}\mathbf{v} = \mathbf{i}$ with `numpy`, compute power and the effect of resistor tolerance, and check the predictions on the board: Network 1 through the buttons and A0, Network 2 with a multimeter, and a divider through A1.
Hardware. Laptop with the lab kit (Parts B to E need no board). For Part F: the demo board with the Arduino and USB cable, a multimeter, a 4.7 kΩ resistor (yellow, violet, red), and a 10 kΩ resistor (brown, black, orange).
Notebook. `lab1.2-circuit-analysis-python.ipynb` in the folder `lab1.2-circuit-analysis-python` of the lab kit. If you saved the Lab 1.1 table, step **F1** adds it to the comparison automatically.
Before the lab. Read Sections 1.6 and 1.7. On paper, write KCL at the nine nodes of Figure 1.32 with the current directions shown, and the Ohm's law equations of R_1 and R_3.
Time. About 100 minutes.
You hand in. `lab1.2_submission.json` on the course portal; keep your notebook with your answers.

Key takeaways

- A circuit is a set of linear equations, one KCL per node and one Ohm's law per resistor: 21 for the grid's 20 unknowns.
- SymPy solves them exactly ($s_2 = 253/130$ V); `numpy`'s $\mathbf{G}\mathbf{v} = \mathbf{i}$ agrees to about 10^{-15} V of rounding.
- $\mathbf{G}$ by inspection: the conductances at a node on the diagonal, minus each shared one off it, so $\mathbf{G}$ is symmetric.
- One resistor 5% high moves a node by up to 27.5 mV (8.5 ADC counts): a real board reads 10 to 20 mV off.
- Divider and ADC share the 3.3 V rail, so the A1 count depends only on the ratio: 4.7 kΩ over 10 kΩ gives 2.245 V.

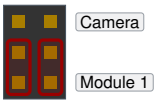
Set up the board for Lab 1.2

Selector. Turn S10 to Module 1 (circuits block). This powers Network 1, Network 2, and the divider header:



Jumpers. Both caps of J7 and J8 on the Module 1 end (pins 2 and 3): A0 reads the touch network and A1 the divider midpoint:

J7 and J8 caps on the Module 1 end



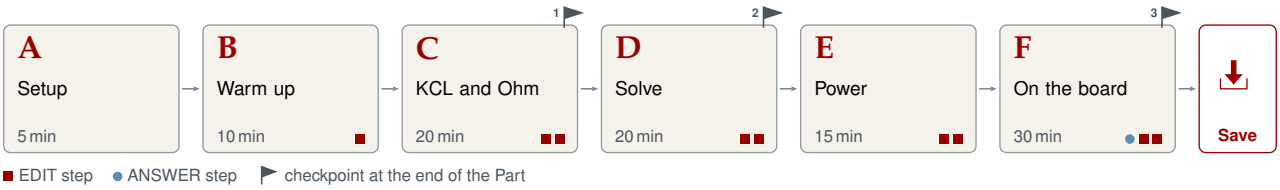
pin 1: camera
pin 2: Arduino A0 (J7), A1 (J8)
pin 3: touch panel (J7), divider (J8)

Plugged in. For step F4, the 4.7 kΩ resistor in sockets J1-J2 (top, R1 on the board, from 3.3 V) and the 10 kΩ resistor in J3-J4 (bottom, R2, to ground) of the divider header, the column of four sockets to the right of the button grid (Figure 1.46). For step F2, the multimeter on DC volts with its black probe on TP9 (ground) and its red probe on each Network 2 test point in turn.

Check. The first notebook cell prints two lines beginning with `[ok]`, the second saying that S10 is on `Module 1` (circuits block). If the reading does not fit, the cell prints a `[!!]` warning instead; check that S10 points at Module 1 and carry on.

Afterwards. Leave the caps of J7 and J8 on the Module 1 end (their usual position) and take the two resistors out of the header.

Figure 1.43 shows the three parts of the circuits block this lab uses, with S10, J7, and J8.



Objectives

- Translate a schematic into linear equations: one KCL equation per node and one Ohm's law equation per resistor.
- Solve them exactly with SymPy, check the result as $\mathbf{G}\mathbf{v} = \mathbf{i}$ with `numpy`, and use the solver for power and resistor tolerance.
- Compare the predictions with the board: buttons on A0, the test points of Network 2 with a multimeter, and a divider on A1.



Figure 1.43: The board for Lab 1.2. The touch panel (Network 1), the resistor grid with its test points (Network 2), and the divider header J1 to J4 are highlighted. S10 is on Module 1, and the caps of J7 and J8 are on the Module 1 end. The two resistors go into the header for step F4 only.

Background

Section 1.6 gave two ways to set up a circuit for a computer. The *branch* form keeps every current as an unknown: for the grid that is 12 resistor currents $i_1, \ldots, i_{12}$, the supply current i_t, and seven node voltages $s_2, \ldots, s_8$, with $s_1 = 3.3$ and $s_9 = 0$, 20 unknowns in all. Its equations are KCL at each node (for example $i_t = i_1 + i_3$ at S1 and $i_1 = i_2 + i_4$ at S2) and Ohm's law for each resistor: if R_k joins Sa to Sb in the direction of its current, then $i_k R_k = s_a - s_b$. SymPy's `solve` takes these 21 equations and returns exact fractions. The *nodal* form keeps only the seven node voltages and builds the 7×7 conductance matrix with the inspection rules of Section 1.6. It is smaller and is what large simulators use. The notebook supplies the machinery for both; you write the parts that need an engineer: the KCL list, the table of which nodes each resistor joins, and the matrix of Network 2 by hand. The notebook's function that builds **G** (step **C3**) works resistor by resistor:

```
def nodal_system(R, known={1: 3.3, 9: 0.0}):
    """G and i of G v = i for resistor values R (ohms)."""
    row = {n: m for m, n in enumerate(UNKNOWN)}  # S2..S8
    G, cur = np.zeros((7, 7)), np.zeros(7)
    for k, (a, b) in EDGES.items():   # R_k joins S_a, S_b
        g = 1.0 / R[k]
        for p, q in ((a, b), (b, a)): # both ends of R_k
            if p in row:
                G[row[p], row[p]] += g
                if q in row:
                    G[row[p], row[q]] -= g
                else:
                    cur[row[p]] += g * known[q]
```

```
    return G, cur
```

Network 2, the symmetric grid next to the buttons, is wired like Figure 1.32 with all twelve resistors 100 Ω: its corner TP1 is on 3.3 V, the opposite corner TP9 is on ground, and test point TP*k* sits at grid node *k*, numbered row by row (TP1, TP2, TP3 across the top; Figure 1.45). The figure numbers S1 to S9 column by column, but because the all-100 Ω grid is symmetric about its TP1-TP9 diagonal, TP*k* has the same voltage as S*k*. It has no buttons; you read it with the multimeter.

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **B1**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT or ANSWER needs your input in the cell first (Section 0.11 explains the tags). Parts B to E are pure Python; the board comes back in Part F.

Part A: Setup

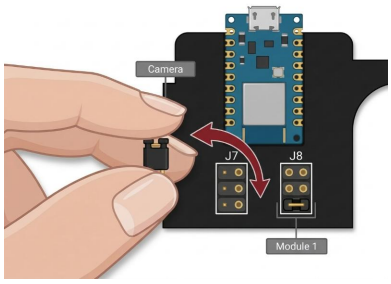
GOAL Connect the board on Module 1; Part F needs it again.

about 5 min

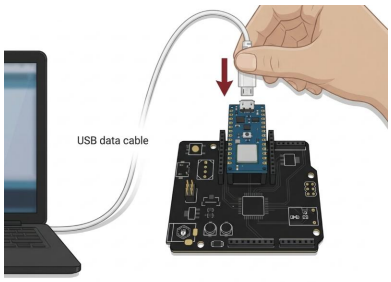
A1 DO Connect the board

Set up the board as in the box above: S10 on Module 1, both caps of J7 and J8 on the Module 1 end. Plug the Nano into your laptop with a USB *data* cable, click the first code cell (A1), and press Shift+Enter. The cell finds the board, uploads the course firmware if the board needs it, checks the selector, and defines `show(sol)`, which prints a SymPy solution as exact values and as decimals. No board today? Set `SIMULATE = True` in this cell to practice on a simulated board, which gives the ideal node voltages.

YOU SHOULD SEE two lines that begin with `[ok]`: the board is connected (firmware `ee105`), and S10 is on `Module 1` (circuits block). A `[!!]` line about the selector is only a warning: if S10 points at Module 1, carry on.



A1 J7 and J8 are single rows of three pins: put each cap on the Module 1 end, over pins 2 and 3. (The picture draws the headers two pins wide.)



A1 Plug the Nano into your laptop with a USB *data* cable.

Part B: Warm up: SymPy solves a small circuit exactly

GOAL Let SymPy solve KVL, KCL, and Ohm's law exactly.

about 10 min

SymPy solves a list of equations for a list of unknowns. Write one equation per law you use: KVL around a loop, KCL at a node, Ohm's law for each resistor, each as `Eq(left, right)`. The call `sp.solve(eqs, unknowns, dict=True)[0]` returns a dictionary that maps each unknown to its exact value. With integer resistor and source values, the answers are exact fractions.

B1 ▸ RUN Example: 5 V across two resistors in series

B1 solves a 5 V source driving $R_1 = 100\,\Omega$ and $R_2 = 200\,\Omega$ in series with three equations: KVL around the loop and Ohm's law for each resistor. Read it before you write your own.

YOU SHOULD SEE `I = 1/60 A = 16.67 mA`, `V1 = 5/3 V = 1.667 V`, and `V2 = 10/3 V = 3.333 V`.

EXPECTED

```
I = 1/60 A = 16.67 mA
V1 = 5/3 V = 1.667 V
V2 = 10/3 V = 3.333 V
```

B1 What `show` prints: each unknown as an exact fraction, then as a decimal (currents also in mA).

B2 EDIT A series-parallel circuit

B2 · EDIT in the notebook

```
1 It, I2, I3, Vm, Req = sp.symbols("It I2 I3 Vm Req")
2 V, R1, R2, R3 = 10, 100, 200, 300
3 eqs_mixed = [...]   # ✎ five Eq(...), with the node voltage Vm
4 sol_mixed = sp.solve(eqs_mixed, [It, I2, I3, Vm, Req],
5                      dict=True)[0]
6 show(sol_mixed)
```

WHAT TO WRITE `eqs_mixed` (line 3), five equations for a 10 V source driving $R_1 = 100\,\Omega$ in series with $R_2 = 200\,\Omega \parallel R_3 = 300\,\Omega$. Use the voltage `Vm` of the node between R_1 and the parallel pair as an unknown: Ohm's law for R_1 (the voltage across it is $V - V_m$), for R_2, and for R_3 (each has V_m across it), KCL at the middle node, and the definition of the equivalent resistance, $V = I_t R_{\text{eq}}$. Line 1 makes the symbols.

HINT write each as `Eq(left, right)`, as in **B1**. The most common mistake is to put the full 10 V across R_1.

DONE WHEN `Req = 220 Ohm`, `It = 1/22 A = 45.45 mA`, `Vm = 60/11 V = 5.455 V`, `I2 = 27.27 mA`, and `I3 = 18.18 mA`.

Part C: KCL and Ohm's law for the touch-panel grid

GOAL Write the 9 KCL and 12 Ohm's law equations of the grid.

about 20 min

The touch panel is the 3×3 grid of Figure 1.32, drawn again in the margin with the current directions (Figure 1.44). S1 is held at 3.3 V and S9 is grounded. Vertical resistors are 100 Ω; horizontal resistors are 50 Ω in **Network 1** (asymmetric) and 100 Ω in **Network 2** (symmetric). Resistor R_k carries current i_k, positive downward or to the right; i_t is the current from the 3.3 V rail into S1.

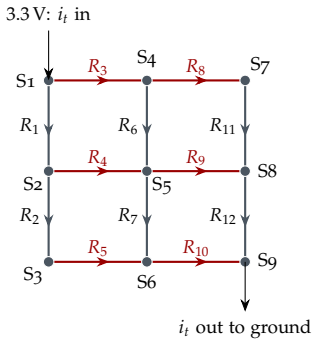


Figure 1.44: The grid of Figure 1.32 with the direction of every current: i_k in R_k flows down or to the right, i_t comes in at S1 and leaves at S9. Vertical resistors (slate) are 100 Ω; horizontal ones (cardinal) are 50 Ω in Network 1 and 100 Ω in Network 2.

C1 EDIT KCL at all nine nodes

C1 · EDIT in the notebook

```
1  s = {n: sp.Symbol(f"s{n}") for n in range(1, 10)}  # volts
2  i = {k: sp.Symbol(f"i{k}") for k in range(1, 13)}  # amperes
3  it = sp.Symbol("it")              # rail current into S1 (A)
4
5  kcl = [
6      Eq(it, i[1] + i[3]),        # S1: rail in; R1, R3 out
7      Eq(i[1], i[2] + i[4]),      # S2: R1 in; R2, R4 out
8      # ✎ S3 ... S9: one Eq per node, in = out
9  ]
10 print(f"{len(kcl)} KCL equations")
```

WHAT TO WRITE the seven KCL equations for S3 to S9 in place of line 8, one `Eq(in, out)` per node with the symbols `i[k]` and `it`. Lines 1 to 3 make the unknowns as SymPy symbols: `s[5]` is the voltage of S5, `i[12]` the current in R_{12}, and `it` the current from the rail. The first two equations, S1 and S2, are given. This list is the only place where the grid's KCL appears; everything after it is built from it.

HINT current in = current out, with the directions of Figure 1.44: an arrow pointing into the node is a current in. S5 has two currents in and two out; S9 sends `it` out to ground.

DONE WHEN the cell prints `9 KCL equations`. A wrong equation shows up only in **D1**, as "No unique solution" or as wrong voltages.

C2 EDIT Complete EDGES: which nodes each resistor joins

C2 · EDIT in the notebook

```
1  # k: (a, b): R_k joins S_a and S_b, i_k flows from a to b
2  EDGES = {
3      1: (1, 2),      # R1: S1 down to S2
4      3: (1, 4),      # R3: S1 right to S4
5      # ✎ R2, R4, R5, ... R12 the same way
6  }
7  print(f"{len(EDGES)} resistors in EDGES")
```

WHAT TO WRITE the ten missing entries of `EDGES` in place of line 5, `k: (a, b)` for R_2, R_4, R_5, …, R_{12}: R_k joins Sa and Sb, and i_k flows from Sa to Sb.

HINT S1, S2, and S3 run down the left column, so R_2 joins S2 and S3: `2: (2, 3)`. A vertical resistor goes from the upper node to the lower one, a horizontal one from left to right.

DONE WHEN the cell prints `12 resistors in EDGES`.

C3 ▸ RUN The Ohm's law equations and the two solvers

C3 sets the values of Network 1, `R_NET1` (50 Ω for the six horizontal resistors R_3, R_4, R_5, R_8, R_9, R_{10} and 100 Ω for the others), turns `EDGES` into the twelve Ohm's law equations $i_k R_k = s_a - s_b$, and prints them for Network 1, so you can check them against the ones you wrote by hand. It also defines the two solvers. `solve_grid(R)` is the branch form: it puts your KCL list and the Ohm's law equations together, fixes $s_1 = 3.3$ V (as the exact fraction 33/10) and $s_9 = 0$, and solves for the 20 unknowns, or stops with "No unique solution." `nodal_system(R)` is the nodal form, the listing of the Background, and `node_voltages(R)` solves it for S1 to S9.

YOU SHOULD SEE twelve lines, from `Eq(100*i1, s1 - s2)`, `Eq(100*i2, s2 - s3)`, and `Eq(50*i3, s1 - s4)` to `Eq(100*i12, s8 - s9)`. Every horizontal resistor shows 50.

EXPECTED

```
Eq(100*i1, s1 - s2)
Eq(100*i2, s2 - s3)
Eq(50*i3, s1 - s4)
...
```

C3 The first three of the twelve lines, in SymPy's notation: the horizontal R_3 shows 50.

✓CHECK **Checkpoint 1** The printout of the twelve Ohm's law equations (C3) matches the list you wrote by hand, and your KCL list has nine equations (C1). Twenty-one equations that describe the whole grid, written by you.

Part D: Solve both networks: SymPy, checked with `numpy`

GOAL Solve both networks exactly, then check them as $\mathbf{G}\mathbf{v} = \mathbf{i}$.

about 20 min

D1 ✎ EDIT Solve both networks exactly

D1 · ✎ EDIT in the notebook

```
1 R_NET2 = ...     # ✎ like R_NET1, all twelve 100 Ohm
2 sol1 = ...       # ✎ solve_grid for Network 1
3 sol2 = ...       # ✎ solve_grid for Network 2
4
5 report(sol1, sol2)   # tables, grid_solution.csv, KCL at S5
```

WHAT TO WRITE `R_NET2`, the resistor dictionary of Network 2 with all twelve resistors 100 Ω (line 1), then `sol1` and `sol2`, the solutions of Network 1 and Network 2 (lines 2 and 3). Line 5 prints both solutions (node voltages as exact fractions and in volts, currents in mA), saves them as `grid_solution.csv`, and checks KCL at S5: in through R_4 and R_6, out through R_7 and R_9.

HINT a dictionary comprehension like the one for `R_NET1` in C3, then `solve_grid` once per network.

DONE WHEN Network 1 gives $s_2 = 253/130 = 1.946$ V and $s_8 = 88/65 = 1.354$ V, Network 2 gives $s_2 = 11/5 = 2.200$ V and $s_5 = 33/20 = 1.650$ V, and KCL at S5 balances exactly: 14.38 mA in and out for Network 1, 11.00 mA for Network 2.

The nodal form keeps only the seven unknown node voltages, s_2 to s_8, and the inspection rules of Section 1.6 give $\mathbf{G}$ directly: G_{nn} is the sum of the conductances $1/R$ of all resistors touching node n; G_{nm}

is minus the conductance of the resistor joining n and m (0 if there is none); and i_n is V_k/R for every resistor joining n to a node held at a known voltage V_k. `nodal_system` applies these rules resistor by resistor, the way circuit simulators such as SPICE do; now you apply them by hand.

D2 EDIT Write **G** for Network 2 and solve it with `numpy`

D2 · EDIT in the notebook

```
1  G2_hand = np.array([
2      # s2   s3   s4   s5   s6   s7   s8
3      [ 3,  -1,   0,  -1,   0,   0,   0],   # S2: touches S1, S3, S5
4      # ✎ rows for S3 ... S8
5  ]) / 100
6  G2, cur2 = nodal_system(R_NET2)        # the script's G and i
7  v2 = ...      # ✎ solve G2_hand v = cur2 (7 node voltages)
8
9  print("Network 2 by NumPy (V):", np.round(v2, 3))
10 match = G2_hand.shape == G2.shape and np.allclose(G2_hand, G2)
11 print("your G matches the script:", match)
12 print("G is symmetric:", np.allclose(G2_hand, G2_hand.T))
13 v1 = np.linalg.solve(*nodal_system(R_NET1))     # Network 1
14 exact = [float(sol[s[n]]) for sol in (sol1, sol2)
15          for n in UNKNOWN]
16 err = float(np.max(np.abs(np.concatenate([v1, v2]) - exact)))
17 print(f"largest difference from SymPy: {err:.1e} V")
18 if match and err < 1e-9:
19     print("✓ Checkpoint 2 passed: NumPy agrees")
```

WHAT TO WRITE rows S3 to S8 of `G2_hand` in place of line 4, with the inspection rules. All resistors are 100 Ω, so write the matrix in units of $1/(100\,\Omega)$ (small integers); line 5 divides by 100. The columns are s_2 to s_8 (line 2); the S2 row is given, since S2 touches S1, S3, and S5. On line 7, `v2`: the seven node voltages from $\mathbf{Gv} = \mathbf{i}$ with your matrix. The cell then solves Network 1 the same way and compares both with the exact SymPy solutions.

HINT the diagonal entry is the number of resistors at the node; put -1 for each neighbor that is not S1 or S9 (those have known voltages and go into **i**, not **G**). `np.linalg.solve(G, i)` solves $\mathbf{Gv} = \mathbf{i}$.

DONE WHEN the cell prints the Network 2 voltages 2.2, 1.65, 2.2, 1.65, 1.1, 1.65, and 1.1 V, `True` after both `your G matches the script` and `G is symmetric`, and a largest difference from SymPy of about 10^{-15} V, the rounding error of floating-point arithmetic.

EXPECTED

```
...
your G matches the script: True
G is symmetric: True
largest difference from SymPy: 8.9e-16 V
✓ Checkpoint 2 passed: NumPy agrees
```

D2 The four check lines under the Network 2 voltages: `True`, `True`, a difference near 10^{-15} V, and the check mark.

CHECK **Checkpoint 2** SymPy gives exact solutions for both networks, KCL at S5 balances exactly (D1), and your **G** and `numpy` agree with them (D2 prints ✓). Exact answers for a circuit with 20 unknowns, checked a second way.

Part E: Power and resistor tolerance

GOAL Find the hottest resistor, and how far a 5% resistor moves each node. about 15 min

E1 EDIT Power in each resistor and from the rail

E1 · EDIT in the notebook

```
P_mW = ...          # ✎ {k: power in resistor k, in mW}
P_rail_mW = ...     # ✎ power from the 3.3 V rail, in mW

total = sum(P_mW.values())
print(f"rail {P_rail_mW:.1f} mW, resistors {total:.1f} mW")
for k in sorted(P_mW, key=P_mW.get, reverse=True)[:4]:
    print(f"R{k} ({R_NET1[k]} Ohm): {P_mW[k]:.1f} mW = "
          f"{P_mW[k] / 100:.0%} of a 0.1 W rating")
```

WHAT TO WRITE `P_mW`, a dictionary {k: power in R_k} in milliwatts with $P = I^2R$ for Network 1 (line 1), and `P_rail_mW`, the power drawn from the rail, $P = 3.3\,\text{V} \times i_t$ (line 2). The two totals must agree.

HINT `sol1[i[k]]` is the current in R_k and `sol1[it]` the rail current, both in amperes (`float()` turns them into numbers); `R_NET1[k]` is in ohms. Multiply watts by 1000 for milliwatts.

DONE WHEN the rail and the twelve resistors both give 97.7 mW; R_1 and R_{12} run hottest at 18.3 mW each (18% of a 0.1 W rating), then R_3 and R_{10} at 12.9 mW.

E2 EDIT Make one resistor `TOL` high at a time

E2 · EDIT in the notebook

```
base = node_voltages(R_NET1)
TOL = 0.05              # ✎ 5 % tolerance; try 0.01 too
shift_mV = {}
for k in range(1, 13):
    R_changed = ...     # ✎ R_NET1 with R_k made TOL larger
    shift_mV[k] = ...   # ✎ largest node change (mV)
print(f"largest change: {max(shift_mV.values()):.1f} mV")
```

WHAT TO WRITE the two lines inside the loop (lines 5 and 6). For each of the twelve resistors in turn, `R_changed` is a copy of `R_NET1` with only R_k made `TOL` larger, and `shift_mV[k]` is the largest change of any node voltage from `base`, in millivolts. `node_voltages(R)` of C3 solves the grid for any resistor values. Run it with `TOL = 0.05`, then set line 2 to `TOL = 0.01` and run E2 and E3 again.

HINT `dict(R_NET1)` makes a copy you can change; changing entry `k` is a second line. `np.max(np.abs(...))` gives the largest change.

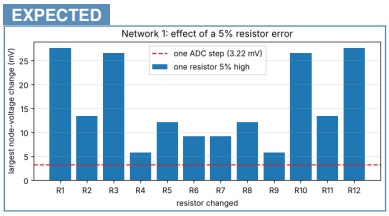
DONE WHEN the cell prints a largest change of 27.5 mV for 5% and 5.6 mV for 1%.

E3 ▸ RUN Plot the tolerance study

E3 plots the twelve changes as bars next to one ADC step (3.22 mV) and prints the worst case.

YOU SHOULD SEE for `TOL = 0.05`, the four corner resistors move a node by 26.5 to 27.5 mV, the others by 5.7 to 13.4 mV, all above one ADC step,

EXPECTED



E3 The bars for 5%: the four corner resistors R_1, R_3, R_{10}, and R_{12} tallest, and every bar above the dashed ADC step.

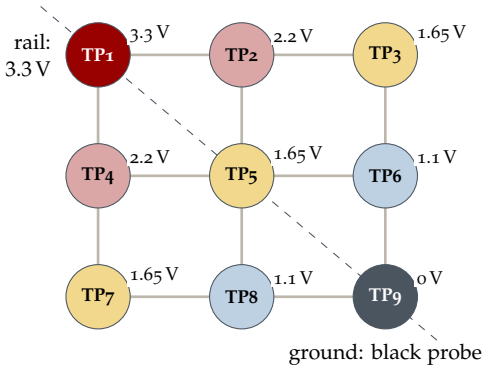


Figure 1.45: The test points of Network 2, numbered row by row, with the voltages of the symmetric solution. Mirror images across the dashed diagonal read the same: TP2 and TP4, TP6 and TP8, and TP3, TP5, TP7.

and `worst case: R1, 27.5 mV = 8.5 ADC steps.` For `TOL = 0.01`, the worst case is 5.6 mV, 1.7 steps (Question 3).

Part F: On the board: buttons, multimeter, divider

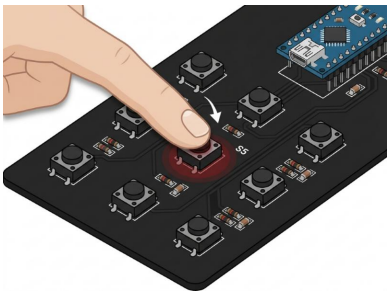
GOAL Check the predictions with A0, a multimeter, and a divider on A1. about 30 min

The board comes back: S10 on Module 1 powers both networks and the divider header, and the caps of J7 and J8 on the Module 1 end connect the buttons to A0 and the divider midpoint to A1.

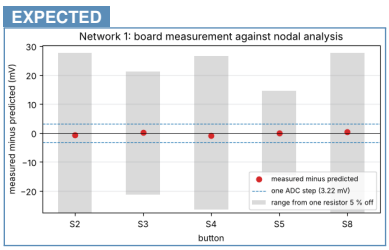
F1 ◆DO Measure five buttons and compare

Your solution predicts the voltage each button puts on A0. Run F1. It asks for S2, S3, S4, S5, and S8 in turn: when a box names a button, hold that button down, then press Enter in the box, and keep holding until its voltage prints. S2 and S4 would read the same on the symmetric Network 2, and S3 and S8 are the closest pair of Network 1. The prompts wait for you, so "Run All" pauses here on purpose. The cell then puts each measurement next to your prediction (`sol1`) and the toolkit's stored values (`hw.TOUCH_NODE_V`), with the difference in millivolts and in ADC counts and the largest shift that one resistor 5% high could cause at that node, adds Lab 1.1's measurements if it finds `touch_panel_table.csv` from that lab, and saves the table as `board_check.csv`.

YOU SHOULD SEE five voltages close to 1.946, 1.185, 2.496, 1.650, and 1.354 V, and a plot of measured minus predicted: on a real board within about 10 to 20 mV for every button, inside the gray bars. One resistor 5% high moves these buttons by 15 to 28 mV (Figure 1.47b).



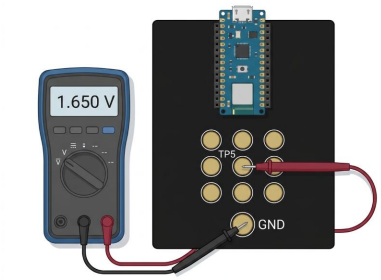
F1 Hold the named button down, press Enter, and keep holding until its voltage prints. (The picture shows S5.)



F1 The simulator's dots sit near 0 (ideal resistors); a real board's within 10 to 20 mV, inside the gray bars.

F2 ANSWER Multimeter readings of TP1 to TP9

Network 2 sits next to the buttons, with its test points TP1 to TP9 numbered row by row (Figure 1.45); the prediction for TPk is s_k of your Network 2 solution `sol2`. A voltmeter goes in parallel with what it measures (Figure 1.21). Set the multimeter to DC volts (the 20 V range, or auto), hold the black probe on TP9 (ground), and touch the red probe to TP1, TP2, . . . , TP9 in turn.



F2 Black probe on TP9, red probe on each test point in turn. (The picture puts the black probe on a pad marked GND; on the board it goes on TP9.)

F2 · ANSWER in the notebook

```
1  tp_measured = {
2      "TP1": None, "TP2": None, "TP3": None,   # ✎ top row (V)
3      "TP4": None, "TP5": None, "TP6": None,   # ✎ middle row
4      "TP7": None, "TP8": None, "TP9": None,   # ✎ bottom row
5  }
6
7  tps = [f"TP{n}" for n in range(1, 10)]
8  pred = [float(sol2[s[n]]) for n in range(1, 10)]
9  meas = [tp_measured[tp] for tp in tps]
10 net2 = pd.DataFrame(
11     {"predicted_V": pred, "multimeter_V": meas},
12     index=tps, dtype=float)
13 net2["diff_mV"] = 1e3 * (net2.multimeter_V - net2.predicted_V)
14 net2.round(3)
```

TYPE IN `tp_measured`: your nine readings in place of each `None`, in the rows of the board (lines 2 to 4). The cell shows them next to the prediction s_k of `sol2` and the difference in millivolts.

UNIT volts, with all the digits the meter shows.

WHERE TO FIND IT the multimeter display, one test point at a time; the layout is Figure 1.45.

YOU SHOULD SEE TP1 near 3.3 V, TP9 at 0 V, and the other seven within a few tens of millivolts of the symmetric solution (Figure 1.47a). The mirror pairs read alike (Question 5).

F3 EDIT Predict the divider voltage

F3 · EDIT in the notebook

```
1  R_TOP, R_BOTTOM = 4.7e3, 10e3   # ✎ your parts (ohms)
2  v_predicted = ...    # ✎ divider rule (V)
3  print(f"predicted V_A1 = {v_predicted:.3f} V")
```

WHAT TO WRITE `v_predicted` (line 2), the divider rule: 3.3 V times $R_{\text{bottom}}/(R_{\text{top}} + R_{\text{bottom}})$. Line 1 holds the two parts, 4.7 kΩ (yellow, violet, red) on top and 10 kΩ (brown, black, orange) at the bottom; change it only if your kit has other values.

HINT `R_TOP` and `R_BOTTOM` are in ohms.

DONE WHEN the cell prints `predicted V_A1 = 2.245 V` for 4.7 kΩ over 10 kΩ.

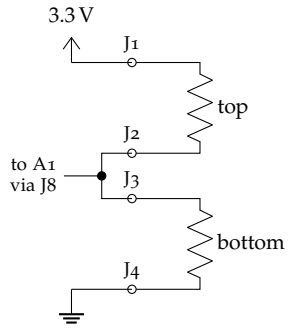
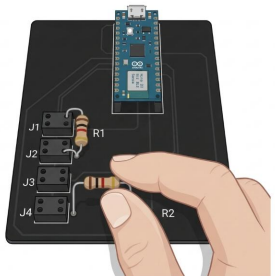


Figure 1.46: The divider header of the circuits block. J1 is the 3.3 V rail, J2 and J3 are both the midpoint (to A1 through J8), and J4 is ground. The top resistor plugs into J1-J2 and the bottom one into J3-J4; the board prints them as R1 and R2 beside the sockets.

F4 DO Plug in the divider and read A1

Plug the 4.7 kΩ resistor into the divider header between J1 and J2 (top, from 3.3 V; printed R1) and the 10 kΩ resistor between J3 and J4 (bottom, to ground; printed R2), as in Figure 1.46. The midpoint, J2 and J3, reaches A1 through J8. Run F4. It reads A1 1000 times and prints the mean next to your prediction, with the noise, and warns with `[!!]` if the reading is more than 0.3 V off.



F4 R1, the top resistor (4.7 kΩ), in J1-J2; R2, the bottom one (10 kΩ), in J3-J4. (The picture labels them only R1 and R2.)

YOU SHOULD SEE `predicted 2.245 V, measured ...` with a reading between about 2.17 and 2.32 V, the range that 5% parts allow, and a noise of a few millivolts.

F5 EDIT Infer the top resistor from the reading

F5 · EDIT in the notebook

```
1 r_top_measured = ...  # ✎ top resistor from v_a1 (ohms)
2 print(f"top resistor {r_top_measured / 1e3:.2f} kOhm measured, "
3       f"{R_TOP / 1e3:.1f} kOhm marked")
```

WHAT TO WRITE `r_top_measured` (line 1), the top resistor that would give exactly the measured voltage: $R_{top} = R_{bottom}(3.3\,V/V_{A1} - 1)$.

HINT `v_a1` is the mean of the readings of F4. Compare the result with the color code: a gold fourth band means 5%.

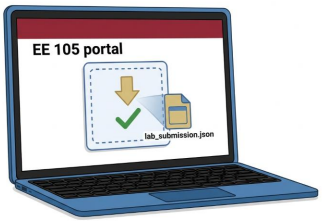
DONE WHEN the cell prints a top resistor within about 5% of 4.7 kΩ (4.47 to 4.94 kΩ) if the bottom resistor is exact, and up to about 10% off, because it absorbs the tolerance of both parts (Question 6).

CHECK **Checkpoint 3** The buttons (F1) and the multimeter (F2) agree with your solutions within a few tens of millivolts, and the divider reads close to your prediction (F4). Paper, Python, the board, and a multimeter all tell the same story about one circuit.

Save and upload

Save RUN Write `lab1.2_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab1.2_submission.json` in the lab folder: your solutions, the power and tolerance numbers, and the board, multimeter, and divider readings. Log in at `ee105.org/portal`, open Lab 1.2, and upload that file.



Save Upload `lab1.2_submission.json` on the course portal under Lab 1.2.

YOU SHOULD SEE `[ok] Saved lab1.2_submission.json`, and then the portal shows your grade. A run on the simulated board counts as practice and earns no credit.

Finish the lab

You solved a 20-unknown circuit exactly with SymPy, checked it as $\mathbf{Gv} = \mathbf{i}$ with `numpy`, found its power and tolerance, and checked it with the board, a multimeter, and a divider. The lab is done when every box is ticked.

- ☐ **B2** EDIT `eqs_mixed`
- ☐ **C1** EDIT the KCL list `kcl`, nine equations
- ☐ **C2** EDIT `EDGES`, twelve resistors
- ☐ **D1** EDIT `R_NET2`, `sol1`, `sol2`
- ☐ **D2** EDIT `G2_hand`, rows S3 to S8, and `v2`
- ☐ **E1** EDIT `P_mW`, `P_rail_mW`
- ☐ **E2** EDIT `R_changed`, `shift_mV`, run for 5% and 1%
- ☐ **F1** DO five buttons measured on your own board
- ☐ **F2** ANSWER `tp_measured`, nine multimeter readings
- ☐ **F3** EDIT `v_predicted`
- ☐ **F4** DO the divider plugged in and A1 read
- ☐ **F5** EDIT `r_top_measured`
- ☐ CHECK Checkpoint 1: twenty-one equations of the grid
- ☐ CHECK Checkpoint 2: exact solutions, checked with `numpy`
- ☐ CHECK Checkpoint 3: paper, Python, board, and meter agree
- ☐ Questions 1 to 6 answered in the notebook
- ☐ **Save** SAVE `lab1.2_submission.json` uploaded on the portal

Typical results

node	Network 1 (V)	Network 2 (V)
S1	3.300	3.30
S2	$253/130 = 1.946$	2.20
S3	$77/65 = 1.185$	1.65
S4	$649/260 = 2.496$	2.20
S5	$33/20 = 1.650$	1.65
S6	$209/260 = 0.804$	1.10
S7	$55/26 = 2.115$	1.65
S8	$88/65 = 1.354$	1.10
S9	0.000	0.00

Table 1.3: Node voltages of the two networks (**D1**). Only Network 1 gives every button its own voltage; in Network 2, TPk reads s_k.

- Network 1, exactly: the node voltages of Table 1.3. Currents in mA: $I_1 = I_{12} = 13.538$, $I_2 = I_5 = I_8 = I_{11} = 7.615$, $I_3 = I_{10} = 16.077$, $I_4 = I_9 = 5.923$, and $I_6 = I_7 = 8.462$; KCL at S5, 14.38 mA in and out.
- Network 2: 3.3, 2.2, 1.65, 2.2, 1.65, 1.1, 1.65, 1.1, and 0 V; 11 mA in R_1, R_3, R_{10}, and R_{12} and 5.5 mA in the other eight. `numpy` agrees with SymPy to about 10^{-15} V (**D2**).

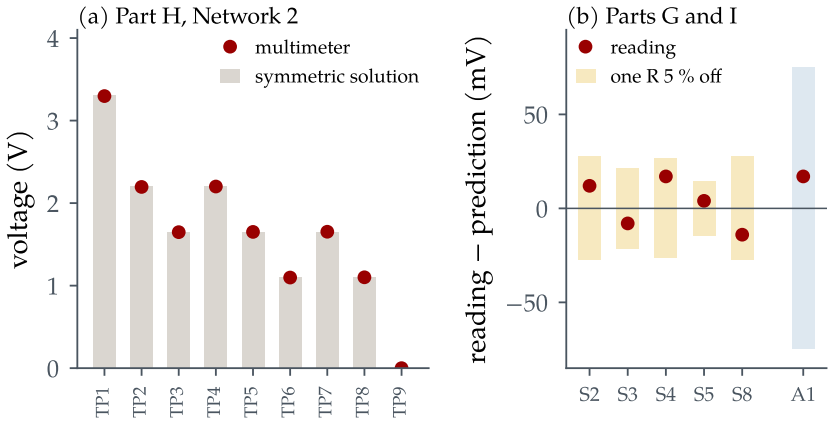


Figure 1.47: Typical board checks in Lab 1.2. (a) Multimeter readings of the nine Network 2 test points (dots) on the symmetric solution (bars); the mirror pairs read alike. (b) The button readings of **F1** and the divider reading of **F4** minus their predictions. The shaded bars show how far one resistor 5% high could move each button (15 to 28 mV; one 5% low moves them up to 29 mV) and how far 5% parts could move A1 (about 2.17 to 2.32 V). A reading inside its bar agrees with the analysis.

- Power in Network 1 (**E1**): 97.7 mW from the rail; R_1 and R_{12} run hottest at 18.3 mW each, then R_3 and R_{10} at 12.9 mW, all under 20% of a 0.1 W rating. One resistor 5% high moves a node by up to 27.5 mV (8.5 ADC steps); 1% by up to 5.6 mV (**E2**).
- Buttons (**F1**): within about 10 to 20 mV of the prediction, and all five within about 40 mV (Figure 1.47b). Because the grid and the ADC reference share the same 3.3 V supply, an error in the supply voltage cancels out of the counts.
- Multimeter (**F2**): within a few tens of millivolts of the symmetric solution (Figure 1.47a), for example 3.297, 2.196, 1.648, 2.201, 1.651, 1.098, 1.653, 1.101, and 0.000 V. The meter measures absolute volts, so if the rail (TP1) is a little off 3.3 V, every reading scales with it. Every node of Network 2 has a Thevenin resistance under 100 Ω, about 10^5 times smaller than the meter's 10 MΩ, so the meter does not disturb the grid.
- Divider (**F3** to **F5**): a prediction of $3.3 \times 10/14.7 = 2.245$ V. With 5% parts the reading can fall anywhere from about 2.17 to 2.32 V, and the top resistor computed from it can be up to about 10% from 4.7 kΩ, because it absorbs the tolerance of both parts.

Troubleshooting

- *"No unique solution" in **D1**.* A KCL equation (**C1**) or an entry of `EDGES` (**C2**) is wrong or missing. Print `kcl` and check each node against Figure 1.44.
- *Node voltages outside 0 to* 3.3 V. A current direction in `EDGES` does not match the KCL list. (If you reversed the same resistor in both places, the solution is still right; that current just comes out negative.)

- `your G matches the script:  False` *in* **D2**. Count again the resistors at each node, and leave out the neighbors S1 and S9: they have known voltages.
- *A button reads far from its prediction.* It was released before the reading finished; run **F1** again and hold until the value prints.
- *The multimeter reads about* 0 V *everywhere.* S10 is not on Module 1, so Network 2 is not powered, or the black probe is not on TP9. Negative readings mean the probes are swapped.
- *A1 is far from the prediction* (**F4** warns). Check that the caps of J7 and J8 are on the Module 1 end, that both resistors are fully seated, and that the 4.7 kΩ part is on top.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. There are nine KCL equations (**C1**) but only eight independent ones. Why? (Hint: add all nine.) Why does SymPy still find a unique solution?
2. What total power does Network 1 draw from the 3.3 V rail, and how is it shared among the resistors (**E1**)? Are the hottest ones within a 0.1 W rating?
3. How far are your button readings (**F1**) and your Lab 1.1 readings from the predictions, in millivolts and in ADC counts? How much of the difference could come from 1% or 5% resistor tolerance (**E2**)?
4. Did your hand-written **G** for Network 2 match the script's (**D2**)? What does each diagonal entry count, and why is **G** symmetric?
5. Which test points of Network 2 read the same voltage (**F2**), and why? From your readings, compute the current in the four resistors around TP5 with Ohm's law and check KCL at TP5. Why does a meter with 10 MΩ input resistance not disturb these readings?
6. What top resistance did you infer from A1 (**F5**)? Is it within the tolerance of the two parts? Why does the exact rail voltage not matter for the ADC count?

Summary

- Current is the flow of charge, $i = dQ/dt$, in amperes; voltage is energy per charge between two points, in volts; power is $P = VI$, and in a resistor $P = I^2R = V^2/R$.
- Ohm's law $V = IR$ relates them, and a uniform conductor has $R = \rho L/A$.
- Ideal elements are defined by their I-V laws: a wire fixes $V = 0$, an open circuit fixes $I = 0$, a voltage source fixes V, a current source fixes I. With the passive sign convention, the powers absorbed by all elements add to zero.
- KCL (charge conservation): the currents into a node sum to the currents out. KVL (energy conservation): voltages around a loop sum to zero. Find the nodes first.
- Series resistances add; parallel conductances add. Dividers: $V_2 = V_s R_2/(R_1 + R_2)$ and $I_1 = I_{\text{in}} R_2/(R_1 + R_2)$. A bridge balances when $R_1/R_2 = R_3/R_4$.
- Voltmeters go in parallel and need large R_{in}; ammeters go in series and need small R_{in}; ohmmeters need an isolated, unpowered element. A fuse is a series link that opens on overcurrent.
- Nodal analysis writes KCL at every unknown node in terms of node voltages, giving $\mathbf{Gv} = \mathbf{i}$, which `numpy` solves in one call.
- Any linear two-terminal network is equivalent to a source V_{th} in series with R_{th} (Thevenin), or to a current source $I_N = V_{\text{th}}/R_{\text{th}}$ in parallel with R_{th} (Norton). A load gets the full V_{th} only if $R_L \gg R_{\text{th}}$.
- Capacitors ($i = C\,dv/dt$, energy $\frac{1}{2}Cv^2$) are open circuits at DC; inductors ($v = L\,di/dt$, energy $\frac{1}{2}Li^2$) are short circuits at DC.
- A resistive touch panel is a grid of dividers. Making the horizontal and vertical resistors unequal gives each of the nine buttons a distinct voltage, and a decoder with thresholds halfway between levels names the button.

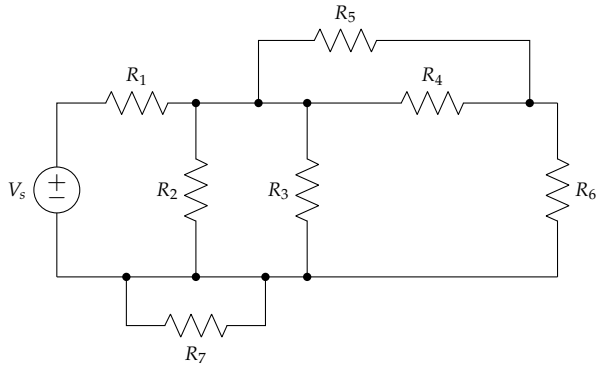
1.12 *Problems*

1.1 ◦ A phone battery is rated 3.7 V and 4000 mAh.

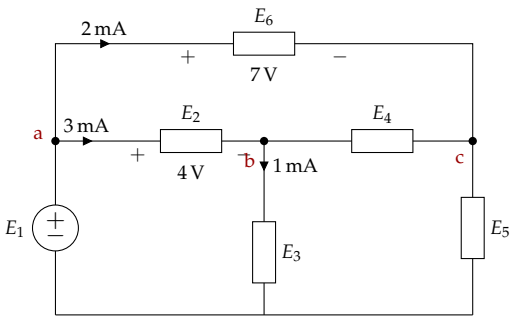
(a) How much charge, in coulombs, does it deliver from full to empty?

(b) How much energy does it store, in joules and in watt-hours?

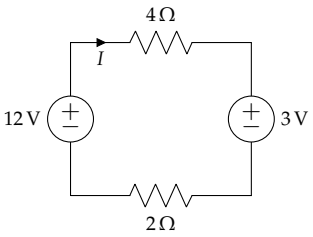
(c) How long does it last if the phone draws an average of 200 mA?



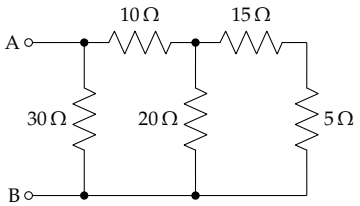
Problem 1.5



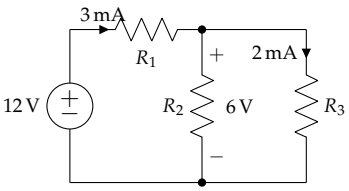
Problem 1.6



Problem 1.7



Problem 1.8



Problem 1.9

Figure 1.48: Circuits for Problems 1.5 to 1.9. Values not shown are what the problem asks you to find. In the circuit of Problem 1.6, each box is an unspecified element; the + sign of each given voltage is at the end where the given current enters.

1.2 ◦ A photodiode in a pulse sensor produces 2.5 µA.

(a) How many electrons per second is that?

(b) How much charge flows in ten minutes?

(c) An LED in the same sensor is driven with 50 mA pulses that last 2 ms, 100 times per second. What is the charge per pulse and the average LED current?

1.3 ◦ A copper trace on a circuit board is 35 µm thick, 0.25 mm wide, and 10 cm long ($\rho = 1.7 \times 10^{-8}\,\Omega\,\text{m}$). Find its resistance, the voltage it drops at 1 A, and the power it dissipates. What is its resistance if you make it twice as wide?

1.4 ◦ Explain each statement using the element definitions of Section 1.2 and Kirchhoff's laws.

(a) Two ideal voltage sources of different values cannot be connected in parallel.

(b) Two ideal current sources of different values cannot be connected in series.

(c) A shorted resistor carries no current.

(d) Adding a resistor in parallel with a network always lowers its equivalent resistance.

1.5 ◦ For the circuit of Problem 1.5 in Figure 1.48:

(a) How many nodes does it have? Which resistors are in parallel, and is any resistor shorted?

(b) With every resistor equal to $1\,\text{k}\Omega$ and $V_s = 6\,\text{V}$, find the current supplied by the source.

1.6 • In the circuit of Problem 1.6 in Figure 1.48, the source E_1 is 10 V, E_2 has 4 V and 3 mA, E_6 has 7 V and 2 mA, and E_3 carries 1 mA downward.

(a) Use KVL to find the voltages of E_3, E_4, and E_5 (each with $+$ at its upper or left end).

(b) Use KCL to find the currents in E_1, E_4 (rightward), and E_5 (downward).

(c) Find the power absorbed by each element and check that the total is zero. Which elements supply power?

1.7 • The loop of Problem 1.7 in Figure 1.48 holds a 12 V source, a 3 V source, and two resistors.

(a) Find the current I and the power absorbed by each of the four elements.

(b) Repeat with the 3 V source turned around.

(c) In which arrangement is the 3 V element being charged, if it is a rechargeable cell?

1.8 ◦ Find the equivalent resistance between terminals A and B of the network of Problem 1.8 in Figure 1.48.

1.9 • In the circuit of Problem 1.9 in Figure 1.48, the source supplies 3 mA, the voltage across R_2 is 6 V, and R_3 carries 2 mA. Find R_1, R_2, and R_3, and check your answer with KVL and KCL.

1.10 • You want to read a signal that ranges from 0 V to 12 V with an ADC whose input range is 0 V to 3.3 V.

(a) Design a voltage divider with a total resistance of $100\,\text{k}\Omega$ that maps 12 V to exactly 3.3 V.

(b) You only have standard values $75\,\text{k}\Omega$ and $27\,\text{k}\Omega$. What is the output at 12 V input, and what 10-bit count does it produce with a 3.3 V reference (Equation (1.16))?

(c) How much power does the divider draw from the 12 V signal?

1.11 ◦ A current of 30 mA enters three resistors in parallel: 100 Ω, 200 Ω, and 300 Ω. Find the voltage across them and the current in each.

1.12 • A 6 mA current source feeds three parallel branches: a 2 kΩ resistor; a 3 kΩ resistor; and a 1 kΩ resistor in series with a 5 kΩ resistor. Find the voltage across the current source, the current in each branch, the voltage across the 5 kΩ resistor, and the power the source delivers.

1.13 •• A sensor interface needs 2.0 V from a 5 V supply, and the next stage loads that output with 10 kΩ.

(a) Design an unloaded divider with $R_1 + R_2 = 10\,\text{k}\Omega$ (R_1 on top) that gives 2.0 V.

(b) What is the output when the 10 kΩ load is connected?

(c) Redesign the divider so that it still gives 2.0 V unloaded and loses no more than 5% when the load is connected. Find the largest allowed R_1 and the matching R_2, and compare the current drawn from the supply with that of part (a).

1.14 • A Wheatstone bridge (Figure 1.19) has $R_1 = 1.00\,\text{k}\Omega$ and $R_2 = 100\,\Omega$, and the unknown is R_4.

(a) The bridge balances with the decade box R_3 at 1234 Ω. Find R_4.

(b) Your decade box goes up to 11 110 Ω. What is the smallest ratio R_2/R_1 that lets you balance the bridge with a 47 kΩ unknown? Pick a convenient power of ten.

(c) Does the supply voltage affect the balance point? Does the resistance of the meter across the outputs?

1.15 • A strain gauge is placed in the bridge of Figure 1.19 with $V_s = 5\,\text{V}$ and $R_1 = R_2 = R_3 = 350\,\Omega$. The gauge is R_4; it reads 350 Ω unstrained and 350.7 Ω under load.

(a) Find $V_x - V_y$ unstrained and under load.

(b) Show that for a small change ΔR in R_4 (all four arms equal to R), $V_x - V_y \approx -\frac{V_s}{4}\,\Delta R/R$.

(c) A single divider (R_3 over R_4) also changes by about the same amount. Why is the bridge still the better way to read the gauge with a 10-bit, 3.3 V ADC?

1.16 ◦ In a powered circuit, a 5 V supply drives R_1 in series with the parallel pair R_2 and R_3.

(a) Describe how to connect a multimeter to measure the voltage across R_2.

(b) Describe how to measure the current through R_2 alone.

(c) Describe how to measure the resistance of R_2.

(d) What happens if, with the meter set to measure current, you touch the probes across R_2 as if measuring voltage?

1.17 • A 10 V source drives two 1 MΩ resistors in series, and you measure the voltage across the bottom one.

(a) What should the reading be?

(b) What does a meter with $R_{\text{in}} = 10\,\text{M}\Omega$ read, and what is the percent error?

(c) What input resistance would keep the error under 1%?

(d) What would a meter with $R_{\text{in}} = 1\,\text{M}\Omega$ read?

1.18 • A 5 V supply drives a 50 Ω load. An ammeter with 1 Ω internal resistance is placed in series to measure the current.

(a) Find the true and the measured current, and the percent error.

(b) A student instead connects the same ammeter directly across the 5 V supply. What current would flow if the supply could deliver it, and how much power would the meter's shunt dissipate? What protects the meter?

1.19 • You build an ammeter from a shunt resistor and a voltmeter whose most sensitive range is 100 mV full scale.

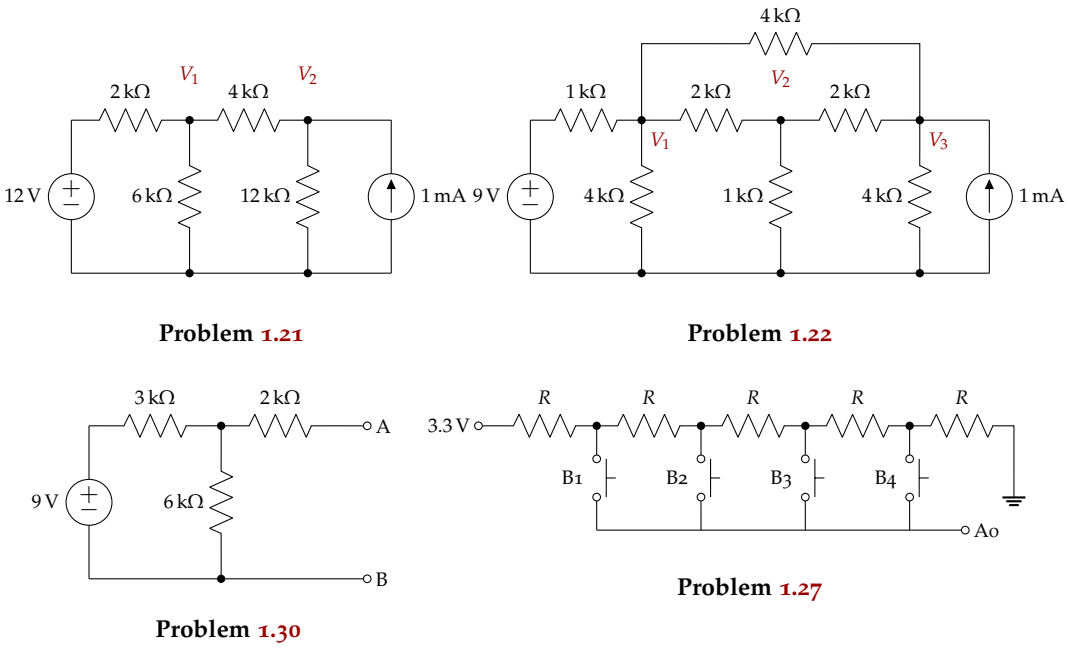
(a) Choose the shunt so that 2 A gives a full-scale reading.

(b) How much power does the shunt dissipate at 2 A?

(c) The shunt measures the current of a 2.5 Ω load on a 5 V supply. What current flows with the shunt in place, and what is the error compared with the current without it?

1.20 ◦ A 5 V USB supply protected by a 1.6 A fuse powers LED strips connected in parallel; each strip behaves like a 20 Ω resistor. How many strips can you connect before the fuse opens? What current flows with one strip too many?

1.21 • For the circuit of Problem 1.21 in Figure 1.49, write the conductance matrix $\mathbf{G}$ and source vector $\mathbf{i}$, solve for V_1 and V_2, and



Problem 1.21

Problem 1.22

Problem 1.27

Problem 1.30

Figure 1.49: Circuits for Problems 1.21, 1.22, 1.27, and 1.30.

find the current in the 4 kΩ resistor, including its direction. Check your answer with `numpy`.

1.22 • For the three-node circuit of Problem 1.22 in Figure 1.49:

(a) Write **G** (in mS) and **i** (in mA) by inspection. Why does G_{13} not vanish?

(b) Solve with `np.linalg.solve`.

(c) Check KCL at node 2 with your numbers.

1.23 •• Return to the bridge of Figure 1.19 with $V_s = 5\,\text{V}$, $R_1 = R_2 = R_3 = 1\,\text{k}\Omega$, and $R_4 = 1.1\,\text{k}\Omega$, and now connect a meter with $500\,\Omega$ of resistance from V_x to V_y.

(a) Find the open-circuit voltage $V_x - V_y$ (meter disconnected).

(b) Find the Thevenin resistance seen by the meter. (Hint: with the source off, the top of the bridge is grounded.)

(c) Find the meter current and its direction.

(d) Check part (c) by nodal analysis with two unknowns in Python.

1.24 ◦ A continuous resistive touch screen is 80 mm wide. Its x film is driven with 3.3 V across its width, and the other film is read by a 10-bit ADC with a 3.3 V reference.

(a) What voltage is read for a touch 20 mm from the grounded edge?

(b) The ADC reports 2.00 V. Where is the finger?

(c) What is the position resolution, in millimeters per ADC count?

1.25 • A Network 1 touch panel with the nominal levels of Figure 1.34 reports these 10-bit readings (convert with Equation (1.16)). For each, name the button the halfway-threshold decoder reports and the distance from that button's level. Which readings should make you suspicious, and what might cause them? (a) 612 counts; (b) 455 counts; (c) 780 counts; (d) 120 counts.

1.26 • In Part F of Lab 1.2 you plug a 4.7 kΩ resistor into sockets J1 and J2 (top) and a 10 kΩ resistor into sockets J3 and J4 (bottom) of the divider header (Figure 1.46).

(a) Predict the A1 voltage and its 10-bit count (Equation (1.16)).

(b) With ±5% resistors, what range of A1 voltages is possible?

(c) How much current and power does the divider draw from the rail?

(d) The rail actually measures 3.25 V. Does the count change? Why or why not?

1.27 •• *Demo board design.* A simpler keypad (Figure 1.49) uses five equal resistors R in series from 3.3 V to ground, with four buttons, B1 to B4, each connecting one internal tap to A0.

(a) Find the four tap voltages and, with Equation (1.16), their 10-bit counts.

(b) The resistors have ±5% tolerance. Find the largest possible shift of any tap voltage, and decide whether detection windows of ±100 counts around each nominal count are safe.

(c) With 1 kΩ resistors, how much current and power does the ladder draw? Why might you choose 10 kΩ instead, and what would limit how large you can go?

1.28 •• *A broken resistor.* On one board, resistor R_4 of Network 1 (between S2 and S5) has cracked open.

(a) Remove R_4 from the `EDGES` table of Lab 1.2 (or from the `numpy` listing of Section 1.11) and find the nine node voltages.

(b) What is the smallest gap between two levels now, in volts and counts?

(c) A decoder still uses the thresholds of Figure 1.34, built from the intact board. Which buttons does it misread, and as what?

(d) How does the measured-level decoder of Lab 1.1 avoid this problem?

1.29 •• *Python design study.* Use the `nodal_system` code of Lab 1.2 with a resistor table in which the six horizontal resistors of the grid have a value R_a while the vertical ones stay at $100\,\Omega$. For R_a from $10\,\Omega$ to $100\,\Omega$, compute the smallest voltage gap between any two of the nine buttons.

(a) Plot the smallest gap against R_a. Which R_a maximizes it, and what is the gap in volts and in 10-bit counts?

(b) How does the board's choice of $50\,\Omega$ compare?

(c) What goes wrong near $R_a = 60\,\Omega$?

1.30 •• For the circuit of Problem 1.30 in Figure 1.49, with terminals A and B:

(a) Find V_{th}, R_{th}, and I_N.

(b) Find the voltage across a $4\,\text{k}\Omega$ load.

(c) The power in a load R_L is $P_L = V_{\text{th}}^2 R_L/(R_{\text{th}} + R_L)^2$. Show that it is largest when $R_L = R_{\text{th}}$ and find that maximum power.

1.31 • A black-box power source reads 4.80 V on a $10\,\text{M}\Omega$ voltmeter. With a $1.00\,\text{k}\Omega$ resistor across its terminals, the reading is 4.00 V.

(a) Find its Thevenin resistance and Norton current.

(b) Predict the terminal voltage with a $500\,\Omega$ load.

(c) Which load resistance receives the most power, and how much?

1.32 ◦ Convert each source to the other form.

(a) A 5 mA current source in parallel with $2\,\text{k}\Omega$: find the Thevenin form.

(b) A 9 V source in series with $3\,\text{k}\Omega$: find the Norton form.

(c) Connect a $1\,\text{k}\Omega$ load to the source of part (a). What current flows in it?

2
Semiconductor Devices

In this chapter you will learn to

- Relate resistivity to carrier density and mobility, and compute the resistance of a piece of doped silicon.
- Describe electrons, holes, donors, and acceptors using both the bond model and the band picture.
- Convert between band gap and photon wavelength, and predict which materials emit or absorb a given color.
- Explain how a pn junction forms a depletion region and why it conducts in only one direction.
- Analyze diode circuits with the diode equation and with the ideal and constant-drop models.
- Find the operating region of a MOSFET, compute its drain current, and use it as a switch, an amplifier, and a CMOS inverter.
- Explain how LEDs, photodiodes, and solar cells convert between electrical and optical energy, and size their drive and readout circuits.
- Use the two ideal op-amp rules to analyze the follower, difference, and transimpedance amplifiers that read currents on the demo board, and check that a single-supply op-amp can produce the output a circuit asks for.
- Measure diode and MOSFET I-V curves on the demo board with the DAC, the Arduino, and Python.

Turn the module selector of your demo board to Module 2 and look at the devices block that it powers (upper right on the board): a small diode in series with a 1 kΩ resistor, a MOSFET with its own

1 kΩ resistor, a digital-to-analog converter (DAC) that can sweep a voltage, and two op-amp circuits that turn currents into voltages the Arduino can read. Every part you met in Chapter 1 obeyed Ohm's law: double the voltage and the current doubles. None of these new parts do. The diode passes almost nothing until about 0.6 V and then lets the current climb steeply. The MOSFET has a third terminal that decides how much current flows between the other two. Elsewhere on the board, in the biosensing block, LEDs turn current into light and photodiodes turn light back into current; in Chapter 3 such a pair measures your pulse, and in Chapter 5 another carries data across the room.

All of these parts, and the microcontroller on the Arduino, are made of semiconductors. This chapter builds from the material to the pn junction and diode, then the transistor, then the devices that emit and detect light. A short section on op-amps then explains how the board turns device currents into voltages, and the chapter ends with a lab in which you sweep the diode and the MOSFET on your own board.

2.1 *Tunable Resistance: Conductors, Insulators, and Semiconductors*

Chapter 1 gave you the resistance of a uniform bar of length L and cross-sectional area A:

$$R = \frac{\rho L}{A}, \tag{2.1}$$

where ρ is the resistivity of the material in Ω cm. If you want a resistor whose value you can change, Equation (2.1) offers three knobs. You can stretch the bar (L), thicken it (A), or change the material itself (ρ). The first two are mechanical and slow. The third is what semiconductors give us: their resistivity can be set chemically, when the material is made, and changed electrically or optically while the circuit runs. That single property is the reason this chapter exists.

WHY DOES RESISTIVITY DEPEND ON THE MATERIAL? Current is moving charge. Suppose a material holds n mobile electrons per cubic centimeter and an electric field $\mathcal{E}$ pushes them along at an average drift velocity v. In one second, every electron within a distance v of a cross section crosses it, so the current density is $J = qnv$, where $q = 1.602 \times 10^{-19}$ C. In most materials the drift velocity is proportional to the field, $v = \mu\mathcal{E}$, and the constant μ is called the *mobility*, in $\mathrm{cm}^2/(\mathrm{V\,s})$. Combining the two gives $J = qn\mu\mathcal{E}$. Ohm's law in this form reads $J = \sigma\mathcal{E}$, so the conductivity is $\sigma = qn\mu$. A semiconductor has two kinds of mobile carrier, electrons (density n) and holes

(density p, introduced in the next section), and both contribute:

$$\sigma = \frac{1}{\rho} = q\left(n\mu_n + p\mu_p\right). \tag{2.2}$$

Mobility is a property of the material and changes only modestly. In lightly doped silicon, $\mu_n \approx 1350\,\mathrm{cm^2/(V\,s)}$ for electrons and $\mu_p \approx 480\,\mathrm{cm^2/(V\,s)}$ for holes. The carrier density, on the other hand, ranges over an almost unimaginable span, and that is what separates the three families of materials in Table 2.1.

KEY IDEA Resistivity is set by how many mobile carriers a material has and how easily they move. Semiconductors let you choose the first number.

Material class	Mobile carriers ($\mathrm{cm^{-3}}$)	Example
Conductor (metal)	10^{22} to 10^{23}, fixed	copper, 8.5×10^{22}
Semiconductor	10^{10} to 10^{20}, adjustable	silicon
Insulator	about 10^{-57}, effectively zero	silicon dioxide

Table 2.1: Density of mobile charge carriers in the three classes of material. Only semiconductors let the designer pick the value.

In a metal every atom donates roughly one electron to a sea of mobile charge, so n is huge and cannot be changed. In an insulator essentially every electron is locked in a chemical bond. A semiconductor sits between the two, and the important word in the middle row of the table is *adjustable*. Figure 2.1 places real materials on a resistivity scale: the same silicon crystal can sit near the insulators or near the metals depending on what has been added to it.

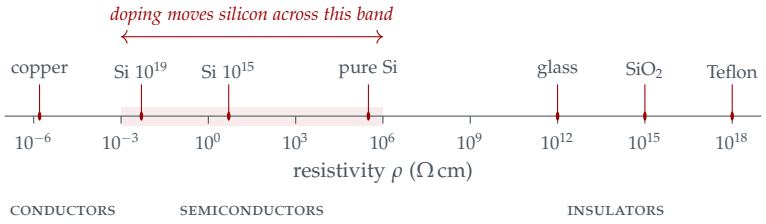


Figure 2.1: Resistivity of common materials at room temperature on a logarithmic scale. "Si 10^{19}" means silicon doped with 10^{19} atoms per $\mathrm{cm^3}$. Doping lets one silicon crystal span about nine decades, from nearly insulating to nearly metallic. Values for glass, SiO_2, and Teflon are typical orders of magnitude.

WHAT CAN YOU BUILD from an element whose resistance you can tune, or whose resistance depends on the voltage across it? There are four answers, and together they cover most of electronics. A resistance that you can switch between very high and very low is a *switch*, the basis of digital logic. A resistance that a small signal can modulate lets a small voltage control a large current, which is an *amplifier*. A resistance that is low for one polarity and high for the other *rectifies* alternating current into direct current, the first step of every power supply. And a resistance that responds to light, heat, or chemistry is a *sensor*; run some of the same structures backward and they generate power or emit light.

WORKED EXAMPLE 2.1 *Materials and their carrier densities*

What are the different types of materials used in electronics, and what are their charge carrier densities?

SOLUTION. Electronics uses three classes of material, distinguished by how many mobile charge carriers they hold (Table 2.1).

Conductors (metals such as copper, aluminum, and gold) have about 10^{22} to 10^{23} free electrons per cm^3, roughly one per atom. Copper has $8.5 \times 10^{22}\,cm^{-3}$. They make wires, traces, and contacts.

Insulators (silicon dioxide, glass, plastics, Teflon) have their electrons locked in bonds by a large energy gap. Their equilibrium density of mobile carriers is about $10^{-57}\,cm^{-3}$, which means zero for any practical purpose. (Section 2.3 shows where this absurdly small number comes from.) They isolate conductors and form the gate oxide of a transistor.

Semiconductors (silicon, germanium, and gallium arsenide) have about $10^{10}\,cm^{-3}$ carriers when pure, and doping raises that anywhere up to about $10^{20}\,cm^{-3}$. Because the density can be chosen, they make diodes, transistors, LEDs, and sensors. ■

2.2 *Silicon, Electrons, and Holes*

SEMICONDUCTORS LIVE IN A NARROW BAND of the periodic table. Silicon, germanium, and carbon (as diamond) sit in group IV and have four valence electrons. Combining a group III element with a group V element gives compound semiconductors such as gallium arsenide (GaAs), gallium nitride (GaN), and gallium phosphide (GaP), and mixing three or four elements gives alloys like AlGaAs and InGaN whose properties can be tuned by composition. Compounds dominate optoelectronics, as you will see in Section 2.9. Silicon dominates everything else.

Silicon has atomic number 14. Its outer shell holds four electrons, and in a crystal each atom shares one electron with each of four neighbors, forming four covalent bonds arranged as a tetrahedron. The resulting *diamond cubic* lattice repeats every 5.43 Å (the side of the cubic unit cell) and packs 5×10^{22} atoms into each cubic centimeter. Figure 2.2 shows the crystal with two impurity atoms that we will need shortly.

THE BOND MODEL flattens this three-dimensional crystal onto the page (Figure 2.3a). Each line between two atoms is a bond, and each bond holds two shared electrons. At absolute zero every electron sits in a bond, nothing is free to move, and pure silicon is an insulator. At room

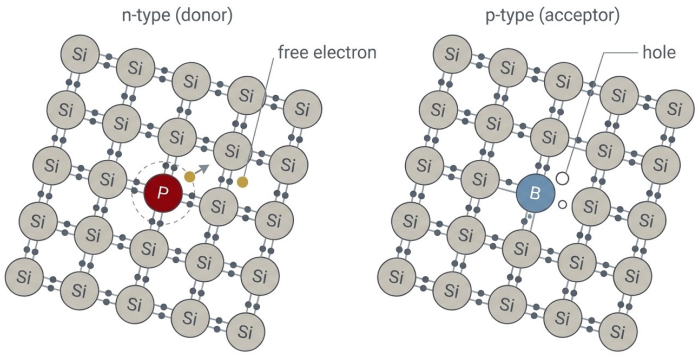
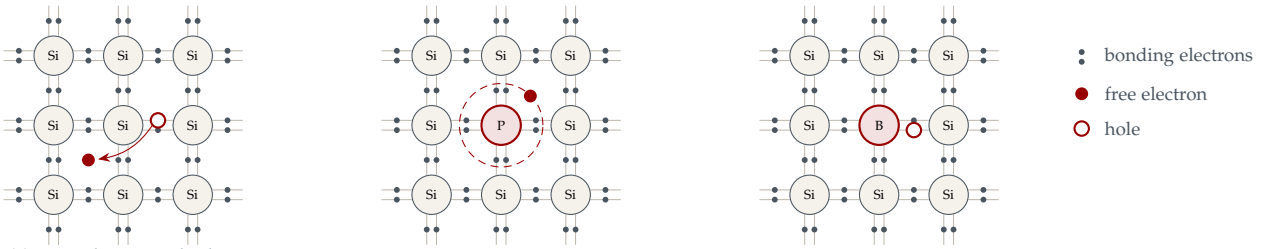


Figure 2.2: A silicon crystal. Each atom bonds to four neighbors. A phosphorus atom (a donor) brings one more electron than its bonds need, and that electron wanders off; a boron atom (an acceptor) brings one fewer, leaving a hole.



(a) pure silicon: one broken bond, one electron, one hole

(b) n-type: phosphorus donor adds a fifth electron

(c) p-type: boron acceptor leaves a hole

Figure 2.3: The bond model of silicon. (a) Thermal energy breaks a bond and makes an electron-hole pair. (b) A group V donor such as phosphorus or arsenic has five valence electrons; four fill bonds and the fifth is loosely held (dashed orbit) and easily freed, leaving a fixed P^+ ion. (c) A group III acceptor such as boron has only three; the empty bond captures an electron from a neighbor, creating a mobile hole and a fixed B^- ion.

temperature the lattice vibrates, and once in a while a bond shakes loose one of its electrons. That electron is now free to wander through the crystal: it is a *conduction electron*. The broken bond it left behind is missing an electron, and a neighboring bonding electron can hop into that vacancy, which moves the vacancy one bond over. The vacancy, called a *hole*, therefore moves too, and since it marks a place where negative charge is missing, it behaves exactly like a positive carrier of charge $+q$. Breaking one bond creates one electron and one hole, so in pure silicon the two densities are equal.

How many bonds are broken? In pure (*intrinsic*) silicon at 300 K, the density of electrons equals the density of holes, and both equal the *intrinsic carrier concentration*

$$n = p = n_i \approx 1 \times 10^{10}\,\mathrm{cm}^{-3} \qquad \text{(silicon, 300 K)}. \tag{2.3}$$

(You will also see 1.5×10^{10} quoted; the exact figure depends on the model used. We use 10^{10}.) Compare that with 5×10^{22} atoms per cm^3: only about one atom in five trillion has lost an electron. That is why pure silicon is a poor conductor. The number climbs steeply with temperature (Figure 2.4), because more lattice vibration breaks more bonds, which is why semiconductor devices leak more current when hot.

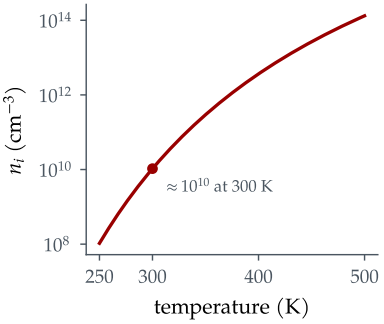


Figure 2.4: Intrinsic carrier concentration of silicon versus temperature, $n_i \propto T^{3/2} e^{-E_g/2kT}$. Heating from 300 K to 400 K raises n_i by a factor of about 350.

Electrons and holes are created in pairs, and they also disappear in pairs: when a free electron meets a hole it can fall back into the bond, a process called *recombination*. Generation and recombination balance in equilibrium, and the balance has a simple consequence known as the *mass action law*:

$$n\,p = n_i^2. \tag{2.4}$$

It holds whether or not the crystal is doped. If you raise one density, the other must fall so that the product stays at $n_i^2 = 10^{20}\,\mathrm{cm}^{-6}$.

Doping is how we choose the carrier density. Replace a few silicon atoms with phosphorus or arsenic from group V (Figure 2.3b). Each has five valence electrons. Four fill the bonds to its neighbors, and the fifth is held so weakly (about 0.045 eV for phosphorus) that room temperature frees it almost immediately. The impurity is called a *donor*, it leaves behind a positive ion fixed in the lattice, and the silicon becomes *n-type* because its mobile carriers are mostly negative electrons. With a donor density $N_D \gg n_i$,

$$n \approx N_D, \qquad p = \frac{n_i^2}{N_D}. \tag{2.5}$$

Replace silicon atoms with boron from group III instead (Figure 2.3c). Boron has only three valence electrons, so one bond is incomplete. It readily accepts an electron from a neighboring bond, which creates a mobile hole and leaves the boron as a fixed negative ion. Boron is an *acceptor*, and the silicon becomes *p-type*, with $p \approx N_A$ and $n = n_i^2/N_A$. Donors and acceptors together are called *dopants*. The more numerous carrier (electrons in n-type, holes in p-type) is the *majority carrier*; the other is the *minority carrier*.

The amounts involved are tiny. A doping of $10^{17}\,\mathrm{cm}^{-3}$ replaces two silicon atoms in every million, yet it multiplies the electron density by ten million. The same rules apply to compound semiconductors. In GaAs, a group VI atom (selenium or tellurium) on an arsenic site has one extra electron and acts as a donor; a group II atom (zinc or beryllium) on a gallium site is one electron short and acts as an acceptor. Silicon, from group IV, can play either role depending on which site it occupies.

Worked Example 2.2 *Mass action in n-type silicon*

Silicon at 300 K is doped with $N_D = 10^{16}$ phosphorus atoms per cm^3. Find the electron and hole densities and the factor by which each changed from the intrinsic value.

Solution. The donors are all ionized at room temperature, and

$N_D \gg n_i$, so $n \approx N_D = 10^{16}\,\mathrm{cm}^{-3}$. From the mass action law,

$$p = \frac{n_i^2}{n} = \frac{(10^{10})^2}{10^{16}} = 1 \times 10^4\,\mathrm{cm}^{-3}.$$

The electron density rose by a factor of $10^{16}/10^{10} = 10^6$, and the hole density fell by the same factor. Holes are now so rare that they play no part in conduction: this is n-type silicon with electrons as the majority carrier. ■

A crystal can hold both kinds of dopant at once. The donors' extra electrons then first fill the empty bonds of the acceptors, and only the excess is free to move. If $N_D > N_A$ the material is n-type with $n \approx N_D - N_A$; if $N_A > N_D$ it is p-type with $p \approx N_A - N_D$. This *compensation* is how manufacturers turn a region of one type into the other: the n^+ source and drain of a transistor are made by adding many more donors to the p-type body than it has acceptors.

Worked Example 2.3 *Compensated silicon*

A p-type wafer has $N_A = 10^{16}\,\mathrm{cm}^{-3}$. One region of it then receives a phosphorus doping of $N_D = 3 \times 10^{16}\,\mathrm{cm}^{-3}$. What type is that region, and what are its electron and hole densities?

Solution. The donors outnumber the acceptors, so the region becomes n-type. Each acceptor captures one donor electron, leaving

$$n \approx N_D - N_A = 3 \times 10^{16} - 1 \times 10^{16} = 2 \times 10^{16}\,\mathrm{cm}^{-3}.$$

The mass action law still holds: $p = n_i^2/n = 10^{20}/(2 \times 10^{16}) = 5 \times 10^3\,\mathrm{cm}^{-3}$. Only two thirds of the donor electrons are free; the rest neutralize the acceptors. The junction between this region and the untouched p-type wafer around it is a pn junction, the subject of Section 2.4. ■

Doping changes resistivity through Equation (2.2). Figure 2.5 shows the measured relation for silicon. The curves are nearly straight lines of slope -1 on log axes, because ρ is inversely proportional to the dopant density. They bend at high doping, where the impurity ions scatter the carriers and the mobility falls (to about $750\,\mathrm{cm}^2/(\mathrm{V\,s})$ for electrons at $10^{17}\,\mathrm{cm}^{-3}$ and below $300\,\mathrm{cm}^2/(\mathrm{V\,s})$ at 10^{18}). At every doping level p-type silicon has the higher resistivity, because holes are slower than electrons.

Worked Example 2.4 *Tuning a resistor with doping*

A silicon bar is 1 cm long, and its cross section is $1\,\mathrm{mm}^2$, which is

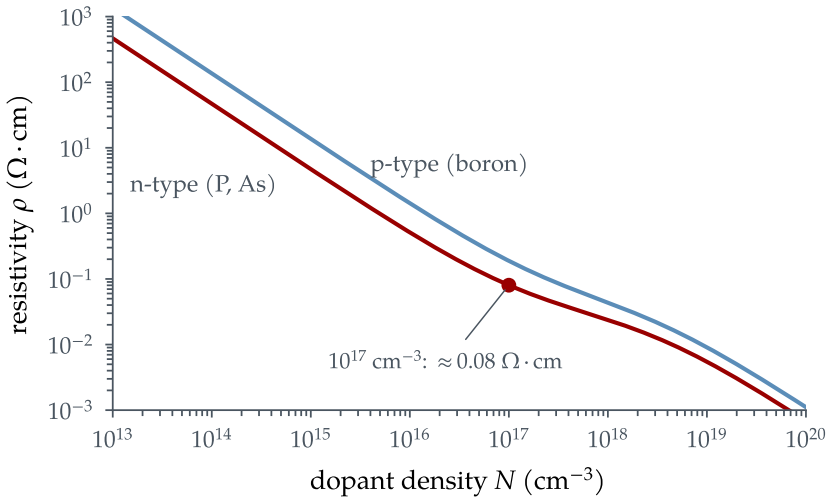


Figure 2.5: Resistivity of silicon at 300 K versus dopant density, computed from an empirical fit to measured mobilities. The marked point is the nanowire example below: arsenic at $10^{17}\,\mathrm{cm}^{-3}$ gives about $0.08\,\Omega\,\mathrm{cm}$.

$0.01\,\mathrm{cm}^2$. Find its resistivity and resistance when it is (a) intrinsic, (b) doped n-type with $N_D = 10^{15}\,\mathrm{cm}^{-3}$, and (c) doped p-type with $N_A = 10^{18}\,\mathrm{cm}^{-3}$. Use $\mu_n = 1350$ and $\mu_p = 480\,\mathrm{cm}^2/(\mathrm{V\,s})$ for light doping, and $\mu_p = 200\,\mathrm{cm}^2/(\mathrm{V\,s})$ at $10^{18}\,\mathrm{cm}^{-3}$.

Solution. In all three cases the geometry factor is

$$L/A = 1\,\mathrm{cm}/0.01\,\mathrm{cm}^2 = 100\,\mathrm{cm}^{-1}.$$

(a) Intrinsic: $n = p = 10^{10}\,\mathrm{cm}^{-3}$ and both carriers count:

$$\sigma = q n_i(\mu_n + \mu_p) = (1.602 \times 10^{-19})(10^{10})(1830),$$

which is $2.93 \times 10^{-6}\,\mathrm{S/cm}$. So $\rho = 3.41 \times 10^{5}\,\Omega\,\mathrm{cm}$ and $R = \rho L/A = 3.41 \times 10^{7}\,\Omega$, about $34\,\mathrm{M\Omega}$.

(b) $N_D = 10^{15}$: $n = 10^{15}$ and $p = 10^{5}$, so holes are negligible. $\sigma = qn\mu_n = (1.602 \times 10^{-19})(10^{15})(1350) = 0.216\,\mathrm{S/cm}$, giving $\rho = 4.62\,\Omega\,\mathrm{cm}$ and $R = 462\,\Omega$.

(c) $N_A = 10^{18}$: $p = 10^{18}$ and $n = 10^{2}$. $\sigma = qp\mu_p = (1.602 \times 10^{-19})(10^{18})(200) = 32.0\,\mathrm{S/cm}$, giving $\rho = 0.0312\,\Omega\,\mathrm{cm}$ and $R = 3.12\,\Omega$.

The same piece of silicon spans more than seven decades of resistance, from $34\,\mathrm{M\Omega}$ to $3\,\Omega$, set entirely by at most 20 parts per million of impurity. ■

Worked Example 2.5 *Resistance of a silicon nanowire*

(a) What is the resistivity of silicon doped with $10^{17}\,\mathrm{cm}^{-3}$ of arsenic? (b) What is the resistance of a wire of this material $1\,\mu\mathrm{m}$

long with a cross section of $0.1\,\mu m^2$?

SOLUTION. (a) A published resistivity chart gives $\rho \approx 0.084\,\Omega\,cm$ at $10^{17}\,cm^{-3}$; the smooth fit in Figure 2.5 reads about $0.08\,\Omega\,cm$, the same within the accuracy of reading a log plot. Notice that $\rho = 1/(qN_D\mu_n)$ with $\mu_n = 1350$ would give only $0.046\,\Omega\,cm$; the measured value is higher because the mobility has dropped to about $740\,cm^2/(V\,s)$ at this doping.

(b) Convert to centimeters first: $L = 1\,\mu m = 10^{-4}\,cm$ and $A = 0.1 \times (10^{-4}\,cm)^2 = 10^{-9}\,cm^2$. Then

$$R = \frac{\rho L}{A} = \frac{(0.084)(10^{-4})}{10^{-9}} = 8.4 \times 10^3\,\Omega = 8.4\,k\Omega.$$

Small cross sections make even well-doped silicon surprisingly resistive. Unit slips are the most common error here: one square micrometer is $10^{-8}\,cm^2$, not 10^{-4}. ■

WORKED EXAMPLE 2.6 *Why silicon?*

Why is silicon the material of choice for electronics?

SOLUTION. Several properties come together in one element. *Its band gap is moderate* (1.12 eV): large enough that few carriers are generated thermally, so junctions leak little at room temperature and keep working well above 100 °C, yet small enough that doping and modest voltages control conduction. *It can be doped both ways*, n-type and p-type, over a range from 10^{13} to $10^{20}\,cm^{-3}$, which makes junctions and complementary transistors possible. *It grows an excellent native oxide.* Heating silicon in oxygen produces SiO_2, a high-quality insulator with a clean interface to the crystal. That oxide is the gate insulator of the MOSFET and the mask used during manufacturing; no other common semiconductor has an oxide this good. *It is abundant and manufacturable.* Silicon is the second most common element in Earth's crust (sand is SiO_2), and it can be grown as nearly perfect single crystals 300 mm in diameter. Finally, it is mechanically strong and conducts heat well, which matters when billions of transistors share one chip. Its main weakness, as Section 2.9 explains, is that it emits light very poorly. ■

WORKED EXAMPLE 2.7 *Bond models of doped silicon*

Draw the atomic (bond) models of n-type and p-type semiconductors.

SOLUTION. Figure 2.3(b) and (c) are the required drawings. Draw a square grid of Si atoms with two electrons (two dots) on each

bond. For *n-type*, replace one Si with a group V atom (P or As). Its four bonds are complete, and a fifth electron, drawn loosely bound near the dopant, is free to move; the dopant itself becomes a fixed positive ion. For *p-type*, replace one Si with a group III atom (B). One of its four bonds has only one electron; mark the missing electron as a hole (an open circle). When a neighboring electron fills it, the hole moves away and the boron becomes a fixed negative ion. In both drawings, the dopant atom stays put; only the electron or the hole is mobile. ■

2.3 *Energy Bands and the Band Gap*

The bond model is good for counting carriers. For anything involving energy, and especially light, we need a second picture. An isolated silicon atom has sharp, discrete energy levels. When 5×10^{22} atoms per cm^3 pack together into a crystal, their outer orbitals overlap, and each sharp level spreads into a *band* of closely spaced allowed energies. Two bands matter. The *valence band* holds the electrons that sit in bonds. The *conduction band* holds electrons that have broken free and can move through the crystal. Between them lies a range of energies where no electron states exist at all: the *band gap*.

A band diagram (Figure 2.6) draws only the two edges that face each other across the gap: the bottom of the conduction band, E_c, and the top of the valence band, E_v. Their difference

$$E_g = E_c - E_v \tag{2.6}$$

is the minimum energy needed to free a bonding electron. Energies at this scale are measured in *electron-volts*: 1 eV is the energy an electron gains moving through 1 V, or 1.602×10^{-19} J. For silicon, $E_g = 1.12\,\text{eV}$.

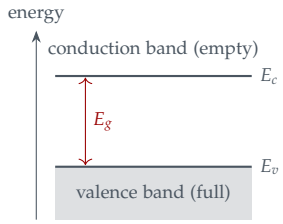


Figure 2.6: Energy band diagram. The conduction band edge E_c and valence band edge E_v are separated by the band gap E_g.

Translating the bond model into this picture is direct. Breaking a bond lifts an electron from the valence band into the conduction band, and the empty state it leaves in the valence band is the hole (Figure 2.7). Electrons, being negative, fall to the lowest available energy, so they collect near the bottom of the conduction band. Holes do the opposite and float to the top of the valence band. A donor creates an electron state just below E_c; its electron needs very little energy to hop up into the conduction band. An acceptor creates an empty state just above E_v, which a valence electron easily fills, leaving a hole behind.

The band gap sorts materials into the three classes of Figure 2.8. A completely full band cannot carry current, for the same reason that water in a completely full, sealed bottle cannot slosh: there is no empty

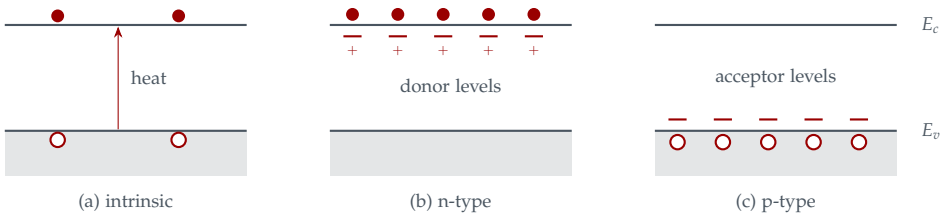


Figure 2.7: Carriers in the band picture. (a) Thermal energy lifts a few electrons across the gap, leaving equal numbers of holes. (b) Donor levels sit just below E_c; their electrons enter the conduction band and leave fixed positive ions. (c) Acceptor levels sit just above E_v; they capture valence electrons and leave holes in the valence band.

state for an electron to move into. A completely empty band cannot carry current either, because it has no electrons. In a metal the highest occupied band is only half full, so electrons can always shift into neighboring empty states, and conduction is easy. In an insulator such as SiO_2 the valence band is full, the conduction band is empty, and the gap (about 9 eV) is far too wide for heat to bridge. A semiconductor has the same full-and-empty arrangement but a modest gap, so a few electrons are thermally excited across it, and doping can add many more.

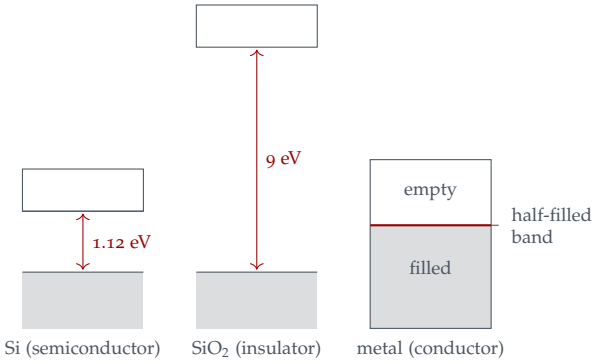


Figure 2.8: Band pictures of a semiconductor, an insulator, and a metal. Shaded regions are filled with electrons. Full bands and empty bands do not conduct; the half-filled band of a metal does.

The width of the gap enters the carrier density exponentially, $n_i \propto e^{-E_g/2kT}$, where $kT = 0.0259\,\text{eV}$ at 300 K. For silicon the exponent is $-1.12/0.0518 \approx -21.6$; for a 9 eV insulator it is -174. That difference of 152 in the exponent is a factor of about 10^{66}. Multiplying silicon's $10^{10}\,\text{cm}^{-3}$ by 10^{-66} lands near 10^{-56}, the same absurdly small order as the $10^{-57}\,\text{cm}^{-3}$ of Table 2.1.

Worked Example 2.8 *How fast n_i grows with temperature*

With its weaker temperature factor included, the intrinsic density is $n_i \propto T^{3/2} e^{-E_g/2kT}$, where Boltzmann's constant is $k = 8.617 \times 10^{-5}\,\text{eV/K}$. By what factor does n_i of silicon rise between 300 K (27 °C) and 350 K (77 °C, a car dashboard in summer)? Treat $E_g = 1.12\,\text{eV}$ as constant.

SOLUTION. Take the ratio, so that the unknown constant cancels:

$$\frac{n_i(350)}{n_i(300)} = \left(\frac{350}{300}\right)^{3/2} \exp\left[\frac{E_g}{2k}\left(\frac{1}{300} - \frac{1}{350}\right)\right].$$

Here

$$\frac{E_g}{2k} = \frac{1.12}{2 \times 8.617 \times 10^{-5}} = 6499\,\mathrm{K}$$

and $1/300 - 1/350 = 4.762 \times 10^{-4}\,\mathrm{K}^{-1}$, so the exponent is 3.095 and the exponential is 22.1. The prefactor is $(1.167)^{3/2} = 1.260$. The ratio is $1.260 \times 22.1 = 27.8$, so n_i rises from 10^{10} to about $2.8 \times 10^{11}\,\mathrm{cm}^{-3}$. The exponential does almost all of the work. The same calculation from 300 to 400 K gives 346, the factor of about 350 quoted in Figure 2.4. A diode's reverse saturation current is proportional to n_i^2 (Section 2.5), so it grows by about $27.8^2 \approx 770$ over the same 50 K. ■

LIGHT CONNECTS TO THE BAND GAP through the photon energy. A photon of wavelength λ carries energy $E = h\nu = hc/\lambda$, where h is Planck's constant and c is the speed of light. With energy in eV and wavelength in nanometers, the constants combine into one convenient number:

$$E\,[\mathrm{eV}] = \frac{hc}{\lambda} = \frac{1240}{\lambda\,[\mathrm{nm}]}. \tag{2.7}$$

KEY EQUATION
$E\,[\mathrm{eV}] = \dfrac{1240}{\lambda\,[\mathrm{nm}]}$
photon energy and wavelength

A photon can lift an electron across the gap, and so be absorbed, only if $h\nu \geq E_g$. Shine light of decreasing wavelength on a semiconductor and it stays transparent until the photon energy reaches E_g, then begins to absorb strongly. That absorption edge is how band gaps are measured. Silicon's gap of 1.12 eV corresponds to $1240/1.12 = 1107$ nm, in the infrared, so silicon absorbs all visible light (which is why a wafer looks gray and opaque) and becomes transparent beyond about 1.1 µm. The process also runs in reverse: when a conduction electron falls back into a hole it can release the energy as a photon of energy close to E_g. That is how an LED works, and Table 2.2 shows why different materials glow in different colors.

Material	E_g (eV)	$\lambda = 1240/E_g$ (nm)	Region
PbTe	0.31	4000	mid-infrared
Ge	0.67	1850	infrared
Si	1.12	1107	near-infrared
GaAs	1.42	873	near-infrared
GaP	2.25	551	green
GaN	3.4	365	ultraviolet
Diamond	5.5	225	deep ultraviolet

Table 2.2: Band gaps of selected semiconductors at room temperature and the photon wavelength that matches each gap. Photons with shorter wavelength than this are absorbed.

WORKED EXAMPLE 2.9 *Band gap from an absorption edge*

A semiconductor sample is transparent to light of wavelength longer than 870 nm and absorbs strongly below it. Estimate its band gap and identify a likely material.

SOLUTION. The absorption edge is where the photon energy equals the gap:

$$E_g = \frac{1240}{870} = 1.43\,\text{eV}.$$

Table 2.2 lists GaAs at 1.42 eV, so the sample is very likely gallium arsenide. Silicon would stay opaque out to about 1107 nm. ■

2.4 *The pn Junction*

CARRIERS MOVE FOR TWO REASONS. The first is *drift*, which we already used to derive Equation (2.2): an electric field pushes holes along the field and electrons against it, at speeds set by the mobility. In a field of 1 kV/cm, an electron in lightly doped silicon drifts at $\mu_n \mathcal{E} = 1350 \times 1000 = 1.35 \times 10^6\,\text{cm/s}$. The second is *diffusion*. Any time carriers are crowded in one place and sparse in another, their random thermal motion spreads them out, the same way a drop of ink spreads through still water. No field is needed. The diffusion current is proportional to how steeply the density changes, $J = qD_n\, dn/dx$ for electrons, and the diffusion coefficient D is tied to mobility by the Einstein relation $D = \mu\,(kT/q)$. Both mechanisms appear in every device in this chapter, and in the pn junction they meet head on.

A *pn junction* is a single crystal that is p-type on one side and n-type on the other. Picture the two halves at the instant they join (Figure 2.9). The p side is full of holes and the n side is full of electrons, so both carriers immediately start to diffuse across the boundary: holes into the n side, electrons into the p side. When they meet, they recombine and vanish. What they leave behind is not neutral. Near the boundary on the n side, the donors have lost their electrons and are now bare positive ions; on the p side, the acceptors are negative ions. These ions are locked into the lattice and cannot move.

The layer of exposed ions is the *depletion region*, so called because it has been emptied of mobile carriers. Its positive and negative charges create an electric field pointing from the n side to the p side, and that field pushes holes back toward p and electrons back toward n, opposing the diffusion. The region grows until drift exactly cancels diffusion and no net current flows. At that point a potential difference,

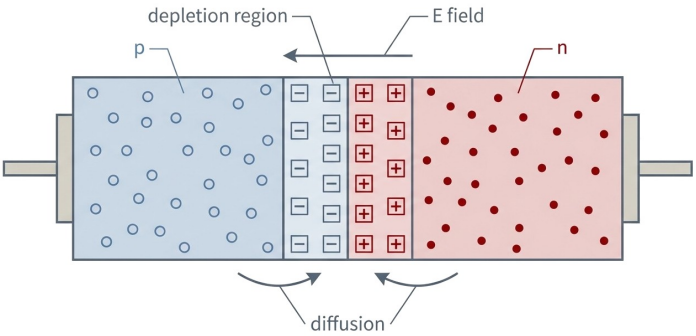
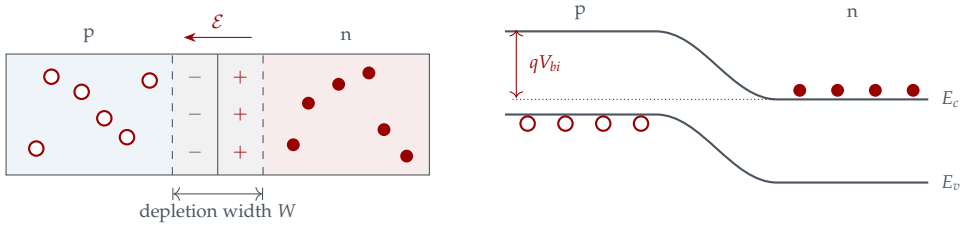


Figure 2.9: A pn junction. Diffusion of holes and electrons across the boundary leaves a depletion region of fixed ions (negative acceptors on the p side, positive donors on the n side) whose electric field holds the remaining carriers back.



(a) structure and charge (b) energy bands

Figure 2.10: The pn junction in equilibrium. (a) Holes (open circles) fill the p side and electrons (dots) fill the n side; the depletion region in between holds only fixed ions, whose field $\mathcal{E}$ points from n to p. (b) The same junction as a band diagram. The bands bend by qV_{bi} across the depletion region, forming an energy hill that electrons on the n side must climb. Hole energy increases downward on this diagram, so holes on the p side face an equal barrier.

the *built-in potential*, exists across the depletion region:

$$V_{bi} = \frac{kT}{q} \ln\left(\frac{N_A N_D}{n_i^2}\right). \tag{2.8}$$

For typical silicon junctions V_{bi} is 0.6 to 0.9 V. You cannot measure it with a voltmeter (contact potentials at the metal leads cancel it), but you will see its fingerprint: it is why a silicon diode needs roughly 0.6 V before it conducts.

In the band diagram of Figure 2.10(b), electrons on the n side face an energy hill of height qV_{bi}. The few that are energetic enough to climb it are balanced by the few minority electrons on the p side that roll down it, so no net current flows.

Worked Example 2.10 *Built-in potential and depletion width*

A silicon junction has $N_A = 10^{17}\,\text{cm}^{-3}$ on the p side and $N_D = 10^{16}\,\text{cm}^{-3}$ on the n side. Find V_{bi} at 300 K. The depletion width of an abrupt junction is $W = \sqrt{(2\varepsilon_s V_{bi}/q)(1/N_A + 1/N_D)}$ with $\varepsilon_s = 11.7\varepsilon_0 = 1.04 \times 10^{-12}\,\text{F/cm}$. Find W and say which side holds most of it.

Solution. With $kT/q = 0.0259\,\text{V}$,

$$V_{bi} = 0.0259 \ln\frac{(10^{17})(10^{16})}{10^{20}} = 0.0259 \ln(10^{13})$$
$$= 0.0259 \times 29.93 = 0.775\,\text{V}.$$

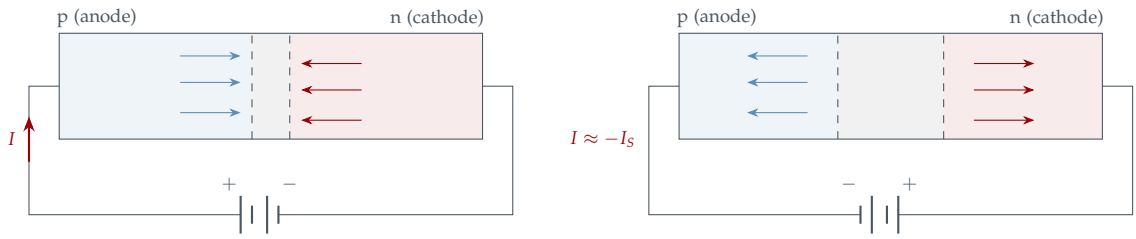


Figure 2.11: Biasing a pn junction. Left: with the anode positive, holes (steel arrows) and electrons (cardinal arrows) are pushed toward the junction, the depletion region narrows, and a large current flows. Right: with the anode negative, carriers are pulled away, the depletion region widens, and only the tiny saturation current I_S flows.

For the width,

$$W = \sqrt{\frac{2(1.04\times10^{-12})(0.775)}{1.602\times10^{-19}}\,(10^{-17}+10^{-16})}$$
$$= 3.33\times10^{-5}\,\text{cm} = 0.33\,\mu\text{m}.$$

The total charge on each side must balance, $N_A x_p = N_D x_n$, so the lightly doped side must give up a longer stretch to supply the same charge. Here $x_n = W N_A/(N_A+N_D) = 0.30\,\mu\text{m}$: about 91 % of the depletion region lies in the lightly doped n side. ■

Now apply a voltage (Figure 2.11). Call the p terminal the *anode* and the n terminal the *cathode*, and define the diode voltage V_D as anode minus cathode.

In *forward bias* ($V_D > 0$), the external source pushes holes into the p side and electrons into the n side, toward the junction. The applied voltage opposes the built-in potential, so the barrier drops from V_{bi} to $V_{bi} - V_D$ and the depletion region narrows. Because the number of carriers able to climb a barrier depends exponentially on its height (a Boltzmann factor $e^{-qV/kT}$), lowering the barrier even slightly lets exponentially more carriers through. Electrons flood into the p side and holes into the n side, where they recombine, and the source keeps resupplying them. Current flows easily: this is the "open door."

In *reverse bias* ($V_D < 0$), the source pulls holes and electrons away from the junction. The barrier rises to $V_{bi} + |V_D|$, the depletion region widens, and almost no majority carriers can cross. The only current left is a tiny flow of minority carriers (electrons generated on the p side, holes on the n side) that wander to the edge of the depletion region and get swept across by its field. Since there are very few minority carriers, this *reverse saturation current* I_S is minute, and it does not grow with reverse voltage because the supply of minority carriers, not the field, limits it. The door is closed.

Push reverse bias far enough and the field in the depletion region becomes strong enough to tear electrons out of bonds, either by direct

tunneling (*Zener breakdown*, in heavily doped junctions at a few volts) or by carriers gaining enough energy to knock loose more carriers in a chain reaction (*avalanche breakdown*, at tens to hundreds of volts). The current then rises sharply at a nearly fixed voltage.[1] Ordinary diodes must stay below their breakdown rating. Zener diodes are designed to work there and serve as voltage references.

[1] A water analogy pictures the diode as a spring-loaded flap valve in a pipe. Pressure in the forward direction must first overcome the spring (the 0.6 V knee), after which water flows freely. Pressure in the reverse direction presses the flap shut, apart from a slight seep (leakage) until the flap breaks (breakdown).

The depletion region also behaves like a capacitor: two layers of opposite fixed charge separated by a width W of silicon. Its capacitance per unit area is $C_j = \varepsilon_s / W$, and because reverse bias widens W, the capacitance falls as reverse voltage rises. Quantitatively, a reverse bias V_R adds to the barrier, and the width of an abrupt junction becomes

$$W = \sqrt{\frac{2\varepsilon_s}{q}\left(\frac{1}{N_A} + \frac{1}{N_D}\right)(V_{bi} + V_R)}, \tag{2.9}$$

which grows as the square root of the total barrier voltage. Photodiodes exploit this: a larger reverse bias means a smaller capacitance and a faster response.

WORKED EXAMPLE 2.11 *Reverse bias and junction capacitance*

Take the junction of the previous example ($V_{bi} = 0.775\,\text{V}$, $W = 0.333\,\mu\text{m}$ at zero bias) and give it an area of $1\,\text{mm}^2$. Find its depletion width and capacitance at zero bias and with a reverse bias of 3.3 V.

SOLUTION. By Equation (2.9), W scales as $\sqrt{V_{bi} + V_R}$, so

$$W(3.3\,\text{V}) = 0.333\sqrt{\frac{0.775 + 3.3}{0.775}} = 0.333 \times 2.293 = 0.764\,\mu\text{m}.$$

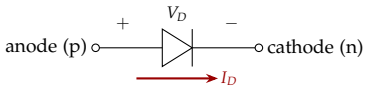
The area is $1\,\text{mm}^2 = 0.01\,\text{cm}^2$, so at zero bias

$$C = \frac{\varepsilon_s A}{W} = \frac{(1.04 \times 10^{-12})(0.01)}{3.33 \times 10^{-5}} = 3.12 \times 10^{-10}\,\text{F} = 312\,\text{pF},$$

and at 3.3 V reverse bias $C = 312/2.293 = 136\,\text{pF}$. A detector whose speed is set by charging this capacitance through a resistor becomes 2.3 times faster when reverse biased. ■

2.5 The Diode

A DIODE IS A PN JUNCTION WITH TWO LEADS, and its job is to pass current in one direction only. In its symbol,



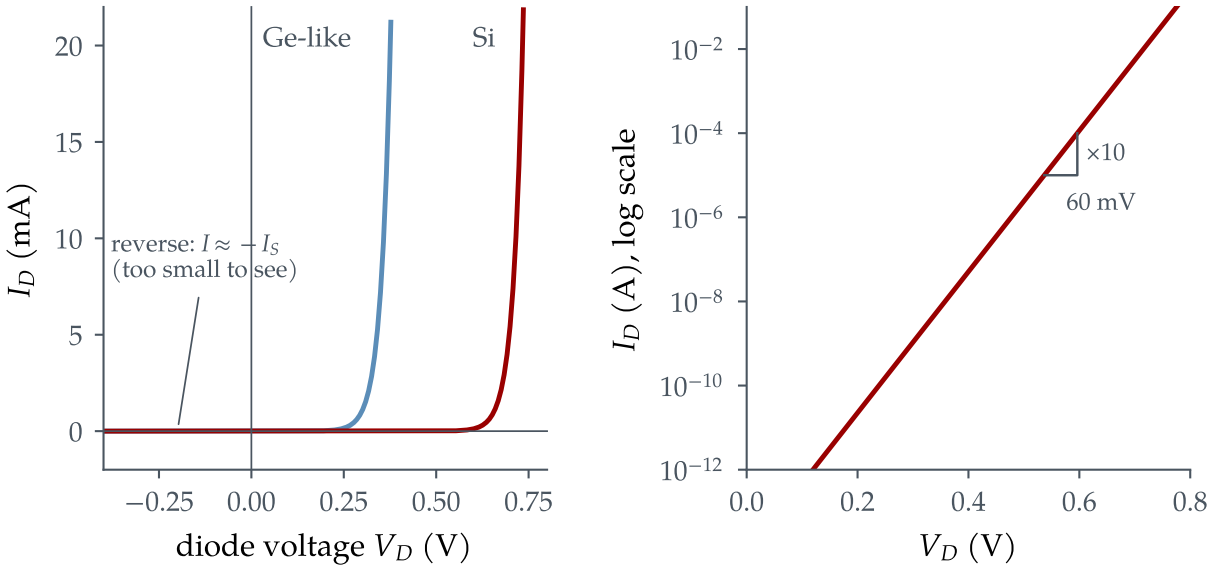


Figure 2.12: The diode equation with $n = 1$ and $V_T = 25.9\,\text{mV}$. Left: on linear axes, a silicon diode ($I_S = 10^{-14}\,\text{A}$) looks like it switches on near 0.6 V; a diode with a larger I_S (here $10^{-8}\,\text{A}$, typical of germanium or a Schottky diode) turns on sooner. Right: on a log scale the silicon curve is a straight line rising one decade for every 60 mV.

the triangle points in the direction of conventional forward current I_D, from anode to cathode, and the bar marks the cathode. On a real part the cathode is marked with a painted band. Forward current flows when $V_D > 0$.

The physics of the last section leads to a compact formula for the current, the *diode equation* (also called the Shockley equation):

$$I_D = I_S\left(e^{V_D/nV_T} - 1\right), \qquad V_T = \frac{kT}{q}. \tag{2.10}$$

KEY EQUATION
$I_D = I_S\left(e^{V_D/nV_T} - 1\right)$
the diode equation

Each symbol has a physical meaning. I_D is the current through the diode, positive from anode to cathode. V_D is the voltage across it, anode minus cathode. I_S is the *reverse saturation current*, the small minority-carrier current that flows in reverse bias. It is proportional to the junction area and to n_i^2, so it rises steeply with temperature; for small silicon diodes it lies between about 10^{-15} and $10^{-9}\,\text{A}$, and $10^{-12}\,\text{A}$ is a convenient round number. $V_T = kT/q$ is the *thermal voltage*, the voltage equivalent of thermal energy: 25.9 mV at 300 K. (Many textbooks round it to 26 mV or even 25 mV; each example below says which value it uses.) n is the *ideality factor*, between 1 and 2. It equals 1 for a junction where current is purely diffusion over the barrier and rises toward 2 when recombination inside the depletion region contributes. The "-1" makes the current exactly zero at $V_D = 0$.

Figure 2.12 plots the equation two ways. On linear axes the curve looks almost like a switch: flat until a *knee* near 0.6 to 0.7 V for silicon, then nearly vertical. The knee is not a property of the equation; the exponential has no corner. It appears because the eye picks out the voltage where the current reaches a "noticeable" value, typically a

milliampere. On a log scale the curve is a straight line. Each factor of 10 in current costs the same voltage step,

$$\Delta V_D = n V_T \ln 10 = 2.303\, n V_T \approx 60\,\text{mV per decade} \quad (n = 1,\ 300\,\text{K}). \tag{2.11}$$

This is the most useful single fact about a diode. Raising the current from 1 mA to 10 mA changes the voltage by only 60 mV, which is why a conducting silicon diode can be treated as a nearly fixed 0.6 V to 0.7 V drop.

In reverse bias, e^{V_D/nV_T} quickly becomes negligible and $I_D \approx -I_S$. Real diodes leak somewhat more than I_S because of surface paths and generation in the depletion region, but the current is still in the nanoampere range, until breakdown. Real diodes also have a series resistance R_s from the bulk silicon and contacts, which adds $I_D R_s$ to the voltage at high current and makes the steep part of the curve lean over.

Worked Example 2.12 *Evaluating the diode equation*

A silicon diode has $I_S = 10^{-12}$ A and $n = 1$. Use $V_T = 26$ mV. Find the current at $V_D = 0.7$, 0.6, 0.1, and -5 V.

Solution. Compute the exponent V_D/V_T first, then the exponential.

$V_D = 0.7$ V: $0.7/0.026 = 26.92$, and $e^{26.92} = 4.93 \times 10^{11}$. So $I_D = 10^{-12}(4.93 \times 10^{11} - 1) = 0.493$ A.

$V_D = 0.6$ V: $0.6/0.026 = 23.08$, and $e^{23.08} = 1.05 \times 10^{10}$, so $I_D = 10.5$ mA.

$V_D = 0.1$ V: $0.1/0.026 = 3.85$, and $e^{3.85} = 46.8$. So the current is $10^{-12}(46.8 - 1) = 4.58 \times 10^{-11}$ A, about 46 pA.

$V_D = -5$ V: $-5/0.026 = -192$, and e^{-192} is essentially zero, so $I_D = 10^{-12}(0 - 1) = -1 \times 10^{-12}$ A $= -I_S$.

Two lessons. First, 0.1 V more forward voltage (0.6 to 0.7 V) multiplies the current by $e^{0.1/0.026} = 47$. Second, half an ampere at 0.7 V would destroy a small-signal diode; in a real circuit, a resistor or the source limits the current long before the diode voltage reaches that value. ■

Worked Example 2.13 *Reading the ideality factor from a decade of current*

(a) The diode on the demo board has an ideality factor of about $n = 1.93$ (the lab finds this value). If the junction alone set the voltage, how much would V_D rise when the current grows tenfold, from 0.25 to 2.5 mA? Use $V_T = 25.9$ mV. (b) The lab sweep of a demo board (Lab 2.1, step B3) shows $V_D = 0.564$ V at 0.244 mA

and 0.745 V at 2.505 mA. What ideality factor do these two points suggest, and why is it too large?

SOLUTION. (a) For $I \gg I_S$, $V_D = nV_T \ln(I/I_S)$, so a tenfold current step costs

$$\Delta V_D = nV_T \ln 10 = (1.93)(0.0259)(2.303) = 115\,\text{mV}.$$

(b) Solving the same relation for n,

$$n = \frac{\Delta V_D}{V_T \ln(I_2/I_1)} = \frac{0.745 - 0.564}{0.0259\,\ln(2.505/0.244)} = \frac{0.181}{0.0603} = 3.00.$$

An ideality factor must lie between 1 and 2, so something other than the junction is adding voltage. That something is the series resistance R_s of the silicon and contacts. The junction accounts for $1.93 \times 0.0259 \times \ln(2.505/0.244) = 0.116\,\text{V}$ of the rise; the other 0.065 V appears across R_s as the current grows by 2.26 mA, which implies $R_s \approx 0.065/0.00226 = 29\,\Omega$, close to the $28\,\Omega$ of the lab's fit. The lab separates the two effects with that fit. ■

TRY IT ON THE BOARD Turn the selector to Module 2 and run `board.monitor("diodes")` in a new cell of the lab notebook, after its first cell has connected to the board. The firmware sweeps DAC channel 0 from 0 to 3.3 V in 16.5 mV steps and prints one line per step: V_{DAC}, V_D, and I_D in mA. Watch how little the middle column changes once the current passes a milliampere.

WORKED EXAMPLE 2.14 *Identify the device*

A two-terminal device is drawn as a block with a p region joined to an n region, with one lead on each side. What is this device? Sketch its I-V characteristic and explain what the threshold voltage is.

SOLUTION. A p region in contact with an n region is a pn junction *diode*. The p lead is the anode and the n lead is the cathode.

Its I-V curve is Figure 2.12 (left): essentially zero current for negative voltages (only $-I_S$, nanoamperes or less) down to the breakdown voltage, still very little current for small positive voltages, and then an exponential rise. The *threshold* (also called turn-on, cut-in, or knee) voltage is the forward voltage at which the current becomes appreciable, about 0.6 V to 0.7 V for silicon, 0.3 V for germanium, and roughly 1.6 V to 3.5 V for visible LEDs depending on color. Physically, it is the forward voltage needed to lower the junction's built-in barrier enough that majority carriers flood across.

Because the curve is exponential, the threshold is a convenient convention rather than a sharp corner: it is the voltage at which the current reaches roughly a milliampere in a typical circuit. ■

2.6 *Diode Circuits and Diode Models*

PUT A DIODE IN A CIRCUIT and you immediately meet a problem that resistor circuits never posed. Kirchhoff's laws still hold, but the diode's equation is exponential, so the circuit equations can no longer be solved with algebra alone. There are three ways forward, from most to least exact: solve the exponential equation numerically, approximate the diode with a simpler model, or solve graphically. Figure 2.13 compares the models.

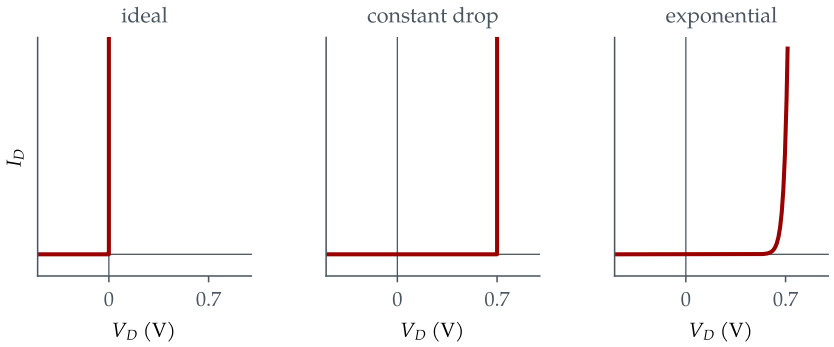


Figure 2.13: Three diode models. The *ideal* diode is a perfect switch: a short circuit for forward current and an open circuit for reverse voltage. The *constant-drop* model adds a fixed 0.7 V (silicon) when the diode conducts. The *exponential* model is the diode equation itself.

The *ideal diode* model treats the diode as a switch that closes for forward current and opens for reverse voltage. It is the right choice when the other voltages in the circuit are tens of volts, so that a fraction of a volt does not matter. The *constant-drop* model says that a conducting diode has $V_D = V_{on}$, with $V_{on} = 0.7\,\text{V}$ for silicon (or 0.3 V for germanium and Schottky diodes, and roughly 1.6 V to 3.5 V for visible LEDs), and that a non-conducting diode carries no current. This is the workhorse model for hand analysis, and it works because of the 60-mV-per-decade rule: even large changes in current barely move the voltage. The *exponential* model is Equation (2.10), used when you need accuracy or when the diode current is small.

Every model calls for the same strategy: *assume* a state (on or off), solve the linear circuit that results, then *check* the assumption. An "on" diode must carry positive current; an "off" diode must see $V_D < V_{on}$. If the check fails, flip the assumption and solve again.

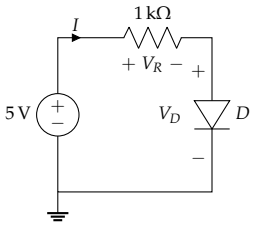


Figure 2.14: A diode in series with a resistor.

WORKED EXAMPLE 2.15 *Diode equation in a series circuit*

Write the ideal diode equation and explain each quantity. A silicon diode is connected in series with a 1 kΩ resistor across a 5 V supply (Figure 2.14). Its saturation current is $I_S = 10^{-12}$ A, with $n = 1$ and $kT/q = 26$ mV. (a) Use the diode equation to find the current when the diode voltage is 0.7 V. (b) Find the diode voltage when the current is 1 mA.

SOLUTION. The ideal diode equation is $I_D = I_S\left(e^{qV_D/nkT} - 1\right)$, where I_D is the diode current (anode to cathode), V_D the voltage across the diode (anode minus cathode), I_S the reverse saturation current set by the junction's doping, area, and temperature, n the ideality factor (1 for an ideal junction), k Boltzmann's constant, T the absolute temperature, and q the electron charge. The combination $kT/q = V_T$ is the thermal voltage, here 26 mV.

(a) At $V_D = 0.7$ V:

$$I_D = 10^{-12}\left(e^{0.7/0.026} - 1\right) \approx 10^{-12}\, e^{26.92}$$
$$= 10^{-12}(4.93 \times 10^{11}) = 0.493 \text{ A}.$$

That is the answer the equation gives, but notice that this circuit cannot deliver it. With the full 5 V across the resistor the current could not exceed $5/1000 = 5$ mA. So in this circuit the diode never reaches 0.7 V. To find where it actually sits, combine KVL, $I = (5 - V_D)/1\,\text{k}\Omega$, with the diode equation and iterate: guess V_D, compute I from KVL, then update $V_D = V_T \ln(I/I_S)$:

Step	V_D used (V)	$I = (5 - V_D)/1000$ (mA)	new V_D (V)
1	0.7000	4.3000	0.5767
2	0.5767	4.4233	0.5775
3	0.5775	4.4225	0.5775

The operating point is $V_D = 0.577$ V with a current of 4.42 mA. The constant-drop model ($V_D = 0.7$ V) would give $I = 4.3/1000 = 4.30$ mA, only 3 % low. The ideal model ($V_D = 0$) gives 5.00 mA, 13 % high.

(b) Solve the diode equation for V_D:

$$V_D = nV_T \ln\left(\frac{I_D}{I_S} + 1\right) = 0.026 \ln\left(\frac{10^{-3}}{10^{-12}} + 1\right)$$
$$= 0.026 \times 20.72 = 0.539 \text{ V}.$$

A current of 1 mA needs only 0.539 V. (To make the circuit carry 1 mA, the resistor would have to be raised to $(5 - 0.539)/10^{-3} = 4.46\,\text{k}\Omega$.) ■

The graphical method makes the iteration visible. KVL around the loop says the current must satisfy $I = (V_S - V_D)/R$, a straight line in the I-V plane called the *load line*. It runs from the open-circuit point ($V_D = V_S$, $I = 0$) to the short-circuit point ($V_D = 0$, $I = V_S/R$). The diode must also obey its own curve. The only point that satisfies both is where they cross, the *operating point* (Figure 2.15). Change the supply and the line slides parallel to itself; change the resistor and it pivots about the open-circuit point. Because the diode curve is so steep, the crossing barely moves in voltage, which is the constant-drop model seen from another angle.

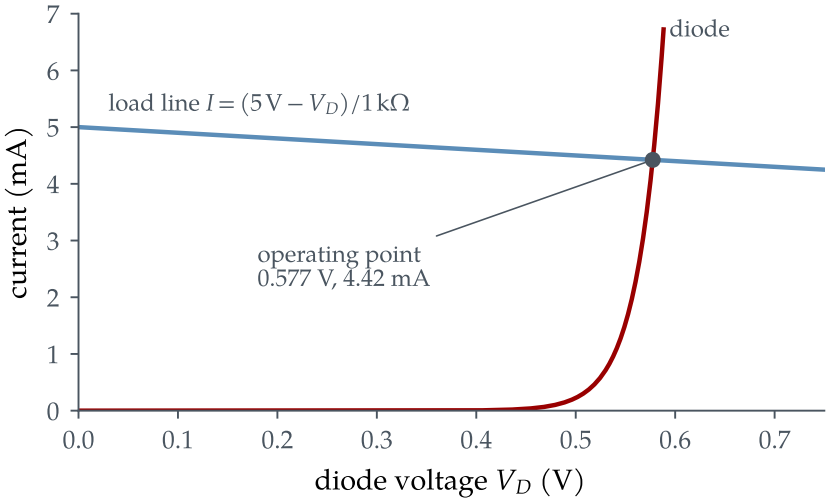


Figure 2.15: Load-line solution of the circuit in Figure 2.14. Only the beginning of the line is shown; it continues down to zero current at $V_D = 5\,\mathrm{V}$.

Worked Example 2.16 *The lab's diode circuit, four ways*

In the lab, DAC channel 0 drives the board's diode through the $1\,\mathrm{k\Omega}$ sense resistor. Take $V_{DAC} = 2.00\,\mathrm{V}$ and predict the current with (a) the ideal-diode model, (b) the constant-drop model, (c) the diode equation with $n = 1.93$ and $I_S = 3.5\,\mathrm{nA}$, and (d) the same diode equation plus a series resistance $R_s = 28\,\Omega$ (the values the lab fit gives). Use $V_T = 25.9\,\mathrm{mV}$. The measured sweep gives about $1.32\,\mathrm{mA}$ at this DAC voltage.

Solution. (a) An ideal diode drops nothing: $I = 2.00/1000 = 2.00\,\mathrm{mA}$, 52 % too high.

(b) Constant drop: $I = (2.00 - 0.70)/1000 = 1.30\,\mathrm{mA}$, within 2 %.

(c) Iterate as in the worked example "Diode equation in a series circuit," with $nV_T = 50.0\,\mathrm{mV}$. Start from $1.30\,\mathrm{mA}$:

$$V_D = 0.0500 \ln(1.30 \times 10^{-3}/3.5 \times 10^{-9}) = 0.641\,\mathrm{V},$$

so $I = (2.00 - 0.641)/1000 = 1.359\,\text{mA}$. Once more: $V_D = 0.643\,\text{V}$ and $I = 1.357\,\text{mA}$, which no longer changes. The exponential model alone predicts 1.36 mA, 3 % high.

(d) With R_s the loop equation is $2.00 = (1028\,\Omega)\,I + nV_T \ln(I/I_S)$, whose solution is $I = 1.32\,\text{mA}$ (junction 0.642 V, plus 0.037 V across R_s, so 0.680 V at the diode's terminals). This matches the measurement.

The constant-drop model does as well as the full model here only because the real diode voltage, 0.68 V, happens to sit close to 0.7 V. At $V_{DAC} = 1.0\,\text{V}$, where the diode voltage is near 0.6 V, the same model is 26 % low (lab Question 3). ■

SMALL-SIGNAL RESISTANCE. Near an operating point, a small change in voltage produces a proportional change in current. The ratio is the slope of the curve, and its inverse is the diode's *dynamic* or *small-signal resistance*. Differentiating Equation (2.10) in forward bias,

$$r_d = \frac{dV_D}{dI_D} = \frac{nV_T}{I_D}. \tag{2.12}$$

At 1 mA a diode with $n = 1$ looks like 26 Ω to small signals, at 10 mA like 2.6 Ω. The diode is a resistor whose value you set with the DC current through it, an early example of the tunable resistance promised in Section 2.1.

RECTIFICATION is the classic diode job. In the half-wave rectifier of Figure 2.16, a diode sits between an AC source and a load resistor. During positive half cycles the diode conducts, and the load sees the input minus about 0.7 V. During negative half cycles the diode is reverse biased, no current flows, and the output is zero. The output never goes negative: AC in, pulsating DC out. Adding a capacitor across the load smooths the pulses into a nearly steady voltage, which is how simple power supplies work. Four diodes in a bridge rectify both half cycles.

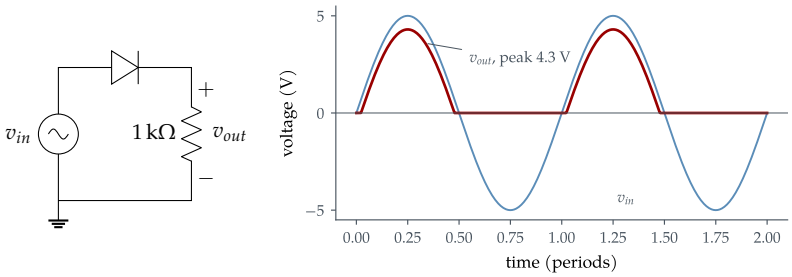


Figure 2.16: Half-wave rectifier with a 5 V peak sine input. With the constant-drop model, the output follows $v_{in} - 0.7\,\text{V}$ while the input exceeds 0.7 V and is zero otherwise, so its peak is 4.3 V.

The same on-or-off reasoning explains reverse-polarity protection (a series diode blocks a backward battery), flyback diodes across relay

and motor coils, and the clamp diodes that protect the Arduino's input pins.

2.7 *The MOSFET*

A DIODE HAS TWO TERMINALS, so only the voltage across it can control its current. The breakthrough of modern electronics was a third terminal. The first transistor, built at Bell Telephone Laboratories on December 16, 1947, by John Bardeen and Walter Brattain in William Shockley's group, was a slab of germanium touched by two closely spaced gold contacts: a small current into one contact controlled a larger current at the other. It amplified like a vacuum tube in a device a few millimeters across, and the three shared the 1956 Nobel Prize in Physics.

The transistor in your phone works on an older idea. A metal plate (the *gate*) held just above a semiconductor surface by a thin insulator can, through its electric field, attract or repel carriers at the surface and so change the current between two contacts. This *field-effect transistor* (FET) was patented by Oskar Heil in 1935,[2] but a reliable metal-oxide-semiconductor FET (MOSFET) came only after engineers learned to grow clean SiO_2 on silicon, around 1960. The chip in your Arduino contains millions of them.

[2] Julius Lilienfeld filed similar patents in the 1920s. Neither inventor could build a working device, because the semiconductor surfaces of the day trapped the charge the gate was meant to control.

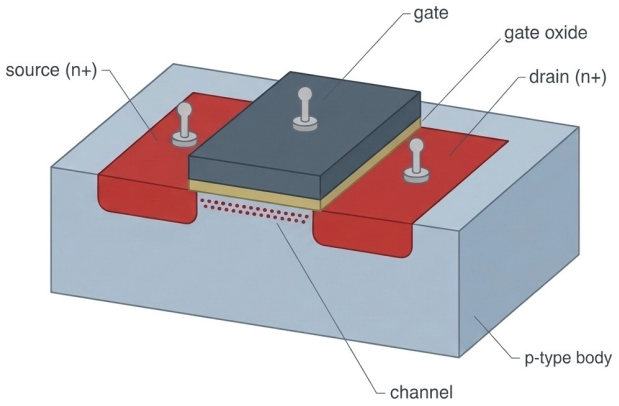


Figure 2.17: Cutaway of an n-channel MOSFET. Two heavily doped n^+ regions (source and drain) sit in a p-type body. The gate electrode is separated from the silicon by a thin oxide. A positive gate voltage pulls electrons to the surface, forming a conducting channel between source and drain.

STRUCTURE. Figure 2.17 shows an n-channel MOSFET (NMOS). The starting material, the *body* or substrate, is p-type silicon. Two heavily doped n^+ regions are formed in it a short distance L apart (the *channel length*, which ranges from micrometers in discrete parts down to tens of nanometers in processors); they are the *source* and the *drain*. The strip of silicon between them is covered by a thin layer of SiO_2, the gate oxide, and on top of that sits the gate electrode, originally aluminum

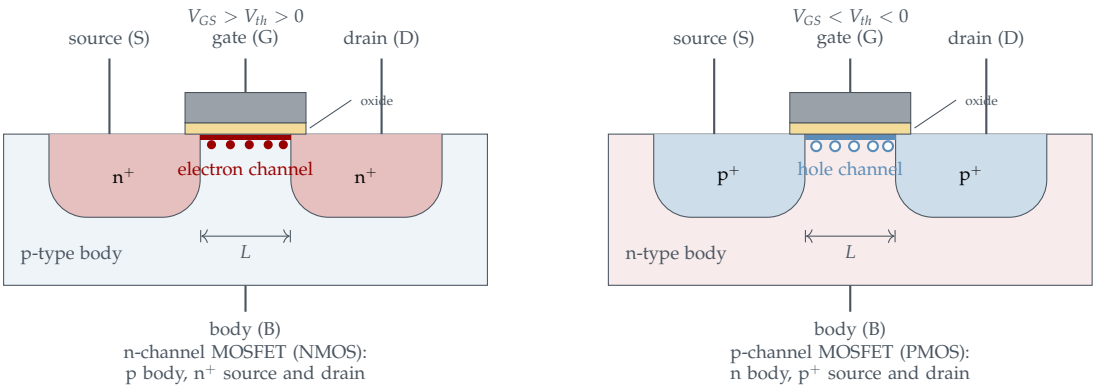


Figure 2.18: Side views of the two kinds of MOSFET with their channels formed. In the NMOS (left), a positive gate voltage attracts electrons to the surface of the p body and inverts it to n-type. In the PMOS (right), every doping and every voltage is reversed: a negative gate voltage attracts holes to the surface of an n body.

(the "metal" in MOS) and now usually heavily doped polysilicon or a metal stack. The width of the strip, measured into the page, is W. Figure 2.18 shows the side view.

Notice that the source and drain each form a pn junction with the body. With the gate at zero, going from source to drain means crossing an n^+p junction and then a pn^+ junction, two diodes back to back. One of them is always reverse biased, so no current flows no matter which way you push. That is the *off* state. (The body is normally tied to the source or to the most negative voltage in the circuit so that these junctions are never forward biased.)

Turning the transistor on. Raise the gate voltage relative to the source, V_{GS}. The gate and the body form a capacitor with the oxide as its dielectric. A positive charge on the gate first pushes the holes in the p body away from the surface, leaving a depletion layer of negative acceptor ions. Push further and the gate begins to attract electrons, which the n^+ source and drain supply in abundance. When enough electrons gather at the surface, a thin layer there behaves as n-type silicon even though the body is p-type. This *inversion layer* connects source to drain like a wire whose conductance the gate controls. It is the *channel*, and the gate voltage at which it forms is the *threshold voltage* V_{th}, typically 0.3 to 1 V in logic transistors and 1 to 3 V in discrete power MOSFETs.

Above threshold, the electron charge in the channel per unit area is the gate capacitance times the excess voltage, $Q_n = C_{ox}(V_{GS} - V_{\text{th}})$, where $C_{ox} = \varepsilon_{ox}/t_{ox}$ is the oxide capacitance per unit area and $V_{GS} - V_{\text{th}}$ is called the *overdrive voltage*. More overdrive means more electrons and a more conductive channel. The gate itself draws no DC current, because the oxide is an insulator. That is the decisive advantage of the MOSFET: it is controlled by a voltage and consumes essentially no power to hold its state.

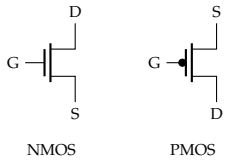


Figure 2.19: Circuit symbols. The PMOS gate carries a bubble (it turns on for a low gate voltage) and its source is drawn at the top.

NMOS AND PMOS. Swap every doping type and you get the p-channel MOSFET (PMOS): an n-type body, p^+ source and drain, and a channel made of holes (Figure 2.18, right). It turns on when the gate is *below* the source by more than $|V_{th}|$, so its threshold is negative and its currents flow from source to drain. Because holes have lower mobility than electrons, a PMOS carries roughly half the current of an NMOS of the same size at the same overdrive. Figure 2.19 shows the two symbols. You can also think of the NMOS as a switch that closes when its gate is high and the PMOS as a switch that closes when its gate is low.

WORKED EXAMPLE 2.17 *MOSFET types and side views*

What is a MOSFET? What are n-channel and p-channel MOSFETs? Draw their side views with each region labeled, and draw their circuit symbols.

SOLUTION. A MOSFET (a metal-oxide-semiconductor field-effect transistor) is a four-terminal device in which the voltage on a gate electrode, insulated from the silicon by a thin oxide, controls the conductivity of a channel between two terminals called the source and the drain. The fourth terminal is the body (substrate), usually tied to the source.

An *n-channel* MOSFET is built on a p-type body with n^+ source and drain; a positive V_{GS} above V_{th} forms a channel of electrons. A *p-channel* MOSFET is built on an n-type body (in an integrated circuit, an n-well) with p^+ source and drain; a negative V_{GS} below $V_{th} < 0$ forms a channel of holes.

The labeled side views are Figure 2.18. Each shows, from top to bottom, the gate electrode, the thin gate oxide, the channel region at the silicon surface between source and drain, the two heavily doped source and drain regions, and the body with its contact. The circuit symbols are Figure 2.19; the PMOS is distinguished by the bubble on its gate (some books instead reverse the arrow on the source). ■

REGIONS OF OPERATION. Hold the gate above threshold and slowly raise the drain voltage V_{DS}. Figure 2.20 shows what happens to the drain current I_D, and the physical picture behind each part of the curve is this.

For small V_{DS} the channel is a uniform sheet of electrons, and the device behaves like a resistor: I_D rises in proportion to V_{DS}. As V_{DS} grows, the voltage between gate and channel shrinks toward the drain end (it is V_{GS} at the source but only $V_{GS} - V_{DS}$ at the drain), so the channel thins out near the drain and the curve bends over. This is the *triode* or *linear* region. When V_{DS} reaches $V_{GS} - V_{th}$, the gate-to-drain

voltage equals the threshold, and the channel just vanishes at the drain end, an event called *pinch-off*. Beyond pinch-off, extra drain voltage drops across the short depleted stretch near the drain, and electrons arriving at the pinch-off point are swept across it by the strong field. The current stops increasing. This is the *saturation* region, and the voltage where it begins is $V_{DS,sat} = V_{GS} - V_{\text{th}}$. If the gate is below threshold, there is no channel at all, and the device is in *cutoff*.

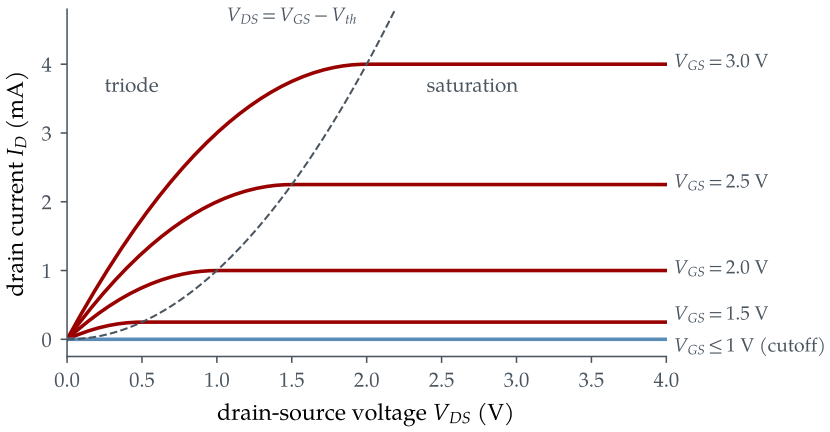


Figure 2.20: Output characteristics of an NMOS with $\mu_n C_{ox} W/L = 2\,\text{mA/V}^2$ and $V_{\text{th}} = 1\,\text{V}$, from Equations (2.14) and (2.15). The dashed parabola $V_{DS} = V_{GS} - V_{\text{th}}$ separates triode from saturation.

THE SQUARE LAW. Adding up the charge along the channel and multiplying by its drift velocity gives the long-channel MOSFET equations. With $k' = \mu_n C_{ox}(W/L)$, in units of A/V^2:

$$\text{cutoff } (V_{GS} < V_{\text{th}}): \qquad I_D = 0, \tag{2.13}$$

$$\text{triode } (V_{DS} < V_{GS} - V_{\text{th}}): \qquad I_D = k'\left[(V_{GS} - V_{\text{th}})V_{DS} - \tfrac{1}{2}V_{DS}^2\right], \tag{2.14}$$

$$\text{saturation } (V_{DS} \ge V_{GS} - V_{\text{th}}): \qquad I_D = \tfrac{1}{2}k'(V_{GS} - V_{\text{th}})^2. \tag{2.15}$$

KEY EQUATION

$I_D = \frac{1}{2}\mu_n C_{ox}\frac{W}{L}(V_{GS} - V_{\text{th}})^2$

saturation, the square law

The two expressions agree at the boundary, as they must. Three facts make them easy to use. The current scales with W/L, so a designer sets a transistor's strength by drawing it wider or shorter. In saturation the current depends only on the overdrive, squared, not on V_{DS}: the MOSFET is a *voltage-controlled current source*. And for very small V_{DS} the V_{DS}^2 term is negligible, so the triode region is a resistor of value

$$R_{on} \approx \frac{1}{k'(V_{GS} - V_{\text{th}})}, \tag{2.16}$$

which the gate voltage tunes. Some textbooks write the saturation current as $K(V_{GS} - V_{\text{th}})^2$, folding the $\frac{1}{2}$ into the constant $K = \mu_n C_{ox} W/2L$. Always check which convention a problem uses.

To find which equation applies, use the same assume-and-check strategy as for diodes. Check V_{GS} against V_{th} first. If the device is on, assume saturation, compute I_D, use the circuit to find V_{DS}, and confirm that $V_{DS} \geq V_{GS} - V_{\text{th}}$. If not, redo the problem with the triode equation.

A PMOS obeys the same equations with every voltage reversed. Write them in terms of the source-to-gate voltage $V_{SG} = -V_{GS}$, the source-to-drain voltage $V_{SD} = -V_{DS}$, and $|V_{\text{th}}|$; the current I_{SD} then flows into the source and out of the drain. The device is on when $V_{SG} > |V_{\text{th}}|$, saturated when $V_{SD} \geq V_{SG} - |V_{\text{th}}|$, and there $I_{SD} = \frac{1}{2}k'_p(V_{SG} - |V_{\text{th}}|)^2$, with $k'_p = \mu_p C_{ox} W/L$.

Worked Example 2.18 *Triode and saturation currents*

An NMOS has $\mu_n C_{ox} = 200\,\mu\text{A/V}^2$, $W/L = 10$, and $V_{\text{th}} = 1.0\,\text{V}$. Find I_D for $V_{GS} = 3.0\,\text{V}$ when (a) $V_{DS} = 0.5\,\text{V}$ and (b) $V_{DS} = 3.0\,\text{V}$. (c) What is the on-resistance for small V_{DS}?

Solution. First, $k' = \mu_n C_{ox}(W/L) = 200 \times 10 = 2000\,\mu\text{A/V}^2 = 2\,\text{mA/V}^2$, and the overdrive is $V_{GS} - V_{\text{th}} = 2.0\,\text{V}$.

(a) $V_{DS} = 0.5 < 2.0$, so the device is in triode:

$$I_D = 2\left[(2.0)(0.5) - \tfrac{1}{2}(0.5)^2\right] = 2\,(1.0 - 0.125) = 1.75\,\text{mA}.$$

(b) $V_{DS} = 3.0 \geq 2.0$, so the device is saturated:

$$I_D = \tfrac{1}{2}(2)(2.0)^2 = 4.0\,\text{mA}.$$

(c) $R_{on} = 1/[k'(V_{GS} - V_{\text{th}})] = 1/(2\,\text{mA/V}^2 \times 2\,\text{V}) = 250\,\Omega$. Check against (a): $0.5\,\text{V}/250\,\Omega$ would be 2.0 mA; the actual 1.75 mA is a bit lower because at 0.5 V the curve has already started to bend. ■

Worked Example 2.19 *A PMOS high-side switch*

A PMOS with $V_{\text{th}} = -1.0\,\text{V}$ and $k'_p = 0.1\,\text{A/V}^2$ has its source on a 3.3 V supply and its drain connected through a $100\,\Omega$ load to ground. An Arduino pin drives the gate. Find the load current and V_{SD} when the pin is low (0 V) and when it is high (3.3 V).

Solution. Pin low: $V_{SG} = 3.3 - 0 = 3.3\,\text{V}$, well above $|V_{\text{th}}| = 1.0\,\text{V}$, so the PMOS is on with an overdrive of 2.3 V. Assume it is deep in triode and use the on-resistance, $R_{on} = 1/(0.1 \times 2.3) = 4.35\,\Omega$. The load current is $3.3/(100 + 4.35) = 31.6\,\text{mA}$, and $V_{SD} = (0.0316)(4.35) = 0.14\,\text{V}$, far below $V_{SG} - |V_{\text{th}}| = 2.3\,\text{V}$, which confirms triode. Pin high: $V_{SG} = 0 < |V_{\text{th}}|$, the PMOS is off, no current flows ($I = 0$), the load sees 0 V, and the whole supply appears across the transistor: $V_{SD} = 3.3\,\text{V}$. A low gate turns the PMOS on,

the opposite of an NMOS. That is why a PMOS switch sits between the supply and the load (on the *high side*), while an NMOS switch sits between the load and ground (on the *low side*), with its source at ground so that a positive gate voltage turns it on. ■

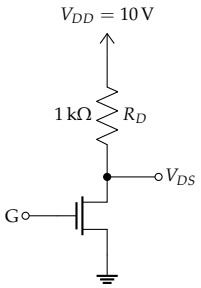


Figure 2.21: NMOS with a drain resistor.

Worked Example 2.20 *NMOS with a drain resistor*

An n-channel MOSFET has a drain resistor $R_D = 1\,\text{k}\Omega$ to a supply $V_{DD} = 10\,\text{V}$ and its source grounded (Figure 2.21). Its parameters are $V_{\text{th}} = 2\,\text{V}$ and $\mu_n C_{ox} W/2L = 0.5\,\text{mA/V}^2$. With $V_{GS} = 4\,\text{V}$, and assuming saturation, find the drain current and the voltage across the resistor.

Solution. The given constant already includes the factor of one half, $K = \mu_n C_{ox} W/2L = 0.5\,\text{mA/V}^2$, so the saturation current is

$$I_D = K(V_{GS} - V_{\text{th}})^2 = 0.5\,(4-2)^2 = 2.0\,\text{mA}.$$

The resistor carries the drain current, so

$$V_R = I_D R_D = (2\,\text{mA})(1\,\text{k}\Omega) = 2.0\,\text{V}.$$

Check the assumption. KVL gives $V_{DS} = V_{DD} - V_R = 10 - 2 = 8.0\,\text{V}$, and saturation requires $V_{DS} \geq V_{GS} - V_{\text{th}} = 2\,\text{V}$. Since $8 \geq 2$, the device is indeed saturated, and the answers stand: 2 mA and 2 V. ■

The MOSFET as an amplifier. In the circuit of Figure 2.21, any change in gate voltage changes the drain current, and the drain resistor converts that current change into a voltage change. The *transconductance* $g_m = dI_D/dV_{GS}$ measures the strength of the first step. Differentiating the square law,

$$g_m = k'(V_{GS} - V_{\text{th}}) = \frac{2I_D}{V_{GS} - V_{\text{th}}}. \tag{2.17}$$

A small signal v_{gs} on the gate therefore produces a drain current $g_m v_{gs}$ and an output voltage $-g_m R_D v_{gs}$ (the minus sign because more current pulls the drain down). The *voltage gain* is $A_v = -g_m R_D$. In the circuit of the worked example "NMOS with a drain resistor," $g_m = 2(2\,\text{mA})/(2\,\text{V}) = 2\,\text{mA/V}$, so $A_v = -2$. Raising R_D raises the gain, up to the point where the larger drop across R_D pushes V_{DS} below $V_{GS} - V_{\text{th}}$ and the transistor leaves saturation (here at $R_D = 4\,\text{k}\Omega$, gain -8). Practical amplifiers use active loads and more transistors to reach gains of hundreds, but every one of them rests on this idea: a small gate voltage steers a large drain current.

Worked Example 2.21 *The off state*

NMOS output curves are often drawn as a family for gate voltages of 1, 2, and 3 V. Add the curve for $V_G < V_{\text{th}}$. What is the state of the transistor then?

Solution. For $V_G < V_{\text{th}}$ no inversion channel forms. Source and drain are separated by two back-to-back pn junctions, so the drain current is zero (apart from a tiny subthreshold leakage) for every V_{DS}. The added curve lies along the V_{DS} axis, exactly like the steel cutoff line at the bottom of Figure 2.20, below all the curves with $V_G > V_{\text{th}}$. The transistor is in *cutoff*: it is off and acts as an open switch. ■

Below threshold the current is small, not zero. A few electrons have enough thermal energy to cross into the channel region even when the gate has not fully inverted it, and their number depends exponentially on the gate voltage, just as the diode current depends exponentially on the junction voltage:

$$I_D \approx I_0\, e^{(V_{GS}-V_{\text{th}})/nV_T} \qquad (V_{GS} < V_{\text{th}}), \tag{2.18}$$

with a slope factor n between about 1.1 and 1.5. On a log plot (Figure 2.22) this *subthreshold* current is a straight line, falling one decade for every $2.3\,nV_T$, which is 65 to 90 mV at room temperature. The current that remains at $V_{GS} = 0$ is the *off current* I_{off}. One leaky transistor is harmless, but a billion of them can burn over a hundred milliwatts doing nothing (see Problem 2.18). Used deliberately, the same exponential lets wearable and implantable sensors run analog circuits on nanoamperes.

The MOSFET as a switch. Drive the gate either fully low (off) or fully high (on, deep in triode, where it looks like the small resistance R_{on}) and the MOSFET becomes an electrically controlled switch. A microcontroller pin can switch loads far larger than it could drive itself, because the gate draws no steady current.

Worked Example 2.22 *Switching an LED with an NMOS*

An Arduino output pin (3.3 V when high) drives the gate of an NMOS whose source is grounded. The drain connects through a resistor R and a red LED ($V_F = 2.0$ V) to a 5 V supply. The MOSFET has $V_{\text{th}} = 1.5$ V and $k' = 0.2\,\text{A/V}^2$. Choose R for an LED current of 20 mA.

Solution. With the gate at 3.3 V, the overdrive is $3.3 - 1.5 = 1.8$ V. A switch should sit deep in triode, so use the on-resistance: $R_{on} = 1/(0.2 \times 1.8) = 2.78\,\Omega$. At 20 mA the transistor drops only $V_{DS} =$

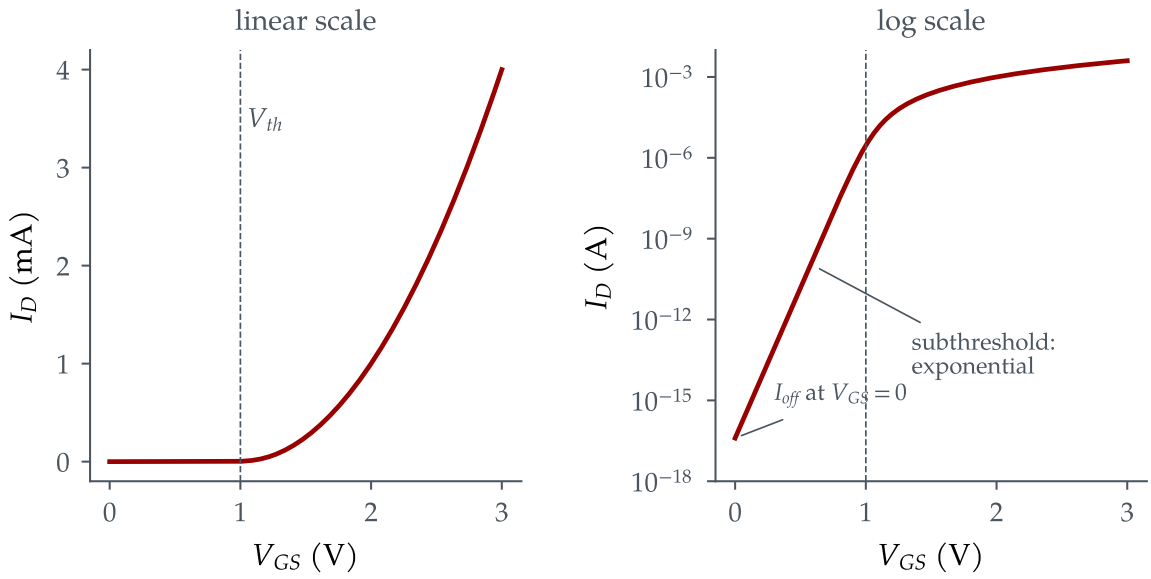


Figure 2.22: Transfer characteristic I_D versus V_{GS} in saturation for the device of Figure 2.20. On linear axes (left), the current appears to start at V_{th} and grow as a parabola. On log axes (right), the current below threshold falls exponentially, here about 90 mV per decade ($n = 1.5$), down to the off current at $V_{GS} = 0$.

$(0.020)(2.78) = 0.056\,\text{V}$, which confirms triode ($0.056 \ll 1.8$). KVL around the loop:

$$R = \frac{5 - 2.0 - 0.056}{0.020} = 147\,\Omega.$$

Use the standard value of $150\,\Omega$. The current is then

$$I = \frac{5 - 2.0}{150 + 2.78} = 19.6\,\text{mA}.$$

With the gate at 0 V the transistor is in cutoff and the LED is dark. The Arduino pin supplies only the tiny charge needed to charge the gate capacitance at each transition. ■

TRY IT ON THE BOARD With the selector on Module 2, run `board.monitor("transfer")` in a new cell of the lab notebook. The firmware holds DAC channel 2 at 3.3 V, sweeps the gate of the board's transistor (a DMN3404L) from 0 to 3.3 V, and prints V_{GS}, V_{DS}, and I_D in mA (its V_{DS} column counts only the 1 kΩ sense resistor). Find the gate voltage at which the current rises off zero; for this part it lies between 1.0 and 2.0 V, typically near 1.5 V. Section 2.11 explains how the board measures I_D and why the current levels off, near 0.6 mA on a real board.

2.8 CMOS Logic

Pair an NMOS with a PMOS and you get the most important circuit in digital electronics, the CMOS (complementary MOS) inverter of Figure 2.23. The PMOS source connects to the supply V_{DD}, the NMOS source to ground, the two drains together form the output, and the two gates together form the input. The capacitor at the output stands for the wiring and the gates of whatever the inverter drives.

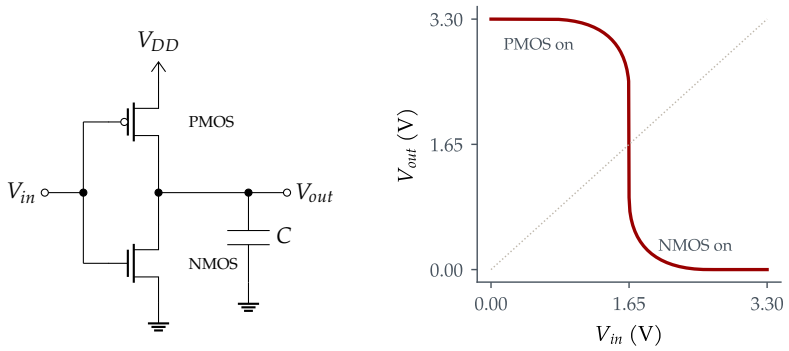


Figure 2.23: CMOS inverter and its voltage transfer characteristic for $V_{DD} = 3.3\,\text{V}$, with matched transistors ($k'_n = k'_p$, $V_{\text{th}} = 0.7\,\text{V}$ for both). The output swings fully from rail to rail and switches sharply at $V_{DD}/2$.

When the input is *low* (0 V), the NMOS has $V_{GS} = 0 < V_{\text{th}}$ and is off. The PMOS has its gate 3.3 V below its source, so it is strongly on and connects the output to V_{DD}. The output is *high*. When the input is *high* (V_{DD}), the roles swap: the NMOS is on and connects the output to ground, the PMOS has $V_{GS} = 0$ and is off, and the output is *low*. The circuit computes the logical NOT.

Worked Example 2.23 *CMOS inverter truth table*

For the circuit of Figure 2.23, fill in V_{out} for $V_{in} = 0$ and $V_{in} = V_{DD}$. What is this device, and where is it used?

Solution.

V_{in}	PMOS	NMOS	V_{out}
0	on	off	V_{DD}
V_{DD}	off	on	0

This is a CMOS *inverter*, the NOT gate. It is the basic cell of all CMOS digital logic: microprocessors, memory, and the microcontroller on the Arduino are built from inverters and their relatives (NAND and NOR gates use the same complementary pull-up and pull-down idea with more transistors). Inverters are also used as buffers to drive long wires or large loads, as clock drivers, and, connected in a ring of an odd number, as oscillators. ■

The beauty of CMOS lies in what it does not do. In either steady

state, one of the two transistors is off, so there is no path from V_{DD} to ground, and the static current is only leakage. Power is drawn mainly when the output switches, to charge or discharge the load capacitance C: each full cycle takes energy CV_{DD}^2 from the supply, so a gate switching at frequency f dissipates

$$P_{dyn} = C\,V_{DD}^2\,f. \tag{2.19}$$

Supply voltages have fallen from 5 V to below 1 V largely because power falls with the square of V_{DD}. The PMOS is usually drawn about twice as wide as the NMOS: with $\mu_n \approx 2\mu_p$, that equalizes their k' values, so the output rises as fast as it falls and switches at $V_{DD}/2$.

A real chip holds millions of gates, and in a given clock cycle only a fraction α of them (the *activity factor*) switch. With N gates of load C each, the total dynamic power is $P = \alpha N C V_{DD}^2 f$.

Worked Example 2.24 *Dynamic power of a processor*

A processor has $N = 10^7$ gates, each driving 2 fF, and runs at $f = 100$ MHz with an activity factor $\alpha = 0.1$. Find its dynamic power at $V_{DD} = 1.0$ V and at 0.8 V, and the energy it spends per clock cycle at 1.0 V.

Solution. At 1.0 V:

$$P = \alpha N C V_{DD}^2 f = (0.1)(10^7)(2\times10^{-15})(1.0)^2(10^8) = 0.20\,\text{W}.$$

At 0.8 V the factor V_{DD}^2 drops from 1.0 to 0.64, so $P = 0.128$ W, 36 % less for the same work. The energy per cycle at 1.0 V is $P/f = 0.20/10^8 = 2$ nJ. Halving the clock would halve the power but not the energy per cycle; lowering V_{DD} reduces both, which is why battery-powered devices run their processors at the lowest supply voltage that still meets the required speed. ■

2.9 *Light-Emitting Diodes*

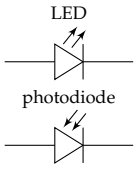


Figure 2.24: Symbols for a light-emitting diode (arrows out) and a photodiode (arrows in). Both are diodes with anode on the left.

Look at the power and status lights of the blocks on your demo board and you are looking at diodes that glow. Optoelectronic devices convert between electrical and optical energy using the bandgap physics of Section 2.3: an LED turns current into light, while a photodiode, phototransistor, or solar cell turns light into current. Figure 2.24 shows the symbols; arrows out mean light emitted, arrows in mean light absorbed.

An LED is a forward-biased diode made from a material in which electrons and holes recombine by emitting light. In forward bias, elec-

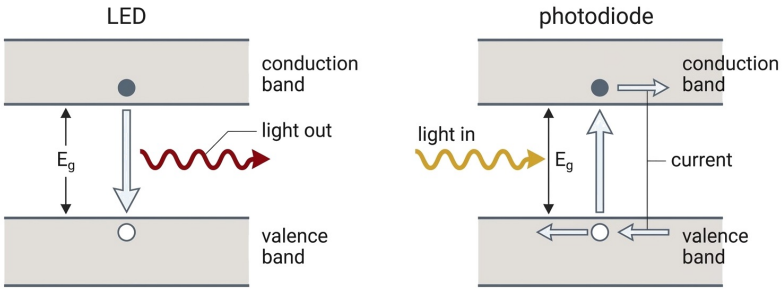


Figure 2.25: Two directions of one process. Left: in a forward-biased LED, electrons and holes meet at the junction and recombine, each emitting a photon of energy close to E_g. Right: in a reverse-biased photodiode, an absorbed photon creates an electron-hole pair that the junction field sweeps apart, producing a current proportional to the light.

trons pour from the n side into the p side and holes from the p side into the n side. Near the junction they meet. An electron at the bottom of the conduction band drops into a hole at the top of the valence band and gives up an energy of about E_g. In an LED that energy leaves as a photon (Figure 2.25), and Equation (2.7) fixes its color:

$$\lambda \approx \frac{1240}{E_g\,[\text{eV}]}\ \text{nm}. \tag{2.20}$$

A red LED needs a gap near 1.9 eV, a green one about 2.3 eV, a blue one about 2.7 eV (Figure 2.26). Real LEDs emit a band of wavelengths roughly 20 to 40 nm wide around this value, because the carriers are spread over a small range of energies near the band edges.

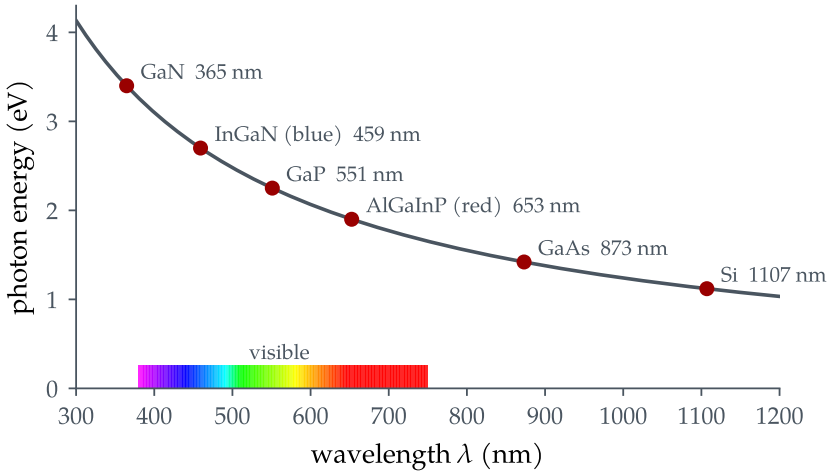


Figure 2.26: Photon energy versus wavelength, $E = 1240/\lambda$, with the band gaps of common optoelectronic materials marked. The colored strip is the visible range, about 380 to 750 nm.

WHY NOT SILICON? Silicon makes poor LEDs, and the reason is momentum. An electron in a crystal has both an energy and a crystal momentum, and a photon carries almost no momentum. For a photon to be emitted, the electron must drop to a hole with the same momentum. In a *direct-gap* semiconductor such as GaAs, InP, or GaN, the bottom of the conduction band and the top of the valence band sit

at the same momentum, so electrons can fall straight down and emit light efficiently. In an *indirect-gap* semiconductor such as silicon or germanium, they sit at different momenta, and recombination must also hand off momentum to a lattice vibration (a phonon). That three-body event is so slow that the electron almost always finds a non-radiative path first, through a defect, and its energy ends up as heat. LEDs are therefore made from III-V compounds: AlGaAs and AlGaInP for red and infrared, GaP and InGaN for green, and InGaN on GaN for blue.[3] Silicon remains an excellent *detector* because absorption does not face the same restriction as strongly.

Alloys let a designer slide E_g continuously between the gaps of their parent compounds (*band-gap engineering*). Raising the indium fraction x in $\mathrm{In}_x\mathrm{Ga}_{1-x}\mathrm{N}$, for instance, shrinks the gap from GaN's 3.4 eV and sweeps the emission from violet through blue to green. One LED chip emits one color; white LEDs, including the one on the demo board, coat a blue InGaN chip with a phosphor that converts part of the blue into a broad yellow-green band.

[3] The 2014 Nobel Prize in Physics went to Isamu Akasaki, Hiroshi Amano, and Shuji Nakamura for efficient blue LEDs. Red and green LEDs had existed since the 1960s, but blue resisted for three decades: high-quality GaN crystals were hard to grow, and nobody could make p-type GaN until Amano and Akasaki activated magnesium dopants with electron-beam irradiation (1989) and Nakamura found that simple thermal annealing worked (1992).

Electrically an LED is just a diode, with a larger turn-on voltage because its gap is larger. Its forward voltage at a typical 20 mA grows with photon energy (Figure 2.27): about 1.2 V for infrared, 1.6 to 2.0 V for red, about 2.1 V for yellow, and 2.5 to 3.5 V for blue, green (InGaN), and white. A useful rule of thumb is that qV_F at normal operating current is close to the photon energy, since every emitted photon is paid for by one electron crossing the junction. (At very low current qV_F can sit slightly below it, because thermal energy supplies the difference.) The white LED on the demo board, whose blue chip emits photons of about 2.7 eV, already carries 1 mA at about 2.6 V.

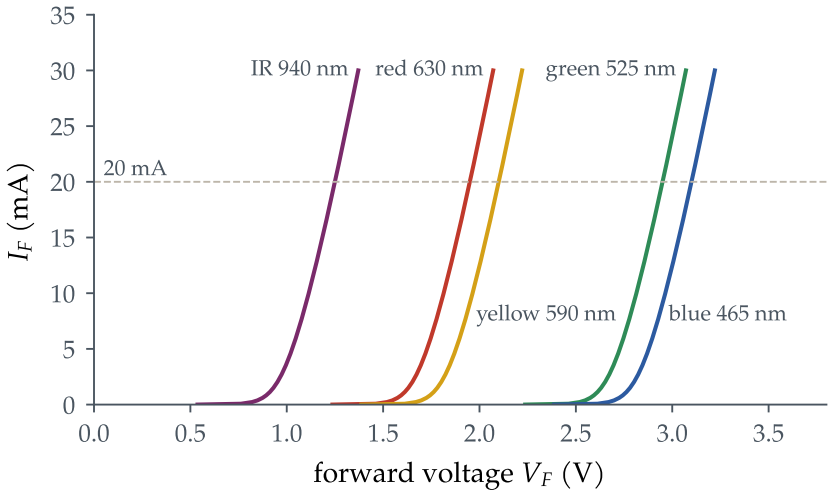


Figure 2.27: Representative *I-V* curves of LEDs of different colors. The forward voltage at 20 mA rises with photon energy. Exact values vary by manufacturer; always check the data sheet.

Because the LED's current rises exponentially, it must never be connected directly across a voltage source: a few tenths of a volt too much would multiply the current many times and burn it out. Instead it is driven through a series resistor that sets the current. The design equation follows from KVL with the constant-drop model:

$$R_S = \frac{V_{IN} - V_F}{I_{LED}}. \tag{2.21}$$

Brightness is roughly proportional to current, so this resistor is also the brightness control.

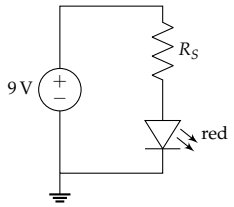


Figure 2.28: LED with a current-limiting resistor.

Worked Example 2.25 *Current-limiting resistor for an LED*

A red LED with $V_F = 1.8\,\text{V}$ is to run at 20 mA from a 9 V battery (Figure 2.28). Find R_S and the power in the resistor and the LED. Then find R_S for a 10 mA red LED ($V_F = 2.0\,\text{V}$) driven from a 3.3 V Arduino pin.

Solution. KVL around the loop: $V_{IN} = V_F + IR_S$, so

$$R_S = \frac{9 - 1.8}{0.020} = \frac{7.2}{0.020} = 360\,\Omega.$$

The resistor dissipates $(0.020)^2(360) = 0.144\,\text{W}$, so a quarter-watt (0.25 W) part is needed. The LED takes $(1.8)(0.020) = 36\,\text{mW}$. Only $1.8/9 = 20\,\%$ of the battery's energy reaches the LED; the rest heats the resistor. That is the price of a resistor driver, and why battery products use more efficient switching LED drivers.

From the 3.3 V pin: $R_S = (3.3 - 2.0)/0.010 = 130\,\Omega$. Arduino pins can supply only a few milliamperes to around 10 mA safely; for larger currents, switch the LED with a transistor as in the NMOS switch example of Section 2.7. ■

Worked Example 2.26 *LED color from band gap*

(a) What wavelength does an LED with $E_g = 2.0\,\text{eV}$ emit, and what color is it? (b) The red LED on a pulse oximeter emits at 660 nm. What band gap does its active layer need?

Solution. (a) $\lambda = 1240/2.0 = 620\,\text{nm}$, which is orange-red. (b) $E_g = 1240/660 = 1.88\,\text{eV}$. AlGaInP or AlGaAs alloys can be tuned to this value; silicon (1.12 eV) could only emit, inefficiently, near 1100 nm, far into the infrared. ■

2.10 Photodiodes, Solar Cells, and Phototransistors

Run the LED process backward and you have a photodetector. A photon with energy $h\nu \geq E_g$ that enters a semiconductor can be absorbed, lifting an electron into the conduction band and leaving a hole: it creates an electron-hole pair (Figure 2.25, right). In a block of uniform silicon the pair simply recombines a few microseconds later, and the light has done nothing useful. Inside or near the depletion region of a pn junction, however, the built-in field separates the pair before it can recombine: the electron is swept to the n side and the hole to the p side. Each absorbed photon therefore pushes about one electron of charge around the external circuit. This light-generated *photocurrent* I_{ph} flows from cathode to anode inside the diode, the reverse direction, and is proportional to the optical power.

How much current does a given light power produce? A beam of power P at wavelength λ carries $P/(hc/\lambda)$ photons per second. If a fraction η of them (the *quantum efficiency*) each produce one electron in the circuit, the photocurrent is

$$I_{ph} = \mathcal{R}\,P, \qquad \mathcal{R} = \frac{\eta\, q\lambda}{hc} = \eta\,\frac{\lambda\,[\text{nm}]}{1240}\ \text{A/W}. \tag{2.22}$$

The *responsivity* $\mathcal{R}$ grows with wavelength (longer wavelength means more photons per watt) until the photon energy drops below E_g, at which point the material becomes transparent and $\mathcal{R}$ falls to zero. For silicon that cutoff is near 1100 nm, and peak sensitivity lies around 800 to 1000 nm.

The total diode current is the ordinary diode equation minus the photocurrent:

$$I = I_S\left(e^{V/nV_T} - 1\right) - I_{ph}. \tag{2.23}$$

Light shifts the whole I-V curve downward by I_{ph} (Figure 2.29), and the operating point determines what kind of device you have.

Photoconductive mode. With a reverse bias, the curve is flat and the current equals $-(I_S + I_{ph})$. The dark current I_S is tiny, so the measured current is essentially the photocurrent, and it is proportional to light over many decades. The reverse bias also widens the depletion region, which collects more photons and lowers the junction capacitance, making the detector fast. This is how photodiodes are used for sensing and communication. The usual readout is a *transimpedance amplifier*, an op-amp with a feedback resistor R_F that converts the current to a voltage $V_{out} = I_{ph}R_F$ while holding the photodiode at a fixed bias. Section 2.11 shows the version on the demo board, and Chapter 3 uses it to read your pulse.

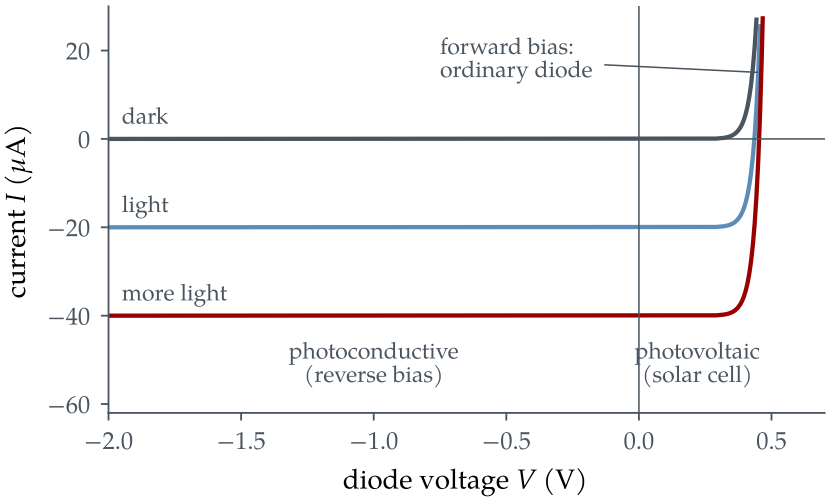


Figure 2.29: I-V curves of a photodiode in the dark and under two light levels (photocurrent 20 and 40 µA). In the third quadrant (reverse bias) the current is flat and proportional to light. In the fourth quadrant ($V > 0$, $I < 0$) the device delivers power: it is a solar cell.

Worked Example 2.27 *Photocurrent and readout voltage*

A silicon photodiode with quantum efficiency 0.8 receives 50 µW of 850 nm light. Find its responsivity, the photocurrent, and the output of a transimpedance amplifier with $R_F = 47\,\text{k}\Omega$.

Solution. From Equation (2.22), the responsivity is

$$\mathcal{R} = 0.8 \times \frac{850}{1240} = 0.548\,\text{A/W}.$$

The photocurrent is $I_{ph} = (0.548)(50 \times 10^{-6}) = 27.4\,\mu\text{A}$, and the amplifier output is

$$V_{out} = (27.4 \times 10^{-6})(47 \times 10^{3}) = 1.29\,\text{V},$$

well suited to the Arduino's 0 to 3.3 V input range. For scale, 50 µW at 850 nm is 2.1×10^{14} photons per second, and 80 % of them become electrons. ■

The photodiode on the demo board is a silicon PIN photodiode (Vishay VEMD2023SLX01): a pn junction with a wide, lightly doped *intrinsic* layer between the p and n regions so that most photons are absorbed inside the depleted zone. Its chip area is only $0.23\,\text{mm}^2$, and a daylight-blocking filter limits its response to roughly 750 to 1050 nm, peaking near 940 nm. Its data sheet shows the two properties discussed above as straight lines on log axes: the reverse light current rises in proportion to irradiance (about 10 µA at $1\,\text{mW/cm}^2$ with a 5 V reverse bias), while the dark current, around a nanoampere at room temperature, grows about tenfold for every 25 °C. The second fact is the steep temperature dependence of n_i (Figure 2.4) showing up in

a real part: dark current comes from carriers generated by heat, and their number grows with n_i. In Chapter 3 the board's LED and photodiode pairs and its optical pulse sensor put these ideas to work.

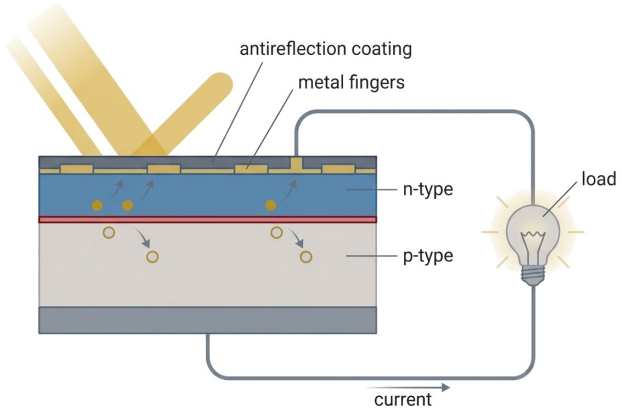


Figure 2.30: A silicon solar cell. Sunlight enters through the thin top n layer and its antireflection coating; electron-hole pairs created near the junction are separated, and the cell drives current through an external load. The metal fingers on top collect the current while shading as little of the surface as possible.

PHOTOVOLTAIC MODE: THE SOLAR CELL. Leave the photodiode unbiased and connect it to a load resistor. Now the photocurrent flows out through the load and builds up a positive voltage across the diode, which forward biases the junction and lets some of the photocurrent leak back through the diode. The operating point lands in the fourth quadrant of Figure 2.29, where $V > 0$ and $I < 0$: the device *delivers* power instead of absorbing it. That is a solar cell (Figure 2.30), which is simply a large-area photodiode.

Solar cells are usually described with the current sign flipped, so that the delivered current is positive: $I = I_{ph} - I_S(e^{V/nV_T} - 1)$ (Figure 2.31). Two points anchor the curve. With the terminals shorted ($V = 0$), the full photocurrent flows: the *short-circuit current* $I_{sc} = I_{ph}$. With no load ($I = 0$), the photocurrent all flows back through the diode, and the voltage rises to the *open-circuit voltage*

$$V_{oc} = nV_T \ln\left(\frac{I_{ph}}{I_S} + 1\right). \tag{2.24}$$

Neither point delivers any power, since $P = VI$ and one factor is zero. Between them lies the *maximum power point*, where the rectangle under the curve has the largest area. The ratio $P_{max}/(V_{oc}I_{sc})$ is the *fill factor*, typically 0.75 to 0.85 for good silicon cells, and the *efficiency* is P_{max} divided by the incident optical power. Commercial silicon panels reach about 20 to 23 %. Because V_{oc} depends only logarithmically on light, a silicon cell produces close to 0.6 V in bright sun and only a little less on a cloudy day, while its current scales directly with the light. Cells are wired in series to raise the voltage and in parallel to raise the current.

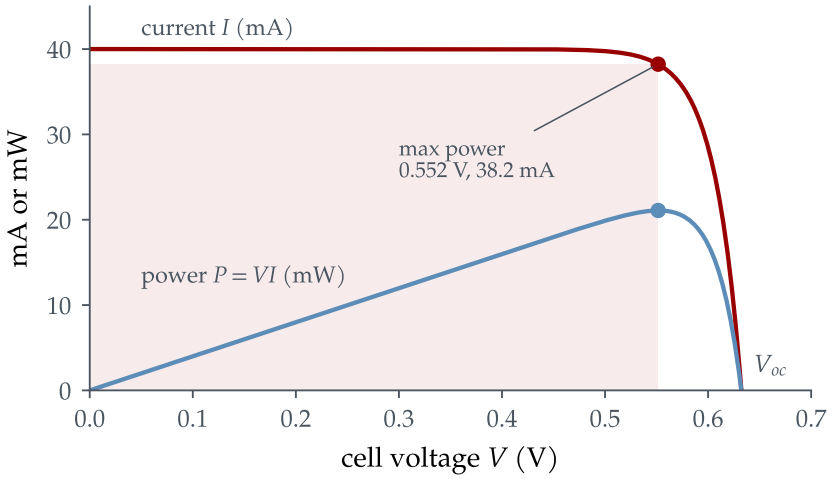


Figure 2.31: Solar cell output with $I_{ph} = 40\,\text{mA}$, $I_S = 10^{-12}\,\text{A}$, and $n = 1$. The shaded rectangle is the maximum delivered power, about 21 mW at 0.55 V.

Worked Example 2.28 *Solar cell operating points*

A small silicon cell has $I_{ph} = 40\,\text{mA}$ in full sun, $I_S = 10^{-12}\,\text{A}$, and $n = 1$; use $V_T = 25.9\,\text{mV}$. Find V_{oc}, the maximum power and the load resistance that extracts it, the fill factor, and the efficiency if the cell receives 100 mW of sunlight.

Solution. Open circuit:

$$V_{oc} = 0.0259 \ln\left(\frac{0.040}{10^{-12}} + 1\right) = 0.0259 \times 24.41 = 0.632\,\text{V}.$$

Short circuit: $I_{sc} = 40\,\text{mA}$.

The delivered power is $P = VI$ with $I = I_{ph} - I_S(e^{V/V_T} - 1)$. It peaks where $dP/dV = 0$, an equation with no closed-form solution, so evaluate P numerically:

V (V)	0.50	0.53	0.55	0.57	0.60
I (mA)	39.8	39.2	38.3	36.4	28.5
P (mW)	19.9	20.8	21.1	20.7	17.1

The maximum is $P_{max} = 21.1\,\text{mW}$ at $V_{mp} = 0.552\,\text{V}$ and $I_{mp} = 38.2\,\text{mA}$. The matching load is $R = V_{mp}/I_{mp} = 14.4\,\Omega$. The fill factor is $21.1/(0.632 \times 40) = 0.83$, and the efficiency is $21.1/100 = 21\,\%$. ■

Worked Example 2.29 *The same cell on a cloudy day*

Clouds cut the light on the cell of the previous example to one tenth, so $I_{ph} = 4\,\text{mA}$. Find the new open-circuit voltage and estimate the maximum power, assuming the fill factor stays at 0.83.

SOLUTION.

$$V_{oc} = 0.0259 \ln\left(\frac{0.004}{10^{-12}} + 1\right) = 0.0259 \times 22.11 = 0.573\,\text{V}.$$

While the current fell tenfold, the voltage fell by only 60 mV, which is $V_T \ln 10$, the voltage step for one decade of current. The maximum power is about $0.83 \times 0.573 \times 4 = 1.9\,\text{mW}$, roughly eleven times less than in full sun. A solar cell's voltage is a poor light meter; its current is an excellent one. ■

PHOTOTRANSISTORS AND LASERS. A *phototransistor* builds the light-sensitive junction into a transistor, which amplifies the photocurrent 100 to 1000 times at the cost of speed and linearity. An *optoisolator* pairs an LED with a photodetector in one package; the same pairing appears as a slotted coupler (an object breaks the beam) and as a reflective pair (light bounces off an object, as in the pulse sensor of Chapter 3). A *laser diode* is an LED inside a tiny mirrored cavity; above a threshold current, stimulated emission produces a narrow, coherent beam for fiber links and optical drives.

2.11 Op-Amps: Turning Currents into Voltages

THE ARDUINO CAN ONLY MEASURE VOLTAGES, and only between 0 and 3.3 V with respect to ground. The devices in this chapter, however, are described by their currents: the diode current, the drain current, the photocurrent. On the demo board, small circuits built around *operational amplifiers* (op-amps) turn each of these currents into a ground-referenced voltage that the ADC can read. This section introduces just enough about op-amps to understand those circuits and the lab that follows. Op-amps for sensor readout, including the inverting amplifier, the limits on gain and bandwidth, filtering, and the complete pulse-oximeter readout chain, are treated in more depth in Chapter 3 (Sensors).

What an op-amp is

An op-amp is an integrated-circuit amplifier with two inputs, a *non-inverting* input marked $+$ and an *inverting* input marked $-$, one output, and two supply pins (Figure 2.32). Its output is a greatly amplified copy of the difference between its inputs,

$$V_{out} = A\,(V_+ - V_-), \tag{2.25}$$

where the *open-loop gain* A is enormous, typically 10^5 to 10^6. The output can never go beyond the voltages on the supply pins. Many

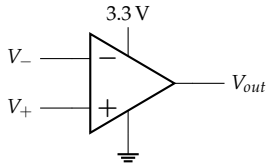


Figure 2.32: Op-amp symbol with the inverting ($-$) and non-inverting ($+$) inputs, the output, and the two supply pins. On the demo board the supplies are 3.3 V and ground, so the output stays between 0 and 3.3 V.

textbooks draw op-amps with a positive and a negative supply (for example 15 V and −15 V), so that the output can swing either way. The op-amps on the demo board run from a *single supply*: the positive supply pin is at 3.3 V and the negative supply pin is at ground. Their output therefore lives between 0 and 3.3 V, the same range as the ADC, and it can never be negative.

The board uses the TLV9102, which holds two op-amps in one small package. It is a *rail-to-rail* part: its inputs work anywhere from 0 to 3.3 V, and its output swings to within a few tens of millivolts of either supply (the "rails"). Its inputs draw only about 10 pA, and its input offset voltage, the small difference between the inputs at which the output is balanced, is typically 0.3 mV.

How large an input difference does it take to drive the output across its whole range? With $A = 10^5$, only $3.3/10^5 = 33\,\mu V$. Any realistic difference larger than that slams the output against one rail or the other, so an op-amp on its own makes a poor amplifier. It becomes useful when part of the output is fed back to the inverting input.

Two rules for negative feedback

When a circuit connects the output back to the inverting input (*negative feedback*), two rules give the behavior of an ideal op-amp:

Rule 1. No current flows into either input: $I_+ = I_- = 0$.

Rule 2. The output takes whatever value makes the two inputs equal: $V_- = V_+$.

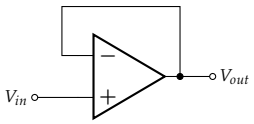
Rule 1 is a property of the chip: its input currents are only picoamperes, negligible next to the microamperes and milliamperes in the board's circuits. (Op-amps with MOSFET inputs owe this to the insulated gate of Section 2.7.) Rule 2 is a property of the feedback. If V_- were slightly below V_+, the huge gain would drive the output up; through the feedback path that raises V_-; and the output keeps moving until the difference has shrunk to microvolts, which for us is zero.

KEY IDEA With negative feedback: no input current, and $V_- = V_+$. Rule 2 holds only while the output it calls for lies between the rails.

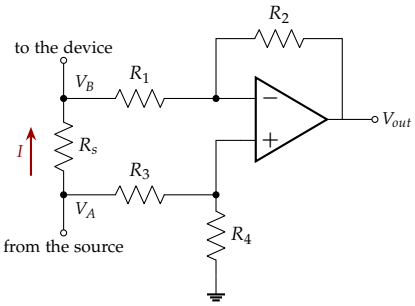
Rule 2 comes with a condition that matters on a single-supply board. It holds only while the output voltage it calls for lies inside the output range, here 0 to 3.3 V. If a circuit asks the output for −0.5 V or 4 V, the output stops at the rail, feedback can no longer pull the inputs together, and V_- no longer equals V_+. Every single-supply design must therefore pass one check: *is the output it asks for between the rails?*

The voltage follower and the non-inverting amplifier

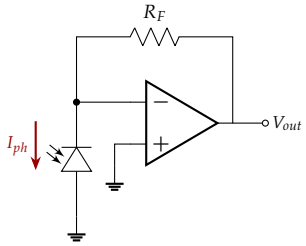
Connect the output straight back to the inverting input and drive the non-inverting input (Figure 2.33a). Rule 2 gives $V_{out} = V_- = V_+ = V_{in}$.



(a) voltage follower



(b) difference amplifier across a sense resistor



(c) transimpedance amplifier

Figure 2.33: The three op-amp circuits of the demo board. (a) Voltage follower: $V_{out} = V_{in}$, with no current drawn from the input. (b) Difference amplifier: with $R_2/R_1 = R_4/R_3$, $V_{out} = (R_2/R_1)(V_A - V_B)$; with four equal resistors it reports the voltage IR_s across the sense resistor. (c) Transimpedance amplifier: the photodiode's cathode sits on the virtual ground, its photocurrent is drawn out of that node and supplied through R_F, and $V_{out} = I_{ph}R_F$.

A gain of one sounds pointless, but this *voltage follower* is a *buffer*: by rule 1 it draws no current from whatever drives it, while its output can supply milliamperes to whatever it drives. It is the circuit to use whenever a node must be read or passed on without being disturbed. Put a divider in the feedback path instead, with R_i from the inverting input to ground and R_f from the output to that input, and the same two rules give the *non-inverting amplifier*,

$$V_{out} = \left(1 + \frac{R_f}{R_i}\right) V_{in}, \tag{2.26}$$

which Chapter 3 derives and uses. The follower is the special case $R_f = 0$ with R_i left out.

Worked Example 2.30 *A follower and a non-inverting amplifier on 3.3 V*

Two 10 kΩ resistors divide 3.3 V down to 1.65 V. (a) A 10 kΩ load is connected directly to the divider's midpoint. What voltage does it see? (b) The load is connected through a voltage follower instead. What does it see now, and where does its current come from? (c) A non-inverting amplifier with $R_f = 20\,\text{k}\Omega$ and $R_i = 10\,\text{k}\Omega$ runs from the board's single 3.3 V supply. Find its output for inputs of 0.40 and 1.50 V.

Solution. (a) The load is in parallel with the bottom resistor, and $10\,\text{k}\Omega \parallel 10\,\text{k}\Omega = 5\,\text{k}\Omega$, so the midpoint drops to $3.3 \times 5/(10+5) = 1.10$ V. Connecting the load pulled the node down by a third.

(b) The follower's input draws no current (rule 1), so the divider stays at 1.65 V, and the output copies it (rule 2). The load sees 1.65 V and draws $1.65/10\,\text{k}\Omega = 0.165\,\text{mA}$, all of it from the op-amp's output, and so from its supply, and none from the divider.

(c) The gain is $1 + 20/10 = 3$. An input of 0.40 V gives 1.20 V, inside the rails, so the formula holds. An input of 1.50 V asks for 4.50 V, which a 3.3 V supply cannot produce: the output stops near

3.3 V, feedback is lost, and the formula no longer applies. The largest input this amplifier handles is $3.3/3 = 1.1$ V. ■

The difference amplifier: reading a current

The simplest way to measure a current is to pass it through a small, known *sense resistor* R_s and measure the voltage across it, $I = (V_A - V_B)/R_s$. The trouble is that neither end of the sense resistor is at ground, while the ADC measures every voltage with respect to ground. The *difference amplifier* of Figure 2.33(b) does the subtraction before the ADC sees anything.

Take all four resistors equal to R. The non-inverting input sits on a divider from V_A, so $V_+ = V_A/2$, and rule 2 puts the inverting input there too, $V_- = V_A/2$. By rule 1, the current that flows from V_B through R_1 into the inverting node must continue through R_2 to the output:

$$\frac{V_B - V_-}{R} = \frac{V_- - V_{out}}{R} \quad \Longrightarrow \quad V_{out} = 2V_- - V_B = V_A - V_B.$$

(With matched pairs, $R_2/R_1 = R_4/R_3$, the output is $(R_2/R_1)(V_A - V_B)$.) The output is the voltage across the sense resistor, now referred to ground:

$$V_{out} = V_A - V_B = IR_s. \tag{2.27}$$

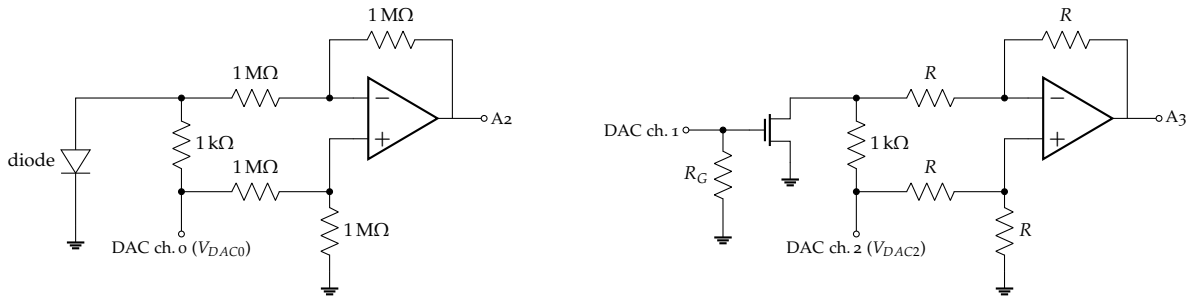
KEY EQUATION
$V_{out} = V_A - V_B = IR_s$
unity-gain difference amplifier across a sense resistor

The devices block uses this circuit twice, each time with $R_s = 1\,\text{k}\Omega$, so one volt at the output means one milliampere through the device; the diode's amplifier uses four 1 MΩ resistors. Three details make the circuit a good fit for the board. The current always flows from the DAC into the device, so $V_A \geq V_B$ and the output is never negative, which a single-supply op-amp requires. The megohm resistors draw only about a microampere from the circuit being measured. And because the subtraction happens before the ADC, the ADC's full resolution applies to the small difference. One ADC count is $3.3\,\text{V}/1024 = 3.22\,\text{mV}$, so the board resolves the current in steps of 3.22 µA and reads up to 3.3 mA.

WORKED EXAMPLE 2.31 *Reading the diode current*

In the diode circuit of Figure 2.34(a) the DAC is set to $V_{DAC} = 2.00$ V, and pin A2 reads 1.32 V. Find the diode current and voltage, the voltages at both op-amp inputs, and the current in the amplifier's input resistor on the diode side.

SOLUTION. The amplifier has unity gain, so the sense resistor carries $V_R = 1.32$ V and $I = 1.32/1\,\text{k}\Omega = 1.32$ mA. KVL around the loop gives $V_D = V_{DAC} - V_R = 2.00 - 1.32 = 0.68$ V, which is also the voltage at the anode node B. The non-inverting input



(a) diode: $I = V_{A2}/1\,\mathrm{k\Omega}$, $V_D = V_{DAC0} - V_{A2}$

(b) transistor: $I_D = V_{A3}/1\,\mathrm{k\Omega}$, $V_{DS} = V_{DAC2} - V_{A3}$

Figure 2.34: The two measurement circuits of the devices block. (a) DAC channel 0 drives the diode through a 1 kΩ sense resistor, and a unity-gain difference amplifier (four 1 MΩ resistors) reports the resistor voltage on pin A2. (b) DAC channel 2 feeds the drain through a second 1 kΩ sense resistor, the source is grounded, and an identical difference amplifier (four equal resistors R) reports the resistor voltage on pin A3. DAC channel 1 drives the gate; the pull-down R_G (1 kΩ) holds the gate at 0 V when the DAC is not driving it. In both circuits one volt on the analog pin is one milliampere.

sits halfway down a divider from the DAC, $V_+ = 1.00\,\mathrm{V}$, and rule 2 puts the inverting input there too, $V_- = 1.00\,\mathrm{V}$. The current in the 1 MΩ resistor from node B to the inverting input is $(0.68 - 1.00)/1\,\mathrm{M\Omega} = -0.32\,\mu\mathrm{A}$: $0.32\,\mu\mathrm{A}$ flows from the inverting node into node B. It is supplied by the output through the feedback resistor, $(1.32 - 1.00)/1\,\mathrm{M\Omega} = 0.32\,\mu\mathrm{A}$, as rule 1 requires. That $0.32\,\mu\mathrm{A}$ is the only current the amplifier adds to the measured circuit, a tenth of one ADC step, so the sense-resistor current and the diode current agree to well within the resolution. ■

Measuring the transistor current

Figure 2.34 shows both measurement circuits of the devices block. The transistor circuit (b) uses exactly the same scheme as the diode. DAC channel 2 feeds the drain through a 1 kΩ sense resistor, the source is grounded, and a second unity-gain difference amplifier across the sense resistor reports its voltage on pin A3. DAC channel 1 drives the gate directly. A pull-down resistor R_G of 1 kΩ from gate to ground holds the gate at 0 V, with the transistor off, whenever the DAC is not driving it (at power-up, for instance). The gate itself draws no current, so the gate voltage is exactly the DAC voltage. The three quantities the lab needs follow from the readings:

$$I_D\,[\mathrm{mA}] = V_{A3}\,[\mathrm{V}], \qquad V_{DS} = V_{DAC2} - V_{A3}, \qquad V_{GS} = V_{DAC1}. \tag{2.28}$$

The sense resistor also limits the current. Even with the transistor fully on, V_{DS} cannot fall below zero, so $I_D \le V_{DAC2}/1\,\mathrm{k\Omega} \le 3.3\,\mathrm{mA}$. That protects the transistor and the DAC and keeps A3 inside the ADC range. The price is that the DAC no longer sets V_{DS} directly: V_{DS} is whatever is left of V_{DAC2} after the drop across the resistor. Every point the circuit can reach lies on or below the line $I_D = (3.3\,\mathrm{V} - V_{DS})/1\,\mathrm{k\Omega}$ (Figure 2.36).

On a real board the drain path holds more than the sense resistor. A

transfer sweep on one board levels off at 0.62 mA with $V_{DAC2} = 3.3\,\text{V}$, not at 3.3 mA. A square-law model of the whole circuit fits that sweep only with a total drain-path resistance R_{path} of about 4.6 kΩ: the 1 kΩ sense resistor plus about 3.6 kΩ more between DAC channel 2 and the drain, with the fully-on channel itself taking the rest of the drop. A3 still reads the drain current, because the difference amplifier sits across the sense resistor alone, but then $V_{DS} = V_{DAC2} - I_D R_{path}$ and $I_D \leq V_{DAC2}/R_{path}$. Lab 2.1 (Section 2.12) measures R_{path} on your board as V_{DAC2}/I_{max}, a value that also takes in the few hundred ohms of the fully-on channel (5.3 kΩ on that board). The examples and problems of this chapter use the circuit as drawn, with the 1 kΩ resistor alone, unless they say otherwise.

WHY NOT A TRANSIMPEDANCE AMPLIFIER? A common way to read a current is to send it straight into the inverting input of an op-amp whose non-inverting input is grounded, with a feedback resistor R_F from the output to that input. Rule 2 holds the input node at 0 V, a *virtual ground*, and rule 1 forces the whole current through R_F. This *transimpedance amplifier* (TIA) gives $V_{out} = -I_{in}R_F$, where I_{in} is the current flowing *into* the virtual-ground node: the current leaves through R_F, and current flows from higher to lower potential, so the output must sit below 0 V. On a single positive supply that is impossible. A single-supply TIA with its non-inverting input at 0 V can read only a current that flows *out of* its input node, and the first layout of the board fell into exactly this trap.

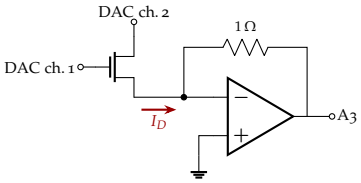


Figure 2.35: The transistor readout of the first board layout: a transimpedance amplifier with a 1 Ω feedback resistor, running from 3.3 V and ground. It could not work (Worked Example 2.32).

WORKED EXAMPLE 2.32 *Why the first transistor circuit read zero*

On the first layout of the board (Figure 2.35), DAC channel 2 drove the drain directly, DAC channel 1 drove the gate, and the source went to the inverting input of a TIA whose non-inverting input was grounded, with a feedback resistor of 1 Ω. The op-amp ran from 3.3 V and ground. Suppose the gate and the drain are both at 2.0 V and the transistor carries 2 mA. (a) What output does the TIA equation predict? (b) What does the single-supply op-amp actually do, and what does A3 read? (c) If the op-amp also had a negative supply, could the ADC resolve the current? (d) What limits the drain current in this circuit?

SOLUTION. (a) The drain current flows from drain to source and out of the source into the virtual-ground node. Rule 1 sends it on through R_F to the output, and since current flows from higher to lower potential, the output must sit below the node: $V_{out} = 0 - I_D R_F = -(2\,\text{mA})(1\,\Omega) = -2\,\text{mV}$.

(b) The output cannot go below its negative supply, which is

ground. It stops at about 0 V, and the feedback stops working: rule 2 fails, and nothing holds the inverting input at 0 V any more. The source current still reaches the output pin through the 1 Ω resistor, so the source sits about 2 mV above ground and the transistor conducts normally, but A3 reads 0 V (a count or two at most) whatever the current. The reading sticks near zero.

(c) No. With a negative supply the output would be negative, which the ADC cannot read at all, and its size would be only 1 mV per mA. One ADC count is 3.22 mV, so the current would have to exceed about 3.2 mA before the reading moved by a single count. A useful scale needs about 1 V per mA, that is, a resistance near 1 kΩ.

(d) Nothing but the transistor and the DAC. The drain went straight to the DAC output and the source to ground through 1 Ω. A small power transistor that is fully on has an on-resistance of ohms or less, so with the gate well above threshold the DAC output was nearly shorted to ground, and the current was set by how much the DAC could deliver, not by anything the experiment controlled.

The corrected circuit of Figure 2.34(b) fixes all three problems. The current flows from the DAC through the sense resistor, in the direction the difference amplifier reads with a positive output; the scale is 1 V per mA; and the resistor caps the current at 3.3 mA. ■

Worked Example 2.33 *From readings to an operating point*

The board's transistor is a DMN3404L, whose threshold lies between 1.0 and 2.0 V (typically 1.5 V). Three readings of the devices block, with the 1 kΩ sense resistor as the whole drain path, are

Reading	V_{DAC1} (V)	V_{DAC2} (V)	V_{A3} (V)
1	1.00	3.30	0.00
2	1.80	3.30	0.85
3	2.50	2.00	1.96

For each, find V_{GS}, I_D, and V_{DS}, and decide the region of operation.

Solution. Use Equation (2.28).

Reading 1: $V_{GS} = 1.00\,\text{V}$, I_D below one ADC step (3.2 µA), and $V_{DS} = 3.30 - 0.00 = 3.30\,\text{V}$. No measurable current flows: the gate is at or below threshold, and the transistor is in cutoff (any subthreshold current is too small to resolve).

Reading 2: $V_{GS} = 1.80\,\text{V}$, $I_D = 0.85\,\text{mA}$, and $V_{DS} = 3.30 - 0.85 = 2.45\,\text{V}$. The transistor conducts, so its threshold is below 1.80 V, and the overdrive $V_{GS} - V_{\text{th}}$ is at most $1.80 - 1.0 = 0.80\,\text{V}$ (0.30 V

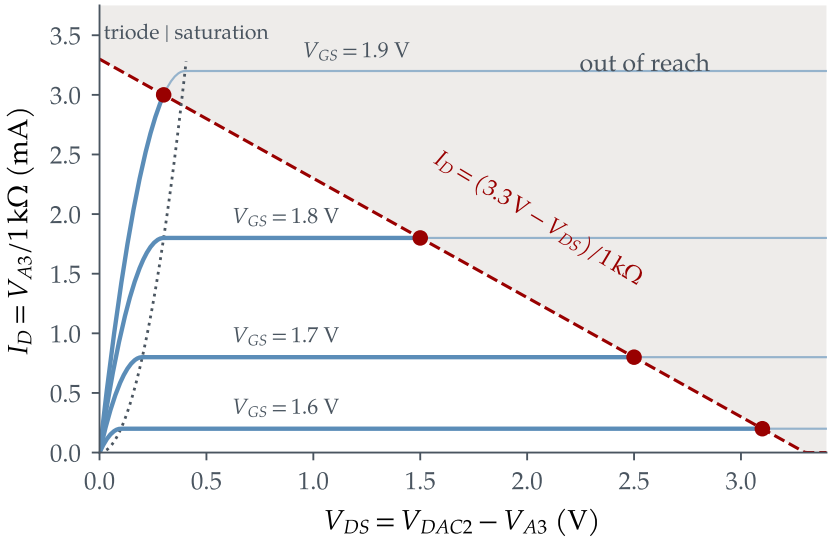


Figure 2.36: The measurement window of the transistor circuit. Thin curves: square-law output curves of an illustrative NMOS ($V_{\text{th}} = 1.5\,\text{V}$, $K = 20\,\text{mA/V}^2$). With the $1\,\text{k}\Omega$ sense resistor alone in the drain path, only points on or below the dashed line $I_D = (3.3\,\text{V} - V_{DS})/1\,\text{k}\Omega$ are reachable; the shaded region is out of reach, so an output sweep traces each curve only up to the line. Dots: the operating points of a transfer sweep at $V_{DAC2} = 3.3\,\text{V}$. The curve for $V_{GS} = 1.9\,\text{V}$ meets the line before it saturates.

for a typical part). V_{DS} exceeds the overdrive whatever the exact threshold, so the transistor is *saturated*: this point lies on the flat part of its output curve.

Reading 3: $V_{GS} = 2.50\,\text{V}$, $I_D = 1.96\,\text{mA}$, and $V_{DS} = 2.00 - 1.96 = 0.04\,\text{V}$. The overdrive is at least $2.50 - 2.0 = 0.50\,\text{V}$, far more than V_{DS}, so the transistor is deep in *triode*. It acts like a small resistor, and the sense resistor sets the current almost by itself: $I_D \approx V_{DAC2}/1\,\text{k}\Omega = 2.0\,\text{mA}$. Do not read an on-resistance from this point. Dividing gives $0.04/(1.96 \times 10^{-3}) \approx 20\,\Omega$, but the data sheet gives about $0.06\,\Omega$ for this part at $V_{GS} = 3\,\text{V}$, so the true V_{DS} is well under a millivolt. The 40 mV is the difference of two much larger numbers, the DAC setting and the A3 reading, and their calibration errors set it (a 1 % DAC gain error alone is 20 mV at 2 V). ■

Worked Example 2.34 *The measurement window*

Model the transistor with the square law, $V_{\text{th}} = 1.5\,\text{V}$ and $K = \frac{1}{2}k' = 20\,\text{mA/V}^2$ (illustrative values). During a transfer sweep the drain DAC stays at $V_{DAC2} = 3.3\,\text{V}$. Find the reading on A3 and the value of V_{DS} for $V_{GS} = 1.8\,\text{V}$ and for $V_{GS} = 1.9\,\text{V}$.

Solution. The sense resistor adds one equation to the transistor's: $I_D = (3.3 - V_{DS})/1\,\text{k}\Omega$, or $I_D\,[\text{mA}] = 3.3 - V_{DS}\,[\text{V}]$. The operating point is where this load line crosses the transistor's curve, exactly as for the diode in Figure 2.15.

$V_{GS} = 1.8\,\text{V}$: the overdrive is 0.3 V. Assume saturation: $I_D = 20(0.3)^2 = 1.80\,\text{mA}$. Then $V_{DS} = 3.3 - 1.80 = 1.50\,\text{V}$, which is at least 0.3 V, so the device is saturated and A3 reads 1.80 V.

$V_{GS} = 1.9\,\text{V}$: the overdrive is 0.4 V. Saturation would give $20(0.4)^2 = 3.20\,\text{mA}$ and $V_{DS} = 0.10\,\text{V} < 0.4\,\text{V}$, so the assumption fails. In triode, with currents in mA and voltages in V,

$$20\left(0.8V_{DS} - V_{DS}^2\right) = 3.3 - V_{DS} \implies 20V_{DS}^2 - 17V_{DS} + 3.3 = 0,$$

whose roots are $(17 \pm 5)/40$. The root 0.55 V exceeds the overdrive and is not a triode solution, so $V_{DS} = 0.30\,\text{V}$ and $I_D = 3.3 - 0.30 = 3.00\,\text{mA}$; A3 reads 3.00 V. Figure 2.36 shows both points. With the 1 kΩ resistor alone, the transfer curve would rise steeply above threshold and then bend over just below 3.3 mA, not because the transistor saturates but because the resistor takes over. A real board's longer drain path (about 4.6 kΩ) does the same at a lower current: its curve levels off near 0.6 mA (Lab 2.1). ■

The transimpedance amplifier: reading a photodiode

For a photodiode the TIA is the right circuit, because the photodiode can be connected so that its current flows in the direction a single supply allows (Figure 2.33c). Its cathode goes to the inverting input and its anode to ground. The photodiode then sits at zero bias, effectively short circuited, and its current is the photocurrent alone. Inside the photodiode the photocurrent flows from cathode to anode (Section 2.10), so it is drawn *out of* the virtual-ground node, and by rule 1 it can only come from the output, through R_F:

$$\frac{V_{out} - 0}{R_F} = I_{ph} \quad \Longrightarrow \quad V_{out} = I_{ph}R_F. \tag{2.29}$$

The ratio $V_{out}/I_{ph} = R_F$ has units of ohms but relates an output voltage to an input current, hence the name *transimpedance*. The biosensing block uses $R_F = 3\,\text{M}\Omega$, so one microampere of photocurrent gives 3 V. Chapter 3 adds the feedback capacitor that sets the bandwidth, the choice of R_F for a given light level, and the second stage that pulls a heartbeat out of the TIA output.

WORKED EXAMPLE 2.35 *The photodiode amplifier on the board*

A photodiode on the biosensing block feeds a TIA with $R_F = 3\,\text{M}\Omega$, read by the 10-bit ADC. (a) What output and ADC count does a photocurrent of 0.50 µA give? (b) What is the largest photocurrent the circuit can read, and how much current does one ADC count represent? (c) With a responsivity of 0.6 A/W at 940 nm,

what optical power fills the range?

SOLUTION. (a) $V_{out} = I_{ph}R_F = (0.50 \times 10^{-6})(3 \times 10^6) = 1.50\,\text{V}$, which is $1.50 \times 1024/3.3 = 465$ counts.

(b) The output tops out at 3.3 V, so $I_{max} = 3.3/(3 \times 10^6) = 1.1\,\mu\text{A}$. One count is $3.22\,\text{mV}/3\,\text{M}\Omega = 1.07\,\text{nA}$.

(c) $P = I_{max}/\mathcal{R} = (1.1 \times 10^{-6})/0.6 = 1.8\,\mu\text{W}$. A couple of microwatts is very little light. That is why the feedback resistor is so large: the circuit is built to see the faint light that makes it back out of a fingertip. ■

Circuit	Relation	Use on the demo board
Voltage follower	$V_{out} = V_{in}$	reading a node without loading it
Difference amplifier	$V_{out} = IR_s$	diode (A2) and transistor (A3): 1 V per mA
Transimpedance amplifier	$V_{out} = I_{ph}R_F$	photodiodes (A6): 3 V per µA

Table 2.3: The three op-amp circuits of this section. All run from 3.3 V and ground, so each must be wired so that its output is positive.

2.12 *Laboratory: Diode and Transistor I-V Characteristics*

Lab 2.1 at a glance

Goal. Sweep the voltages on the diode and the MOSFET of the devices block, read their currents through sense resistors and difference amplifiers, smooth the readings with a running average, and extract the diode's knee voltage, ideality factor, and series resistance and the MOSFET's drain-path resistance, threshold voltage, and saturation boundary.
Hardware. Demo board with the Arduino Nano 33 BLE Sense, USB data cable, laptop with the lab kit.
Notebook. `lab2.1-diode-transistor.ipynb` in the lab-kit folder `lab2.1-diode-transistor`. It saves each sweep to a CSV file in that folder: `diode_sweep.csv`, `mosfet_transfer.csv`, and `mosfet_output.csv`.
Before the lab. Review the diode equation and the constant-drop model (Sections 2.5 and 2.6), the MOSFET square law (Section 2.7), and the measurement circuits (Section 2.11). Then set `SIMULATE = True` in step A1 and run the notebook from top to bottom: every cell works on a simulated board, so you can write and test your fits at home. The simulated board copies one real demo board, so expect numbers close to, but not the same as, your board's.
Time: about 95 minutes: connecting (10 min), the diode (40 min), and the MOSFET (45 min).
You hand in. `lab2.1_submission.json` on the course portal; keep your notebook with your answers.

Key takeaways

- One ADC count is 3.2 µA through the 1 kΩ sense resistor; a 5-point running average cuts the noise 2.2-fold without bending the curves.
- The diode reaches 0.5 mA at 0.61 V and 2.5 mA at 0.77 V, so a 0.7 V constant drop is within 3 % for 2 to 3.3 V of drive.
- Two points a decade apart give $n \approx 3$, because IR_s (70 mV at 2.5 mA) is not junction voltage; a fit with R_s gives $n = 1.93$, $R_s = 28\,\Omega$.
- Measure the circuit, not the schematic: the drain path is 5.3 kΩ, not 1 kΩ, so I_D stops at 0.62 mA and $V_{DS} = V_{DAC2} - I_D R_{path}$.
- $\sqrt{I_D}$ is straight in V_{GS} while saturated: fit 0.05 mA to $0.6\,I_{max}$ for $V_{\text{th}} \approx 1.4$ V. Output curves saturate at $V_{DS} \approx V_{GS} - V_{\text{th}}$.

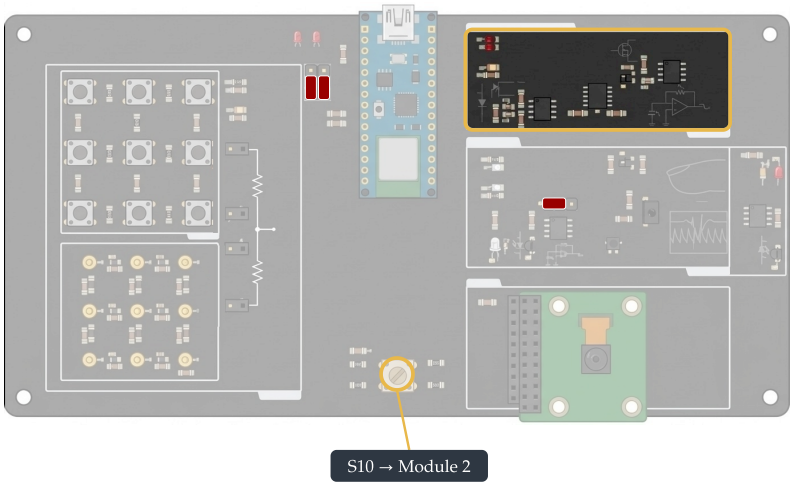
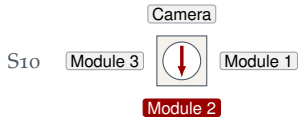


Figure 2.37: The board for Lab 2.1. Only the devices block (highlighted, upper right) is used. S10 is on Module 2; no jumper moves and nothing is plugged in.

SET UP THE BOARD FOR LAB 2.1

Selector. Turn S10 to Module 2 (devices block, upper right of the board; see Figure 0.16):



Jumpers. No change. The devices block does not use J5, J7, or J8; leave them where they are (the caps of J7 and J8 normally sit on the Module 1 end).

Plugged in. Nothing besides the USB cable. Module 2 does not power the circuits block, so parts left in the divider header from another lab do no harm.

Check. The first notebook cell runs `module` and confirms that the selector is on `Module 2` (devices block). If the reading does not fit, it prints a warning; check that S10 points at Module 2 and carry on. The power light of the devices block is on.

No jumper moves in this lab, so there is nothing to restore afterwards. Figure 2.37 shows the devices block and S10 on the board, and Figure 2.38 shows the lab running.



Objectives

- Use a DAC as a programmable voltage source, and a sense resistor with a difference amplifier and the ADC as a current meter.
- Lower the noise of a sweep by averaging readings and with a running average, without bending the curve.
- Measure the forward I-V curve of a diode; find its knee voltage, ideality factor, and series resistance; and compare the ideal, constant-drop, and exponential models with the data.
- Measure the transfer (I_D versus V_{GS}) and output (I_D versus V_{DS}) characteristics of an NMOS; find the current limit and the resistance of its drain path, and then V_{th}, K, and $V_{DS,sat}$.

Background

The two measurement circuits are those of Figure 2.34, explained in Section 2.11. The AD5593R chip (I^2C address `0x10`) supplies the voltages: its 12-bit DAC channels give 1.65 mV per code, so codes 0 to 2000 span 0 to 3.3 V. Channel 0 drives the diode, channel 1 the transistor's gate, and channel 2 the transistor's drain supply. Both currents are read the same way, through a 1 kΩ sense resistor and a unity-gain difference amplifier (four 1 MΩ resistors and an op-amp), so in both cases the reading in volts is the current in milliamperes:

$$\begin{aligned} &\text{diode:} && I = V_{A2}, && V_D = V_{DAC0} - V_{A2}; \\ &\text{transistor:} && I_D = V_{A3}, && V_{DS} = V_{DAC2} - I_D R_{path}, \quad V_{GS} = V_{DAC1}. \end{aligned}$$

One ADC count (3.22 mV) is 3.22 µA, and no current above 3.3 mA can be read, or even reached. Table 2.4 gives the names the notebook uses for each channel and pin.

The transistor line differs from Equation (2.28): R_{path} is the whole resistance between DAC channel 2 and the drain, not only the 1 kΩ sense resistor. On the demo board used to write this lab the drain current stops at 0.62 mA with 3.3 V on the drain supply, so the path measures about 5.3 kΩ, the fully-on channel included (Section 2.11).

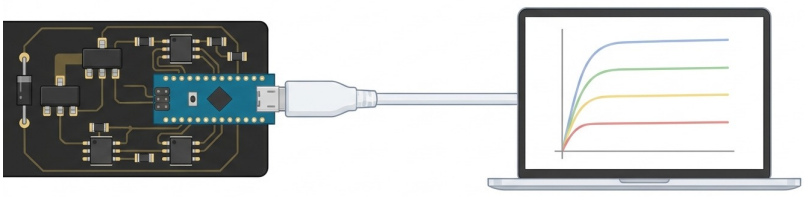


Figure 2.38: Lab 2.1 running. The DACs on the devices block set the voltages, the difference amplifiers report the currents, and the notebook plots the MOSFET's output curves as they arrive.

A3 still reads the sense resistor alone, so $I_D = V_{A3}/1\,\text{k}\Omega$ holds, but the lab measures R_{path} on your board (**C2**) before anything uses V_{DS}.

quantity	where it comes from	in the notebook
diode drive V_{DAC}	DAC channel 0, through the 1 kΩ sense resistor	`hw.DAC_DIODE`
resistor voltage V_R	difference amplifier (gain 1) to pin A2	`hw.PIN_DIODE`
gate voltage V_{GS}	DAC channel 1 (pull-down R_G to ground)	`hw.DAC_GATE`
drain supply V_{DAC2}	DAC channel 2, through the drain path R_{path} (the 1 kΩ sense resistor and more)	`hw.DAC_DRAIN`
$V_{A3} = I_D \times 1\,\text{k}\Omega$	difference amplifier (gain 1) to pin A3	`hw.PIN_NMOS`

Table 2.4: The channels and pins of the devices block and their names in the notebook. The source of the transistor is grounded.

Diode. With the series resistance R_s of the silicon and contacts included, the diode voltage at current $I \gg I_S$ is

$$V_D = nV_T \ln \frac{I}{I_S} + IR_s. \tag{2.30}$$

Where IR_s is negligible, two points give $n = \Delta V_D / [V_T \ln(I_2/I_1)]$. In this lab the *knee voltage* is the V_D at which the current reaches 0.5 mA. The constant-drop model predicts $I = (V_{DAC} - 0.7\,\text{V})/1\,\text{k}\Omega$.

Transistor. With $K = \frac{1}{2}k'$, saturation gives $\sqrt{I_D} = \sqrt{K}\,(V_{GS} - V_{\text{th}})$, a straight line in V_{GS} whose x-intercept is V_{th} and whose slope squared is K. Setting $V_{DS} = V_{GS} - V_{\text{th}}$ in the saturation equation shows that the triode and saturation regions meet on the parabola $I_D = KV_{DS}^2$. An output curve lies above this parabola in triode and below it in saturation, so it crosses the parabola at $V_{DS,sat}$. Keep the measurement window of Figure 2.36 in mind, with R_{path} in place of 1 kΩ: no measured point can lie above the load line $I_D = (V_{DAC2} - V_{DS})/R_{path}$, and the current can never exceed V_{DAC2}/R_{path}.

Noise and the running average. Each reading carries about one ADC count of random noise. The firmware averages 16 readings at every point of a sweep (`AVERAGE` in **A2**) and rounds the mean to a whole count. Then, right after each sweep, the notebook replaces every measured voltage x_k by the *centered running average* of the $n = 5$ points around it,

$$\bar{x}_k = \frac{1}{n} \sum_{j=-(n-1)/2}^{(n-1)/2} x_{k+j} = \frac{x_{k-2} + x_{k-1} + x_k + x_{k+1} + x_{k+2}}{5}, \tag{2.31}$$

with `ee105.running_average(x, n=5)`; at the two ends a point averages only the points that exist. The sum of n values grows n times while its random noise grows only $\sqrt{n}$ times, so the noise drops by $\sqrt{5}$, a factor of 2.2 (Figure 2.39b). Five points of a sweep in 16.5 mV steps span 82 mV, while the diode and transfer curves bend over a few

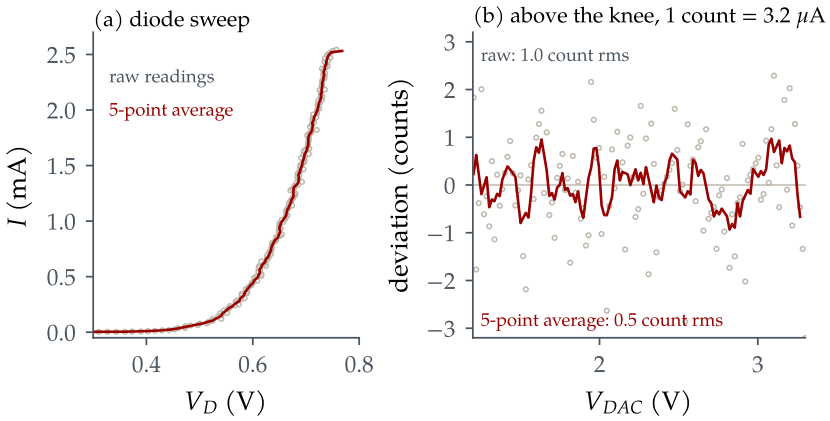


Figure 2.39: The 5-point running average on a real diode sweep (the `diode_sweep.csv` of a demo board, 4 readings per point). (a) Raw readings (circles) and the running average (line) lie on top of each other at full scale. (b) Above the knee, each value minus a smooth curve fitted to the whole sweep, in ADC counts: the raw readings scatter by 1.0 count rms, the running average by 0.5 count, a factor of 2.2, close to $\sqrt{5}$. The last two points of the average, whose windows are cut off, are not drawn in (b).

hundred millivolts, so the average does not bend them; only the cut-off window at the steep top of the diode sweep leaves the last point about 0.01 mA low. The notebook keeps the raw readings beside the smoothed ones, plots both, and uses the smoothed values in every later step.

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **B1**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT needs your code in the cell first, on the lines marked ✎ (Section 0.11 explains the tags). On the simulated board every step works. The simulated board copies one real demo board, the one whose numbers this section quotes, so its results are close to, but not the same as, yours.

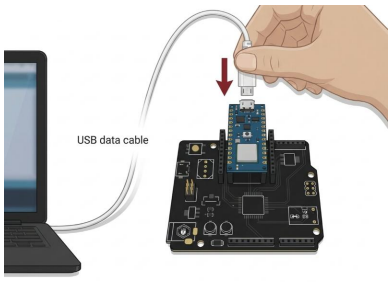
Part A: Connect

GOAL Connect the board on Module 2 and find the circuits of the devices block.

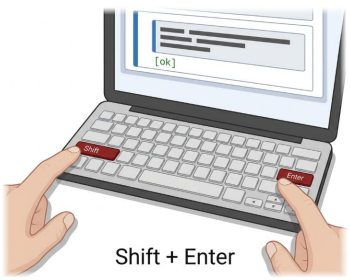
about 10 min

A1 ◆ DO Connect the board

Turn S10 to Module 2 (devices block); no jumper changes and no parts. Plug the Nano into your laptop with a USB *data* cable, click the first code cell (A1), and press Shift+Enter. The cell finds the board, uploads the course firmware if the board needs it, and checks the selector. No board today? Set `SIMULATE = True` in this cell to practice on a simulated board.



A1 Plug the Nano into your laptop with a USB *data* cable.



A1 Click the first code cell and press Shift+Enter.

YOU SHOULD SEE two lines that begin with `[ok]`: the board is connected (firmware `ee105`), and S10 is on `Module 2` (devices block). A `[!!]` line about the selector is only a warning: if S10 points at Module 2, carry on.

A2 ▸ RUN Constants of the measurement circuits

A2 sets the constants the analysis uses: the two sense resistors `R_SENSE` and `R_DRAIN` in kilohms, so that a voltage in volts divided by them is a current in milliamperes, the thermal voltage `VT`, the number of readings the firmware averages at every sweep point, `AVERAGE = 16`, and the length of the running average, `N_SMOOTH = 5`. If a board had other sense resistors, this cell is the only place to change.

YOU SHOULD SEE `sense resistors 1 kOhm (diode) and 1 kOhm (MOSFET drain), VT = 25.9 mV`, then one ADC count = 3.22 mV = 3.22 µA, the two averaging settings, and a reminder that a 1 kΩ drain path would allow 3.3 mA, while **C2** measures the real limit.

✓CHECK **Checkpoint 1** On the devices block, find the diode and its 1 kΩ sense resistor, the MOSFET and its 1 kΩ drain resistor, the gate pull-down resistor, and the two difference amplifiers (the TLV9102 op-amp and its 1 MΩ resistors). Match each one to Figure 2.34. You can point to every part of the two measurement circuits you are about to use.

Part B: The diode

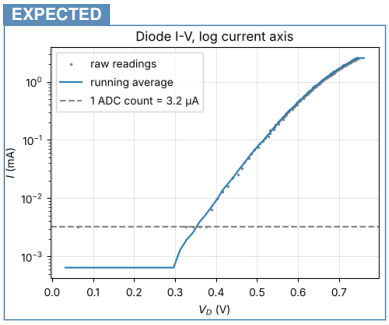
GOAL Sweep the diode and extract its knee voltage, ideality factor, and series resistance. about 40 min

The DAC drives the 1 kΩ resistor and the diode in series (Figure 2.34a). The difference amplifier reports the resistor voltage V_R, so the current is $I = V_R/1\,\text{k}\Omega$ (in mA when V_R is in volts), and KVL gives the diode voltage $V_D = V_{DAC} - V_R$.

B1 ▸ RUN Sweep the diode, smooth it, and plot it

`board.sweep` steps DAC channel 0 from 0 to 3.3 V in 16.5 mV steps (codes 0 to 2000 in steps of 10, 201 points), reads A2 at each step (the mean of 16 readings), and draws the points as they are measured. B1 then replaces V_R by its 5-point running average (Equation (2.31)), computes the current and the diode voltage from it, saves everything to `diode_sweep.csv`, and draws three plots. The raw reading stays in the table as `v_r_raw`; every later step uses the smoothed values. The plots show where the DAC voltage goes (V_R and V_D against V_{DAC}) and the *I-V* curve on linear and on logarithmic current axes, raw readings as gray dots and the running average as a line. B1 warns with `[!]`

EXPECTED



B1 The right panel of three. Above the dashed one-count line the curve runs nearly straight on the log axis up to about 0.3 mA and bends over above; below it the current is too small to read.

`Almost no current` if the largest current stays below 0.5 mA.

YOU SHOULD SEE `201 points saved to diode_sweep.csv, largest current 2.53 mA` on a real board, a scatter below one count, and almost no current until V_D reaches about 0.5 V, then a steep rise (Figure 2.41). Above the knee, V_D levels off near 0.7 V and V_R takes the rest of the DAC voltage.

B2 EDIT Knee voltage

B2 · EDIT in the notebook

```
1  v_knee = ...   # V_D (V) where I first reaches 0.5 mA
2  print(f"knee voltage (I = 0.5 mA): {v_knee:.3f} V")
```

WHAT TO WRITE `v_knee` (line 1), the diode voltage where the current first reaches 0.5 mA: find the first point at or above 0.5 mA and interpolate between it and the point before.

HINT `np.argmax(i_ma >= 0.5)` is the index `k` of the first `True`. `np.interp(0.5, xs, ys)` interpolates; take `xs` and `ys` from the two points `k - 1` and `k` of `i_ma` and `v_d`.

DONE WHEN the cell prints a knee voltage near 0.60 to 0.62 V (0.610 V on a real board).

B3 EDIT Ideality factor from two points

B3 · EDIT in the notebook

```
1  def nearest(i_target):
2      """(I in mA, V_D in V) of the point nearest i_target mA."""
3      k = np.argmin(np.abs(i_ma - i_target))
4      return i_ma[k], v_d[k]
5
6
7  def n_two_point(i_lo, i_hi):
8      """n from the points nearest i_lo and i_hi (mA)."""
9      (i1, v1), (i2, v2) = nearest(i_lo), nearest(i_hi)
10     return ...   # n = (V2 - V1) / (VT ln(I2 / I1))
11
12
13 for lo, hi in [(0.25, 2.5), (0.025, 0.25)]:
14     n2 = n_two_point(lo, hi)
15     print(f"n from the points near {lo} and {hi} mA: {n2:.2f}")
```

WHAT TO WRITE the return line of `n_two_point` (line 10). For $I \gg I_S$ the diode equation gives $V_D = nV_T \ln(I/I_S)$, so two points on the curve give

$$n = \frac{V_{D2} - V_{D1}}{V_T \ln(I_2/I_1)}.$$

`nearest` (lines 1 to 4) finds the measured point closest to a target current. The loop uses `n_two_point` twice: with the 0.25 and 2.5 mA points (the textbook's choice) and with two points ten times lower.

HINT `np.log` is the natural logarithm; `VT` is 0.0259 V.

DONE WHEN the first estimate is near 3 and the second one near 2 (a real board: 3.00 and 1.98).

Did you know? Each factor of 10 in diode current costs only about 60 mV ($n = 1$, 300 K). Raising the current from 1 to 10 mA moves the diode voltage by just 60 mV, which is why a conducting silicon diode can be treated as a nearly fixed drop.

The first estimate is well above 2, the largest value the physics allows. At high current part of the measured voltage is IR_s across the diode's series resistance, not the junction (**B4**, Question 2).

B4 EDIT Fit with series resistance

B4 · EDIT in the notebook

```
1  def diode_v(i_a, n, log_is, rs):
2      """V_D (V) at i_a (A); log_is = ln(I_S in A)."""
3      return n * VT * (np.log(i_a) - log_is) + i_a * rs
4
5
6  use = i_ma >= 0.02           # leave out the lowest points
7  p0 = (2, np.log(1e-9), 10)   # n = 2, I_S = 1 nA, Rs = 10 Ohm
8  p, cov = curve_fit(...)   # ✎ diode_v to (I in A, V_D)
9  n_fit, is_fit, rs_fit = ...   # ✎ n, I_S (A), Rs (Ohm)
10 print(f"fit with Rs, {use.sum()} points: n = {n_fit:.2f}, "
11       f"I_S = {is_fit:.2e} A, Rs = {rs_fit:.0f} Ohm")
12 print(f"I x Rs at 2.5 mA: {2.5 * rs_fit:.0f} mV")
```

WHAT TO WRITE `p, cov` from `curve_fit` (line 8), then `n_fit, is_fit, rs_fit` (line 9). The model `diode_v` is Equation (2.30), with $\ln I_S$ as its parameter (fits behave better when a parameter is not a tiny number like 10^{-9}). Fit it to every point of `use`, above 0.02 mA: the currents in amperes as `x`, the diode voltages as `y`. The last line prints IR_s at 2.5 mA, the voltage that inflates the two-point estimate of **B3**.

HINT call `curve_fit(diode_v, x, y, p0=p0)`; `p0` (line 7) starts the fit at $n = 2$, $I_S = 1\,\text{nA}$, $R_s = 10\,\Omega$. The second parameter is $\ln I_S$, so take `np.exp` of it.

DONE WHEN n is between 1 and 2 and R_s is a few tens of ohms. A full sweep on a real board gave $n = 1.93$, $R_s = 28\,\Omega$, and $I_S = 3.5\,\text{nA}$, so $IR_s = 71\,\text{mV}$ at 2.5 mA (the readings of Table 2.5 give $n \approx 1.3$ and $R_s \approx 56\,\Omega$).

B5 EDIT Constant-drop model

B5 · EDIT in the notebook

```
1  V_ON = 0.7                   # constant drop for silicon (V)
2  i_cd = ...   # ✎ mA for every v_dac; zero below V_ON
3
4  for v in [1.0, 2.0, 3.3]:
5      meas = np.interp(v, v_dac, i_ma)
6      cd = np.interp(v, v_dac, i_cd)
7      err = 100 * (cd - meas) / meas
8      print(f"V_DAC = {v:.1f} V: measured {meas:.2f} mA, "
9            f"constant drop {cd:.2f} mA ({err:+.0f} %)")
```

WHAT TO WRITE `i_cd` (line 2), the constant-drop prediction for every DAC voltage of the sweep. The model says a conducting diode has $V_D = V_{on} = 0.7\,\text{V}$, so in this circuit $I = (V_{DAC} - V_{on})/1\,\text{k}\Omega$ when $V_{DAC} > V_{on}$ and $I = 0$ otherwise. The loop compares the constant-drop current with the measurement at 1.0, 2.0, and 3.3 V.

HINT `np.maximum(x, 0)` sets negative values to zero, and `R_SENSE` is in kilohms, so the result is in milliamperes.

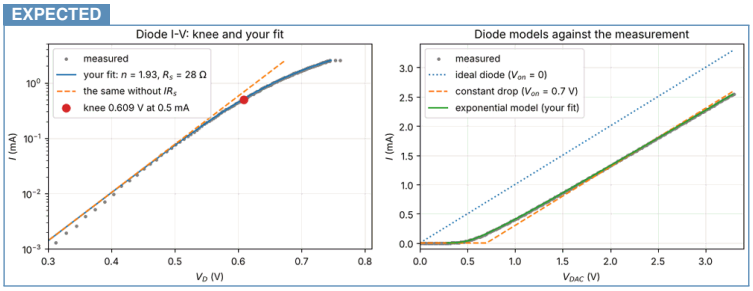
DONE WHEN the constant drop is within a few percent at 2.0 and 3.3 V (about 3% high at 3.3 V) and about 26% low at 1.0 V (Question 3).

B6 ▸ RUN Plot the fit and the diode models

B6 draws two plots: your data on a log current axis with the knee, your fit, and the same fit without the IR_s term; and the measurement against V_{DAC} with the ideal, constant-drop, and exponential models (Figure 2.13).

YOU SHOULD SEE the line `✓n = ...  with series resistance, between 1 and 2` at the end.

EXPECTED



B6 Left: your fit with R_s follows the data from 0.02 mA up, while the same fit without IR_s runs off to the left above about 0.3 mA. Right: the exponential model lies on the data, the constant drop close above about 2 V, and the ideal diode far off.

✓CHECK **Checkpoint 2** The diode plot shows the knee voltage (**B6**), and the fit with R_s gives n between 1 and 2 (**B6** prints ✓). You characterized a real diode from one sweep, the way a device engineer does.

Part C: The MOSFET

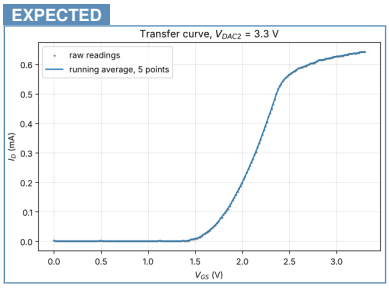
GOAL Measure the transistor's transfer curve and its drain path, then extract V_{th}, K, and $V_{DS,sat}$. about 45 min

DAC channel 1 sets the gate; the source is grounded, so V_{GS} is the gate DAC voltage. DAC channel 2 feeds the drain through the drain path, of which A3 sees only the 1 kΩ sense resistor (Figure 2.34b). So $I_D = V_{A3}/1\,\text{k}\Omega$, but $V_{DS} = V_{DAC2} - I_D R_{path}$, and the current can never exceed V_{DAC2}/R_{path}. Part C therefore sweeps the transfer curve first (**C1**), finds R_{path} from its top (**C2**), and only then computes V_{DS} (**C3**) for the threshold fit and the output curves.

C1 ▸ RUN Sweep the transfer curve and plot it

`board.transistor_transfer` holds the drain supply at $V_{DAC2} = 3.3$ V and sweeps the gate from 0 to 3.3 V in 16.5 mV steps (201 points, 16 readings each), drawing the curve as it is measured. C1 keeps the reading as `v_a3_raw`, replaces `v_a3` by its 5-point running average, computes $I_D = V_{A3}/1\,\text{k}\Omega$, saves the table to `mosfet_transfer.csv`, and plots I_D against V_{GS} on a linear and on a log current axis. The

EXPECTED



C1 The left panel of two: flat near zero up to about 1.5 V, a rise over about one volt, then a flat top near 0.6 mA, set by the drain path.

arrays `vgs` and `i_tr` hold V_{GS} and the smoothed I_D for the next steps.

YOU SHOULD SEE `201 points saved to mosfet_transfer.csv; largest current 0.624 mA at V_DAC2 = 3.30 V` on a real board, and a curve that stays near zero up to about 1.6 V and levels off near 0.62 mA above about 2.6 V. On the way it passes 0.19 mA at 1.98 V and 0.55 mA at 2.48 V. A 1 kΩ path would let it climb toward 3.3 mA: the flat top is the drain path, not the transistor.

C2 EDIT Current limit and drain-path resistance

C2 · EDIT in the notebook

```
1  full_on = vgs >= 3.0                # gate far above threshold
2  i_max = ...    # ✎ mean I_D (mA) of the full_on points
3  r_path = ...   # ✎ V_DAC2 / i_max (kOhm)
4  print(f"current limit I_max = {i_max:.3f} mA "
5        f"({full_on.sum()} points, V_GS >= 3.0 V)")
6  print(f"drain path R_path = {r_path:.2f} kOhm: "
7        f"{r_path - R_DRAIN:.2f} kOhm more than R_DRAIN")
```

WHAT TO WRITE `i_max` (line 2), the current limit: the mean I_D of the points with $V_{GS} \geq 3.0$ V (`full_on`, line 1); and `r_path` (line 3), the drain-path resistance $R_{path} = V_{DAC2}/I_{max}$ in kilohms. With the gate far above threshold the transistor is fully on, and almost all of V_{DAC2} drops along the path. The fully-on channel still has a resistance of its own, about $1/[2K(V_{GS} - V_{th})]$, a few hundred ohms here; V_{DAC2}/I_{max} lumps it into R_{path}, so those points end up at $V_{DS} \approx 0$.

HINT `i_tr[full_on]` picks the currents of those points, and `.mean()` averages them. `V_DAC2` holds the drain supply in volts; volts over milliamperes is kilohms.

DONE WHEN the cell prints $I_{max} = 0.62$ mA and $R_{path} = 5.3$ kΩ, 4.3 kΩ more than `R_DRAIN`, on a real board (the simulator: 0.63 mA and 5.2 kΩ).

C3 EDIT Write `id_vds(v_dac2, v_a3)`

C3 · EDIT in the notebook

```
1  def id_vds(v_dac2, v_a3):
2      """(I_D in mA, V_DS in V) from V_DAC2 and V_A3 (V)."""
3      i_ma = ...      # ✎ Ohm's law, 1 kOhm sense resistor
4      v_ds = ...      # ✎ KVL: supply minus the drop on R_path
5      return i_ma, v_ds
6
7
8  tr["i_ma"], tr["v_ds"] = id_vds(tr["v_dac2"], tr["v_a3"])
9  tr.to_csv("mosfet_transfer.csv", index=False)  # now with V_DS
10 vds_tr = tr["v_ds"].to_numpy()
11 print(f"V_DS from {vds_tr.max():.2f} V (off) "
12       f"to {vds_tr.min():.2f} V (fully on)")
```

WHAT TO WRITE the body of `id_vds` (lines 3 and 4): the drain current `i_ma` in mA and `v_ds` in volts, from the drain supply V_{DAC2} and the A3 reading. It must work on single numbers and on whole table columns. Lines 8 to 12 then add V_{DS} to the transfer table, save it again, and print the range of V_{DS}.

HINT Ohm's law for the sense resistor (`R_DRAIN` is in kilohms), and KVL around the drain loop with the whole path, `r_path` from C2. Plain arithmetic works on numbers and on columns alike.

Did you know? The first transistor, built at Bell Labs on December 16, 1947, by John Bardeen and Walter Brattain in William Shockley's group, was a slab of germanium touched by two closely spaced gold contacts. The three shared the 1956 Nobel Prize in Physics.

DONE WHEN the cell prints V_{DS} from 3.30 V (off) to about 0 V (fully on). A few hundredths of a volt below zero is normal: R_{path} was made from the mean of the fully-on points.

C4 EDIT Threshold voltage from $\sqrt{I_D}$

C4 · EDIT in the notebook

```
1  fit = (i_tr >= 0.05) & (i_tr <= 0.6 * i_max)  # saturated
2  vth = ...    # x-intercept of the line (V)
3  K = ...      # slope squared (mA/V^2)
4  k_bend = np.argmax(i_tr >= 0.9 * i_max)   # levels off here
5  rise = vgs < vgs[k_bend]
6  vth_250ua = float(np.interp(0.25, i_tr[rise], vgs[rise]))
7  saturated = bool(np.all(vds_tr[fit] >= vgs[fit] - vth))
8  print(f"{fit.sum()} points in the fit: Vth = {vth:.2f} V, "
9        f"K = {K:.3f} mA/V^2, k' = 2K = {2 * K:.3f} mA/V^2")
10 print(f"fit points saturated, V_DS >= V_GS - Vth: {saturated}")
11 print(f"V_GS at I_D = 0.25 mA: {vth_250ua:.2f} V")
```

WHAT TO WRITE `vth` and `K` (lines 2 and 3) of a straight line through $\sqrt{I_D}$ against V_{GS} for the points of `fit` (line 1): from 0.05 mA (about 15 ADC counts) up to 0.6 I_{max}, 0.37 mA on a real board. That is the rising part of the curve, where the transistor is surely saturated. Higher up, V_{DS} has fallen so far that the transistor leaves saturation and the path takes over. The x-intercept of the line is V_{th} and its slope squared is K. The cell then checks, with the V_{DS} of C3, that every point in the fit had $V_{DS} \geq V_{GS} - V_{\text{th}}$, and finds the gate voltage for 0.25 mA, the current at which the DMN3404L data sheet defines $V_{GS(th)}$ (1.0 to 2.0 V for the bare transistor).

HINT `np.polyfit(x, y, 1)` returns the slope m and the intercept b; the line $mx + b$ is zero at $x = -b/m$.

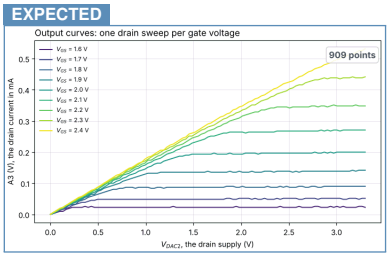
DONE WHEN the cell prints `True` for the saturation check. A real board gave 30 points in the fit, $V_{\text{th}} = 1.42$ V, $K = 0.58$ mA/V^2, and 2.07 V for 0.25 mA, above the data sheet's range: in the board's circuit the current rises far more slowly with V_{GS} than in the bare transistor (Question 5).

C5 ▸ RUN Output curves

C5 holds the gate at 1.6 to 2.4 V in 0.1 V steps (`VGS_STEPS`) and, for each, sweeps V_{DAC2} from 0 to 3.3 V in 33 mV steps (101 points, 16 readings each), one curve after another on a live plot. Because the current can never exceed V_{DAC2}/R_{path}, these gate voltages keep the saturation currents below that limit, about 0.02 to 0.55 mA by the fitted model. C5 smooths each drain sweep on its own with the running average, computes I_D and V_{DS} with your `id_vds`, and saves the table to `mosfet_output.csv`. If your I_{max} or your threshold differs much from these, shift `VGS_STEPS` so that the top step gives a saturation current $K(V_{GS} - V_{\text{th}})^2$ of about 0.9 I_{max} with your values from C4, and run C5 again.

YOU SHOULD SEE `909 points (9 drain sweeps) saved to mosfet_output.csv`, and the load line at $V_{DAC2} = 3.3$ V, at most 0.62 mA.

EXPECTED



C5 The live plot at the end: one curve per gate voltage against V_{DAC2}, each rising and then flat, the lowest near zero, none above about 0.6 mA.

C6 EDIT Write `vds_sat(vds, i_d)`

C6 · EDIT in the notebook

```
1  def vds_sat(vds, i_d):
2      """V_DS where I_D = K V_DS^2 is crossed (NaN if never)."""
3      above = i_d - K * vds ** 2    # > 0 triode, < 0 saturation
4      ...   # last index with above > 0; interpolate to zero
5
6
7  rows = []
8  for v_gs, g in out.groupby("v_gs"):
9      if v_gs > vth + 0.15:                # clearly on
10         vs = vds_sat(g.v_ds.to_numpy(), g.i_ma.to_numpy())
11         rows.append({"v_gs": v_gs, "v_gs_minus_vth": v_gs - vth,
12                      "vds_sat": vs})
13 sat_table = pd.DataFrame(rows)
14 sat_table["ratio"] = sat_table.vds_sat / (sat_table.v_gs - vth)
15 sat_table.round(3)
```

WHAT TO WRITE the body of `vds_sat` in place of line 4. In triode the current is $K[2(V_{GS}-V_{th})V_{DS}-V_{DS}^2]$, in saturation $K(V_{GS}-V_{th})^2$; the two meet on the parabola $I_D = KV_{DS}^2$. Line 3 computes `above`, positive where the curve lies above the parabola (triode) and negative below it (saturation). Find the last point where `above > 0` and interpolate between it and the next point to where `above` is zero (Figure 2.40). Return `np.nan` if the curve never crosses: a curve near threshold is saturated almost everywhere, and the highest one may meet the load line before it saturates (Question 6). The rest of the cell tabulates $V_{DS,sat}$ against $V_{GS}-V_{th}$ for every curve that is clearly on.

HINT `np.nonzero(above > 0)[0]` lists the triode points; take the last one, `k`. `np.interp(0, xs, ys)` needs increasing `xs`, and `above[k + 1]` is the smaller of the two values.

DONE WHEN the table shows $V_{DS,sat}$ close to $V_{GS}-V_{th}$ (a `ratio` near 1) for at least two curves, and `NaN` for a curve that never crosses (on the simulator, which copies a real board: ratios of 0.93 to 1.0 for 1.6 to 2.3 V, and `NaN` for 2.4 V).

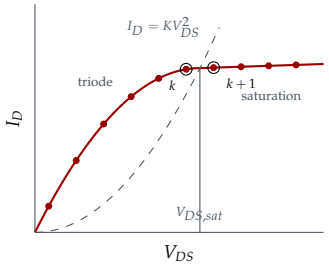


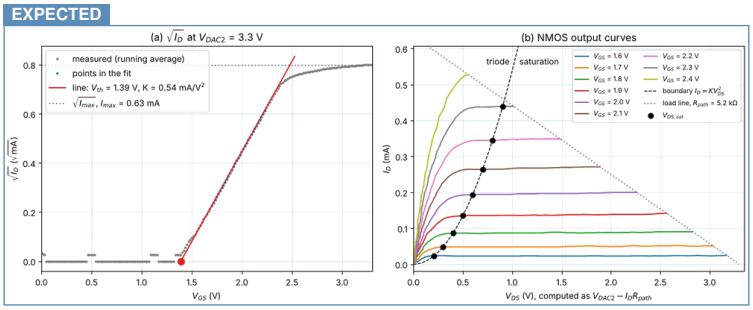
Figure 2.40: How `vds_sat` works. The points of an output curve lie above the dashed boundary in triode (`above` > 0) and below it in saturation. Point k is the last one above; interpolating `above` between k and $k+1$ to zero gives $V_{DS,sat}$.

C7 RUN Plot $\sqrt{I_D}$ and the output curves

C7 draws the two results of Part C side by side, as in Figure 2.42: (a) $\sqrt{I_D}$ against V_{GS} with the points in the fit, your line, and $\sqrt{I_{max}}$; (b) every output curve against the computed V_{DS}, with the boundary $I_D = KV_{DS}^2$, the load line of $V_{DAC2} = 3.3$ V through your R_{path}, and your $V_{DS,sat}$ points.

YOU SHOULD SEE the line ✓`V_th = ...  is inside the data sheet range` at the end.

EXPECTED



C7 (a) The fitted points lie on the line, which meets the axis near 1.4 V; the data bend over to $\sqrt{I_{max}}$ above about 2.5 V. (b) Each curve rises in triode, crosses the dashed boundary at a black dot, and flattens; the top curve meets the load line first, and no point lies above that line.

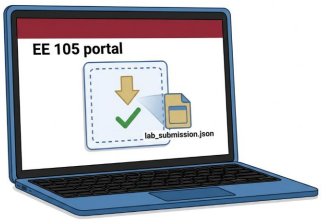
✓CHECK **Checkpoint 3** The output curves show both regions, triode on the left of the dashed boundary and nearly flat saturation on the right (**C7**), and the $\sqrt{I_D}$ fit gives a threshold in the data sheet's range (**C4**). You can read a transistor's behavior, and the circuit around it, straight off its curves.

If you have time. Fit a straight line to $\ln I$ against V_D between 0.02 and 0.3 mA, where IR_s is small: its slope is $1/(nV_T)$ and its intercept $\ln I_S$. A real board gives $n = 2.06$ and $I_S = 6.1$ nA, a little above the fit with R_s, because IR_s is already about 8 mV at 0.3 mA.

Save and upload

Save ▸ RUN Write `lab2.1_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab2.1_submission.json` in the lab folder: your fitted numbers and the raw sweeps behind them, from which the portal recomputes, with the same running average, the knee, R_{path}, and the threshold. Log in at `ee105.org/portal`, open Lab 2.1, and upload that file. The three CSV files stay in the lab folder for a report.



Save Upload `lab2.1_submission.json` on the course portal under Lab 2.1.

YOU SHOULD SEE `[ok] Saved lab2.1_submission.json`, and then the portal shows your grade. A run on the simulated board counts as practice and earns no credit.

FINISH THE LAB

You characterized the two devices every chip is built from: a diode I-V curve with its knee, ideality factor, and series resistance, and the transfer and output curves of a MOSFET with its drain path, V_{th}, K,

and $V_{DS,sat}$. The lab is done when every box is ticked.

☐	**B1**	▸ RUN	your own diode sweep, raw and smoothed, `diode_sweep.csv`
☐	**B2**	✎ EDIT	`v_knee`
☐	**B3**	✎ EDIT	`n_two_point`
☐	**B4**	✎ EDIT	`p, cov, n_fit, is_fit, rs_fit`
☐	**B5**	✎ EDIT	`i_cd`
☐	**C1**	▸ RUN	your own transfer sweep, `mosfet_transfer.csv`
☐	**C2**	✎ EDIT	`i_max, r_path`
☐	**C3**	✎ EDIT	`id_vds`
☐	**C4**	✎ EDIT	`vth, K`
☐	**C5**	▸ RUN	at least four output curves above 0.1 mA
☐	**C6**	✎ EDIT	`vds_sat`
☐		✓ CHECK	Checkpoint 1: the parts of both circuits found on the board
☐		✓ CHECK	Checkpoint 2: knee marked; n between 1 and 2 with R_s
☐		✓ CHECK	Checkpoint 3: both regions; V_{th} in the data-sheet range
☐			Questions 1 to 6 answered in the notebook
☐	**Save**	⭳ SAVE	`lab2.1_submission.json` uploaded on the portal

Typical results

Diode. Table 2.5 and Figure 2.41 show readings from one sweep on a demo board. The numbers in the list come from a full sweep on another board, the one that also gave Figure 2.39, taken through steps **B1** to **B5**. Your numbers will differ somewhat from board to board.

- Below about 0.4 V on the DAC the current is a count or two at most, and nearly all of the DAC voltage appears across the diode. The raw readings scatter by about one ADC count about the running average (0.92 count with 4 readings per point).
- The knee (0.5 mA, **B2**) lies near 0.60 to 0.62 V (0.610 V).
- The two-point estimate from the points near 0.25 and 2.5 mA is about $n = 3$ (**B3**). The fit with series resistance (**B4**) gives $n = 1.93$, $R_s = 28\,\Omega$, and $I_S = 3.5\,\text{nA}$ (the five readings of the table give $n \approx 1.3$, $R_s \approx 56\,\Omega$, and $I_S \approx 1 \times 10^{-11}$ A). On your board n should land between 1 and 2 and R_s at a few tens of ohms.
- The constant-drop model (**B5**) is within a few percent from $V_{DAC} =$ 2 to 3.3 V (1 % low at 2.0 V, 3 % high at 3.3 V: 2.60 mA against 2.53 mA), but about 26 % low at 1.0 V.

V_{DAC} (V)	I (mA)	V_D (V)
0.40	0.01	0.39
0.81	0.24	0.57
1.63	0.97	0.66
2.46	1.74	0.72
3.28	2.50	0.78

Table 2.5: Five of the diode readings plotted in Figure 2.41.

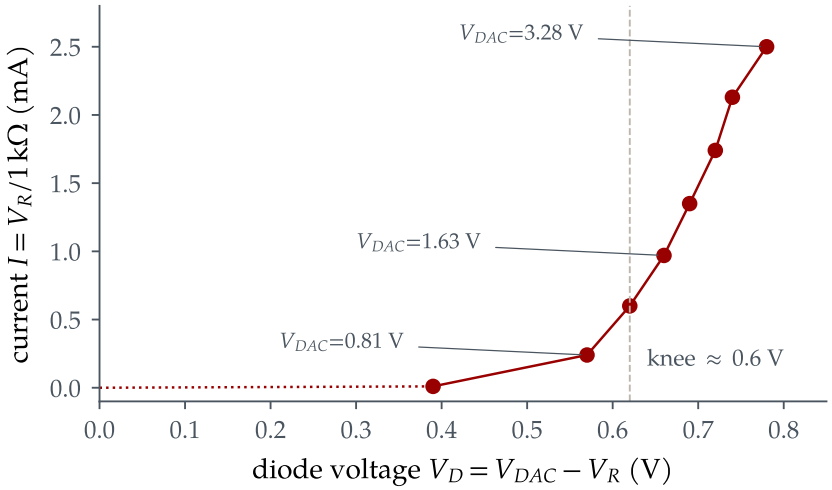


Figure 2.41: Representative diode data from one sweep on a demo board (every 25th point of the sweep). Below about 0.4 V the current is too small to resolve; the knee is near 0.6 V, and 2.5 mA needs only 0.78 V.

Transistor. Figure 2.42 shows the transfer sweep of the same real board and a set of output curves from the simulator, which copies that board.

- A3 reads zero, or a count or two, until V_{GS} nears threshold: below 0.01 mA up to $V_{GS} = 1.60$ V. Then the current rises over about one volt (0.19 mA at 1.98 V, 0.25 mA at 2.07 V, 0.55 mA at 2.48 V) and levels off at 0.62 mA above about 2.6 V: $R_{path} = 5.3\,\mathrm{k\Omega}$ (**C2**), 4.3 kΩ more than the sense resistor, about 0.5 kΩ of it the fully-on channel.
- The square-law fit over 0.05 to 0.37 mA (30 points, **C4**) gives $V_{\mathrm{th}} = 1.42$ V and $K = 0.58\,\mathrm{mA/V^2}$, with every point saturated. With the fit range of a 1 kΩ path, 0.1 to 1.5 mA, the fit would take in the flat top and give $V_{\mathrm{th}} = 0.26$ V.
- On the output curves, gate voltages near threshold give small, flat curves; from about 1.8 V up a triode rise is followed by a flat saturation plateau below the limit. The 2.4 V curve would need about 3.9 V of drain supply to saturate and ends on the load line in triode. No point lies above the line $I_D = (3.3\,\mathrm{V} - V_{DS})/R_{path}$.
- $V_{DS,sat}$ comes out close to $V_{GS} - V_{\mathrm{th}}$ (**C6**); not exactly, because the

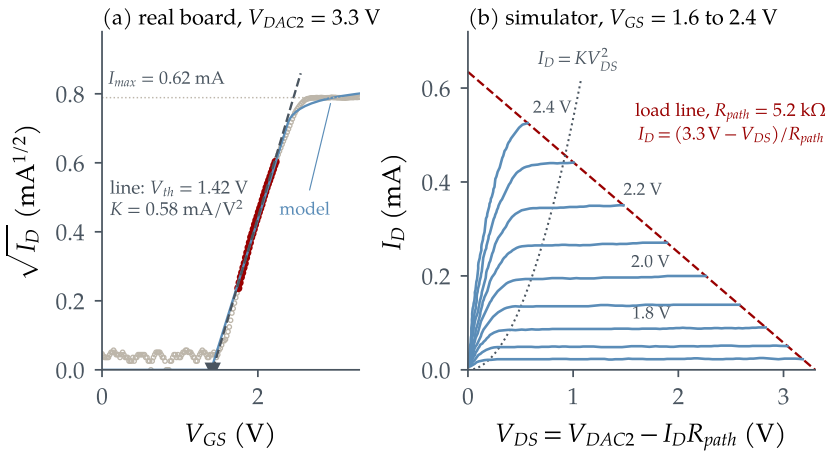


Figure 2.42: Transistor results. (a) The transfer sweep of a real demo board at V_{DAC2} = 3.3 V, after the running average, plotted as $\sqrt{I_D}$: near zero below about 1.6 V, a straight run through the fitted points from 0.05 mA to 0.6 I_{max} (dots), then flat at I_{max} = 0.62 mA, where the drain path, not the transistor, sets the current. The dashed line meets the axis at V_{th} = 1.42 V, and its slope squared is K = 0.58 mA/V². The thin curve is the fitted model of the whole circuit: the square law with V_{th} = 1.40 V and K = 0.53 mA/V² behind a 4.6 kΩ drain path. (b) Output curves from the simulator for V_{GS} = 1.6 to 2.4 V, analyzed as in **C5** to **C7**, with the simulator's own R_{path} = 5.2 kΩ. The curves rise in triode, cross the parabola $I_D = KV_{DS}^2$ (dotted) near $V_{DS,sat}$, and flatten; the 2.4 V curve meets the dashed load line before it saturates. Every curve ends on that line.

square law is only an approximation this close to threshold, and R_{path} includes the channel of the fully-on transistor.

Troubleshooting

- *All currents near zero, or the warning "Almost no current."* The selector is not on Module 2, or the devices block is not powered (its power light, M2 Power, is off). Turn S10 to Module 2 and run **A1** again.
- *The notebook cannot open the board.* Another program (a second notebook, a serial terminal, or the Arduino IDE) holds the serial port. Close it; only one program can use the port at a time.
- *I_D levels off near* 0.6 mA, *far below* 3.3 mA. That is the drain path, not a broken transistor: **C2** measures R_{path}, and **C3** uses it.
- *Your numbers differ from a classmate's or from the simulation.* That is normal: diodes, transistors, and drain paths vary from board to board. Any error in a DAC's gain also goes straight into V_D and V_{DS} (1 % at 3.3 V is 33 mV).
- *The lowest points scatter on the log plot.* Below about 0.02 mA a reading is only a few ADC counts, so one count of noise changes the current by tens of percent. Leave those points out of the fits.
- *`curve_fit` fails or returns n outside 1 to 2 (**B4**).* Pass `p0=p0`, and check that the currents are in amperes, not milliamperes.
- *V_{th} well below* 1 V *(**C4**).* The fit must use only 0.05 mA $\leq I_D \leq$ 0.6 I_{max}; points near I_{max} are limited by the drain path, not by the transistor.

- *V_{DS} looks wrong, or the output curves run past the load line.* $V_{DS} = V_{DAC2} - I_D R_{path}$, neither the DAC voltage nor $V_{DAC2} - V_{A3}$: check `r_path` (**C2**) and `id_vds` (**C3**).
- *Few output curves rise above* 0.1 mA*, or none of them saturates.* Shift `VGS_STEPS` in **C5** so that the top step gives about 0.9 I_{max} in saturation.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. What is the gain of the difference amplifiers in Figure 2.34? Why must the voltage across a sense resistor be measured differentially instead of with a single analog pin? What is the smallest current one ADC count resolves?
2. The two-point estimate from the 0.25 and 2.5 mA points (**B3**) gives an ideality factor near 3. Why is this unrealistic, and what does the fit with series resistance (**B4**) give? Why do the lowest-current points (0.01 mA and below) pull a fit so far that we leave them out?
3. Where does the constant-drop model (0.7 V, **B5**) predict the diode current well, and where is it worst? Why is it still the model used for hand analysis?
4. What current limit I_{max} and drain-path resistance R_{path} did you find (**C2**)? How much of R_{path} is not the 1 kΩ sense resistor, and why does R_{path} also hold some of the transistor's own resistance? Had you assumed $R_{path} = 1\,\text{k}\Omega$, what V_{DS} would you have computed for the fit point nearest 0.3 mA, and what would the saturation check of **C4** then be worth?
5. What V_{th} and K did you get from the $\sqrt{I_D}$ fit (**C4**), and what is k'? Is V_{th} inside the data-sheet range, and why is the gate voltage for 0.25 mA higher? (Problem 2.31 works through this with sample numbers.)
6. On your output curves (**C7**), mark the triode and saturation regions and compare $V_{DS,sat}$ with $V_{GS} - V_{\text{th}}$ (**C6**). Why does the curve for the highest gate voltage end on the load line before it saturates? (Find the V_{DAC2} it would need.)

Summary

- $\sigma = q(n\mu_n + p\mu_p)$. Metals hold 10^{22} to 10^{23} carriers per cm^3, insulators essentially none, and semiconductors 10^{10} to 10^{20}, set by doping.

- Pure silicon has $n_i \approx 10^{10}\,\text{cm}^{-3}$ at 300 K. Donors give $n \approx N_D$, acceptors $p \approx N_A$ (the net value when both are present), and always $np = n_i^2$.
- Silicon has a band gap of 1.12 eV. A gap E_g absorbs and emits light near $\lambda\,[\text{nm}] = 1240/E_g\,[\text{eV}]$.
- A pn junction has $V_{bi} = V_T \ln(N_A N_D / n_i^2)$. Forward bias lowers the barrier; reverse bias raises it and widens the depletion region as $\sqrt{V_{bi} + V_R}$.
- $I_D = I_S(e^{V_D/nV_T} - 1)$, with $V_T = 25.9\,\text{mV}$ at 300 K: a decade per $2.3\,nV_T$ (60 mV for $n = 1$), about 0.7 V when conducting, and $r_d = nV_T/I_D$.
- A MOSFET is off below V_{th}, a tunable resistor in triode, and a current source $\frac{1}{2}k'(V_{GS} - V_{\text{th}})^2$ in saturation ($V_{DS} \geq V_{GS} - V_{\text{th}}$). A PMOS obeys the same laws with all voltages reversed. CMOS draws almost no static power; dynamic power is $\alpha N C V_{DD}^2 f$.
- LEDs need direct-gap III-V materials and a series resistor $(V_{IN} - V_F)/I$. Photodiodes give $I_{ph} = \mathcal{R}P$; solar cells deliver power between I_{sc} and V_{oc}.
- With negative feedback an ideal op-amp draws no input current and holds $V_- = V_+$, but only while the output it needs lies between the rails (0 to 3.3 V on the board). A difference amplifier across a 1 kΩ sense resistor reads 1 V per mA; a single-supply TIA gives $V_{out} = I_{ph} R_F$ for a current drawn out of its input node.
- On the devices block, $I_D\,[\text{mA}] = V_{A3}\,[\text{V}]$, $V_{DS} = V_{DAC2} - I_D R_{path}$, and $I_D \leq V_{DAC2}/R_{path}$; with the 1 kΩ sense resistor alone $R_{path} = 1\,\text{k}\Omega$, and on a real board about 4.6 kΩ plus the fully-on channel (the current levels off near 0.6 mA).

2.13 *Problems*

2.1 ◦ Silicon at 300 K is doped with $N_A = 5 \times 10^{16}$ boron atoms per cm^3. Is it n-type or p-type? Find the majority and minority carrier densities.

2.2 ◦ A region of silicon at 300 K contains $N_D = 4 \times 10^{16}\,\text{cm}^{-3}$ donors and $N_A = 1 \times 10^{16}\,\text{cm}^{-3}$ acceptors. Is it n-type or p-type? Find its electron and hole densities.

2.3 ◦ The resistance of a pure silicon bar falls when it is heated, while the resistance of a copper wire rises. Explain both behaviors in terms of carrier density and mobility.

2.4 • You need a $10\,\text{k}\Omega$ resistor on a chip, made from a strip of n-type silicon with $N_D = 10^{16}\,\text{cm}^{-3}$ ($\mu_n = 1200\,\text{cm}^2/(\text{V s})$) that is $1\,\mu\text{m}$ thick and $2\,\mu\text{m}$ wide.

(a) Find the resistivity.

(b) How long must the strip be?

2.5 ◦ Use Equation (2.7) to answer both parts.

(a) At what wavelength does GaN ($E_g = 3.4\,\text{eV}$) absorb or emit? Which part of the spectrum is this?

(b) Optical fiber links use 1550 nm light. What is the photon energy? Could a silicon photodiode detect it? Could a germanium one ($E_g = 0.67\,\text{eV}$)?

2.6 ◦ A pulse oximeter uses a red LED at 660 nm and an infrared LED at 940 nm.

(a) Find the photon energy of each in eV.

(b) Can a silicon photodiode detect both? Explain.

(c) How many photons per second does $1\,\mu\text{W}$ of 940 nm light carry?

2.7 • A silicon pn junction has $N_A = 10^{18}\,\text{cm}^{-3}$ and $N_D = 10^{16}\,\text{cm}^{-3}$. At 300 K ($V_T = 25.9\,\text{mV}$, $\varepsilon_s = 1.04 \times 10^{-12}\,\text{F/cm}$), find

(a) the built-in potential,

(b) the zero-bias depletion width and the fraction of it on the n side,

(c) the junction capacitance per unit area.

2.8 • A photodiode is a p^+n junction with $N_A = 10^{18}\,\text{cm}^{-3}$, $N_D = 10^{15}\,\text{cm}^{-3}$, and an area of $1\,\text{mm}^2$. At 300 K ($V_T = 25.9\,\text{mV}$, $\varepsilon_s = 1.04 \times 10^{-12}\,\text{F/cm}$):

(a) find the built-in potential;

(b) find the depletion width and the capacitance at zero bias and at a reverse bias of 3.3 V;

(c) by what factor does the reverse bias speed up a response that is set by charging this capacitance through a fixed resistor?

2.9 ◦ A diode has $I_S = 10^{-14}\,\text{A}$ and $n = 1$; use $V_T = 25.9\,\text{mV}$. Find V_D at 1 mA and at 10 mA, the difference between them, and the small-signal resistance at 10 mA.

2.10 ◦ An Arduino pin at 3.3 V drives a red LED ($V_F = 1.9\,\text{V}$) through a 220 Ω resistor (Figure 2.43). Find the LED current with the constant-drop model. Which resistor would give 10 mA?

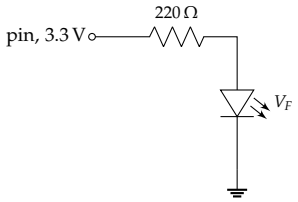


Figure 2.43: Problem 2.10.

2.11 • Figure 2.44 shows a diode logic gate with inputs A and B that are each either 0 V or 3.3 V. Use the constant-drop model with $V_{on} = 0.7\,\text{V}$.

(a) Make a table of V_{out} for all four input combinations. Which logic function is this?

(b) Find the current in each diode when A is high and B is low, and when both are high (assume identical diodes).

(c) Why can't a chain of such gates be extended indefinitely?

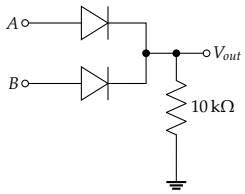


Figure 2.44: Problem 2.11: a diode gate.

2.12 •• (Python) A diode with $I_S = 10^{-14}\,\text{A}$, $n = 1$, and an internal series resistance $R_s = 10\,\Omega$ is connected through an external 100 Ω resistor to a 3.3 V source. Use $V_T = 25.9\,\text{mV}$.

(a) Write KVL as a single equation in the current I and solve it numerically (for example with `scipy.optimize.brentq`).

(b) Find the junction voltage and the voltage at the diode's terminals.

(c) Compare I with the constant-drop estimate. Which effect does the constant-drop model miss here?

2.13 • A half-wave rectifier (Figure 2.16) is fed by a 12 V peak sine wave and drives a 1 kΩ load. With the constant-drop model, find

(a) the peak output voltage and peak current,

(b) the peak reverse voltage the diode must withstand,

(c) the fraction of each cycle during which the diode conducts.

2.14 ◦ An NMOS has $k' = \mu_n C_{ox} W/L = 1\,\text{mA/V}^2$ and $V_{\text{th}} = 0.8\,\text{V}$. Find its operating region and I_D for

(a) $V_{GS} = 0.5\,\text{V}$, $V_{DS} = 2\,\text{V}$;

(b) $V_{GS} = 2\,\text{V}$, $V_{DS} = 0.4\,\text{V}$;

(c) $V_{GS} = 2\,\text{V}$, $V_{DS} = 3\,\text{V}$.

2.15 •• The NMOS in Figure 2.45 has $k' = 2\,\text{mA/V}^2$ and $V_{\text{th}} = 1\,\text{V}$, with $R_D = 2\,\text{k}\Omega$ and $V_{DD} = 5\,\text{V}$.

(a) For $V_{GS} = 2\,\text{V}$, find I_D and V_{DS} and verify the region.

(b) For $V_{GS} = 3\,\text{V}$, show that the saturation assumption fails, and find I_D and V_{DS} in the triode region.

(c) At the bias of part (a), find g_m and the small-signal voltage gain v_{ds}/v_{gs}.

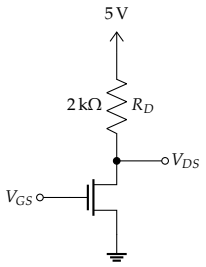


Figure 2.45: Problem 2.15.

2.16 • A PMOS has $V_{\text{th}} = -0.8\,\text{V}$ and $k'_p = 0.5\,\text{mA/V}^2$. Its source is at 3.3 V and its gate at 0 V. Find the region of operation and the current I_{SD} when the drain is at

(a) 0.5 V;

(b) 2.8 V.

(c) Which way does the current flow through the device?

2.17 • An NMOS below threshold carries 10 nA at $V_{GS} = 0.30\,\text{V}$ and 1.0 µA at $V_{GS} = 0.48\,\text{V}$.

(a) Find the subthreshold slope in mV per decade and the slope factor n ($V_T = 25.9\,\text{mV}$).

(b) Estimate the off current at $V_{GS} = 0$.

2.18 • A logic gate drives a 10 fF load and switches at an average rate of 1 GHz.

(a) Find its dynamic power at $V_{DD} = 1.2\,\text{V}$ and at 0.8 V. By what factor does lowering the supply reduce power?

(b) A chip holds 10^9 transistors, each with a subthreshold slope of 80 mV/decade and $I_D = 1\,\mu\text{A}$ at $V_{GS} = V_{\text{th}}$. Treat every transistor as off with the full V_{DD} across it. Find the off current per transistor and the total leakage power at $V_{DD} = 1.0\,\text{V}$ for $V_{\text{th}} = 0.4\,\text{V}$ and for $V_{\text{th}} = 0.3\,\text{V}$.

(c) Use your answers to explain why designers cannot simply keep lowering V_{DD} and V_{th} together.

2.19 • A blue LED ($V_F = 3.0\,\text{V}$ at 15 mA) is powered from a 5 V USB supply.

(a) Choose a standard resistor (nearest E24 value below the exact answer) and find the actual current and the power in the resistor and the LED.

(b) Someone proposes driving the same LED from 3.3 V with a 20 Ω resistor. If V_F varies from part to part between 2.9 V and 3.1 V, what range of currents results? Why is this a poor design?

2.20 • A silicon photodiode with a quantum efficiency of 0.70 receives 20 µW of light at 940 nm. Find its responsivity, its photocurrent, and the feedback resistor a transimpedance amplifier needs to produce a 2.0 V output.

2.21 • A solar panel has 36 cells in series. Each cell has $V_{oc} = 0.62\,\text{V}$ and $I_{sc} = 3.0\,\text{A}$ in full sun ($1000\,\text{W/m}^2$), and the panel's fill factor is 0.78. The panel area is $0.35\,\text{m}^2$. Find the panel's open-circuit voltage, maximum output power, and efficiency.

2.22 ◦ A non-inverting amplifier runs from the board's single 3.3 V supply, with $R_f = 47\,\text{k}\Omega$ and $R_i = 10\,\text{k}\Omega$.

(a) Find its gain and its output for $V_{in} = 0.25\,\text{V}$.

(b) What does it output for $V_{in} = 0.80\,\text{V}$, and why?

(c) What is the largest input it amplifies correctly?

(d) How much current does it draw from the source of V_{in}?

2.23 ◦ In the diode circuit of Figure 2.34(a), DAC channel 0 is at 1.20 V and pin A2 reads 0.55 V.

(a) Find the diode current and the diode voltage.

(b) What current does the constant-drop model predict?

(c) Find the voltages at the op-amp's two inputs.

2.24 • A photodiode on the biosensing block delivers up to $0.8\,\mu\text{A}$ in bright light.

(a) With $R_F = 3\,\text{M}\Omega$, what is the largest TIA output, and how much room is left below 3.3 V?

(b) What R_F would put $0.8\,\mu\text{A}$ at 3.0 V?

(c) How much photocurrent does one ADC count (3.22 mV) represent with each resistor?

(d) Which terminal of the photodiode must connect to the inverting input of a single-supply TIA, and why?

2.25 • In the difference amplifier of Figure 2.33(b), $R_1 = R_2 = R_4 = 1\,\text{M}\Omega$, but R_3 is 1 % high ($1.01\,\text{M}\Omega$).

(a) Find V_{out} in terms of V_A and V_B.

(b) In the diode circuit with $V_{DAC} = 3.3\,\text{V}$, the true voltage across the sense resistor is 2.50 V. What does A2 read, and how large is the current error in microamperes and in ADC counts?

(c) In the transistor circuit with $V_{DAC2} = 3.3\,\text{V}$ and a true drain current of $10\,\mu\text{A}$, what does A3 read? What does this say about measuring small currents?

2.26 •• The difference amplifier's input resistor R_1 connects the device node B to the inverting input, so the amplifier draws a little current from the circuit it measures.

(a) For an ideal op-amp and four equal resistors R, show that the current flowing from node B into R_1 is $(V_B - V_A/2)/R$.

(b) In the diode circuit with $V_{DAC} = 3.3\,\text{V}$, $V_D = 0.78\,\text{V}$, and $R = 1\,\text{M}\Omega$, find this current and the error it causes in the measured diode current. Compare it with one ADC step.

(c) What would the error be with four $10\,\text{k}\Omega$ resistors?

2.27 •• An engineer proposes to keep the transimpedance amplifier of the first board layout (Figure 2.35), but to connect its non-inverting input to a reference $V_{ref} = 2.0\,\text{V}$ and to use $R_F = 1\,\text{k}\Omega$. The op-amp still runs from 3.3 V and ground.

(a) At what voltage does the source sit? Find V_{out} for $I_D = 1.0\,\text{mA}$.

(b) What range of I_D can this circuit read?

(c) What are the largest V_{GS} and V_{DS} the DACs (0 to 3.3 V) can now apply? Why is this a poor choice for a transistor whose threshold is near 1.5 V?

(d) Show that, whatever V_{ref} is chosen, the largest readable $I_D R_F$ plus the largest V_{GS} equals 3.3 V.

2.28 ◦ On the devices block, with the $1\,\text{k}\Omega$ sense resistor as the whole drain path, $V_{DAC1} = 2.2\,\text{V}$, $V_{DAC2} = 1.5\,\text{V}$, and A3 reads 1.46 V.

(a) Find V_{GS}, I_D, and V_{DS}.

(b) In which region is the DMN3404L (V_{th} between 1.0 and 2.0 V)?

(c) What is the largest current this setting of V_{DAC2} could ever produce?

2.29 • Model the board's transistor with the square law, $V_{\text{th}} = 1.5\,\text{V}$ and $K = \frac{1}{2}k' = 20\,\text{mA/V}^2$, in the circuit of Figure 2.34(b) with the $1\,\text{k}\Omega$ sense resistor as the whole drain path (see Figure 2.36).

(a) For $V_{GS} = 1.7\,\text{V}$, find the saturation (plateau) current and $V_{DS,sat}$.

(b) What is the smallest V_{DAC2} that brings the transistor to the plateau?

(c) For $V_{GS} = 1.9\,\text{V}$, show that no $V_{DAC2} \le 3.3\,\text{V}$ reaches the plateau.

2.30 •• (Demo board) During the diode sweep of Lab 2.1 you read 0.50 mA at $V_D = 0.60\,\text{V}$ and 2.0 mA at 0.66 V. Assume the diode equation with $I \gg I_S$ and $V_T = 25.9\,\text{mV}$.

(a) Find the ideality factor n and the saturation current I_S.

(b) Predict V_D at 5 mA.

(c) Would you expect the measured voltage at 5 mA to be higher or lower than your prediction? Why? Could the board's diode circuit even reach 5 mA?

2.31 •• (Demo board, Python) In the transfer sweep of Lab 2.1 with $V_{DAC2} = 3.3\,\text{V}$, the board's drain path, from DAC channel 2 through the 1 kΩ sense resistor to the drain, measures $R_{path} = 5.0\,\text{k}\Omega$ (step C2 of the lab), and A3 reads 0.080 V at $V_{GS} = 1.80\,\text{V}$ and 0.320 V at $V_{GS} = 2.20\,\text{V}$. Treat R_{path} as a resistor in series with the drain.

(a) Find I_D and V_{DS} at both points, and the largest current the circuit can deliver.

(b) Assuming the square law $I_D = K(V_{GS} - V_{\text{th}})^2$, find V_{th} and K (hint: $\sqrt{I_D}$ is linear in V_{GS}), and check that both points were saturated.

(c) Predict I_D at $V_{GS} = 2.80\,\text{V}$ if the transistor stayed saturated. Show that the circuit cannot deliver it, and find the actual I_D and V_{DS}.

(d) Write a few lines of Python that fit V_{th} and K to all the suitable points of a measured sweep instead of just two.

3
Sensors

In this chapter you will learn to

- Describe a sensor as a transducer and place it in the five-stage chain that ends in a number on a screen.
- Read a sensor's transfer curve for sensitivity, range, linearity, resolution, and noise, and compute an RMS noise level and an SNR.
- Model thermistors and RTDs, read them with a voltage divider or a Wheatstone bridge, and account for self-heating, response time, and calibration.
- Convert between voltage and ADC code, find the temperature resolution of a measurement, and pick a sampling rate that avoids aliasing.
- Design op-amp readout circuits for sensors: buffers, gain stages, difference amplifiers for bridges, and transimpedance amplifiers.
- Apply the Beer-Lambert law to explain why a pulse oximeter needs two wavelengths.
- Compute SpO_2 from the AC and DC parts of red and infrared PPG signals, and extract heart rate by peak detection and by the FFT.
- Build a thermometer and a pulse oximeter with the demo board (Labs 3.1 and 3.2).

Rest your fingertip on the small black window in the biosensing block of the demo board (selector on Module 3) and type `sensors` into the serial monitor. Two traces start to scroll: one for red light and one for infrared. Each one dips about once a second, in step with your pulse. Nothing touched your blood and no cuff was inflated,

yet within a minute you can read your heart rate and the fraction of your hemoglobin that is carrying oxygen. The window hides several LEDs, a photodiode, an amplifier, and an analog-to-digital converter, all inside one optical sensor chip (the MAX86916) that reports numbers to the Arduino over two wires.

This chapter builds up to that measurement one layer at a time. We start with what a sensor is and how to describe one with numbers. Then we read a simple sensor, a temperature-dependent resistor, with the voltage divider from Chapter 1, and we learn how an analog voltage becomes a digital code. Op amps, introduced in Chapter 2, let us amplify weak signals and turn a photodiode current into a voltage. Filters and the Fourier transform let us separate a heartbeat from drift and noise. Finally the physics of light in blood, the Beer-Lambert law, explains why two colors of light are enough to measure blood oxygen. The two labs at the end of the chapter put these ideas on the bench: a thermistor thermometer (Lab 3.1) and a pulse oximeter (Lab 3.2).

3.1 *What Is a Sensor?*

A sensor converts a physical quantity into an electrical one. Temperature becomes a change in resistance. Light becomes a current. Pressure bends a thin membrane and changes a capacitance. The general name for a device that converts energy or information from one form to another is a *transducer*[1], and a sensor is a transducer whose output is electrical and whose job is measurement.

Figure 3.1 shows the idea. The input may be motion, light, sound, temperature, a magnetic field, or a chemical concentration; the output is an analog voltage or current that follows it. Before a computer can use that signal it usually needs *conditioning*: amplification, filtering, and conversion to digital form.

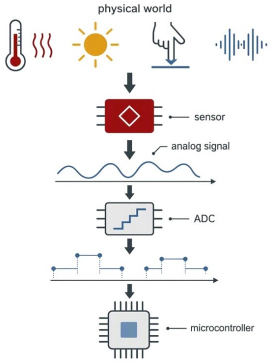


Figure 3.1: Transduction. A physical stimulus becomes a voltage, current, or resistance change, and the ADC turns it into numbers the microcontroller can process.

[1] A loudspeaker is also a transducer, but it runs the other way: an electrical signal becomes sound. Devices that turn electrical signals into physical action are called *actuators*.

The sensing chain

Every sensing system you will build in this course follows the same five stages, shown in Fig. 3.2. The pulse oximeter, the smartwatch on your wrist, and the bedside monitor in a hospital all have them, even when several stages live inside one chip.

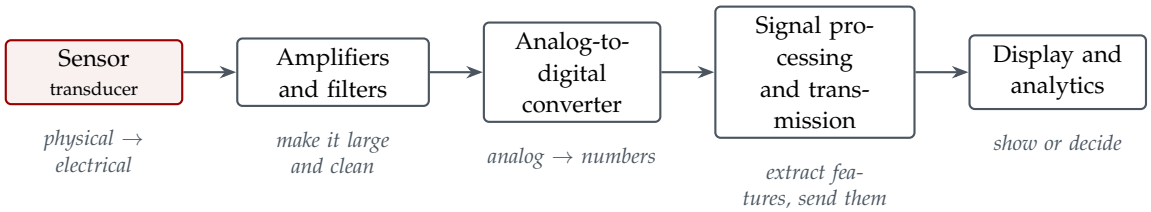


Figure 3.2: The five stages of a sensing system. The *analog front end* (AFE) is everything between the sensing element and the digital numbers: the first three stages.

WORKED EXAMPLE 3.1 *The five steps of a sensing system*

Draw the system block diagram of a sensing system. Name the five steps and explain why each is needed.

SOLUTION. The diagram is Fig. 3.2.

1. **Sensor.** Converts the physical quantity (temperature, light, pressure) into an electrical signal. Without it there is nothing to measure.
2. **Amplifiers and filters.** Sensor signals are often tiny (nanoamps from a photodiode, millivolts from a bridge) and ride on noise and drift. Amplification raises the signal to fill the ADC input range; filtering removes frequencies that carry no information.
3. **Analog-to-digital converter.** A microcontroller only handles numbers, so the conditioned voltage must be sampled in time and quantized in amplitude.
4. **Signal processing and transmission.** Raw samples are turned into useful quantities (heart rate, temperature in degrees) and sent over USB, Bluetooth, or Wi-Fi.
5. **Display and analytics.** A person or an algorithm sees the result and acts on it: a number on a phone, an alarm, a trend over weeks.

■

How sensors appear in circuits

From the circuit's point of view, sensors come in three flavors. A *variable passive element* is a resistor, capacitor, or inductor whose value changes with the stimulus; you read it with a circuit, such as a voltage divider, whose output depends on that value. A *switch* opens or closes when the stimulus crosses a threshold, like a float switch or the board's push buttons. A sensor with its own electronics is either *analog* (it outputs a voltage for the microcontroller's built-in ADC) or *digital* (it has its own ADC and talks over a serial bus such as I^2C). The demo board has both: the thermistor you plug into the voltage-divider header of the circuits block is a variable resistor read on pin A1, and the optical sensor of the biosensing block is a digital sensor that drives its own LEDs, digitizes the photodiode current, and sends numbers over I^2C. (The blocks are labeled in Fig. 0.16.)

Table 3.1 lists the common families. The electrical questions are always the same: how large is the output, how fast does it change, and how much noise rides on it? The next section gives them precise names.

Category	What it detects	Examples
Position	linear or angular position	potentiometer, rotary encoder, Hall sensor
Proximity, motion	an object nearby or moving	ultrasonic ranger, reflective optical, PIR, capacitive proximity
Inertial	acceleration, rotation	accelerometer, gyroscope, tilt switch
Pressure, force	force or pressure on a surface	strain gauge, load cell, piezoelectric disk
Optical	light intensity	photodiode, phototransistor, photoresistor, solar cell
Image, magnetic	light patterns, fields	CMOS camera (Chapter 4), Hall sensor, compass
Media	gas, humidity, moisture	metal-oxide gas sensor, soil probe
Temperature	heat	thermistor, RTD, thermocouple, IC sensor, IR thermometer
Specialized	sound, radiation, chemistry	microphone, Geiger tube, glucose biosensor

Table 3.1: Common sensor families. The Arduino Nano 33 BLE Sense on your board already carries several of them: an inertial measurement unit, a microphone, a light and gesture sensor, and pressure, temperature, and humidity sensors.

3.2 *Characterizing a Sensor*

EVERY SENSOR HAS A TRANSFER FUNCTION, the relation between the stimulus S_{in} and the electrical output S_{out}:

$$S_{\text{out}} = f(S_{\text{in}}). \tag{3.1}$$

We find f by *calibration*: apply known inputs (known temperatures, known glucose concentrations), record the outputs, and fit a curve. The calibration curve then runs backward during use. You measure S_{out} and solve for the S_{in} that must have caused it.

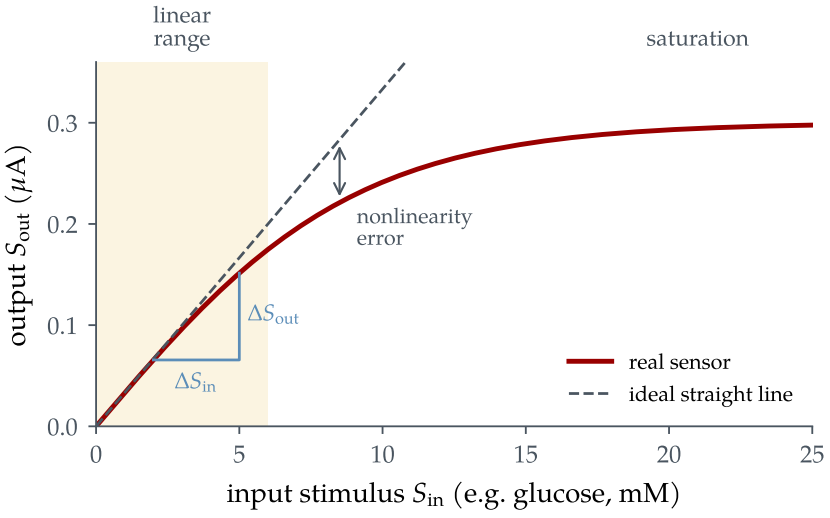


Figure 3.3: A typical sensor transfer curve, modeled on an enzymatic glucose sensor. Near the origin the output grows in proportion to the input, and the slope is the sensitivity. At high input the sensor saturates. The gap between the real curve and the straight line is the nonlinearity error.

Figure 3.3 shows a transfer curve and the features we care about.

Sensitivity is the slope of the transfer curve, the change in output per unit change in input:

$$S = \frac{\Delta S_{\text{out}}}{\Delta S_{\text{in}}} \approx \frac{\mathrm{d}S_{\text{out}}}{\mathrm{d}S_{\text{in}}}. \tag{3.2}$$

A glucose sensor might have a sensitivity of tens of nanoamps per millimolar; a thermistor divider on the board gives tens of millivolts per degree. Higher sensitivity means a given change in the world produces a larger, easier to measure change in the signal.

Range is the span of inputs the sensor can measure; beyond it the output saturates or the sensor is damaged. *Linearity* describes how closely the curve follows a straight line. A linear sensor has one sensitivity across its range, so two calibration points suffice; a nonlinear one, like the thermistor of the next section, needs a model or a lookup table.

Resolution is the smallest change in input that produces a detectable change in output. It is set by noise in the sensor and electronics, and by the step size of the ADC that digitizes the signal (Section 3.4).

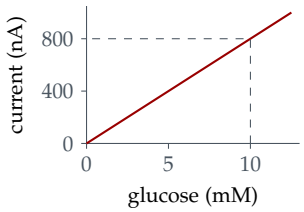


Figure 3.4: Glucose sensor calibration line. A glucose sensor of this type outputs a current, plotted on the vertical axis.

Worked Example 3.2 *Sensitivity of a glucose sensor*

The calibration line of a glucose sensor rises from 0 nA at 0 mM to about 800 nA at 10 mM (Fig. 3.4). Calculate the sensitivity.

Solution. The curve is a straight line through the origin, so any two points give the same slope:

$$S = \frac{\Delta I}{\Delta c} = \frac{800\,\text{nA} - 0\,\text{nA}}{10\,\text{mM} - 0\,\text{mM}} = 80\,\text{nA/mM}.$$

Each additional millimole per liter of glucose adds about 80 nA of sensor current. Blood glucose in a healthy fasting adult sits near 5 mM, where this sensor would output about 400 nA. ■

Noise and signal-to-noise ratio

Noise is the random part of a sensor's output, the part unrelated to the input. It comes from temperature fluctuations, electromagnetic interference (the 60 Hz hum from power lines), instability in the sensing element or the electronics, and mechanical effects such as vibration or bubbles in a fluid sensor. If you hold the input fixed and record many samples, the readings scatter around a mean value. The size of that scatter is measured by the *root mean square* (RMS) value of the deviations x_i from the mean:

$$x_{\text{rms}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}}. \tag{3.3}$$

The *signal-to-noise ratio* compares the signal you care about with this noise level. It is often quoted in decibels:

$$\mathrm{SNR} = \frac{\text{signal}}{x_{\mathrm{rms}}}, \qquad \mathrm{SNR}_{\mathrm{dB}} = 20\log_{10}\frac{\text{signal}}{x_{\mathrm{rms}}}. \tag{3.4}$$

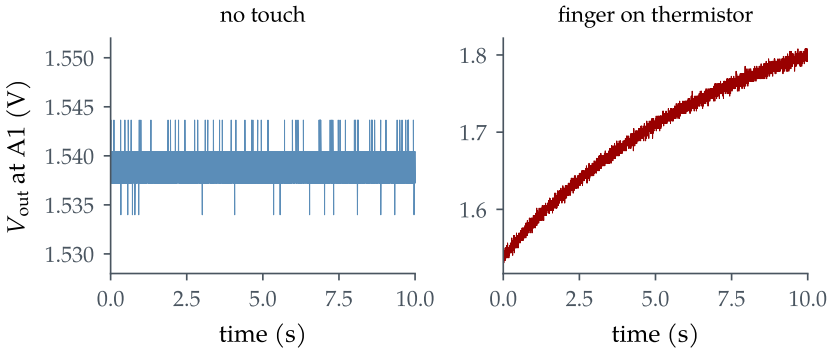


Figure 3.5: Simulated 10 s recordings of the board's thermistor divider (thermistor on top, Fig. 3.9) at 1 ms per sample and a room temperature of 22 °C. Left: nothing touches the sensor, and the output jumps between a few neighboring ADC levels near 1.54 V. Right: a finger warms the thermistor and the voltage climbs by about 0.25 V, far above the noise.

Figure 3.5 shows what this looks like on the board. With no finger on the thermistor the trace is a band of noise only a few millivolts wide, and the ADC steps are visible as discrete levels. A finger raises the voltage by a quarter of a volt, so the temperature change is easy to see.

> **Worked Example 3.3** *RMS noise and SNR of the thermistor reading*
>
> Six readings of the divider output with nothing touching the sensor are 1.539, 1.542, 1.536, 1.539, 1.545, and 1.533 V. When a finger touches the thermistor the mean output rises by 250 mV. Find the RMS noise and the SNR.
>
> Solution. The mean is $\bar{V} = 1.539$ V. The deviations are 0, +3, −3, 0, +6, and −6 mV. Their squares sum to $0+9+9+0+36+36 = 90\,\mathrm{mV}^2$, so
>
> $$x_{\mathrm{rms}} = \sqrt{\frac{90}{6}}\ \mathrm{mV} = \sqrt{15}\ \mathrm{mV} = 3.87\ \mathrm{mV}.$$
>
> The SNR is $250/3.87 = 64.5$, or $20\log_{10} 64.5 = 36.2$ dB. The touch is about 65 times larger than the noise, so a single reading is enough to tell whether a finger is present. ■

Differential measurement

Some noise is shared. A chemical sensor responds to its target molecule, but also responds to temperature and humidity, which drift through the day. A standard trick is to build two identical sensing elements, expose only one of them to the target, and subtract their

outputs. Anything common to both (drift, interference, temperature) cancels, and only the response to the target remains (Fig. 3.6). The Wheatstone bridge of the next section and the difference amplifier of Section 3.5 are the circuit versions of this idea.

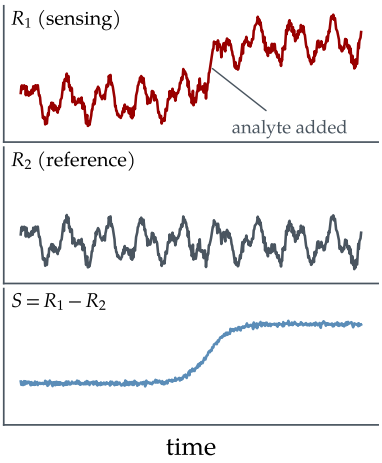


Figure 3.6: Differential measurement. R_1 senses the analyte and R_2 is an identical reference that does not. Both see the same drift and noise; their difference keeps only the response.

3.3 *Temperature Sensors and Resistive Interfaces*

Temperature is the quantity engineers measure most often, and the course uses it as the first sensing example. Four sensor types cover nearly every application (Fig. 3.7).

A *thermistor* is a resistor made of a semiconducting metal oxide whose resistance changes strongly with temperature. In the common *negative temperature coefficient* (NTC) type, resistance falls as the temperature rises, because heat frees more charge carriers in the oxide. A *positive temperature coefficient* (PTC) type does the opposite. Thermistors are cheap, small, and very sensitive (a few percent change per degree), but strongly nonlinear.

A *resistance temperature detector* (RTD) is a length of pure metal, usually platinum, whose resistance rises slowly and almost linearly with temperature. The standard part is the Pt100, which has exactly 100 Ω at 0 °C. RTDs are less sensitive than thermistors (about 0.39 % per degree) but very stable and accurate.

A *thermocouple* is a junction of two different metals that produces a few tens of microvolts per degree (the Seebeck effect); it survives extreme temperatures but needs careful amplification. An *IC sensor* such as the TMP36 outputs 10 mV per degree (750 mV at 25 °C), linear and easy to use from about −40 °C to 125 °C.

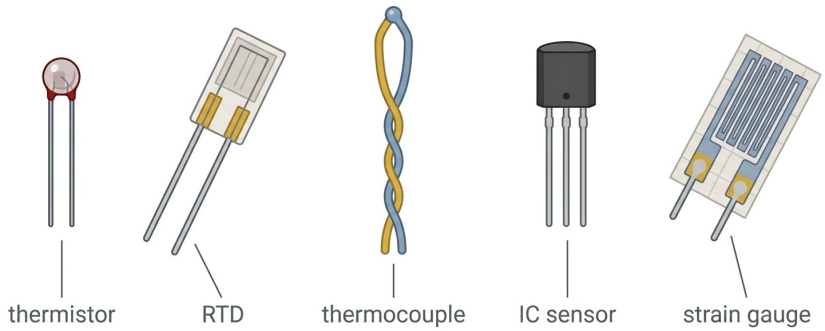


Figure 3.7: Common temperature sensors, plus a strain gauge. A thermistor is a small bead of metal-oxide ceramic. An RTD is a thin platinum film or wire. A thermocouple is a junction of two different metals. An IC sensor such as the TMP36 is a transistor circuit in a three-pin package. A foil strain gauge, read with the same circuits, is shown for comparison.

The thermistor model

KEY EQUATION

$R_T = R_0 e^{\beta(1/T - 1/T_0)}$

NTC thermistor: resistance falls exponentially as T rises.

Over a moderate temperature range an NTC thermistor follows the *beta equation*

$$R_T = R_0 \exp\left[\beta\left(\frac{1}{T} - \frac{1}{T_0}\right)\right], \tag{3.5}$$

where R_T is the resistance at absolute temperature T (in kelvin), R_0 is the resistance at a reference temperature T_0 (almost always 25 °C = 298.15 K), and β is a material constant in kelvin. The data sheet gives R_0, T_0, and β. The thermistors used in class are "10K 3950" parts: $R_0 = 10\,\text{k}\Omega$ at 25 °C and $\beta = 3950\,\text{K}$. During a measurement the circuit tells you R_T, and you solve Eq. (3.5) for the unknown T:

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{\beta}\ln\frac{R_T}{R_0}. \tag{3.6}$$

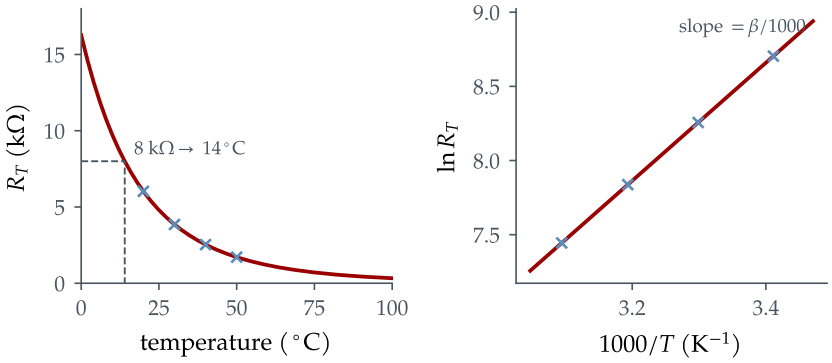


Figure 3.8: Calibrating a thermistor with $R_0 = 4.8\,\text{k}\Omega$ at 25 °C and $\beta = 3977\,\text{K}$ (the example values). Left: resistance falls steeply with temperature; a reading of 8 kΩ means 14 °C. Right: plotting $\ln R_T$ against $1000/T$ turns the curve into a straight line whose slope is $\beta/1000$.

Taking the logarithm of Eq. (3.5) gives $\ln R_T = \beta/T + \text{const}$, a straight line in $1/T$ (Fig. 3.8, right). That is how β is found in practice: measure the resistance at a few known temperatures, plot $\ln R_T$ against $1/T$, and read the slope.

WORKED EXAMPLE 3.4 *Reading a thermistor calibration curve*

A thermistor has $R_0 = 4800\,\Omega$ at $T_0 = 25\,°\text{C}$ and $\beta = 3977\,\text{K}$. The circuit reports $R_T = 8000\,\Omega$. What is the temperature?

SOLUTION. Use Eq. (3.6) with $T_0 = 298.15\,\text{K}$:

$$\begin{aligned}\frac{1}{T} &= \frac{1}{298.15} + \frac{1}{3977}\ln\frac{8000}{4800} \\ &= 3.3540\times10^{-3} + 1.2844\times10^{-4} = 3.4825\times10^{-3}\,\text{K}^{-1},\end{aligned}$$

so $T = 287.15\,\text{K} = 14.0\,°\text{C}$. The answer is below 25 °C, as it must be: an NTC resistance above R_0 means a temperature below T_0. Figure 3.8 shows the same reading graphically. ■

Worked Example 3.5 *Finding β from two calibration points*

You place a thermistor in a water bath. It reads 10.0 kΩ at 25 °C and 3.60 kΩ at 50 °C. Find β and predict the resistance at body temperature, 37 °C.

Solution. Write Eq. (3.5) at both points and divide:

$$\beta = \frac{\ln(R_1/R_2)}{1/T_1 - 1/T_2} = \frac{\ln(10.0/3.60)}{1/298.15 - 1/323.15} = \frac{1.0217}{2.5948 \times 10^{-4}} = 3937\,\text{K}.$$

Then at $T = 310.15\,\text{K}$,

$$R_{37} = 10.0\,\text{k}\Omega \times \exp\left[3937\left(\frac{1}{310.15} - \frac{1}{298.15}\right)\right] = 6.00\,\text{k}\Omega.$$

■

RTDs

A platinum RTD follows the Callendar-Van Dusen equation. Above 0 °C,

$$R_T = R_0\left(1 + AT + BT^2\right), \tag{3.7}$$

with T in degrees Celsius, $R_0 = 100\,\Omega$ for a Pt100, $A = 3.9083 \times 10^{-3}/°\text{C}$, and $B = -5.775 \times 10^{-7}/°\text{C}^2$ (below 0 °C a term $C(T-100)T^3$ with $C = -4.183 \times 10^{-12}/°\text{C}^4$ is added). A Pt100 reads 109.73 Ω at 25 °C and 114.38 Ω at 37 °C, only about 0.39 Ω per degree, which is why RTDs are usually read with a bridge.

Reading a resistive sensor: the voltage divider

A microcontroller measures voltage, not resistance, so we need a circuit that turns R_T into a voltage. The simplest is the voltage divider from Chapter 1. On the demo board it is built in the *voltage-divider header* of the circuits block: four single-pin sockets, J1 to J4, in a column to the right of the button grid (Fig. 3.10). J1 (top) carries 3.3 V, J2 and J3 are both the divider midpoint, and J4 (bottom) is ground, so the top part (printed R1 on the board) plugs into J1–J2 and the bottom part (R2) into J3–J4. The midpoint reaches the Arduino's analog pin A1 through jumper J8 when its cap sits on the Module 1 end, pins 2 and 3 (on the Camera end it connects A1 to the camera instead). In this chapter and in Lab 3.1 the thermistor goes on *top* (J1–J2) and a fixed 10 kΩ resistor on the *bottom* (J3–J4), as in Fig. 3.9. When an example uses a different arrangement, it says so.

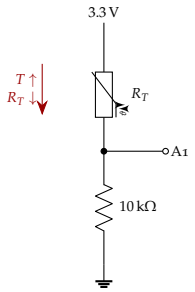


Figure 3.9: The thermistor divider on the demo board, used throughout this chapter and in Lab 3.1: an NTC thermistor from 3.3 V to pin A1 (header sockets J1–J2) and a 10 kΩ resistor from A1 to ground (J3–J4). Warming the thermistor lowers R_T and raises V_{out}.

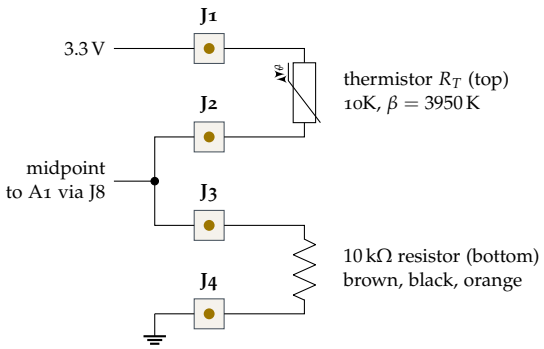


Figure 3.10: The voltage-divider header of the circuits block (printed Module 1 on the board), to the right of the button grid. J1 is 3.3 V, J2 and J3 are the midpoint, J4 is ground. For Lab 3.1 the thermistor plugs into J1–J2 (R1 on the board) and the 10 kΩ resistor into J3–J4 (R2); with the selector on Module 1 and the cap of J8 on the Module 1 end, A1 reads the midpoint. This is the circuit of Fig. 3.9.

The divider rule gives

$$V_{\text{out}} = 3.3\,\text{V} \times \frac{R}{R_T + R}, \qquad R = 10\,\text{k}\Omega, \tag{3.8}$$

and solving for the thermistor resistance,

$$R_T = R\left(\frac{3.3\,\text{V}}{V_{\text{out}}} - 1\right). \tag{3.9}$$

At 25 °C the thermistor equals the fixed resistor, so $V_{\text{out}} = 1.65$ V, half the supply. Touch the thermistor and R_T falls, so V_{out} rises (Fig. 3.11). Swap the two parts and the output falls instead.

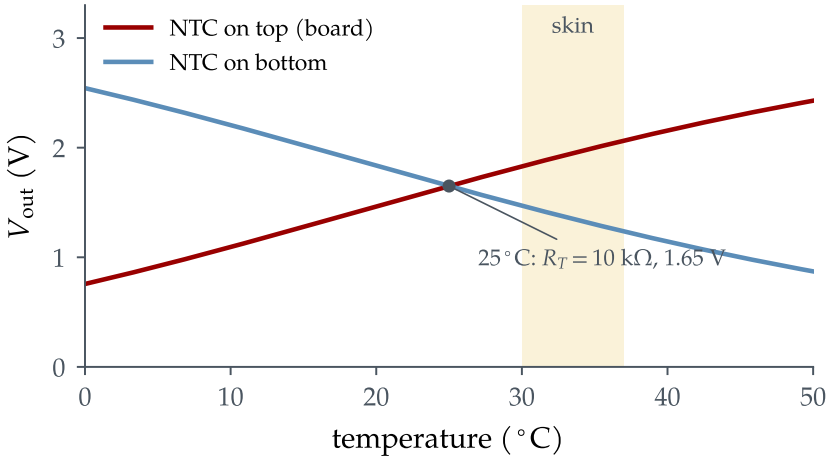


Figure 3.11: Divider output versus temperature for a 10K 3950 NTC thermistor and a 10 kΩ fixed resistor, with the thermistor on top (as on the board) or on the bottom. Both curves cross at 25 °C, where $R_T = R$. Near there the curves are close to their steepest: the slope is 36.7 mV/K at 25 °C and peaks at 37.5 mV/K near 18 °C.

The sensitivity of the divider is the slope of Fig. 3.11. By the chain rule,

$$\frac{\mathrm{d}V_{\text{out}}}{\mathrm{d}T} = \frac{\mathrm{d}V_{\text{out}}}{\mathrm{d}R_T}\frac{\mathrm{d}R_T}{\mathrm{d}T} = \left(-\frac{V_S R}{(R_T + R)^2}\right)\left(-\frac{\beta R_T}{T^2}\right). \tag{3.10}$$

At 25 °C, with $R_T = R = 10\,\text{k}\Omega$, the first factor is $V_S/(4R)$ and the second is $\beta R/T^2$, so the slope is $3.3\,\text{V} \times 3950/(4 \times 298.15^2) = 36.7\,\text{mV}$

per kelvin (a change of one kelvin and one degree Celsius are the same size). Near a room temperature of 22 °C it is a little larger, 37.2 mV per kelvin, and at skin temperature it is smaller. The slope is largest when the fixed resistor equals the thermistor resistance at the temperature you care about most (Problem 3.6).

TRY IT ON THE BOARD Turn the selector to Module 1, check that the caps of J7 and J8 are on the Module 1 end, and plug the thermistor and the 10 kΩ resistor into the header as in Fig. 3.10. In a new cell of a lab notebook, after its first cell has connected, run `board.monitor("divider", seconds=20)` and pinch the thermistor bead: A1 climbs by a few hundred millivolts over about ten seconds. The command `temperature` prints the reading converted to degrees with Eqs. (3.9) and (3.6). The cell sends `stop` when the time is up.

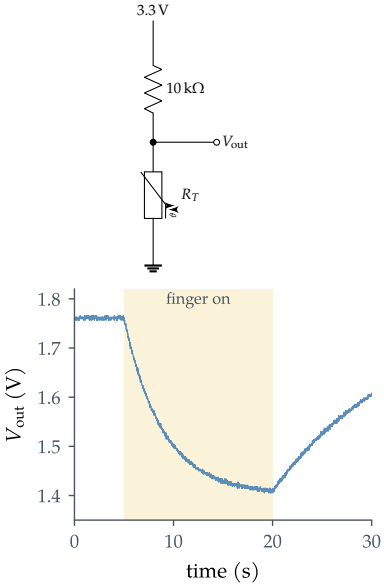


Figure 3.12: The thermistor on the bottom, which swaps the board's arrangement. Top: fixed resistor on top, thermistor on the bottom. Bottom: representative V_{out} (simulated) when a finger rests on the thermistor from 5 s to 20 s.

WORKED EXAMPLE 3.6 *Thermistor on the bottom*

On the board, and in Lab 3.1, the thermistor sits on top of the divider. This example uses the opposite arrangement (Fig. 3.12): 3.3 V, then $R = 10\,\text{k}\Omega$, then the thermistor R_T to ground. (i) Write an expression for $R_T = f(V_{out})$. (ii) Sketch V_{out} when you touch the sensor and explain the direction of the change.

SOLUTION. (i) The output is taken across the thermistor, so

$$V_{out} = 3.3\,\text{V}\,\frac{R_T}{R + R_T} \implies V_{out}(R + R_T) = (3.3\,\text{V})\,R_T,$$

and collecting the R_T terms,

$$R_T = \frac{R\,V_{out}}{3.3\,\text{V} - V_{out}} = \frac{(10\,\text{k}\Omega)\,V_{out}}{3.3\,\text{V} - V_{out}}.$$

(ii) Your finger is warmer than the room, so the thermistor heats up. For an NTC part R_T falls, the thermistor takes a smaller share of the 3.3 V, and V_{out} *decreases*. With representative values (a 10K 3950 thermistor going from a room temperature of 22 °C to about 32 °C), R_T goes from 11.4 kΩ to 7.38 kΩ and V_{out} falls from 1.76 V to 1.40 V (Fig. 3.12). Check with part (i): $R_T = 10 \times 1.40/(3.3 - 1.40) = 7.37\,\text{k}\Omega$. When you lift your finger, the thermistor cools and V_{out} creeps back up. The change is gradual because the thermistor bead has to exchange heat with your skin. Compared with the board's arrangement only the direction of the change is different; the sensitivity at 25 °C is the same 36.7 mV per kelvin (Fig. 3.11). ■

Measurement current and self-heating

A resistive sensor can only be read by passing a current through it, and that current heats it. In the board's divider the measurement current is $I = V_S/(R_T + R)$, and the thermistor dissipates

$$P_T = I^2 R_T = \frac{V_S^2 \, R_T}{(R_T + R)^2}. \tag{3.11}$$

This power warms the bead above its surroundings by

$$\Delta T_{\text{self}} = \frac{P_T}{\delta}, \tag{3.12}$$

where δ is the *dissipation constant*: the power that holds the sensor one degree above its surroundings. Data sheets of small epoxy-bead thermistors give about 1 to 3 mW per kelvin in still air, and more when the bead touches something that carries heat away, such as skin or water. The thermistor then reports its own, slightly warmer, temperature. For a fixed R, Eq. (3.11) is largest when $R_T = R$ (set $\mathrm{d}P_T/\mathrm{d}R_T = 0$ to check), where it equals $V_S^2/(4R)$. Self-heating is reduced by a lower supply voltage, a larger resistance, or by powering the divider only for the moment of each reading.

WORKED EXAMPLE 3.7 *Self-heating in the board's divider*

(a) Find the measurement current and the power in the thermistor of Fig. 3.9 at 25 °C. (b) With $\delta = 2\,\text{mW/K}$, how much does the bead warm itself? (c) A student replaces the 10 kΩ bottom resistor with 1 kΩ "to get more current and more signal." Repeat (a) and (b), and find the new sensitivity at 25 °C.

SOLUTION. (a) $I = 3.3\,\text{V}/(10\,\text{k}\Omega + 10\,\text{k}\Omega) = 165\,\mu\text{A}$. Half the supply, 1.65 V, falls across the thermistor, so $P_T = 1.65\,\text{V} \times 165\,\mu\text{A} = 0.272\,\text{mW}$. Since $R_T = R$, this is also the most the thermistor can ever dissipate in this divider, $V_S^2/(4R)$.

(b) $\Delta T_{\text{self}} = 0.272\,\text{mW}/(2\,\text{mW/K}) = 0.14\,\text{K}$. That is about one and a half ADC steps in temperature (Section 3.4 finds 0.088 °C per step), small next to the 10 °C of a finger touch, and nearly constant, so a calibration (below) absorbs most of it.

(c) $I = 3.3\,\text{V}/11\,\text{k}\Omega = 300\,\mu\text{A}$ and $P_T = (300\,\mu\text{A})^2 \times 10\,\text{k}\Omega = 0.90\,\text{mW}$, so the bead runs 0.45 K warm, more than three times worse. The sensitivity from Eq. (3.10) is

$$\frac{(3.3\,\text{V})(1\,\text{k}\Omega)}{(11\,\text{k}\Omega)^2} \times \frac{(3950\,\text{K})(10\,\text{k}\Omega)}{(298.15\,\text{K})^2} = 12.1\,\text{mV/K},$$

a third of the 36.7 mV/K of the board's divider. The change made the signal smaller and the self-heating larger. ■

Response time

A thermistor does not follow a temperature change instantly. The bead has a heat capacity C_{th} (joules per kelvin) and exchanges heat with its surroundings, at temperature T_s, through a thermal resistance R_{th} (kelvins per watt). The heat flow $(T_s - T)/R_{\text{th}}$ warms the bead:

$$C_{\text{th}} \frac{\mathrm{d}T}{\mathrm{d}t} = \frac{T_s - T}{R_{\text{th}}}. \tag{3.13}$$

This is the equation of the RC circuit in Chapter 1, with temperature in place of voltage and heat flow in place of current. After a step change of the surroundings, the bead approaches its final temperature exponentially:

$$T(t) = T_{\text{final}} + (T_{\text{start}} - T_{\text{final}})\, e^{-t/\tau}, \qquad \tau = R_{\text{th}} C_{\text{th}}. \tag{3.14}$$

After one time constant the change is 63 % complete, after 3τ 95 %, and after 5τ 99.3 %. Figure 3.13 shows a finger touch. Warming is fast because a fingertip pressed on the bead is a low thermal resistance; cooling into still air is several times slower because air is a poor conductor of heat. The heat capacity is the same both ways, so the ratio of the two time constants is the ratio of the two thermal resistances. Lab 3.1 measures both by fitting Eq. (3.14) to a recording.

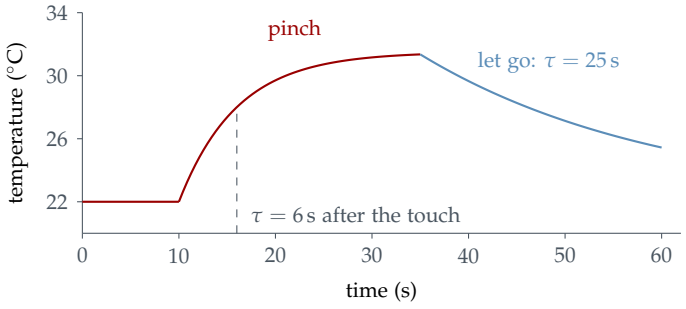


Figure 3.13: First-order model of a finger touch on the thermistor, as recorded in Lab 3.1: room at 22 °C, a pinch from 10 s to 35 s that would settle at 31.5 °C, then release. The warm-up time constant is 6 s; the cool-down in still air is 25 s. These are typical values; your recording gives its own.

Worked Example 3.8 *Time constant from a finger touch*

The bead sits at a room temperature of 22.0 °C. At $t = 0$ you pinch it; it reads 28.0 °C at $t = 6.0$ s and settles at 31.5 °C. (a) Find the warm-up time constant. (b) What does it read at 3 s? (c) When is the change 95 % complete? (d) After you let go, the bead cools with $\tau = 25$ s. How long until it is within 0.5 °C of room temperature?

Solution. (a) The fraction of the change still to go at 6 s is

$$\frac{31.5 - 28.0}{31.5 - 22.0} = \frac{3.5}{9.5} = 0.368 = e^{-6.0/\tau},$$

so $\tau = 6.0\,\text{s}/\ln(9.5/3.5) = 6.0\,\text{s}$. (A remaining fraction of 0.368 is e^{-1}: 6 s is exactly one time constant.)

(b) $T(3\,\text{s}) = 31.5 - 9.5\,e^{-3/6} = 31.5 - 5.76 = 25.7\,^\circ\text{C}$.

(c) At $3\tau = 18\,\text{s}$.

(d) Starting from 31.5 °C, the gap of 9.5 °C must shrink to 0.5 °C: $e^{-t/25\,\text{s}} = 0.5/9.5$, so $t = (25\,\text{s})\ln 19 = 74\,\text{s}$. The 25 s of cool-down recorded in Lab 3.1 covers only one time constant, which is why the fitted final value of the cool-down is less certain than that of the warm-up. ■

Calibration against a reference

The beta equation with data sheet values is only as good as those values. A "10K 3950" part is typically specified to 1 % in R_0 and 1 % in β, and the fixed resistor of the divider is a 1 % or 5 % part. Differentiating Eq. (3.6) shows what a small fractional error ϵ in the computed R_T does (it can come from R_0, or from R in Eq. (3.9)):

$$\Delta T \approx -\frac{T^2}{\beta}\,\epsilon. \tag{3.15}$$

Near room temperature $T^2/\beta = 298.15^2/3950 = 22.5\,\text{K}$, so each percent of resistance error moves the reading by about 0.23 °C. An error in β does nothing at T_0 and grows away from it: a 1 % error in β moves a reading at 37 °C by about 0.12 °C.

The board can therefore be *precise* (it resolves changes of 0.1 °C) without being *accurate* (its absolute reading may be off by a few tenths of a degree). To fix accuracy, compare the sensor with a reference thermometer. A *one-point* calibration measures both at one temperature and stores the difference as an offset. It removes errors that shift every reading by about the same amount, such as a resistor tolerance or self-heating. A *two-point* calibration at two well-separated temperatures also corrects the slope, $T_{\text{true}} = a\,T_{\text{meas}} + b$. With more points you can refit R_0 and β themselves, as in the example "Finding β from two calibration points." Without any reference, the temperatures you report come from the model alone; Lab 3.1 is in that situation unless your bench has a reference thermometer.

Worked Example 3.9 *One-point and two-point calibration*

A thermistor thermometer built like the board's (same divider, data sheet values in the code) is checked against a reference thermometer. In the room the reference reads 21.0 °C and the thermistor 21.8 °C; in a warm water bath the reference reads 37.0 °C and the thermistor 38.1 °C. (a) Find a one-point (offset) correction from

the room reading and the error it leaves in the bath. (b) Find a two-point correction $T_{\text{true}} = a\,T_{\text{meas}} + b$. (c) Correct a reading of 30.0 °C.

SOLUTION. (a) The offset is $21.0 - 21.8 = -0.8$ °C. Applied to the bath reading it gives $38.1 - 0.8 = 37.3$ °C, still 0.3 °C high: the error is not the same at both temperatures.

(b) The slope is $a = (37.0 - 21.0)/(38.1 - 21.8) = 16.0/16.3 = 0.9816$, and the offset is $b = 21.0 - 0.9816 \times 21.8 = -0.40$ °C.

(c) $T_{\text{true}} = 0.9816 \times 30.0 - 0.40 = 29.0$ °C. Both corrections are only valid between and near the calibration points; the two-point line also assumes the error grows linearly with temperature. ■

The Wheatstone bridge

A divider has a problem with small changes. A Pt100 in a divider with a 100 Ω resistor moves the output by only about 0.1 V between 0 °C and 37 °C, on top of a 1.65 V offset. Amplifying the change would also amplify the large offset and saturate the amplifier. The *Wheatstone bridge* fixes this by building two dividers side by side and taking the difference of their outputs (Fig. 3.14).

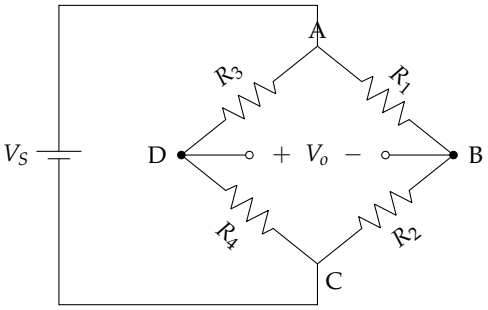


Figure 3.14: Wheatstone bridge. The left divider (R_3, R_4) sets node D and the right divider (R_1, R_2) sets node B. The output is the difference $V_o = V_D - V_B$. In the examples the sensor sits in the R_1 position.

Node D sits at $V_S R_4/(R_3 + R_4)$ and node B at $V_S R_2/(R_1 + R_2)$, so

$$V_o = V_D - V_B = V_S\left(\frac{R_1}{R_1 + R_2} - \frac{R_3}{R_3 + R_4}\right) = V_S\,\frac{R_1 R_4 - R_2 R_3}{(R_1 + R_2)(R_3 + R_4)}. \tag{3.16}$$

The bridge is *balanced*, $V_o = 0$, when $R_1 R_4 = R_2 R_3$. Now make R_1 a sensor whose resistance changes from R to $R + \delta R$, and set the other three resistors equal to R. Equation (3.16) becomes

$$V_o = V_S\,\frac{\delta R}{4R + 2\,\delta R} \approx V_S\,\frac{\delta R}{4R} \qquad (\delta R \ll R). \tag{3.17}$$

The output is now zero when the sensor is at its reference value and grows in proportion to δR. There is no large offset to fight, and any change that affects all four resistors equally (such as a uniform temperature shift of matched resistors) cancels. A drift in V_S does not upset the balance point, although it does scale the output.

KEY IDEA A balanced bridge outputs zero, so all of its output is signal. A differential amplifier can then apply large gain without saturating.

WORKED EXAMPLE 3.10 *Pt100 bridge at body temperature*

A Pt100 RTD is placed in the R_1 position of a bridge with $R_2 = R_3 = R_4 = 100\,\Omega$ and $V_S = 3.3\,\text{V}$. Find V_o at 37 °C exactly and with the small-change approximation.

SOLUTION. From Eq. (3.7), $R_1 = 100(1 + 3.9083 \times 10^{-3} \times 37 - 5.775 \times 10^{-7} \times 37^2) = 114.38\,\Omega$, so $\delta R = 14.38\,\Omega$. Exactly,

$$V_o = 3.3\left(\frac{114.38}{214.38} - \frac{1}{2}\right) = 0.1107\,\text{V}.$$

The approximation gives $3.3 \times 14.38/400 = 0.1186\,\text{V}$, about 7 % high. Here $\delta R/R = 0.14$ is not very small, so the $2\,\delta R$ term in the denominator matters. ■

WORKED EXAMPLE 3.11 *A strain gauge in a bridge*

A strain gauge is a thin metal foil pattern glued to a surface; when the surface stretches, the foil gets longer and thinner and its resistance rises. The fractional change is $\delta R/R = \text{GF} \times \varepsilon$, where ε is the strain (fractional change in length) and GF is the gauge factor, about 2 for metal foil. A $350\,\Omega$ gauge with GF = 2.0 is strained by $\varepsilon = 500 \times 10^{-6}$ (500 microstrain) in a bridge with $V_S = 3.3\,\text{V}$. Find V_o.

SOLUTION. $\delta R = 350 \times 2.0 \times 500 \times 10^{-6} = 0.35\,\Omega$, and

$$V_o \approx V_S \frac{\delta R}{4R} = 3.3 \times \frac{0.35}{1400} = 0.825\,\text{mV}.$$

The exact formula gives 0.8246 mV; with $\delta R/R = 0.001$ the approximation is excellent. A signal under a millivolt is far too small for a 3.3 V, 10-bit ADC whose step is 3.2 mV, so a strain bridge is always followed by an amplifier (Section 3.5). ■

Other resistive and capacitive sensors

Many sensors in Table 3.1 are read the same way: a metal-oxide gas sensor and a soil-moisture probe are resistors that change with their stimulus. Humidity and pressure sensors often use a capacitor $C = \epsilon A/d$ instead, read with an AC signal or a timing circuit. The ideas of this section carry over: the divider or bridge turns the change into a voltage, the measurement current heats the sensor a little, the sensor needs time to respond, and a reference fixes its accuracy.

3.4 *Analog-to-Digital Conversion*

The divider produces a voltage, but the Arduino can only store numbers. An analog-to-digital converter (ADC) does two things to a voltage. It *samples* it, measuring it at discrete times, and it *quantizes* it, rounding each measurement to one of a finite set of levels.

Resolution and quantization

An N-bit ADC divides its input range, from 0 V to a reference voltage V_{ref}, into 2^N steps. The size of one step is the *least significant bit* (LSB):

$$\text{LSB} = \frac{V_{\text{ref}}}{2^N}, \qquad \text{code} = \left\lfloor \frac{V_{\text{in}}}{\text{LSB}} \right\rfloor, \qquad V_{\text{in}} \approx \text{code} \times \text{LSB}. \tag{3.18}$$

The Arduino on the demo board uses a 10-bit conversion with $V_{\text{ref}} = 3.3\,\text{V}$, so $2^{10} = 1024$ codes (0 to 1023) and an LSB of $3.3/1024 = 3.22\,\text{mV}$. Every voltage within one LSB maps to the same code, so the ADC cannot tell them apart. The difference between the true voltage and the voltage the code represents is the *quantization error*. The simple conversion $V \approx \text{code} \times \text{LSB}$, used throughout this book, gives the bottom edge of the step, so it can read low by up to one LSB; reading each code as the middle of its step, $(\text{code} + \frac{1}{2}) \times \text{LSB}$, limits the error to half an LSB (Fig. 3.15). For a thermistor reading the difference is a few hundredths of a degree. More bits give smaller steps: a 12-bit converter at 3.3 V has an LSB of 0.81 mV, and the 18-bit converter inside an optical PPG chip resolves about four parts per million of its range.

KEY EQUATION
$\text{LSB} = V_{\text{ref}}/2^N$
One ADC step. The board: 3.3 V and 10 bits give 3.22 mV.

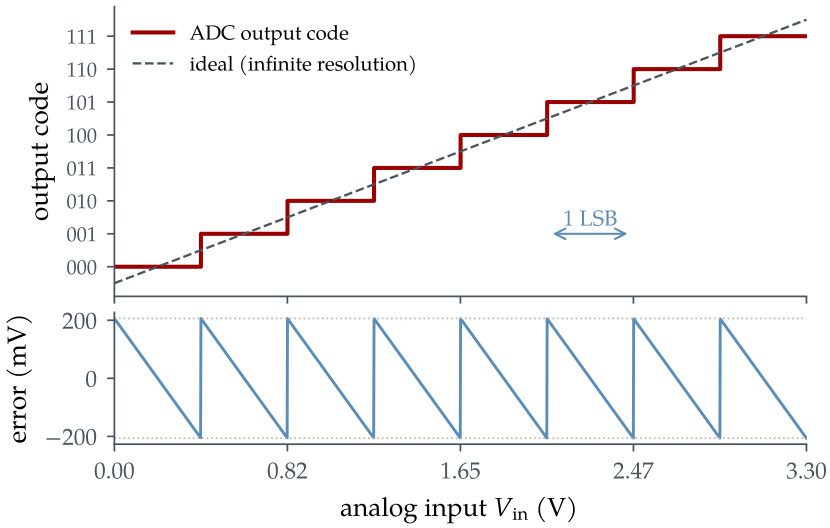


Figure 3.15: Quantization in a 3-bit ADC with $V_{\text{ref}} = 3.3\,\text{V}$ (8 levels, LSB = 0.41 V), chosen so the steps are visible. Top: the output code is a staircase that follows the ideal line. Bottom: the quantization error is a sawtooth between ± half an LSB when each code is read as the middle of its step.

The resolution of the whole measurement is the ADC step divided by the sensor sensitivity. That is the number that matters to the user.

Worked Example 3.12 *From ADC code to temperature*

The board's thermistor divider (Fig. 3.9, 10K 3950 part) returns the code 560. (a) What temperature does this represent? (b) What is the temperature resolution near 25 °C?

Solution. (a) $V_{\text{out}} = 560 \times 3.3/1024 = 1.805\,\text{V}$. From Eq. (3.9), $R_T = 10\,(3.3/1.805 - 1) = 8.29\,\text{k}\Omega$. Then Eq. (3.6) gives

$$\frac{1}{T} = \frac{1}{298.15} + \frac{1}{3950}\ln\frac{8.29}{10}, \qquad T = 302.44\,\text{K} = 29.3\,^\circ\text{C}.$$

(b) Near 25 °C the divider sensitivity is 36.7 mV per degree (Eq. (3.10)). One LSB is 3.22 mV, so the smallest temperature step the board can register is $3.22/36.7 = 0.088\,^\circ\text{C}$. The noise in Fig. 3.5 spans a few LSB, so in practice you average several samples. ■

Worked Example 3.13 *Sensitivity and resolution at room temperature*

In Lab 3.1 the room is at 22 °C and a pinch warms the bead to about 31.5 °C. (a) Find V_{out} and the ADC code at 22 °C. (b) Find the sensitivity and the temperature per LSB at 22 °C. (c) Find the *chord slope* $\Delta V/\Delta T$ between 22 °C and 31.5 °C, and the SNR of the rise if the RMS noise is 3 mV.

Solution. (a) $R_T = 10\,e^{3950(1/295.15 - 1/298.15)} = 11.44\,\text{k}\Omega$, so $V_{\text{out}} = 3.3 \times 10/(10 + 11.44) = 1.539\,\text{V}$, code $\lfloor 1.539/0.003223 \rfloor = 477$.

(b) Equation (3.10) with $R_T = 11.44\,\text{k}\Omega$ and $T = 295.15\,\text{K}$:

$$\frac{3.3 \times 10}{(21.44)^2} \times \frac{3950 \times 11.44}{295.15^2} = 37.2\,\text{mV/K},$$

so one LSB is $3.22/37.2 = 0.087\,^\circ\text{C}$.

(c) At 31.5 °C, $R_T = 7.54\,\text{k}\Omega$ and $V_{\text{out}} = 1.882\,\text{V}$. The rise is $1.882\,\text{V} - 1.539\,\text{V} = 0.343\,\text{V}$ over 9.5 °C, a chord slope of 36.1 mV/K. It is the average of the slope over the range, which falls from 37.2 mV/K at 22 °C to about 34 mV/K at 31.5 °C. The SNR is $20\log_{10}(343/3) = 41\,\text{dB}$. ■

Sampling rate and aliasing

The ADC also takes samples at a fixed rate f_s. How fast is fast enough? The *Nyquist criterion* says that a signal containing frequencies up to $f_{\max}$ can be captured without loss only if

$$f_s > 2 f_{\max}. \tag{3.19}$$

If you sample too slowly, a high frequency does not disappear. It shows up disguised as a lower frequency, an effect called *aliasing*. A component at frequency f sampled at f_s appears at $|f - kf_s|$ for the integer k that brings it closest to zero. Figure 3.16 shows 60 Hz power-line hum sampled at 50 Hz: the samples trace out a convincing but false 10 Hz wave. Once aliased, the false signal cannot be removed in software, which is why real systems place an analog low-pass filter in front of the ADC.

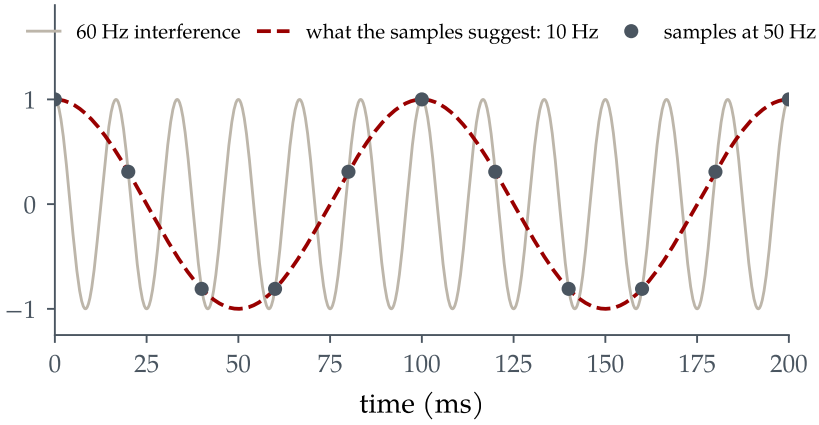


Figure 3.16: Aliasing. A 60 Hz wave sampled 50 times per second produces samples that lie on a 10 Hz wave. Nothing in the samples reveals that the true frequency was higher.

For heart rate the requirements are gentle. A resting heart beats about once per second, and even a racing one stays under 3.5 Hz (210 bpm). To keep the shape of the pulse you want several harmonics, say up to 15 Hz, so anything above 30 Hz works. The PPG chip on the board can sample at 50 to 3200 samples per second. Even its lowest setting, 50 samples per second, is comfortably above what a heartbeat needs. In Lab 3.2 the samples reach your notebook at about 130 per second (Section 3.8).

Worked Example 3.14 *Room-light flicker in a PPG*

Many LED and fluorescent room lights flicker at 120 Hz, twice the power-line frequency, and some of that light can reach the photodiode. Where does the flicker appear if the PPG is sampled at (a) 130, (b) 100, and (c) 400 samples per second? (d) Does it disturb a heart-rate measurement?

Solution. (a) The Nyquist frequency is 65 Hz, below 120 Hz, so the flicker aliases. With $k = 1$, $|120 - 130| = 10$ Hz. (b) $|120 - 100| = 20$ Hz. (c) The Nyquist frequency is now 200 Hz; the flicker is captured correctly at 120 Hz. (d) In (a) and (b) the false components sit at 10 and 20 Hz, well above the heart-rate band of 0.7 to 3.5 Hz, so the lab's band-pass filter removes them. Sampling near

119 or 121 samples per second would put the alias near 1 Hz, where it looks like a pulse. That is one reason the chip cancels ambient light itself, and why you cup a hand over the finger. ■

WORKED EXAMPLE 3.15 *Reading the thermistor for ten seconds*

In an Arduino sketch, a loop reads the divider on the pin named `m1_vd_input` with the function `analogRead()`, prints the raw code, increments a counter, and waits `delay(1)` before the next pass. It stops when the counter reaches a limit. Which pin should `m1_vd_input` be, and what counter limit records about ten seconds?

SOLUTION. The divider midpoint is wired to analog pin A1 (Fig. 3.9), so the sketch defines `m1_vd_input` as `A1`. Each pass waits 1 ms, so ten seconds is about $10\,\text{s}/1\,\text{ms} = 10\,000$ passes, and the stop test is `counter == 10000`. The real period is a little longer than 1 ms because the conversion and the `println` also take time; if the exact rate matters, record `millis()` with each sample. To convert a printed code to volts, multiply by 3.3/1024. (The course firmware's `divider` command and the Lab 3.1 notebook do this for you, and the notebook records the time of every reading.) ■

WORKED EXAMPLE 3.16 *Why ADCs and DACs*

What is the purpose of the ADC and the DAC in sensor systems?

SOLUTION. Sensors produce analog signals, and processors handle numbers. The *ADC* samples and quantizes the conditioned sensor voltage so the microcontroller can store, filter, and analyze it. The *DAC* goes the other way: it turns a number from the processor into an analog voltage that drives something in the real world. On the demo board the AD5593R chip provides the DAC channels. They set the LED drive in the biosensing block and sweep the diode and transistor bias in the devices block, while the Arduino's own 10-bit ADC reads the results on pins A2, A3, and A6. In an oximeter readout circuit (Section 3.5) a DAC also sets the LED current and supplies a voltage that cancels the large DC part of the photodiode signal so the small pulsatile part can be amplified. ■

3.5 *Op-Amp Circuits for Sensor Readout*

SENSOR SIGNALS ARE USUALLY TOO SMALL or in the wrong form for an ADC: a fraction of a millivolt from a strain bridge, a few hundred nanoamps from a photodiode, a divider output that must not be loaded. Chapter 2 introduced the tool for all three, the *operational am-*

plifier (op amp). Here is the short version. An op amp amplifies the difference between its two inputs (Fig. 3.17),

$$V_{\text{out}} = A\,(V_+ - V_-), \tag{3.20}$$

with an open-loop gain A of 10^5 to 10^6. Used alone it simply jams its output against a supply rail: with $A = 1000$, $V_+ = 1\,\text{V}$, and $V_- = 0.5\,\text{V}$, Eq. (3.20) asks for 500 V. With *negative feedback*, resistors from the output back to the inverting input, the ideal rules of Chapter 2 apply and the gain is set by resistor ratios. Chapter 2 used them for the difference amplifier that measures the diode current on the board and for the transimpedance amplifier that reads a photodiode. This section builds on those circuits and designs the readout for the sensors of this chapter: buffers that do not load a sensor, gain stages, difference amplifiers for bridges on a single 3.3 V supply, and transimpedance amplifiers, ending with a complete oximeter front end.

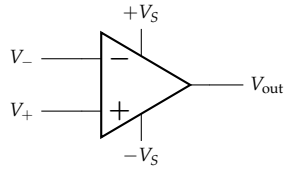


Figure 3.17: Op-amp symbol: inverting input (−), non-inverting input (+), output, and the two supply pins. On the demo board the lower supply pin is ground and the upper one 3.3 V.

Worked Example 3.17 *The op amp and its terminals*

What is an op amp? Draw one and label each terminal.

Solution. An op amp is an integrated circuit amplifier with a very large voltage gain that amplifies the difference between its two inputs, $V_{\text{out}} = A(V_+ - V_-)$. It has five essential terminals (Fig. 3.17): the *non-inverting input* V_+ (a rise here raises the output), the *inverting input* V_- (a rise here lowers the output), the *output*, and the *positive and negative supply* pins, which also set the limits of the output swing. ■

Worked Example 3.18 *The golden rules*

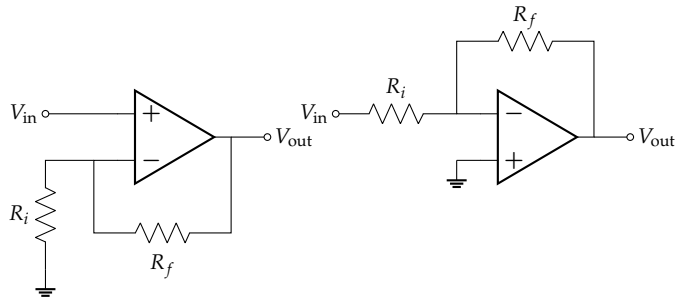
What are the golden rules of an op amp?

Solution. They are the ideal rules of Chapter 2:

1. No current flows into the inputs: $I_+ = I_- = 0$.
2. With negative feedback, the output adjusts so that the two inputs are at the same voltage: $V_+ = V_-$.
3. The output can supply whatever current the circuit requires (within its supply rails).

Rule 2 follows from the huge gain: a millivolt between the inputs would ask for hundreds of volts, so feedback moves the output until the difference is negligible. Picture the op amp as a water valve: the inputs are balloons that sense pressure but pass no water (rule 1), and a tiny pressure difference between them swings a valve that feeds the output from a high-pressure tank or a vacuum, which supply any flow (rule 3). ■

Non-inverting and inverting amplifiers



(a) non-inverting (b) inverting

Figure 3.18: The two basic feedback amplifiers. (a) Non-inverting: the input drives V_+ and the divider R_f, R_i feeds back a fraction of the output. (b) Inverting: V_+ is grounded, so V_- is a virtual ground and the input current V_{in}/R_i flows through R_f.

Worked Example 3.19 *The non-inverting amplifier*

Derive $V_{out} = f(V_{in})$ for Fig. 3.18(a), where V_{in} drives the non-inverting input, R_i runs from the inverting input to ground, and R_f runs from the output to the inverting input.

Solution. By rule 2, $V_- = V_+ = V_{in}$. By rule 1, no current enters the inverting input, so the current through R_f (from the output toward the inverting node) equals the current through R_i (from the inverting node to ground):

$$\frac{V_{out} - V_{in}}{R_f} = \frac{V_{in} - 0}{R_i}.$$

Multiply by R_f and collect terms:

$$V_{out} = V_{in}\left(1 + \frac{R_f}{R_i}\right) = V_{in}\,\frac{R_i + R_f}{R_i}.$$

The gain is positive (output in phase with input) and at least 1. With $R_f = 0$ and R_i removed, the circuit becomes a *voltage follower* or buffer with gain exactly 1. Because the input drives an op-amp input directly, it draws no current and does not load the sensor. ■

Worked Example 3.20 *The inverting amplifier*

Derive $V_{out} = f(V_{in})$ for Fig. 3.18(b). Then evaluate it for $R_i = 10\,\text{k}\Omega$ and $R_f = 100\,\text{k}\Omega$.

Solution. V_+ is grounded, so rule 2 puts V_- at 0 V. The inverting node is called a *virtual ground*: it sits at ground potential without being connected to ground. The current in R_i is $(V_{in} - 0)/R_i$, and

by rule 1 all of it continues through R_f:

$$\frac{V_{\text{in}} - 0}{R_i} = \frac{0 - V_{\text{out}}}{R_f} \implies V_{\text{out}} = -\frac{R_f}{R_i} V_{\text{in}}.$$

With these values the gain is $-100/10 = -10$: a 0.1 V input gives -1 V. The minus sign means the output is inverted (a 180° phase shift for a sine wave). The input resistance seen by the source is just R_i, because the other end of R_i is at virtual ground. On a board with a single positive supply this circuit needs a reference above ground on V_+, as the next subsections show. ■

KEY IDEA Non-inverting: $1 + R_f/R_i$, high input impedance, no sign flip. Inverting: $-R_f/R_i$, input impedance R_i.

The non-inverting amplifier draws no input current, which suits a high-impedance sensor. The inverting amplifier has a fixed input resistance and can add several inputs through separate resistors into the virtual ground (a *summing amplifier*).

Buffering a sensor

A voltage divider is a source with an internal resistance. By Thevenin's theorem (Chapter 1), the thermistor divider looks from A1 like a voltage $V_S R/(R_T + R)$ in series with $R_T \parallel R$, which is 5 kΩ at 25 °C. Anything connected to A1 that draws current forms a second divider with that resistance and pulls the reading down. The Arduino's ADC input draws very little, so the board reads the divider directly; a voltmeter, a filter, or a long cable may not. The cure is the voltage follower of Fig. 3.18(a) with $R_f = 0$: its input draws no current, and its output can drive the load.

WORKED EXAMPLE 3.21 *Loading the thermistor divider*

At 25 °C the board's divider gives 1.650 V. You connect a circuit with an input resistance of 50 kΩ to A1 and convert the reading to temperature as usual. (a) What voltage and temperature do you get? (b) What changes with a voltage follower between A1 and the load?

SOLUTION. (a) The Thevenin source is 1.650 V behind 5 kΩ, so the 50 kΩ load sees $1.650 \times 50/(5 + 50) = 1.500$ V. Equation (3.9) turns that into $R_T = 10\,(3.3/1.500 - 1) = 12.0$ kΩ and Eq. (3.6) into 20.95 °C: an error of −4.05 °C with nothing wrong with the sensor. (b) The follower's input draws no current (rule 1), so the divider stays at 1.650 V, and its output copies that voltage (rule 2) while supplying the 33 µA the load needs (rule 3). The reading is correct again. ■

Single-supply circuits and the difference amplifier

The op amps on the demo board (the devices block uses a TLV9102) run from one 3.3 V supply, so their outputs can only move between about 0 V and 3.3 V. A circuit that would need a negative output simply clips at 0 V. The fix is to measure from a *reference voltage* instead of ground. In the difference amplifier of Chapter 2 (Fig. 3.19) the resistor that went to ground on the non-inverting side now goes to V_{ref}. Rules 1 and 2 give

$$V_{\text{out}} = V_{\text{ref}} + \frac{R_f}{R_i}\,(V_2 - V_1). \tag{3.21}$$

The output sits at V_{ref} when the two inputs are equal and moves up or down from there with the difference. Choose $V_{\text{ref}} = 1.65\,\text{V}$ (half the supply) for a difference that can have either sign, and a value near 0 V for one that only grows. Whatever the two inputs share, such as the 1.65 V that both midpoints of a balanced bridge sit at, does not appear at the output. That makes the difference amplifier the natural partner of the Wheatstone bridge. (On the board, the diode and transistor measurements of Chapter 2 use the same circuit with all four resistors equal, a gain of 1.)

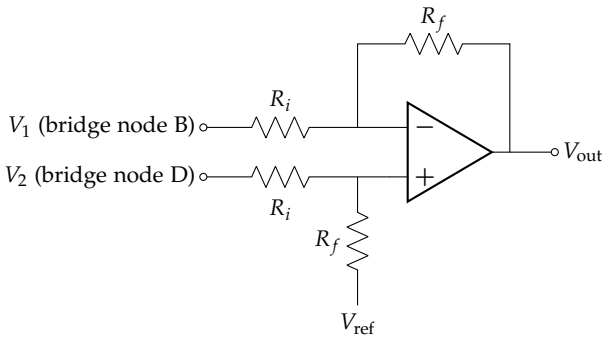


Figure 3.19: Difference amplifier with a reference input, here reading the two midpoints of the Wheatstone bridge of Fig. 3.14. The output is $V_{\text{ref}} + (R_f/R_i)(V_2 - V_1)$; the voltage the two inputs share cancels.

Worked Example 3.22 *Amplifying a Pt100 bridge on a single supply*

The Pt100 bridge of the example "Pt100 bridge at body temperature" ($V_S = 3.3\,\text{V}$, the other three arms $100\,\Omega$) drives the difference amplifier of Fig. 3.19 with $R_i = 10\,\text{k}\Omega$, $R_f = 200\,\text{k}\Omega$, and $V_{\text{ref}} = 0\,\text{V}$. The op amp runs from 3.3 V and has a gain-bandwidth product of 1 MHz. (a) Find V_{out} at 0, 37, and 50 °C. (b) Find the temperature per ADC step near 37 °C with and without the amplifier. (c) Estimate the amplifier's bandwidth.

Solution. (a) The gain is $R_f/R_i = 20$. At 0 °C the bridge is balanced and $V_{\text{out}} = 0$. At 37 °C the bridge gives 0.1107 V, so $V_{\text{out}} = 2.214\,\text{V}$. At 50 °C, $R_1 = 119.40\,\Omega$, the bridge gives $3.3\,(119.40/219.40 - 0.5) = 0.1459\,\text{V}$, and $V_{\text{out}} = 2.918\,\text{V}$, still below

the 3.3 V rail. (A real op amp cannot quite reach 0 V; a V_{ref} of a few hundred millivolts keeps the low end clear.)

(b) Node B is $V_S R_2/(R_1+R_2)$, so the bridge slope is $V_S R_2/(R_1+R_2)^2 \times \mathrm{d}R_1/\mathrm{d}T$. At 37 °C, $\mathrm{d}R_1/\mathrm{d}T = 100\,(A+2BT) = 0.3866\,\Omega/\mathrm{K}$ and

$$\frac{3.3 \times 100}{(214.38)^2} \times 0.3866 = 2.78\,\mathrm{mV/K}.$$

Read directly, one 3.22 mV step is 1.16 °C. After the gain of 20 the slope is 55.5 mV/K and one step is 0.058 °C.

(c) The amplifier's noise gain is $1 + R_f/R_i = 21$, so the bandwidth is about 1 MHz/21 = 48 kHz, far more than a temperature needs. The bridge nodes have Thevenin resistances near 50 Ω, which add to the 10 kΩ input resistors and lower the gain by about 0.5 %; an instrumentation amplifier, with buffered inputs, avoids even that. ■

Practical limits

Real op amps fall short of the ideal in predictable ways. The output cannot swing beyond the supply rails, so on a 3.3 V single-supply board anything that asks for more than 3.3 V (or less than 0 V) clips. The open-loop gain falls with frequency, so a closed-loop gain G has a bandwidth of roughly GBW/G, where the gain-bandwidth product GBW is a data sheet number (1 MHz is common; for the difference amplifier use the noise gain $1 + R_f/R_i$ in place of G). Small input bias currents and offset voltages are multiplied by the gain. For slow signals such as a heartbeat or a temperature, rail clipping and offsets matter far more than bandwidth.

The transimpedance amplifier

A photodiode produces a current proportional to the light that hits it (Chapter 2), and the *transimpedance amplifier* (TIA) of Chapter 2 turns that current into a voltage (Fig. 3.20). The photodiode connects directly to the virtual ground of an inverting amplifier with no input resistor, so it sits at zero bias, and the photocurrent I_S, which cannot enter the op amp, must flow through R_f:

$$\frac{0 - V_{\text{out}}}{R_f} = I_S \quad \Longrightarrow \quad V_{\text{out}} = -I_S R_f. \tag{3.22}$$

For a sensor readout three design choices matter. R_f sets the volts per amp, and with it the largest current before the output reaches the rail, $V_{\max}/R_f$. A capacitor C_f in parallel with R_f limits the bandwidth to $f_c = 1/(2\pi R_f C_f)$, which removes noise above the signal band and keeps the circuit stable. On a single supply the photodiode is turned

around, so its current flows *out of* the virtual ground and the output is positive, $V_{\text{out}} = +I_S R_f$.

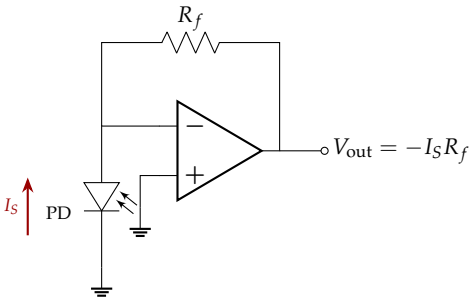


Figure 3.20: Transimpedance amplifier. The photocurrent I_S flows into the virtual ground and must leave through R_f, so $V_{\text{out}} = -I_S R_f$. Reversing the photodiode gives a positive output, which suits a single-supply board (Fig. 3.21).

Worked Example 3.23 *Sizing a photodiode readout*

A photodiode delivers $I_S = 10\,\mu\text{A}$ at full light, and the next stage wants $-3\,\text{V}$ at that light level. Choose R_f. Then choose C_f so the readout keeps frequencies up to 10 Hz, enough for a PPG.

Solution. From Eq. (3.22), $R_f = -V_{\text{out}}/I_S = 3\,\text{V}/10\,\mu\text{A} = 300\,\text{k}\Omega$. For the filter, $C_f = 1/(2\pi R_f f_c) = 1/(2\pi \times 300\,\text{k}\Omega \times 10\,\text{Hz}) = 53\,\text{nF}$; a standard 47 nF part gives $f_c = 11\,\text{Hz}$. ■

The biosensing block of the demo board has two such readouts, one for each LED and photodiode pair. Pair 1, which belongs to this module, is drawn in Fig. 3.21. The firmware command `light` turns its LED on and prints the photodiode reading. Pair 2 is the light link of Chapter 5.

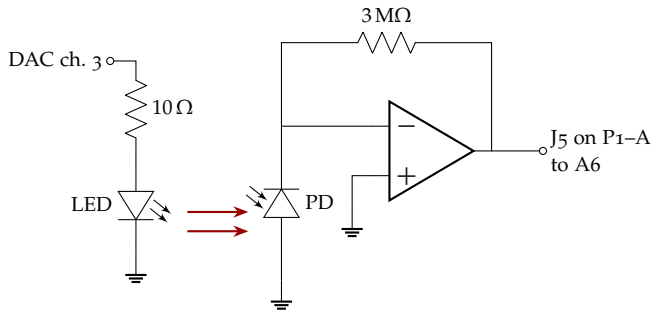


Figure 3.21: Optical pair 1 of the biosensing block. DAC channel 3 drives the LED through 10 Ω. The photodiode is connected so that its current flows out of the virtual ground, so the output is $+I_{\text{PD}} \times 3\,\text{M}\Omega$ and rises with light. Jumper J5 with its cap on P1–A (pins 1 and 2) connects it to A6 (A–P2 selects pair 2, the RX photodiode, instead).

Worked Example 3.24 *The board's photodiode amplifier*

With the selector on Module 3 and J5 on P1–A, suppose A6 reads code 31 with the LED of pair 1 off and code 372 with it on. (a) Find the photocurrent in each case and the current that one ADC step represents. (b) How much photocurrent does the LED

add? (c) What is the largest photocurrent the circuit can read? (d) Stray capacitance of about 1 pF sits across the 3 MΩ resistor. What bandwidth does that allow?

SOLUTION. (a) Code 31 is $31 \times 3.22\,\text{mV} = 0.0999\,\text{V}$, so

$$I = \frac{0.0999\,\text{V}}{3\,\text{M}\Omega} = 33.3\,\text{nA}$$

(room light and dark current). Code 372 is 1.199 V, so $I = 400\,\text{nA}$. One step is $3.22\,\text{mV}/3\,\text{M}\Omega = 1.07\,\text{nA}$. (b) $400 - 33.3 = 367\,\text{nA}$. (c) The output reaches the 3.3 V rail at $3.3\,\text{V}/3\,\text{M}\Omega = 1.1\,\mu\text{A}$; any more light reads as code 1023. (d) $f_c = 1/(2\pi \times 3\,\text{M}\Omega \times 1\,\text{pF}) = 53\,\text{kHz}$, ample for a pulse and for the millisecond light pulses of Chapter 5. A large R_f gives many volts per amp but a low bandwidth for a given capacitance; that is the main trade-off of a TIA. ■

An oximeter readout chain

A complete oximeter can be built from discrete parts (Fig. 3.22). A driver circuit turns the red and green (or infrared) LEDs on in turn and sets their current from a DAC. A TIA (stage 1) converts the photodiode current to a voltage that contains both a large DC part and a small pulsatile AC part. Amplifying that directly would saturate on the DC part. So stage 2 subtracts a DAC voltage equal to the DC level and amplifies only what is left. For the circuit in Fig. 3.22, rules 1 and 2 give

$$V_{\text{out}} = V_{\text{DAC}} - \frac{R_{2f}}{R_2}\,(V_1 - V_{\text{DAC}})\,, \tag{3.23}$$

so when V_{DAC} equals the DC part of V_1, only the AC part is multiplied by the gain R_{2f}/R_2. Stage 2 is the inverting amplifier with a reference: V_{DAC} plays the role of V_{ref} and keeps the output inside the supply. The MAX86916 on the demo board does the same job inside the chip, with ambient-light cancellation and a high-resolution ADC in place of the DAC subtraction (Section 3.8).

WORKED EXAMPLE 3.25 *Removing the DC before amplifying*

Suppose the pair-1 photodiode of Fig. 3.21 were used for a PPG: 0.40 µA of DC photocurrent with a pulsatile part of 1 % peak to peak, a stage-1 TIA with $R_f = 3\,\text{M}\Omega$ and a positive output ($V_1 = +IR_f$), and the board's 10-bit ADC. (a) Find the DC level and the AC swing of V_1, and the AC swing in ADC steps. (b) Why not simply amplify V_1 by 100? (c) Stage 2 uses $V_{\text{DAC}} = 1.20\,\text{V}$ and $R_{2f}/R_2 = 100$. Between which voltages does the output swing, and how many ADC steps is that?

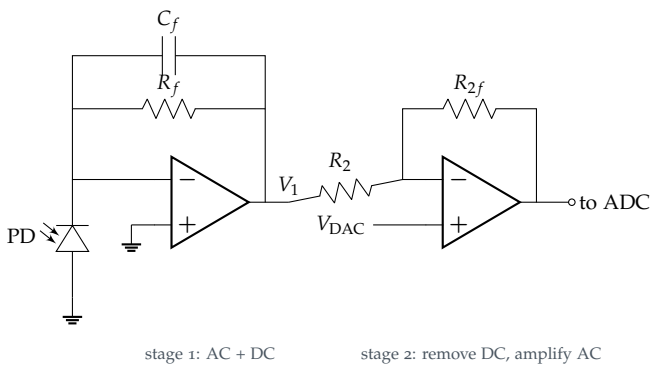


Figure 3.22: A two-stage photodiode readout. Stage 1 is a TIA with a filter capacitor. Stage 2 subtracts a DAC voltage equal to the DC level of V_1 and amplifies the remaining pulsatile signal.

SOLUTION. (a) $V_1 = 0.40\,\mu\text{A} \times 3\,\text{M}\Omega = 1.20\,\text{V}$ DC. The AC part is $4\,\text{nA} \times 3\,\text{M}\Omega = 12\,\text{mV}$ peak to peak, only $12/3.22 = 3.7$ ADC steps: the pulse would be a staircase a few steps tall.

(b) $1.20 \times 100 = 120\,\text{V}$. The output would sit at the rail and the pulse would be lost.

(c) V_1 swings from 1.194 to 1.206 V. Equation (3.23) gives $1.20 - 100\,(1.194 - 1.20) = 1.80\,\text{V}$ and $1.20 - 100\,(1.206 - 1.20) = 0.60\,\text{V}$: an inverted swing of 1.2 V peak to peak centered on 1.20 V, or 372 ADC steps. ■

3.6 *Signals in Time and Frequency*

A RAW PPG RECORDING mixes three things: the pulse wave from your heartbeat (around 1 Hz), slow changes from breathing and movement (well below 1 Hz), and fast noise from electronics and small motions (tens of hertz and up). They overlap in time but sit in different places in *frequency*. The Fourier transform is the tool that moves a signal from one view to the other.

The Fourier transform

Any repeating signal can be written as a sum of sine waves at a fundamental frequency and its multiples (harmonics), each with its own amplitude and phase. A square wave, for example, is the sum of its odd harmonics with amplitudes falling as $1/n$:

$$x(t) = \frac{4}{\pi}\left[\sin(2\pi f_0 t) + \tfrac{1}{3}\sin(2\pi\,3 f_0 t) + \tfrac{1}{5}\sin(2\pi\,5 f_0 t) + \cdots\right]. \quad (3.24)$$

The *Fourier transform* finds those amplitudes. Its plot against frequency, the *spectrum*, shows one spike for each sine component (Fig. 3.23). Sharp edges need many high harmonics; smooth, slowly varying signals need only a few low ones.

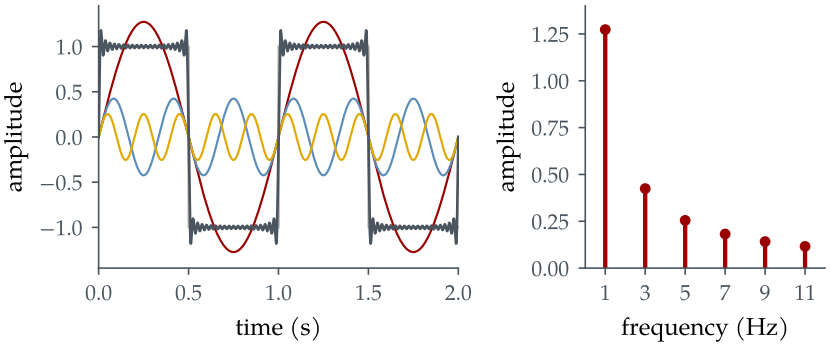


Figure 3.23: Left: a 1 Hz square wave (gray) and its first three odd harmonics at 1, 3, and 5 Hz; the dark curve adds the first twenty odd harmonics and already looks square. Right: the spectrum, one line per harmonic, with heights $4/(\pi n)$.

For sampled data we use the *discrete Fourier transform* (DFT). For N samples $x[n]$,

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N}, \qquad k = 0, 1, \ldots, N-1, \tag{3.25}$$

and the inverse transform rebuilds the signal from its components, $x[n] = \frac{1}{N}\sum_k X[k]\, e^{j2\pi kn/N}$. Bin k corresponds to the frequency kf_s/N, so the spacing between bins is

$$\Delta f = \frac{f_s}{N} = \frac{1}{T_{\text{record}}}, \tag{3.26}$$

the inverse of the recording length. A 10-second recording resolves frequencies 0.1 Hz apart, which is 6 bpm in heart-rate units. The *fast Fourier transform* (FFT) is an efficient algorithm for computing the DFT; `numpy.fft` implements it.

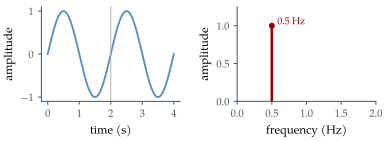


Figure 3.24: A sine with a period of 2 s (left) has a single Fourier component at 0.5 Hz (right).

Worked Example 3.26 *Fourier component of a sine*

A signal is a pure sine wave that completes one full cycle every 2 s (it returns to zero, going upward, at 2 s and 4 s). Draw its Fourier component.

Solution. The period is $T = 2$ s, so the frequency is $f = 1/T = 0.5$ Hz. A pure sine contains only one frequency, so the spectrum is a single line at 0.5 Hz whose height is the sine's amplitude (Fig. 3.24). There is no DC line because the sine averages to zero, and no harmonics because the wave is perfectly smooth. (A two-sided FFT would show a mirror line at -0.5 Hz as well.) ■

Filters

A *filter* keeps some frequencies and removes others (Fig. 3.25). A *low-pass* filter keeps frequencies below its cutoff f_c and smooths away fast noise; a *high-pass* filter keeps those above f_c and removes drift and DC;

a *band-pass* filter keeps a band, such as 0.5 Hz to 5 Hz for a heartbeat; a *notch* removes a narrow band, such as 60 Hz hum. The RC circuit of Chapter 1 is the simplest analog low-pass, with $f_c = 1/(2\pi RC)$, and so is the C_f across a TIA's feedback resistor. Its gain falls gradually through the cutoff:

$$|H(f)| = \frac{1}{\sqrt{1 + (f/f_c)^2}}, \tag{3.27}$$

which is $1/\sqrt{2} = 0.707$ (-3 dB) at f_c and falls by a factor of ten for every decade above it. Swapping R and C gives a high-pass whose gain is $(f/f_c)/\sqrt{1 + (f/f_c)^2}$. In software, a few lines of Python filter recorded data, and higher-order digital filters, such as the Butterworth filters of Section 3.9, fall much faster above the cutoff.

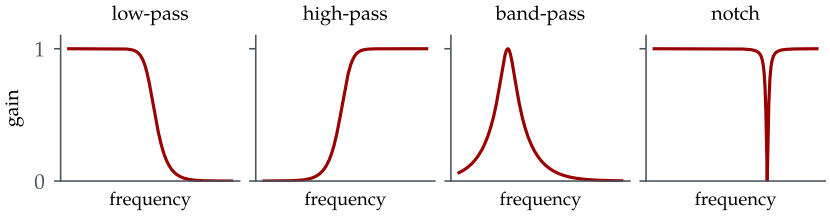


Figure 3.25: Gain versus frequency (log scale) for the four basic filter types. Each passes the frequencies where its gain is near 1 and blocks those where it is near 0.

Worked Example 3.27 *Filter outputs*

Two signals pass through filters. The first is a slow square-ish wave with fast small wiggles riding on it, sent into a low-pass filter (LPF). The second is a slow, smooth sine wave sent into a high-pass filter (HPF). Draw the outputs.

Solution. Think in frequency. The first signal is a slow square wave (fundamental plus odd harmonics) plus a fast wiggle. An LPF whose cutoff sits between them removes the wiggle and keeps the square shape; the output is the clean square wave (Fig. 3.26, top right, solid). If the cutoff were very low, below the third harmonic, only the fundamental would survive and the output would look like a sine at the square wave's frequency (dashed). The second signal contains a single low frequency. An HPF with a cutoff above that frequency blocks it, so the output is essentially zero, a flat line. If the cutoff were far below the sine frequency, the sine would pass unchanged. ■

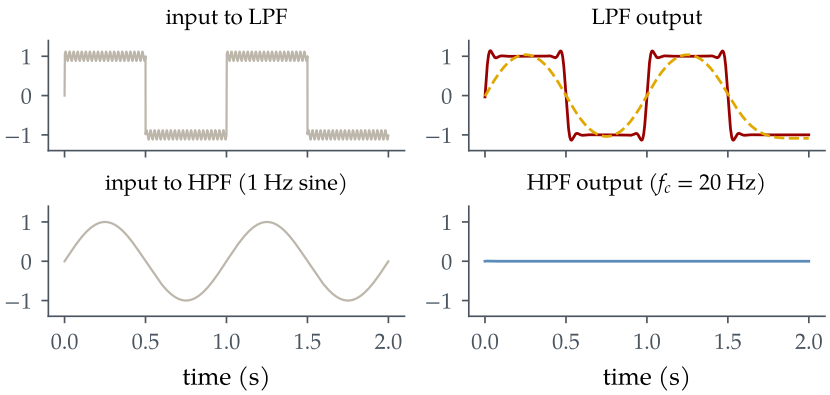


Figure 3.26: Filter outputs with representative numbers. Top: a 1 Hz square wave with a 40 Hz wiggle; a 15 Hz low-pass removes the wiggle (solid), and a 1.5 Hz low-pass leaves mostly the fundamental (dashed). Bottom: a 1 Hz sine through a 20 Hz high-pass comes out as nearly nothing.

Worked Example 3.28 *An anti-aliasing filter on the divider*

A 1 µF capacitor is added from A1 to ground (Fig. 3.27). (a) Find the cutoff frequency at 25 °C. (b) By how much does it reduce 60 Hz hum? (c) Does it slow down the reading of a finger touch?

Solution. (a) The capacitor sees the divider's Thevenin resistance, $R_T \parallel R = 10\,\text{k}\Omega \parallel 10\,\text{k}\Omega = 5\,\text{k}\Omega$ at 25 °C, so $f_c = 1/(2\pi \times 5\,\text{k}\Omega \times 1\,\mu\text{F}) = 31.8\,\text{Hz}$. (At 32 °C, $R_T \parallel R = 4.25\,\text{k}\Omega$ and the cutoff moves up to 37.5 Hz; it depends a little on temperature.) (b) Eq. (3.27) at 60 Hz: $1/\sqrt{1+(60/31.8)^2} = 0.47$, or −6.6 dB. The hum is roughly halved, before any sampling can alias it. (c) No. The RC time constant is $5\,\text{k}\Omega \times 1\,\mu\text{F} = 5\,\text{ms}$, a thousand times shorter than the thermal time constant of the bead (about 6 s, Section 3.3). A thermometer could use a much larger capacitor and a much lower cutoff (Problem 3.22). ■

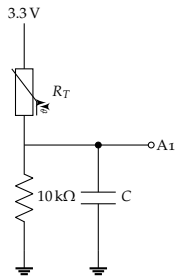


Figure 3.27: A capacitor from A1 to ground turns the thermistor divider into a low-pass filter whose resistance is the divider's Thevenin resistance, $R_T \parallel R$.

3.7 *Optical Biosensing and the Beer-Lambert Law*

Wearable devices now track heart rate, breathing rate, skin temperature, and blood oxygen around the clock, and these *vital signs* are most useful together and over time. Several of them come from one optical sensor pressed against the skin (Fig. 3.28).

Blood carries oxygen bound to *hemoglobin*, and each hemoglobin molecule is either oxygenated (HbO_2) or deoxygenated (Hb). The *oxygen saturation* is the fraction that carries oxygen:

$$SO_2 = \frac{C_{HbO_2}}{C_{HbO_2} + C_{Hb}}, \tag{3.28}$$

where C denotes concentration. Healthy arterial blood is 95 % to 100 %

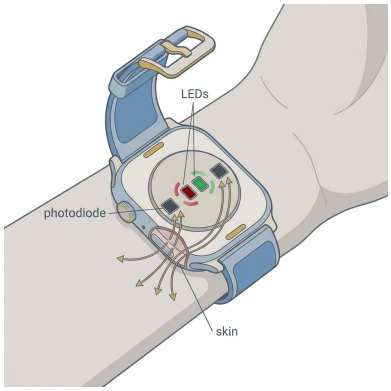


Figure 3.28: An optical sensor on the back of a smartwatch. LEDs shine into the wrist and photodiodes a few millimeters away collect the light that scatters back out. The same principle runs the oximeter on the demo board.

saturated. A value below about 90 % (*hypoxemia*) calls for medical attention. A *pulse oximeter* estimates arterial saturation through the skin; its reading is written $\mathrm{SpO_2}$, the "p" standing for pulse. It works because the two forms of hemoglobin absorb light differently.

The Beer-Lambert law

Shine light of intensity I_0 through a sample of thickness ℓ that contains an absorbing substance at concentration c (Fig. 3.29). Each thin slice $\mathrm{d}x$ absorbs the same *fraction* of the light that reaches it, so the intensity decays exponentially with depth. In base-10 form,

$$I = I_0\,10^{-\varepsilon c \ell}, \qquad T = \frac{I}{I_0}, \qquad A = -\log_{10} T = \varepsilon c \ell. \tag{3.29}$$

Here T is the *transmittance*, the fraction of light that gets through (between 0 and 1), and A is the dimensionless *absorbance*. The *molar absorption coefficient* (also called molar extinction coefficient) ε, in $\mathrm{M^{-1}\,cm^{-1}}$, says how strongly one mole per liter of the substance absorbs at the chosen wavelength; it depends strongly on wavelength. The concentration c is in M (moles per liter) and the path length ℓ in cm. When several substances absorb, their absorbances add: $A_{\text{total}} = \sum_i \varepsilon_i c_i \ell$. Some texts use the natural-log form $I = I_0 e^{-\varepsilon' c x}$, which is the same law with $\varepsilon' = \varepsilon \ln 10$.

KEY EQUATION

$A = -\log_{10}(I/I_0) = \varepsilon c \ell$

Beer-Lambert law: absorbance grows linearly with concentration and path length.

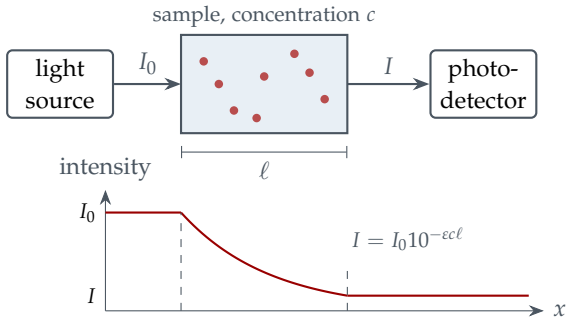


Figure 3.29: Beer-Lambert law. Light of intensity I_0 enters a sample of thickness ℓ with absorbers at concentration c. The intensity decays exponentially inside the sample and stays constant after it leaves, where a photodetector measures I.

WORKED EXAMPLE 3.29 *Absorbance of benzene in a cuvette*

Light at 254 nm is attenuated to 16 % of its incident value after passing through a 50 mM solution of benzene in a 1.0 mm quartz cuvette. Find the absorbance A and the molar absorption coefficient ε. What would the transmittance be through a 2.0 mm cuvette?

SOLUTION. The transmittance is $T = 0.16$, so $A = -\log_{10} 0.16 = 0.796 \approx 0.80$. With $c = 0.050\,\mathrm{M}$ and $\ell = 0.10\,\mathrm{cm}$,

$$\varepsilon = \frac{A}{c\ell} = \frac{0.796}{(0.050)(0.10)} = 159\,\mathrm{M^{-1}\,cm^{-1}} \approx 160\,\mathrm{M^{-1}\,cm^{-1}}.$$

Doubling the path doubles the absorbance to $A = 1.59$, so $T = 10^{-1.59} = 0.0256$, or 2.6 %. Equivalently, $T_{2\,\mathrm{mm}} = (0.16)^2$: each millimeter passes 16 % of what enters it. ■

Worked Example 3.30 *The terms of the Beer-Lambert law*

Write the Beer-Lambert law and explain each term.

Solution. $A = -\log_{10}(I/I_0) = \varepsilon c \ell$, or equivalently $I = I_0\, 10^{-\varepsilon c \ell}$. I_0 is the light intensity entering the sample and I the intensity leaving it. $I/I_0 = T$ is the transmittance. A is the absorbance, a dimensionless logarithmic measure of attenuation. ε is the molar absorption coefficient of the substance at the wavelength used ($\mathrm{M^{-1}\,cm^{-1}}$), a property of the molecule that changes with color. c is the concentration of the absorbing substance (M) and ℓ is the optical path length through the sample (cm). The law says absorbance is proportional to both concentration and path length. ■

Why two colors of light

Figure 3.30 shows how strongly the two forms of hemoglobin absorb across the red and near-infrared. In the red, near 660 nm, deoxyhemoglobin absorbs about ten times more strongly than oxyhemoglobin; that is why oxygen-poor venous blood looks darker. In the near-infrared, past about 800 nm, the order flips and oxyhemoglobin absorbs more. The two curves cross at the *isosbestic point*, near 800 nm, where both forms absorb equally.

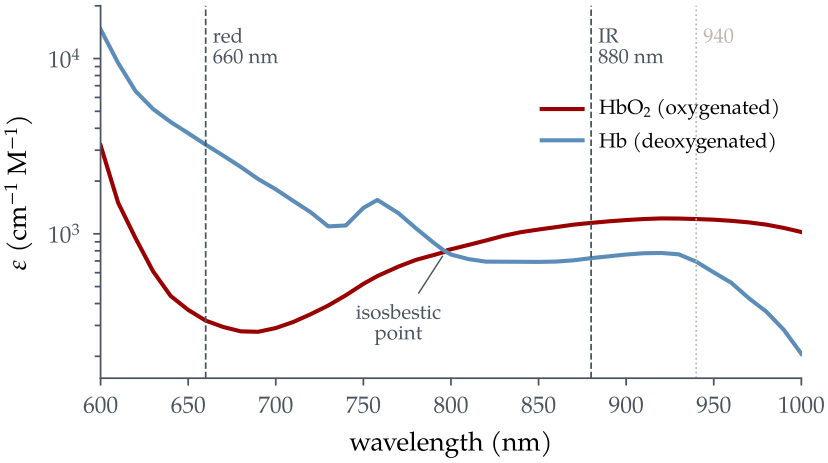


Figure 3.30: Approximate molar extinction coefficients of oxyhemoglobin and deoxyhemoglobin, from S. Prahl's tabulation (Oregon Medical Laser Center), on a log scale. The LEDs of the board's optical sensor sit near 660 nm and 880 nm; many clinical probes use 940 nm for the infrared channel.

Worked Example 3.31 *Two wavelengths*

Why do we need two different lights in an oximeter?

Solution. There are two unknowns, the concentrations of HbO_2 and Hb (or equivalently the total hemoglobin and the saturation), so we need two independent measurements. By Beer-Lambert, the absorbance at each wavelength is $A_\lambda = [\varepsilon_{HbO_2}(\lambda)\, C_{HbO_2} + \varepsilon_{Hb}(\lambda)\, C_{Hb}]\,\ell$. Choosing one wavelength where Hb absorbs much more (red, 660 nm) and one where HbO_2 absorbs more (infrared, 880 nm to 940 nm) makes the two equations as different as possible, so the ratio of the two signals changes strongly with saturation. A single wavelength cannot separate a change in oxygenation from a change in the amount of blood or the path length. Normalizing each wavelength by its own steady level and comparing the two (Section 3.8) also cancels factors they share, such as LED brightness and the thickness of the finger. The static absorption of skin pigment drops out only approximately, which is one reason oximeters have been found to read too high in some patients with darker skin. ■

Worked Example 3.32 *Two absorbers, two wavelengths*

A 1.00 cm cuvette holds diluted blood. Its absorbance is 0.0451 at 660 nm and 0.0534 at 880 nm. Use the coefficients $\varepsilon_{HbO_2} = 320$ and $\varepsilon_{Hb} = 3227$ at 660 nm, and $\varepsilon_{HbO_2} = 1154$ and $\varepsilon_{Hb} = 726$ at 880 nm (all in $M^{-1}\,cm^{-1}$, read from Fig. 3.30). Find the two concentrations and the saturation.

Solution. Absorbances add, so with $x = C_{HbO_2}$, $y = C_{Hb}$, and $\ell = 1$ cm:

$$320\,x + 3227\,y = 0.0451,$$
$$1154\,x + 726\,y = 0.0534.$$

The determinant is $320 \times 726 - 3227 \times 1154 = -3.4916 \times 10^6$, and Cramer's rule gives

$$x = \frac{0.0451 \times 726 - 3227 \times 0.0534}{-3.4916 \times 10^6} = 40.0\,\mu M,$$
$$y = \frac{320 \times 0.0534 - 1154 \times 0.0451}{-3.4916 \times 10^6} = 10.0\,\mu M.$$

The saturation is $40.0/(40.0 + 10.0) = 80\,\%$. In a cuvette the path is known and the light goes straight through, so the two equations can be solved exactly. In a finger neither is true, which is why the oximeter works with ratios and an empirical calibration instead. ■

Light in tissue

A finger is not a clean cuvette. Light in the skin scatters many times and is absorbed by melanin, water, tissue, venous blood, and arterial blood. Only arterial blood changes from moment to moment: with each heartbeat the arteries swell slightly and absorb a little more light. That small, pulsing change is what a pulse oximeter measures (Fig. 3.31).

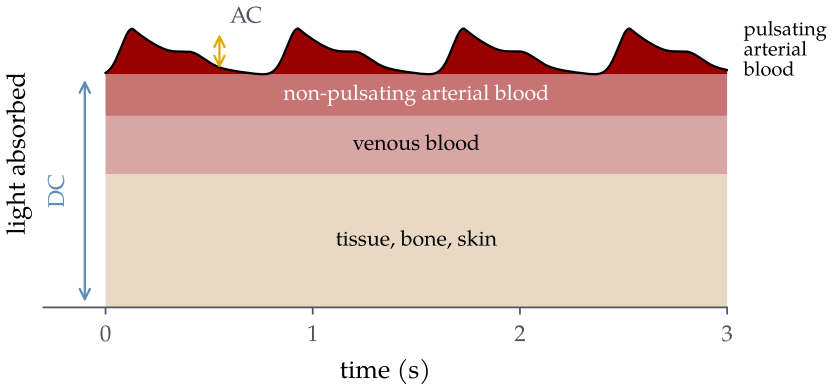
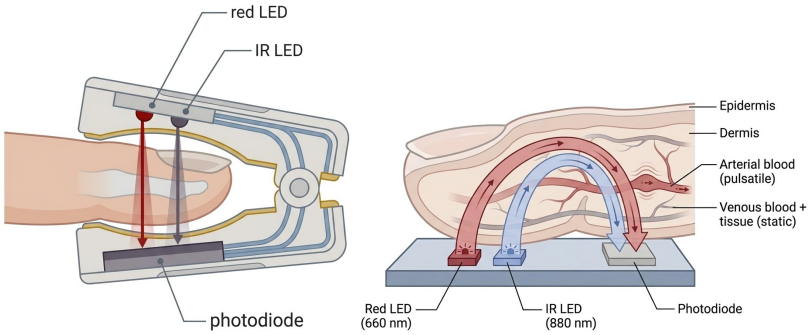


Figure 3.31: Where the light goes. Tissue, venous blood, and the steady part of arterial blood absorb a constant amount (the DC level). Only the volume of arterial blood pulses with each heartbeat, adding a small AC component on top.

There are two ways to arrange the optics. In *transmission mode* (Fig. 3.32a) the LEDs sit on one side of a fingertip or earlobe and the photodiode on the other, and the light must pass all the way through. This gives strong signals but only works where tissue is thin enough to shine through. In *reflectance mode* (Fig. 3.32b) the LEDs and photodiode sit side by side on the same surface. Light that scatters through the tissue in curved, banana-shaped paths returns to the surface and reaches the photodiode. Reflectance sensors work almost anywhere on the body, which is what makes wrist and forehead wearables possible; the forehead gives the largest pulsatile signal, the chest and legs the weakest.



(a) transmission (b) reflectance

Figure 3.32: The two optical arrangements. (a) A transmission-mode clip: red and infrared LEDs above the fingernail shine through the fingertip to a photodiode underneath. (b) Reflectance mode, as on the demo board: light from the LEDs travels through the epidermis and dermis along curved paths and returns to a photodiode beside them. Pulsing arterial blood modulates the returned light; static tissue and venous blood set the steady level.

3.8 *Photoplethysmography and Pulse Oximetry*

THE SIGNAL FROM THE PHOTODIODE over time is called a *photoplethysmogram* (PPG): a record (gram) of volume changes (plethysmo-) measured with light (photo-). Figure 3.33 shows red and infrared PPGs like the ones the board records. Each has two parts. The *DC component* is the large, steady level set by light absorbed in tissue, bone, venous blood, and non-pulsating arterial blood. The *AC component* is the small ripple on top, caused by the surge of arterial blood with each heartbeat. The AC part is typically around one percent of the DC part and rarely more than a few percent.

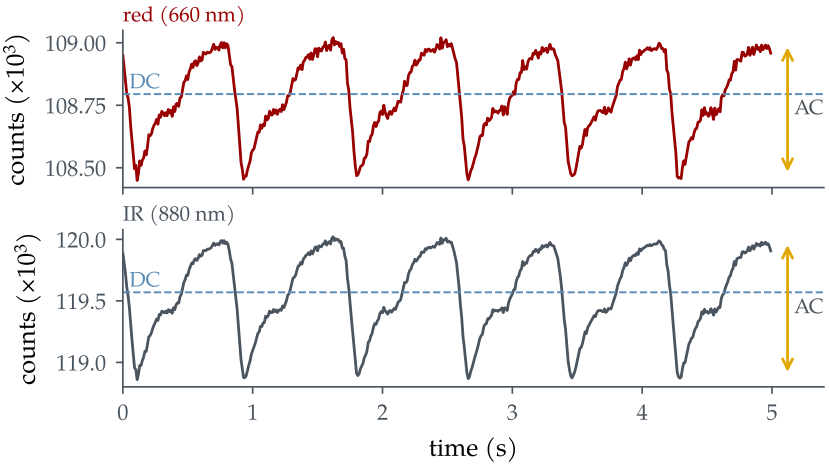


Figure 3.33: Simulated red and infrared PPGs in raw ADC counts, with levels similar to the board's recordings (about 109 000 counts red and 120 000 counts infrared). The light reaching the photodiode drops sharply when each pulse of blood arrives (systole) and recovers slowly. The DC level is the mean; the AC amplitude is the peak-to-peak swing. The AC/DC ratio is about 0.5% for red and 0.9% for IR here.

Notice the direction. More blood absorbs more light, so the detected intensity *falls* when a pulse arrives. Many displays flip the PPG so the beat points upward; the board's raw data are not flipped.

The ratio of ratios

The raw red and infrared signals are hard to compare directly. The two LEDs have different brightness, the photodiode has different sensitivity at the two wavelengths, and tissue absorbs them differently. The fix is to divide each signal's AC part by its own DC part (Fig. 3.34). The normalized quantity AC/DC measures how much of the light the pulsing blood removes, and LED brightness and detector gain cancel out of it. The pulse oximeter then compares the two normalized signals through the *ratio of ratios*:

$$R = \frac{\mathrm{AC}_{\mathrm{red}}/\mathrm{DC}_{\mathrm{red}}}{\mathrm{AC}_{\mathrm{IR}}/\mathrm{DC}_{\mathrm{IR}}}. \tag{3.30}$$

KEY EQUATION

$$R = \frac{\mathrm{AC}_{\mathrm{red}}/\mathrm{DC}_{\mathrm{red}}}{\mathrm{AC}_{\mathrm{IR}}/\mathrm{DC}_{\mathrm{IR}}}$$

The ratio of ratios: one number that tracks oxygen saturation.

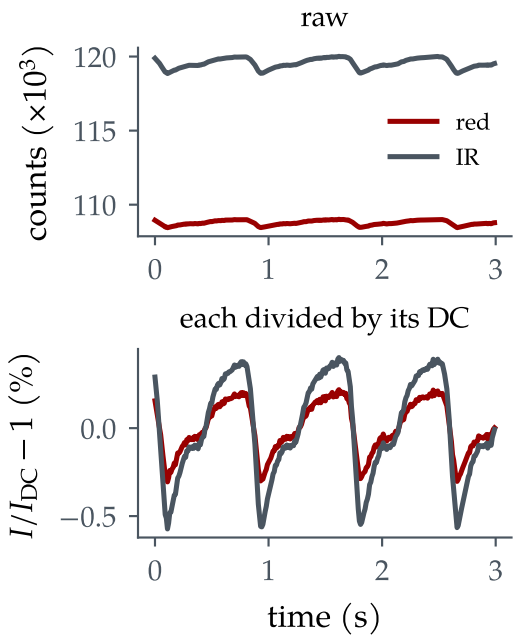


Figure 3.34: Normalization. Top: raw red and IR signals have different DC levels and AC amplitudes. Bottom: dividing each by its own DC level puts them on the same footing. The ratio of the two normalized swings is R.

Why does R track saturation? When a pulse arrives, the arterial blood adds a small absorbance ΔA_λ at each wavelength, and by Eq. (3.29) the detected light drops from I_{DC} to $I_{DC}\,10^{-\Delta A_\lambda}$. For a small pulse, $10^{-\Delta A} = e^{-\Delta A \ln 10} \approx 1 - \Delta A \ln 10$, so

$$\frac{\text{AC}}{\text{DC}} \approx (\ln 10)\,\Delta A_\lambda = 2.303\,\Delta A_\lambda. \tag{3.31}$$

The factor $\ln 10$ is the same at both wavelengths and cancels in R. By Beer-Lambert, ΔA_λ is set by the mix of HbO_2 and Hb in the pulsing blood. Writing S for the saturation and $\Delta\ell$ for the change in path length,

$$R = \frac{\Delta A_{\text{red}}}{\Delta A_{\text{IR}}} = \frac{\varepsilon_{\text{HbO}_2,\text{red}}\,S + \varepsilon_{\text{Hb,red}}\,(1-S)}{\varepsilon_{\text{HbO}_2,\text{IR}}\,S + \varepsilon_{\text{Hb,IR}}\,(1-S)}. \tag{3.32}$$

The concentration and $\Delta\ell$ cancel because they appear at both wavelengths. When S is high the blood absorbs little red and plenty of IR, so R is small. When S falls, red absorption climbs and IR absorption drops, so R grows. Measuring R therefore tells us S.

Worked Example 3.33 *From AC/DC to absorbance*

A recording gives AC/DC = 1.0 % in the infrared and 0.40 % in the red. (a) How much absorbance does one pulse of arterial blood add at each wavelength? (b) Find R. (c) What saturation does Eq. (3.32) predict with the coefficients of the example "Two absorbers, two wavelengths"?

Solution. (a) By Eq. (3.31), $\Delta A_{\text{IR}} = 0.010/2.303 = 4.34 \times 10^{-3}$

and $\Delta A_{\text{red}} = 0.0040/2.303 = 1.74 \times 10^{-3}$. A heartbeat changes the absorbance of the fingertip by a few thousandths. (b) $R = 0.40/1.0 = 0.40$ (the same as $\Delta A_{\text{red}}/\Delta A_{\text{IR}}$). (c) Solving Eq. (3.32) for S (Problem 3.28) gives

$$S = \frac{3227 - 726R}{(3227 - 320) + R\,(1154 - 726)} = \frac{3227 - 290.4}{2907 + 171.2} = 0.954.$$

The pure Beer-Lambert model says 95.4 %; the empirical line below says 100 % for the same R. The next subsection explains the difference. ■

Calibration

Equation (3.32) assumes light travels the same path at both wavelengths. In real tissue red and infrared light scatter differently and travel different average paths, so the pure Beer-Lambert curve misses by several percent. Manufacturers therefore calibrate each oximeter design empirically. They record R from volunteers whose arterial blood is sampled and analyzed at the same time, and fit a curve. A common simplified version, used in the lab, is the straight line

$$\text{SpO}_2\,(\%) = 110 - 25R. \tag{3.33}$$

Figure 3.35 compares this line with the Beer-Lambert prediction computed from the extinction coefficients in Fig. 3.30.

KEY EQUATION

$\text{SpO}_2 = 110 - 25R$

Empirical calibration used in the lab.

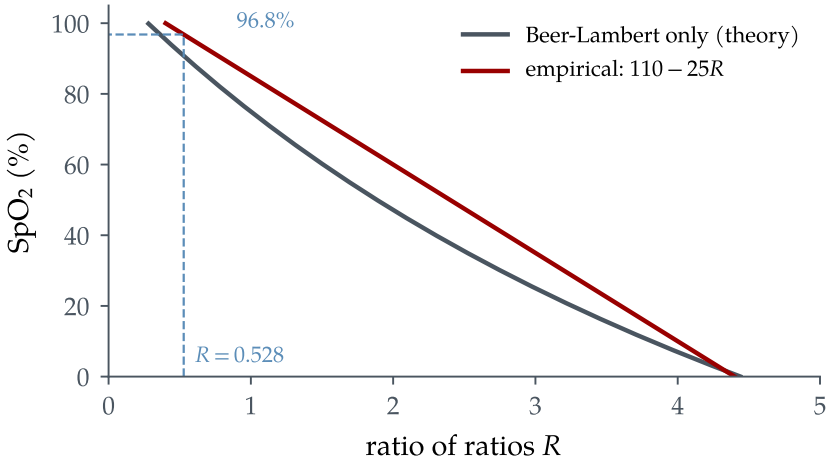


Figure 3.35: Calibration curves. The gray curve is Eq. (3.32) with the approximate coefficients at 660 nm and 880 nm. The cardinal line is the empirical calibration used in the lab. The two disagree mainly because scattering makes the red and IR paths unequal. The dashed lines mark $R = 0.528$, the value in the next worked example.

WORKED EXAMPLE 3.34 *Computing SpO_2*

What are the AC and DC parts of a PPG signal? If $\text{Red}_{\text{AC}} = 20\,\text{mV}$, $\text{Red}_{\text{DC}} = 625\,\text{mV}$, $\text{NIR}_{\text{AC}} = 20\,\text{mV}$, and $\text{NIR}_{\text{DC}} = 330\,\text{mV}$,

calculate $\mathrm{SpO_2}$.

SOLUTION. The DC part is the steady level due to light absorbed by tissue, bone, venous blood, and non-pulsating arterial blood. The AC part is the pulsating component caused by the change in arterial blood volume with each heartbeat. Then

$$R = \frac{20/625}{20/330} = \frac{0.0320}{0.0606} = 0.528,$$

$$\mathrm{SpO_2} = 110 - 25(0.528) = 96.8\,\%.$$

This is a normal, healthy reading. Note that the AC amplitudes are equal here; what matters is that the red AC is a smaller *fraction* of its DC level than the IR AC is of its own. ■

WORKED EXAMPLE 3.35 *Readings of 0% and 100%*

Under what conditions will an oximeter read 0 % and 100 %?

SOLUTION. By Eq. (3.28), saturation is 100 % when every hemoglobin molecule carries oxygen, $C_{\mathrm{Hb}} = 0$, and 0 % when none does, $C_{\mathrm{HbO_2}} = 0$. On the empirical calibration line, $110 - 25R = 100$ gives $R = 0.4$, and $110 - 25R = 0$ gives $R = 4.4$. So the reading is 100 % when the red pulsatile signal, relative to its DC level, is only 0.4 times the infrared one; it would be 0 % when the red ratio is 4.4 times the infrared one. In practice oximeters are calibrated only down to about 70 %, because volunteers cannot safely be made more hypoxic than that, so readings far below that range are extrapolations. ■

WORKED EXAMPLE 3.36 *A reading above 100%*

A student computes $R = 0.25$ from a recording. What $\mathrm{SpO_2}$ does the formula give, and what should the student conclude?

SOLUTION. $\mathrm{SpO_2} = 110 - 25(0.25) = 103.75\,\%$. That is physically impossible: saturation cannot exceed 100 %. The value of R is outside the range where the calibration line is valid, which usually means the measurement is off: a weak or noisy red AC signal, motion, ambient light leaking in, or too much finger pressure. A commercial oximeter would clip the displayed value at 100 % or reject the reading. The student should check finger placement and recompute R over a clean stretch of data. (On the demo board, readings just above 100 % are common even with a clean recording, as the next example shows; $R = 0.25$ is well below that range.) ■

Worked Example 3.37 *A real recording from the board*

Figure 3.36 is a real recording from a demo board. For the shaded beat the analysis of Lab 3.2 finds AC = 518 and DC = 108 894 counts in the red channel, and AC = 1538 and DC = 120 447 counts in the infrared. (a) Find AC/DC for each channel, R, and SpO_2. (b) Convert the DC levels to photocurrent (62.5 pA per count). (c) Is the result reasonable?

Solution. (a) For the red channel, AC/DC is $518/108\,894 = 0.476\,\%$; for the infrared, $1538/120\,447 = 1.277\,\%$. So $R = 0.476/1.277 = 0.373$ and $SpO_2 = 110 - 25(0.373) = 100.7\,\%$.

(b) $108\,894 \times 62.5\,\text{pA} = 6.81\,\mu\text{A}$ (red) and $120\,447 \times 62.5\,\text{pA} = 7.53\,\mu\text{A}$ (infrared).

(c) The recording is clean and the channels behave as they should (the infrared AC/DC is the larger, as for well-oxygenated blood), yet the reading is above 100 %. The fault is in the calibration, not the measurement: $110 - 25R$ is a generic line that was never fit to this sensor, and healthy fingers on this board give R between about 0.36 and 0.45, right at the 100 % end of the line. A commercial oximeter is calibrated for its own optics against blood samples. ■

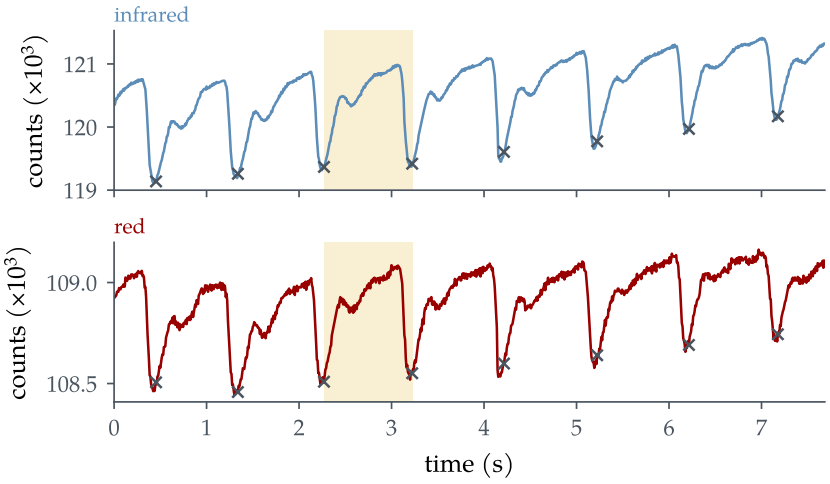


Figure 3.36: About 7.7 s of a real PPG recorded with the demo board (the file `ppg_backup.csv` of Lab 3.2, about 130 samples per second). Crosses mark the beats found by the lab's band-pass filter and peak finder; the heart rate is 62.5 bpm. The shaded beat is analyzed in the example "A real recording from the board."

Worked Example 3.38 *One bad beat*

The beat-by-beat analysis of a recording gives R = 0.38, 0.41, 0.37, 0.95, 0.40, 0.39, and 0.42; during the fourth beat the finger moved. Compare SpO_2 from the mean and from the median of these values.

SOLUTION. The mean is $3.32/7 = 0.474$, giving $110 - 25(0.474) = 98.1\,\%$. Sorted, the values are 0.37, 0.38, 0.39, 0.40, 0.41, 0.42, 0.95, so the median is the fourth, 0.40, giving 100.0 %. Without the bad beat the mean would be $2.37/6 = 0.395$ and SpO_2 100.1 %. One motion artifact moved the mean by almost two percentage points but left the median alone, which is why the lab reports the median. ■

The analog front end on the board

Section 3.5 built an oximeter readout from separate parts. The demo board uses an integrated *analog front end* (AFE), the MAX86916 optical sensor, which places the LEDs, the photodiode, and all of the conditioning electronics in a package a few millimeters long. Its block diagram, like that of its close relative the MAX30102, is summarized in Fig. 3.37.

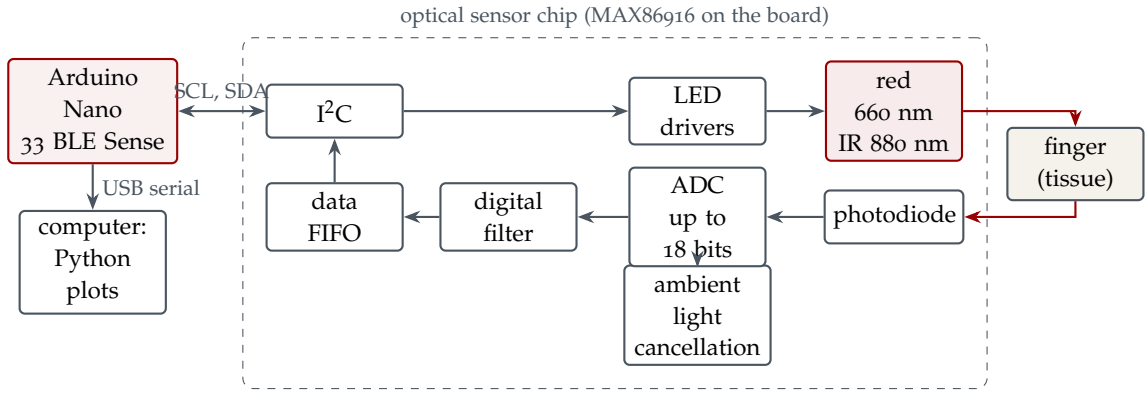


Figure 3.37: The analog front end of the board's oximeter. Inside the sensor chip, LED drivers pulse the red and IR LEDs in turn; the photodiode current is corrected for ambient light, digitized by a high-resolution ADC, filtered, and queued in a FIFO memory. The Arduino reads the FIFO over I^2C and streams the numbers to the computer.

WORKED EXAMPLE 3.39 *The analog front end*

What is an AFE, an analog front end?

SOLUTION. The analog front end is the circuitry between a sensing element and the digital world: everything that prepares the raw analog signal for conversion. For an optical sensor it includes the LED drivers that set the light level, the photodiode readout (a TIA), amplifiers and filters, offset or ambient-light cancellation, and the ADC. In the five-stage chain of Fig. 3.2 it covers the first three stages. The MAX86916 on the board is an integrated AFE: one chip does all of this and reports digital samples over I^2C. ■

A few numbers from the data sheet and from the board's driver code shape what you see in the lab. The LEDs sit near 660 nm and 880 nm. The chip's sample rate is set by a 3-bit register to one of 50,

100, 200, 400, 800, 1000, 1600, or 3200 samples per second. Longer LED pulses give more ADC bits, from 15 bits at 69 µs to 18 bits at 411 µs. The board's driver selects 400 samples per second, 411 µs pulses, and a full-scale range of 16 384 nA, so one count of the 18-bit result is $16\,384\,\text{nA}/2^{18} = 62.5\,\text{pA}$ of photocurrent, and the largest possible reading is $2^{18} - 1 = 262\,143$ counts. A raw reading of 120 000 counts means 7.5 µA. The Arduino fetches the newest red and infrared values from the chip and sends each pair over USB with a time stamp, so the notebook receives fewer pairs than the chip makes: about 130 per second (120 to 135 in practice). Always compute the sampling rate from the time column, never from the chip setting.

3.9 *Heart Rate from the PPG*

HEART RATE IS THE EASIER of the two vital signs to pull from a PPG, because it only needs the timing of the beats, not their exact size. It takes four steps (Fig. 3.38).

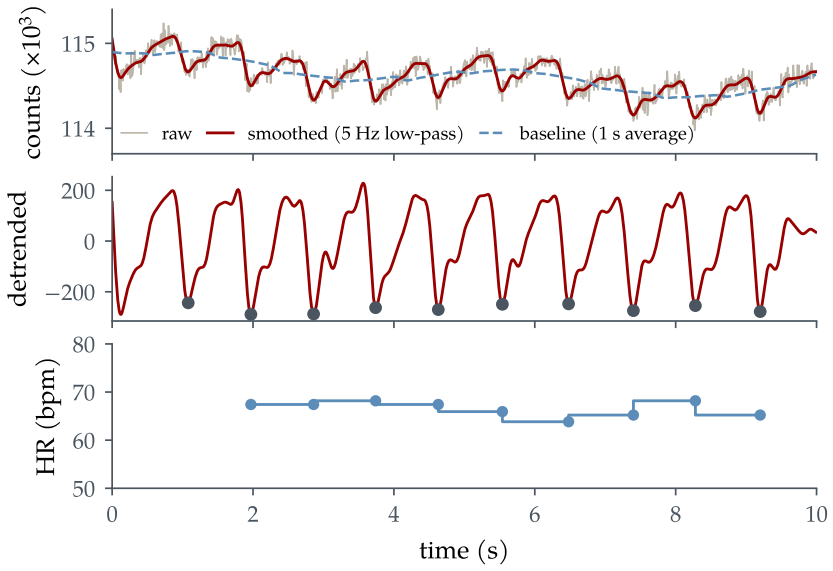


Figure 3.38: Heart-rate pipeline on a simulated 10 s red PPG at 100 samples per second. Top: raw data (gray), the smoothed signal after a 5 Hz low-pass filter, and the slow baseline (1 s moving average). Middle: smoothed signal minus baseline, with each beat's sharp drop marked. Bottom: instantaneous heart rate 60/IBI for each pair of beats.

Smoothing: a low-pass filter with a cutoff of a few hertz removes electronic noise and small movements, the flat noise floor in the spectrum. *Drift correction:* subtracting a slow baseline (a one-second moving average, or a high-pass filter below 0.5 Hz) removes the wander caused by breathing, finger pressure, and temperature. *Peak detection:* each beat now produces one sharp drop, which a peak finder (`find_peaks` in `scipy.signal`) locates; a minimum spacing of 0.4 s (150 bpm) and a minimum prominence keep it from counting the small

dicrotic bump after each beat. *Heart rate:* the time between beats is the *inter-beat interval* (IBI), and

$$\mathrm{HR} = \frac{60}{\mathrm{IBI}}, \qquad \text{IBI in seconds.} \tag{3.34}$$

Per-beat values show *heart-rate variability*; averaging the IBIs gives one resting value. Lab 3.2 does the first two steps at once with a band-pass filter from 0.5 to 4 Hz: above breathing and drift, and high enough to keep heart rates up to 240 bpm and the second harmonic of a resting pulse. The filter is run forward and then backward over the recording (`sosfiltfilt`), so the beats do not shift in time.

WORKED EXAMPLE 3.40 *Heart rate from peak positions*

After smoothing and drift correction, a red PPG recorded at 100 samples per second has beats at sample indices 112, 203, 294, 384, and 476. Find the instantaneous and average heart rate.

SOLUTION. The index differences are 91, 91, 90, and 92 samples. At 100 samples per second the IBIs are 0.91, 0.91, 0.90, and 0.92 s. Equation (3.34) gives instantaneous rates of 65.9, 65.9, 66.7, and 65.2 bpm. The mean IBI is 0.91 s, so the average heart rate is $60/0.91 = 65.9$ bpm. Note the denominator: average the IBIs first, then invert. (Averaging the four rates gives nearly the same answer here, but not in general.) ■

Heart rate from the spectrum

A second method uses the FFT. A steady heartbeat is a repeating signal, so its spectrum has a strong line at the heart rate frequency, plus weaker harmonics at two and three times that frequency (Fig. 3.39). Find the tallest line in the band where heart rates live, about 0.7 to 3.5 Hz (42 to 210 bpm), and multiply by 60. The Lab 3.2 notebook also rebuilds the signal from that single component with the inverse FFT; the result is a pure sine that ticks once per beat. Keeping four components brings back more of the waveform's shape, and keeping all of them returns the original signal.

WORKED EXAMPLE 3.41 *Heart rate from the FFT*

A 10 s PPG recording has its largest spectral line in the heart-rate band at 1.1 Hz. (a) What is the heart rate? (b) How precisely can a 10 s record pin down the heart rate? (c) Lab 3.2 records for 25 s at about 130 samples per second. How many samples is that, and what is the bin spacing?

SOLUTION. (a) 1.1 beats/s $\times$ 60 s/min $=$ 66 bpm, consistent with the peak-detection result. (b) By Eq. (3.26), bins are $1/10\,\mathrm{s} = 0.1$ Hz

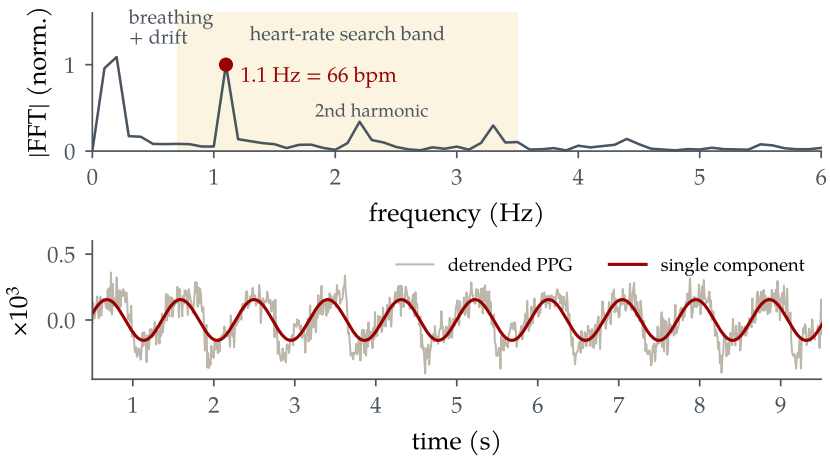


Figure 3.39: Top: spectrum of the 10 s PPG of Fig. 3.38. Breathing and drift sit below 0.5 Hz; the heartbeat is the tallest line in the search band, at 1.1 Hz; its second harmonic appears near 2.2 Hz. Bottom: the signal rebuilt from that single component (cardinal) beats in step with the PPG.

apart, which is $0.1 \times 60 = 6$ bpm. A heart rate between bins is reported at the nearest one, so the answer is only good to about ± 3 bpm. Doubling the record length halves the bin spacing. (c) $25 \times 130 = 3250$ samples. The bin spacing is $1/25\,\text{s} = 0.04\,\text{Hz}$, or 2.4 bpm, so the peak and FFT methods should agree within about 2.4 bpm. The number of samples does not matter for the spacing; only the length of the record does. ■

Peak detection times every beat but can be fooled by one motion artifact; the FFT averages over the window and ignores isolated glitches, but gives one rate per window with resolution $1/T_{\text{record}}$.

3.10 *Laboratory: Thermistor Temperature Sensor*

Lab 3.1 at a glance

Goal: turn the thermistor divider of the circuits block into a thermometer. Measure its noise, record a finger touch, convert volts to degrees with the beta model, fit its time constants, and compare its sensitivity and resolution with Sections 3.3 and 3.4.
Hardware: demo board (circuits block, voltage-divider header), a "10K 3950" NTC thermistor (10 kΩ at 25 °C, $\beta = 3950$ K), a 10 kΩ resistor (brown, black, orange), USB data cable, laptop.
Notebook: the folder `lab3.1-thermistor` of the lab kit holds the notebook `lab3.1-thermistor.ipynb` and an old recording, `temp_data_with_finger.csv`, that you will diagnose.
Before the lab: read Section 3.3 and compute by hand (a) the A1 voltage at a room temperature of 22 °C (answer: about 1.54 V)

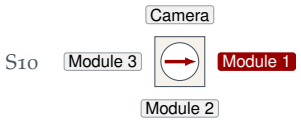
and (b) the temperature for $V_{out} = 1.805\,V$ (answer: 29.3 °C). To try the notebook at home, set `SIMULATE = True` in step A1.
Time: about two hours.
You hand in: `lab3.1_submission.json` on the course portal; keep your notebook with your answers.

Key takeaways

- An NTC thermistor's resistance falls about 4.5 % per degree near room temperature; the beta model turns R_T into a temperature.
- In the divider, A1 reads 1.65 V at 25 °C, half the supply, and rises about 37 mV per degree as the bead warms.
- One step of the 10-bit ADC is 3.22 mV, about 0.09 °C. The noise at rest is about as large: 3 mV RMS, 1 LSB.
- A finger raises A1 by about 0.3 V, 100 times the noise: an SNR of about 40 dB, so one reading shows that a finger is there.
- The bead responds like an RC circuit: 63 % of a change after one time constant, a few seconds to warm, 20 to 60 s to cool in air.

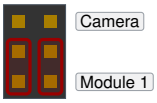
Set up the board for Lab 3.1

Selector. Turn S10 to Module 1 (circuits block, which also powers the divider header):



Jumpers. Put the caps of J7 and J8 on the Module 1 end (pins 2 and 3), so A1 reads the voltage divider (this is their usual position):

J7 and J8 caps on the Module 1 end

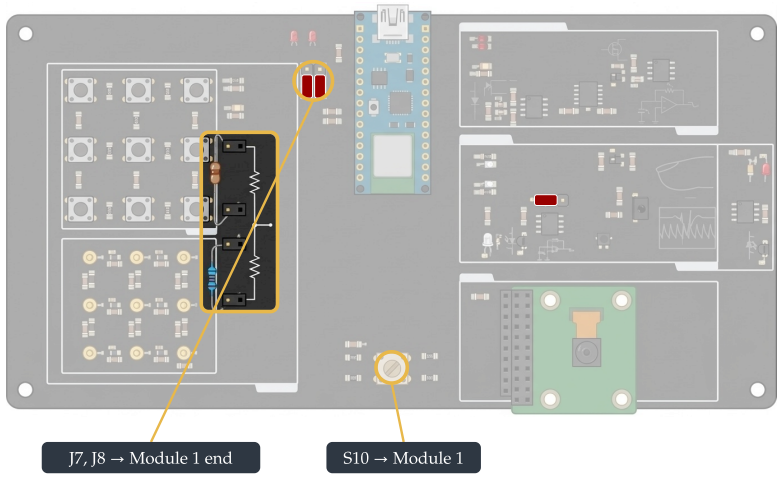


pin 1: camera
pin 2: Arduino A0 (J7), A1 (J8)
pin 3: touch panel (J7), divider (J8)

Parts. Plug the thermistor (the small glass bead on two thin leads) into J1–J2, the top position (R1) from 3.3 V to the midpoint. Plug the 10 kΩ resistor into J3–J4, the bottom position (R2) from the midpoint to ground (Fig. 3.10). Neither part has a polarity.

Check. The first notebook cell confirms that the selector is on `Module 1` (circuits block) and prints the A1 voltage: about 1.46 to 1.65 V at a room temperature of 20 to 25 °C.

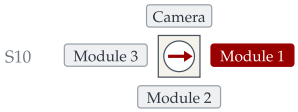
After the lab, leave the caps of J7 and J8 on the Module 1 end and return the thermistor and the resistor to your kit. The header, J7 and J8, and



LAB 3.1: SET UP THE BOARD

Uses: divider header J1 to J4

Selector S10: turn it to **Module 1**



Jumpers

J7 and J8 → both caps on the **Module 1** end



pin 1: camera
pin 2: A0 (J7), A1 (J8)
pin 3: touch panel (J7), divider (J8)

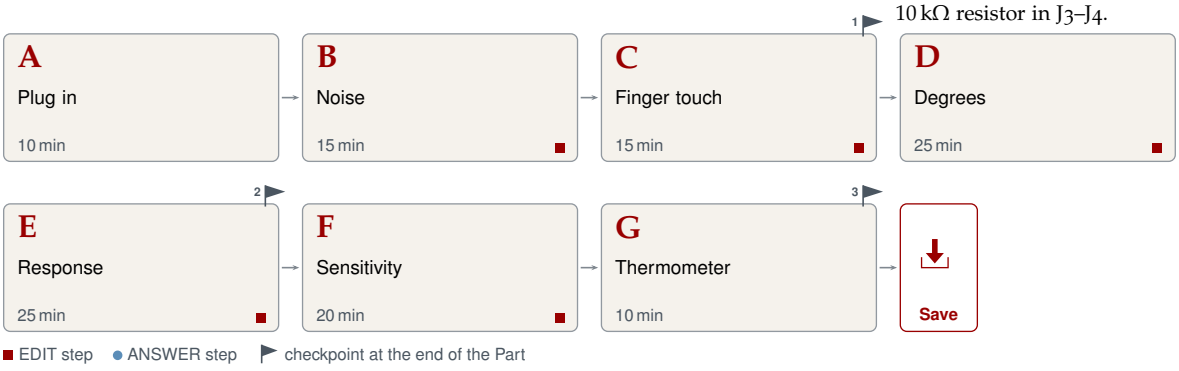
Plug in and place

10 kΩ thermistor (glass bead) in J1–J2 (top, R1, from 3.3 V).

10 kΩ resistor in J3–J4 (bottom, R2, to ground).

Figure 3.40: The board for Lab 3.1. Only the divider header J1 to J4 (highlighted) is used. S10 is on Module 1 and the caps of J7 and J8 are on the Module 1 end; the thermistor goes in J1–J2 and the 10 kΩ resistor in J3–J4.

the selector are labeled in Fig. 0.16, and Fig. 3.40 shows them with the settings of this lab.



Objectives

- Measure the RMS noise of a sensor reading and compute its signal-to-noise ratio.
- Convert the divider voltage into the thermistor resistance and then into temperature with the beta model.
- Fit first-order time constants to the warm-up and cool-down of the bead.
- Compare the measured sensitivity with the formula and find the temperature resolution set by the ADC and by the noise.

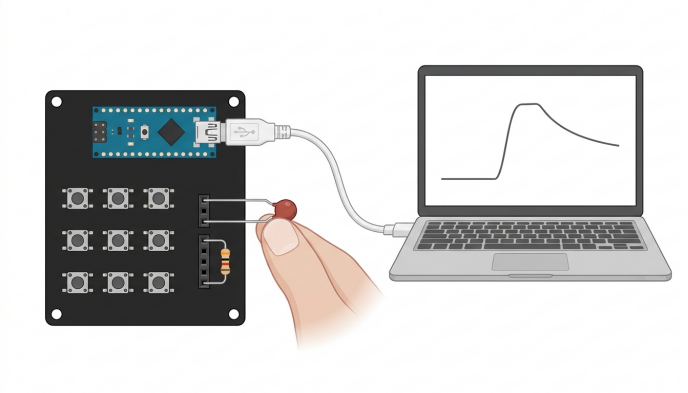


Figure 3.41: Lab 3.1 on the bench: the thermistor bead in the top position of the divider header, pinched between thumb and finger, with the 10 kΩ resistor below it. The laptop shows the temperature rising during the pinch and falling slowly after it.

Background

Everything you need is in Section 3.3. The divider of Fig. 3.9 gives $V_{\text{out}} = 3.3\,\text{V} \times R/(R + R_T)$, inverted by Eq. (3.9) to find R_T, and the beta model, Eq. (3.6), turns R_T into a temperature. The noise and SNR definitions are Eqs. (3.3) and (3.4); the sensitivity is Eq. (3.10); the response follows Eq. (3.14). The worked examples "RMS noise and SNR of the thermistor reading," "From ADC code to temperature," "Sensitivity and resolution at room temperature," "Self-heating in the board's divider," and "Time constant from a finger touch" are this lab done on paper. The simulated board (`SIMULATE = True`) has a room at 22 °C and a finger that warms the bead toward 31.5 °C.

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **B1**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT needs your code in the cell first, on the lines marked ✎ (Section 0.11 explains the tags).

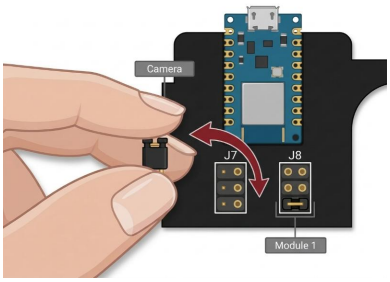
Part A: Plug in the thermistor and connect

GOAL Build the thermistor divider and read about 1.5 V on A1.

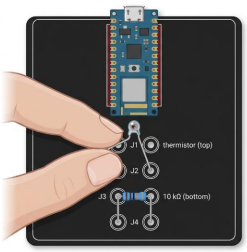
about 10 min

A1 DO Wire the divider and connect

1. Turn the selector S10 to Module 1 (circuits block).
2. Put both caps of J7 and J8 (the two small headers just left of the Arduino) on the Module 1 end, pins 2 and 3, so A1 reads the divider header.
3. The divider header is the column of four single sockets to the right of the button grid: J1 is 3.3 V, J2 and J3 are the midpoint (to A1),



A1 J7 and J8 are single rows of three pins: put each cap on the Module 1 end, over pins 2 and 3. (The picture draws the headers two pins wide.)



A1 Thermistor in J1–J2 (top), 10 kΩ resistor in J3–J4 (bottom). (The picture draws the header as two columns; on the board it is one column of four sockets.)

J4 is ground (Fig. 3.10). Plug the **thermistor** into **J1–J2** (top, R1, from 3.3 V) and the **10 kΩ resistor** (brown, black, orange) into **J3–J4** (bottom, R2, to ground).

4. Plug the Nano into your laptop with a USB *data* cable, click the first code cell (A1), and press Shift+Enter. The cell connects, uploads the course firmware if the board needs it, checks the selector, and prints A1.

With the thermistor R_T on top, $V_{\text{out}} = 3.3\,\text{V} \times R/(R + R_T)$, so at room temperature (20 to 25 °C, R_T = 12.5 to 10 kΩ) A1 reads about 1.46 to 1.65 V. Outside 1.0 to 2.3 V the cell warns you to check the jumpers and the parts. No board today? Set `SIMULATE = True` in this cell.

YOU SHOULD SEE two lines that begin with `[ok]` (S10 on `Module 1`), then `A1 =` about 1.46 to 1.65 V: 1.54 V at 22 °C, 1.65 V at 25 °C.

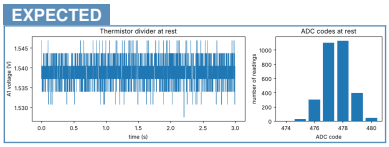
Part B: Noise at rest

GOAL Measure how noisy the sensor is when nothing happens.

about 20 min

B1 ▸ RUN Record 3 s at rest

Keep your hands away from the bead (a warm hand nearby or your breath also changes its temperature) and run B1. It records 3 s of A1 at one reading per millisecond (3000 readings) with `board.read`, saves them as `thermistor_rest.csv`, and plots the trace and a histogram of the ADC codes.



B1 A flat band a few ADC codes high (one code is 3.22 mV) and, at the right, a histogram of 3 to 6 bars side by side.

YOU SHOULD SEE a flat trace whose readings jump between a few neighboring ADC codes, like the left panel of Fig. 3.5: one code is 3.3 V/1024 = 3.22 mV (one LSB), and the histogram uses 3 to 6 codes.

B2 ✎ EDIT Mean and RMS noise

B2 · ✎ EDIT in the notebook

```
1 v_rest = ...    # ✎ mean voltage at rest (V)
2 x_rms = ...     # ✎ RMS noise (V) of the deviations
3 lsb = x_rms / hw.ADC_V_PER_COUNT
4 print(f"mean {v_rest:.4f} V, RMS noise {1000 * x_rms:.2f} mV "
5       f"= {lsb:.2f} LSB")
6 print(f"(np.std gives {1000 * rest.volts.std(ddof=0):.2f} mV)")
```

WHAT TO WRITE `v_rest`, the mean voltage at rest (line 1), and `x_rms`, the RMS noise from the definition, Eq. (3.3) (line 2): the root of the mean of the squared deviations x_i of the readings from their mean. The cell expresses it in LSB and compares it with `np.std`.

HINT the readings are `rest.volts`; the deviations are `rest.volts - v_rest`. Square them, average, take the square root.

DONE WHEN the mean is 1.46 to 1.65 V and the RMS noise about 3 mV, about 1 LSB, the same as `np.std`.

B3 ▸ RUN A recording that went wrong

The file `temp_data_with_finger.csv` comes from an earlier year of this lab: 10 000 raw ADC codes of A1, about 1 ms apart, recorded with the first board sketch while a student touched the thermistor. B3 plots its first 200 readings next to yours and prints a few numbers. You explain what went wrong in Question 6.

YOU SHOULD SEE `old file: 10000 readings, mean 0.075 V, 54% of them exactly 0, largest 0.348 V`, and a trace that looks nothing like yours.

Part C: Record a finger touch

GOAL Record a finger touch and compute the signal-to-noise ratio.

about 15 min

C1 ◆ DO Record a finger touch (60 s)

C1 records A1 for 60 s, one reading every 50 ms (1200 readings), and plots it live with `board.live`. Follow the instructions in the plot title:

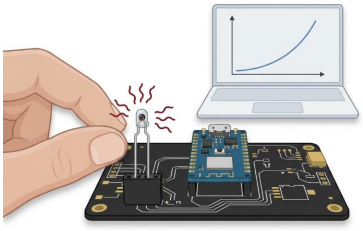
1. Run C1, hands off for the first 10 s.
2. Pinch the thermistor bead between your thumb and finger for 25 s (the bead only, not the resistor).
3. Let go and keep your hands away until the end.

The data are saved as `thermistor_finger.csv`.

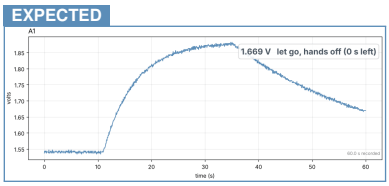
YOU SHOULD SEE A1 flat for 10 s, rising along a curve that slows down while you hold the bead, and falling slowly after you let go at 35 s (Fig. 3.13); then a line like `1201 readings over 60 s`. Your levels depend on the room and your finger.

C2 ▸ RUN Check the recording

C2 prints the voltage before the touch, the highest voltage of a 1 s moving average, and the rise between them, which is the *signal* of this sensor, and plots the recording. If it reports that the recording does not look like a finger touch, fix the wiring and run **C1** again. Only if the sensor is broken and cannot be replaced, set `USE_BACKUP = True` in C2 and run it again: it then uses a *simulated* 60 s recording instead (say so in your answers; the portal gives no credit for it).



C1 Pinch the bead itself between thumb and finger; do not touch the resistor.



C1 Flat for 10 s, a rise of 0.25 to 0.4 V that slows down while you pinch, and a slower fall after you let go.

YOU SHOULD SEE before the touch about 1.46 to 1.65 V (as in **A1**), and a rise of 0.25 to 0.4 V.

C3 EDIT Signal-to-noise ratio

C3 · EDIT in the notebook

```
1 snr = ...       # ✎ signal (rise) divided by the RMS noise
2 snr_db = ...    # ✎ the same in decibels
3 print(f"signal {1e3 * rise:.0f} mV, noise {1e3 * x_rms:.2f} mV")
4 print(f"SNR {snr:.0f} = {snr_db:.1f} dB")
5 if snr_db > 30:
6     print(f"✓ Checkpoint 1 passed: a clear finger signal, "
7           f"SNR {snr_db:.1f} dB (above 30 dB)")
```

WHAT TO WRITE `snr`, the signal (`rise` from **C2**) divided by the RMS noise `x_rms` of **B2** (line 1), and `snr_db`, the same in decibels, Eq. (3.4) (line 2).

HINT $\mathrm{SNR}_{\mathrm{dB}} = 20\log_{10}(\text{signal}/x_{\mathrm{rms}})$. The worked example: 250 mV over 3.87 mV is 36.2 dB.

DONE WHEN the cell prints an SNR of about 36 to 42 dB and the line `✓Checkpoint 1 passed` (above 30 dB).

EXPECTED

```
signal 336 mV, noise 2.98 mV
SNR 113 = 41.0 dB
✓ Checkpoint 1 passed: a clear finger
signal, SNR 41.0 dB (above 30 dB)
```

C3 The SNR as a ratio and in decibels, then the checkpoint line. Your numbers will be your own.

✓CHECK Checkpoint 1 Your RMS noise is a few millivolts (**B2**), the finger-touch plot shows a clear rise (**C2**), and the SNR is above about 30 dB (**C3** prints ✓). Your sensor sees a finger more than 30 times above its own noise.

Part D: From volts to degrees

GOAL Turn a voltage into a temperature with the divider and the beta model. about 25 min

Two steps turn a reading into a temperature (Section 3.3). First the divider: with the thermistor on top and $R = 10\,\mathrm{k\Omega}$ at the bottom, $R_T = R\,(3.3\,\mathrm{V}/V_{\mathrm{out}} - 1)$, Eq. (3.9). Then the beta model: $1/T = 1/T_0 + \ln(R_T/R_0)/\beta$, Eq. (3.6), with T in kelvin. The "10K 3950" part has $R_0 = 10\,\mathrm{k\Omega}$ at $T_0 = 25\,^{\circ}\mathrm{C} = 298.15\,\mathrm{K}$ and $\beta = 3950\,\mathrm{K}$.

Did you know? A thermistor changes its resistance by a few percent per degree. A platinum Pt100 sensor changes only about 0.39 % per degree, and an IC sensor such as the TMP36 gives 10 mV per degree. Your divider gives about 36.7 mV per degree at 25 °C.

D1 EDIT Write `volts_to_ohms(v)` and `ohms_to_celsius(r)`

D1 · EDIT in the notebook

```
1  R_FIXED = 10e3                        # ohms, bottom resistor
2  R0, T0, BETA = 10e3, 298.15, 3950.0   # "10K 3950" part
3
4  def volts_to_ohms(v):
5      """Thermistor resistance (ohms) from the A1 voltage."""
6      # ✎ YOUR CODE HERE: R_T (ohms) from the A1 voltage v
7
8  def ohms_to_celsius(r):
9      """Temperature (degrees Celsius) from the beta model."""
10     # ✎ YOUR CODE HERE: temperature (C), beta model
```

```
11
12 v560 = hw.adc_to_volts(560)
13 r560 = volts_to_ohms(v560)
14 print(f"code 560: {v560:.3f} V, {r560 / 1e3:.2f} kOhm, "
15       f"{ohms_to_celsius(r560):.1f} C")
```

WHAT TO WRITE the bodies of the two functions: `volts_to_ohms` returns the thermistor resistance in ohms from the A1 voltage (line 6), and `ohms_to_celsius` the temperature in degrees Celsius from the beta model (line 10). Lines 1 and 2 hold the constants; `hw.VREF` is 3.3 V.

HINT the divider solved for R_T, and the beta model solved for $1/T$. Subtract 273.15 to get Celsius from kelvin. Both must work on whole columns, so use `np.log`, not `math.log`.

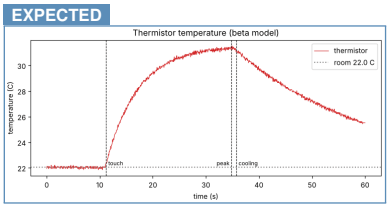
DONE WHEN the check line prints `code 560:  1.805 V, 8.29 kOhm, 29.3 C`.

D2 ▸ RUN Convert the recording to degrees

D2 converts your recording with your two functions, saves the columns `R_ohm` and `T_C` to `thermistor_finger.csv`, and marks the room temperature (the mean of the first 5 s), the touch (the first time the 1 s average rises 0.3 °C above room), the peak, and the start of the cool-down (the first time after the peak that the average falls 0.2 °C below it). It plots temperature against time.

YOU SHOULD SEE a room temperature of 20 to 25 °C and a peak of 30 to 34 °C, with the touch near 10 s, the peak near 35 s, and the cooling right after it.

EXPECTED



D2 The dotted line is the room temperature. The three dashed lines mark the touch, the peak, and the start of the cooling: each must sit where your curve turns.

Part E: Response time

GOAL Fit time constants to the warm-up and the cool-down.

about 25 min

The bead has to take up heat from your finger and later give it to the air. Like the RC circuit of Chapter 1, with a thermal resistance and a heat capacity in place of R and C, it approaches its final temperature exponentially, Eq. (3.14):

$$T(t) = T_{\text{final}} + (T_{\text{start}} - T_{\text{final}})\, e^{-t/\tau},$$

where t counts from the start of the segment. After one time constant τ, 63 % of the change is done; after 3τ, 95 %.

E1 ✎ EDIT Write `first_order` and fit both time constants

E1 · ✎ EDIT in the notebook

```
1 def first_order(t, T_final, T_start, tau):
2     """First-order response with time constant tau."""
3     # ✎ YOUR CODE HERE: T(t) from the formula above
4
5 tw = t[i_on:i_peak + 1] - t[i_on]   # warm-up, from the touch
```

```
Tw = T[i_on:i_peak + 1]
tc, Tc = t[i_cool:] - t[i_cool], T[i_cool:]   # cool-down

p_warm, _ = curve_fit(first_order, tw, Tw,
                      p0=[...])   # ✎ T_final, T_start, tau
p_cool, _ = curve_fit(first_order, tc, Tc,
                      p0=[...])   # ✎ cool-down guesses
tau_warm, tau_cool = p_warm[2], p_cool[2]
print(f"warm-up:   tau {tau_warm:.1f} s, end {p_warm[0]:.1f} C"
      f" (95 % after {3 * tau_warm:.0f} s)")
print(f"cool-down: tau {tau_cool:.1f} s, end {p_cool[0]:.1f} C"
      f" (95 % after {3 * tau_cool:.0f} s)")
```

WHAT TO WRITE the body of `first_order` (line 3), the formula above, and the starting guesses `p0 = [T_final, T_start, tau]` of the two fits (lines 10 and 12). `curve_fit` fits the warm-up (from the touch to the peak, lines 5 and 6) and the cool-down (from the start of the cooling to the end, line 7).

HINT take the guesses from the plot of **D2**: the warm-up goes from room temperature toward the peak in a few seconds, the cool-down from the peak back toward room temperature in about 20 s. Rough guesses are enough; `curve_fit` refines them.

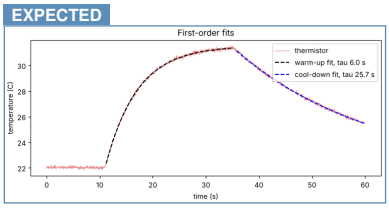
DONE WHEN both fits converge: a warm-up τ of a few seconds to about 15 s, a longer cool-down τ of 20 to 60 s, and a cool-down final temperature near room temperature (the simulator gives about 6 and 26 s).

E2 ▸ RUN Plot the fits

E2 draws both fits over the recording.

YOU SHOULD SEE both dashed fits on the data, the cool-down the slower one, as in Fig. 3.42, and the line `✓The cool-down (tau ...) is slower than the warm-up.`

EXPECTED



E2 Both dashed fits lie on the data. The legend gives the two time constants; the cool-down's is the longer one.

✓CHECK **Checkpoint 2** Both fits follow the data, and the cool-down time constant is longer than the warm-up one (**E2** prints ✓). You measured a thermal time constant: the RC idea of Chapter 1 with heat in place of charge.

Part F: Sensitivity and resolution

GOAL Find how many millivolts one degree gives, and the smallest change you can see. about 15 min

The sensitivity of the divider is the slope $S = dV_{\text{out}}/dT$ of Fig. 3.11. By the chain rule, Eq. (3.10),

$$S = \frac{V_S R}{(R_T + R)^2} \cdot \frac{\beta R_T}{T^2},$$

which Section 3.3 evaluates at 25 °C ($R_T = R$): 36.7 mV per °C. Your data give a second estimate, the chord slope $\Delta V/\Delta T$ between room temperature and the peak. Without a reference thermometer the temperatures come from the beta model, so this checks the circuit and the formula at your temperatures; it is not an independent calibration.

F1 EDIT Write `sensitivity(T_c)` and the chord slope `S_data`

F1 · EDIT in the notebook

```
1  def sensitivity(T_c):
2      """dV_out/dT (V per degree) at T_c (Celsius)."""
3      # ✎ YOUR CODE HERE: S from the formula (V/degree)
4
5  v_peak = v_smooth.iloc[i_peak]
6  S_data = ...   # ✎ V per degree, room to peak
7
8  s25 = 1000 * sensitivity(25)
9  s_room = 1000 * sensitivity(T_room)
10 print(f"formula at 25 C:          {s25:.1f} mV/C (book 36.7)")
11 print(f"formula at room, {T_room:.1f} C: {s_room:.1f} mV/C")
12 print(f"your data, room to peak: {1000 * S_data:.1f} mV/C")
13 lsb_c, noise_c = hw.ADC_V_PER_COUNT / S_data, x_rms / S_data
14 print(f"resolution: one LSB = {lsb_c:.3f} C, "
15       f"RMS noise = {noise_c:.3f} C")
```

WHAT TO WRITE the body of `sensitivity` (line 3), S from the formula above in volts per degree, and `S_data` (line 6), the chord slope of your recording: the change in voltage over the change in temperature from room to peak. The cell then gives the temperature resolution: the temperature change worth one LSB, and the one equal to the RMS noise.

HINT convert `T_c` to kelvin and find R_T from the beta model first (`R0`, `T0`, `BETA` of **D1**). For the chord use `v_peak` (line 5), `v_before` (**C2**), `T_peak`, and `T_room` (**D2**).

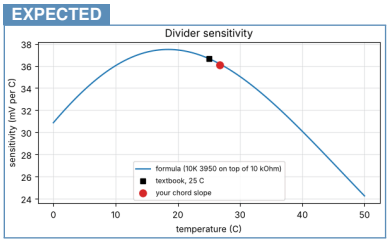
DONE WHEN the formula gives 36.7 mV/°C at 25 °C and 37.2 mV/°C at 22 °C, your chord slope is about 36 mV/°C, and one LSB is about 0.09 °C.

F2 ▸ RUN Plot the sensitivity

F2 plots the formula from 0 to 50 °C, the textbook point at 25 °C, and your chord slope at the middle of your temperature range.

YOU SHOULD SEE a curve that is largest, about 37.5 mV/°C, near 18 °C and falls to about 24 mV/°C at 50 °C, with your point close to it (Question 5).

EXPECTED



F2 The curve is the formula. The square is the textbook point, 36.7 mV/°C at 25 °C; your round point should sit on the curve or just beside it.

Part G: Live thermometer

GOAL Turn the board into a live thermometer. about 10 min

G1 ◆DO Live thermometer

Your two functions of **D1** turn the board into a thermometer. Run G1: it runs for 10 s. Pinch the thermistor bead while it runs, as in **C1**; the plot shows the voltage, and the title and a thermometer bar show the temperature.

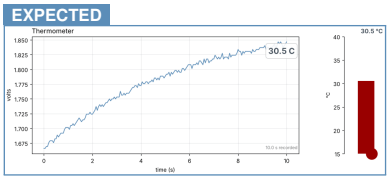
YOU SHOULD SEE a plausible room temperature in the title at first, climbing while you hold the bead.

✓CHECK **Checkpoint 3** The live thermometer shows a plausible temperature (**G1**), and your chord slope is close to the formula (**F1**). You built a working electronic thermometer from a bead, a resistor, and a formula.

Optional, calibration. If your bench has a reference thermometer, note its reading and your room temperature from **D2**. The difference is a one-point calibration offset (Section 3.3).

TRY IT ON THE BOARD In a new cell, run `board.monitor("temperature", seconds=20)`. The firmware converts A1 to degrees with the same divider and beta equations and prints a reading twice a second; pinch the bead and compare with your **G1** thermometer. The cell sends `stop` when the time is up.

EXPECTED



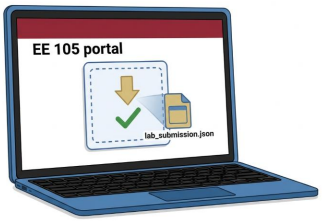
G1 The voltage climbs while you pinch; the box and the bar show the temperature. Yours starts at room temperature, near 1.5 V and 20 to 25 °C; the simulated bead was still warm.

Save and upload

Save ▸RUN Write `lab3.1_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab3.1_submission.json` in the lab folder: your numbers and the recordings behind them, from which the portal recomputes the room and peak temperatures. Log in at `ee105.org/portal`, open Lab 3.1, and upload that file. `thermistor_finger.csv` stays in the lab folder for a report.

YOU SHOULD SEE `[ok] Saved lab3.1_submission.json`, and then the portal shows your grade. A run on the simulated board, or with `USE_BACKUP = True`, earns no credit.



Save Upload `lab3.1_submission.json` on the course portal under Lab 3.1.

Finish the lab

You built a thermometer from a bead, a resistor, and a formula, and measured its noise, signal-to-noise ratio, time constants, sensitivity, and resolution. The lab is done when every box is ticked.

- ☐ **B2** EDIT `v_rest`, `x_rms`
- ☐ **C1** DO your own 60 s recording, not the backup
- ☐ **C3** EDIT `snr`, `snr_db`
- ☐ **D1** EDIT `volts_to_ohms`, `ohms_to_celsius`
- ☐ **E1** EDIT `first_order` and the two `p0`
- ☐ **F1** EDIT `sensitivity`, `S_data`
- ☐ **G1** DO the live thermometer, with a pinch
- ☐ CHECK Checkpoint 1: noise, touch, SNR above 30 dB
- ☐ CHECK Checkpoint 2: both fits; the cool-down slower
- ☐ CHECK Checkpoint 3: live thermometer; chord slope
- ☐ Questions 1 to 7 answered in the notebook
- ☐ **Save** SAVE `lab3.1_submission.json` uploaded on the portal

Typical results

Quantity	Typical result on a real board
A1 at rest	1.46 to 1.65 V at 20 to 25 °C (1.54 V at 22 °C, 1.65 V at 25 °C)
RMS noise at rest (B2)	about 3 mV, about 1 LSB; readings spread over 3 to 6 neighboring codes
Rise during the pinch (C2, C3)	0.25 to 0.4 V; SNR about 36 to 42 dB
Room and peak temperature (D2)	room 20 to 25 °C; peak 30 to 34 °C
Time constants (E1)	warm-up a few seconds to about 15 s; cool-down 20 to 60 s
Sensitivity (F1)	formula 36.7 mV/K at 25 °C, 37.2 mV/K at 22 °C; chord slope about 36 mV/K
Resolution	about 0.09 °C per LSB
Self-heating	about 0.27 mW at 25 °C, about 0.14 °C

Table 3.2: Expected results for Lab 3.1. Time constants depend on the bead, how firmly you pinch, and air movement in the room.

Table 3.2 lists typical numbers. The rest trace looks like the left panel of Fig. 3.5, and the touch like Fig. 3.13; Fig. 3.42 shows a whole recording with its fits. A real finger is not a perfect step: pressure changes during the pinch make the warm-up less exponential, and the bead keeps warming slowly. The fits still give a usable time constant. The peak stays below body temperature because fingertip skin is cooler than the body core and the bead loses heat through its leads and to the air. The cool-down fit sees only about one time constant, so its final value is less certain than the warm-up's. Without a reference thermometer the temperatures come from the beta model, so the sensitivity comparison checks the circuit and the formula, not the accuracy of the thermometer.

Troubleshooting

- *A1 near* 0 V*, or a pattern that repeats every* 16.7 ms *(*60 Hz*):* A1 is floating. Check that the selector is on Module 1, that the caps of J7 and J8 are on the Module 1 end, and that both parts sit firmly in the header.
- *A1 falls when you pinch:* the thermistor and the resistor are swapped; the thermistor belongs in J1–J2.
- *A1 steady near* 3.3 V*:* the bottom resistor is missing or not making contact, so nothing pulls the midpoint down. *Steady near* 0 V *with no hum:* the thermistor is missing or its lead is broken (measure it: about 10 kΩ at 25 °C, 11.4 kΩ at 22 °C). The leads are thin and break; use a spare.
- *No rise, or a small one:* hold the bead itself, firmly, for the whole 25 s. Cold fingers give a smaller rise; warm your hands first.
- *The check line of* **D1** *prints other numbers:* the thermistor must be on top, $R_T = R\,(3.3/V - 1)$, and the beta model works in kelvin.
- *The fit fails (***E1***):* usually you let go late or kept a hand near the bead during the cool-down. Record **C1** again and follow the title, then run C2 to E1 again.
- *No board:* set `SIMULATE = True` in **A1** to practice.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. What RMS noise did you measure (**B2**), in millivolts and in LSB? How many ADC codes does the reading use at rest (**B1**), and what does that say about where the noise comes from?

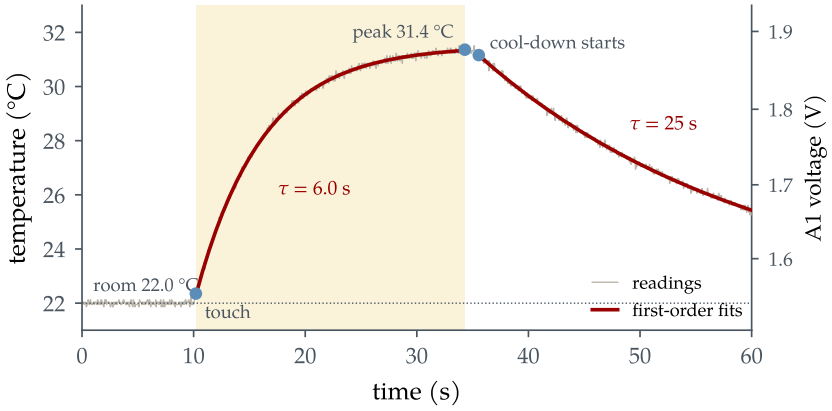


Figure 3.42: A typical Part C recording (simulated board) after Parts D and E: room 22.0 °C, the bead pinched from about 10 to 35 s (shaded from the detected touch to the peak), then let go. Look for a quick warm-up, here τ = 6.0 s, and a slower cool-down, here τ = 25 s; the red curves are the first-order fits. The right axis gives A1, which rises by about 0.34 V.

2. What SNR did you get (**C3**), in decibels? Compare it with the worked example (250 mV signal, 3.87 mV noise, 36.2 dB). Can one reading tell whether a finger is on the sensor?
3. What room and peak temperatures did you measure (**D2**)? Are they plausible? Why is the peak below body temperature?
4. Compare your two time constants (**E1**). Why does the bead cool more slowly than it warms? Is the fitted final temperature of the cool-down close to room temperature?
5. Compare your sensitivity (**F1**) with 36.7 mV/K. Why does the formula give a different value at your room temperature? What is the smallest temperature change your setup can detect, from the ADC step and from the noise?
6. What went wrong in the old recording `temp_data_with_finger.csv` (**B3**)? Give two pieces of evidence from the plot and the numbers.
7. Use Eqs. (3.11) and (3.12) with your room temperature and $\delta = 2\,\mathrm{mW/K}$ to estimate how much the bead warms itself. Does it matter for your measurement?

3.11 *Laboratory: Pulse Oximeter and PPG*

Lab 3.2 at a glance

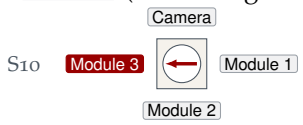
Goal: record red and infrared PPG signals from your fingertip with the demo board, compute your heart rate two ways, estimate your SpO_2, and see how finger pressure and other conditions change the signal.
Hardware: demo board (biosensing block with the MAX86916 optical sensor), USB data cable, laptop. Nothing is plugged in.
Notebook: `lab3.2-pulse-oximeter.ipynb` in the folder `lab3.2-pulse-oximeter` of the lab kit, with `ppg_backup.csv`, a real recording from a demo board (Fig. 3.36) for when your sensor will not cooperate.
Before the lab: read Sections 3.7 to 3.9, and write your code at home on the simulated board (`SIMULATE = True` in step A1). Come with warm hands.
Time: about two hours.
You hand in: `lab3.2_submission.json` on the course portal; keep your notebook with your answers.

Key takeaways

- A PPG is mostly DC: the pulsing part, AC/DC, is only 0.3 to 2 % of the signal in infrared and 0.13 to 0.8 % in red.
- Each heartbeat is a *dip*: more blood absorbs more light. A 0.5 to 4 Hz band-pass filter keeps the pulse and removes the drift.
- Two methods give one heart rate: 60 over the mean beat interval, and the tallest FFT line. They agree within $60/T$, 2.4 bpm for 25 s.
- SpO_2 comes from R, the red AC/DC over the infrared AC/DC. A healthy finger gives R near 0.4: about 100 % on the lab line $110 - 25R$.
- A number is only as good as its signal: with no finger, the code still prints a heart rate and an SpO_2, both computed from noise.

Set up the board for Lab 3.2

Selector. Turn S10 to Module 3 (biosensing block):



Jumper. Put the cap of J5, the small header lying on its side in the biosensing block next to the printed label P1-A-P2, on P1–A (pins 1 and 2), so A6 reads the photodiode of optical pair 1, the pair that belongs to this module:



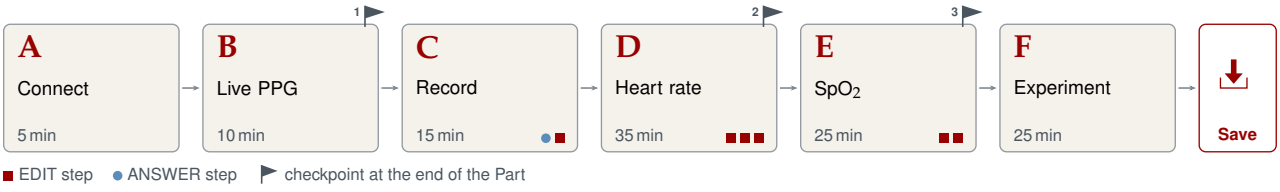
cap on P1–A: `A6` reads pair 1
cap on A–P2: `A6` reads RX (pair 2)

The optical sensor itself talks to the Arduino over I^2C, so J5 does not change the PPG data; it matters for the `light` command and for the LiFi lab of Chapter 5, which uses A–P2.

Parts. Nothing to plug in.

Check. The first notebook cell prints two lines that begin with `[ok]`, the second saying that S10 is on `Module 3` (biosensing block). If the reading does not fit, the cell prints a `[!!]` warning instead; check that S10 points at Module 3 and carry on.

Nothing needs to be restored afterwards. Fig. 3.43 shows the biosensing block, J5, and S10 on the board.



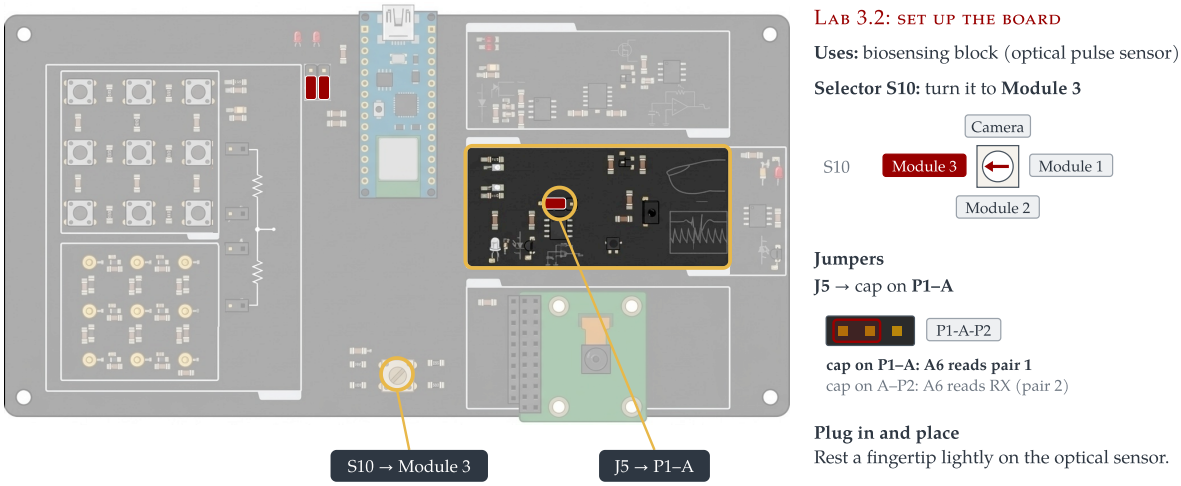


Figure 3.43: The board for Lab 3.2. Only the optical pulse sensor of the biosensing block (highlighted) is used. S10 is on Module 3 and the cap of J5 on P1–A; nothing is plugged in.

Objectives

- Explain how pulsing arterial blood produces the AC and DC parts of a PPG, and record red and infrared PPGs of good quality.
- Band-pass filter the PPG, detect the beats, and compute the heart rate from the inter-beat intervals and from the FFT.
- Compute the ratio of ratios R beat by beat and estimate SpO_2 with the calibration line $110 - 25R$.
- Test how finger pressure, a missing finger, and (if you wish) exercise change the numbers.

Not a medical device. The demo board is a teaching tool with a generic calibration. A strange SpO_2 value in this lab says more about the measurement than about your health.

Background

Sections 3.7 to 3.9 are the background: the Beer-Lambert law and the two wavelengths, the AC and DC parts of the PPG, the ratio of ratios, Eq. (3.30), the calibration line, Eq. (3.33), and heart rate from peaks, Eq. (3.34), and from the FFT, whose bins are $1/T$ apart (Eq. (3.26)). The board measures in reflectance mode (Fig. 3.32b), and the raw data are not flipped: each beat is a *dip*.

Figure 3.44 shows the PPG circuit on the board. The optical sensor (U5) runs its core logic from a 1.8 V supply made by a small linear regulator (U4, MCP1703T-1802E) from the 3.3 V rail, with 1 µF capacitors on its input and output. The LEDs draw their pulse current from 3.3 V through the VLED+ pin, decoupled by 0.1 µF (C5). The Arduino talks to the sensor over I^2C on the SCL and SDA lines. The open-drain interrupt output $\overline{\text{INT}}$ is pulled up to 3.3 V by a 10 kΩ resistor (R32) and

brought out to test point TP10. The chip itself pulses the LEDs, reads the photodiode, and digitizes the result (Fig. 3.37), so there are no analog signals to wire. One count is 62.5 pA of photocurrent, and the samples reach the notebook at about 130 per second (Section 3.8). The notebook does all the work: it uploads the course firmware, records the sensor, and analyzes the data, so you need neither the Arduino IDE nor its Serial Plotter.

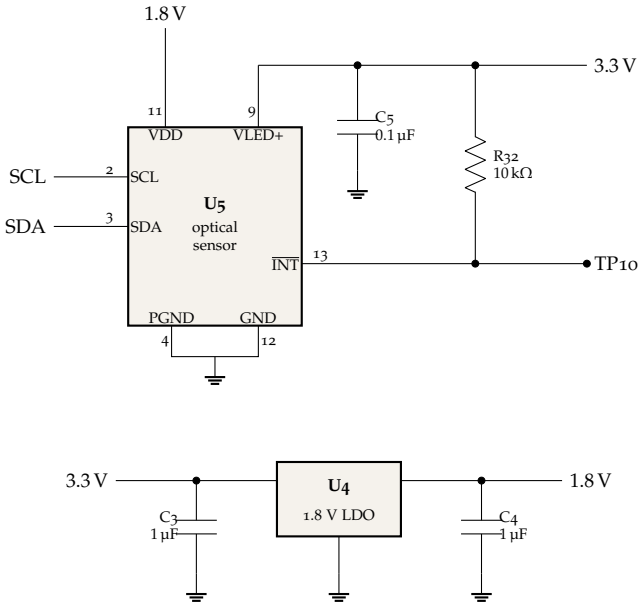


Figure 3.44: The PPG circuit of the biosensing block. Top: the optical sensor (U5) with its I²C lines, LED supply decoupling (C5), and the pull-up (R32) on its interrupt output. Bottom: the 1.8 V regulator (U4, MCP1703T-1802E) for the sensor core. The pin numbers are those of the board schematic, which draws the symbol of the closely related MAX30102; the part fitted is the MAX86916. Both answer at I²C address 0x57.

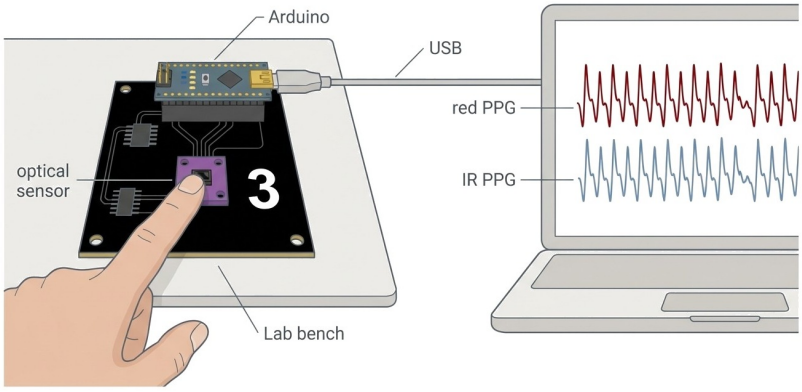


Figure 3.45: Lab setup: a fingertip resting lightly on the optical sensor of the biosensing block (printed Module 3 on the board), the Arduino connected by USB, and live red and infrared traces on the laptop.

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **B1**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT or ANSWER needs your input in the cell first (Section 0.11 explains the tags).

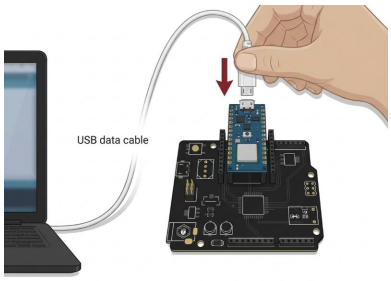
Part A: Connect

GOAL Connect the board on Module 3 with J5 on P1–A. about 5 min

A1 DO Connect the board

Set up the board as in the box above: S10 on Module 3, the cap of J5 on P1–A. Plug the Nano into your laptop with a USB *data* cable, click the first code cell (A1), and press Shift+Enter. The cell finds the board, uploads the course firmware if the board needs it, and checks the selector. No board today? Set `SIMULATE = True` in this cell to practice on a simulated board.

YOU SHOULD SEE two lines that begin with `[ok]`: the board is connected (firmware `ee105`), and S10 is on `Module 3` (biosensing block). A `[!!]` line about the selector is only a warning: if S10 points at Module 3, carry on.



A1 Plug the Nano into your laptop with a USB *data* cable.

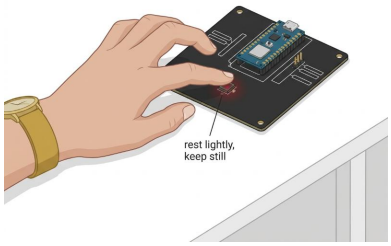
Part B: Live PPG

GOAL See your own pulse in red and infrared light. about 10 min

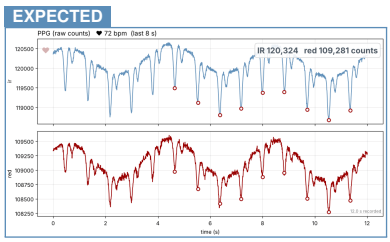
B1 DO Watch your pulse live (12 s)

The sensor pulses a red LED (about 660 nm) and an infrared LED (about 880 nm) and measures the light that scatters back out of your finger (Fig. 3.32).

1. Rest the pad of your index finger *lightly* on the small sensor window, right on top of it (Fig. 3.45). Do not press.
2. Rest your hand and forearm on the table and keep still and quiet: talking and moving show up in the signal.
3. Cup your other hand over the finger to block room light.
4. Run B1. It plots both channels live for 12 s and then prints the mean counts. Stop it early with the stop button; run it again if you moved.
5. Run B1 once more with no finger on the sensor. Write down the mean IR and red counts with and without a finger (Question 1).



B1 Fingertip *on* the sensor window, resting lightly; cup your other hand over it. (The picture draws the fingertip beside the window and leaves out the cupped hand.)



B1 Both traces dip once per heartbeat, by about 1 % of their level. Each detected beat gets a mark, and the title shows the heart rate.

YOU SHOULD SEE within a few seconds both traces settle and dip sharply once per heartbeat, on top of a slow wander from breathing (like Fig. 3.33). With a finger, about 100 000 to 120 000 counts in each channel; with no finger, a few thousand counts at most.

✓CHECK **Checkpoint 1** The live red and IR traces respond to your finger and show a clear pulse in both. You are watching your own heartbeat in light that scattered back out of your finger.

Part C: Record and look at the raw data

GOAL Record 25 s of PPG and find its sample rate and DC levels.

about 15 min

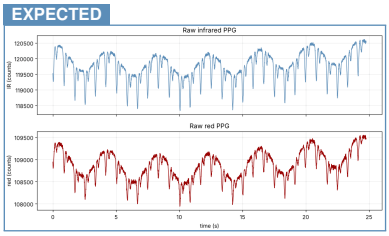
C1 ◆DO Record 25 s of PPG

Rest your fingertip on the sensor as in **B1**, cup your other hand over it, keep still, and run C1. It calls `board.ppg(seconds=25)`, which returns a table with the columns `t_s` (time from the board's clock, in seconds), `ir`, and `red` (raw counts), saves it as `ppg_rest.csv`, and plots both channels. Keep still for the whole 25 s; if you moved, run C1 again. Each heartbeat pushes a little more blood into the finger, the blood absorbs more light, and the signal *dips*: the large steady level is the DC part, the small dip and recovery the AC part (Fig. 3.33). To zoom in, add `axes[1].set_xlim(0, 8)` before `plt.tight_layout()`.

If your sensor will not give a clean pulse after a few tries, set `USE_BACKUP = True` in C1 and run it again: it then loads `ppg_backup.csv`, a real recording from a demo board. That file was saved without times; its `t_s` column assumes 130 samples per second. The portal grades only your own recording, so go back to `USE_BACKUP = False` once your sensor works.

YOU SHOULD SEE the line `Recording 25 s ...`, then 3250 samples over about 25 s and two traces with a dip at every heartbeat.

EXPECTED



C1 A large steady level, about 100 000 to 120 000 counts, with a small dip at every beat, on a slow wander from breathing. About 30 dips in 25 s at rest.

C2 ✎EDIT Sample rate and DC levels

C2 · ✎EDIT in the notebook

```
fs = ...          # ✎ samples per second, from t
dc_ir = ...       # ✎ DC level of the IR channel (counts)
dc_red = ...      # ✎ DC level of the red channel (counts)

PA_PER_COUNT = 62.5   # photocurrent per ADC count (pA)
print(f"sample rate fs = {fs:.1f} samples/s")
print(f"DC IR  = {dc_ir:9,.0f} counts = {dc_ir * PA_PER_COUNT /
    1e6:.2f} µA of photocurrent")
```

```
print(f"DC red = {dc_red:9,.0f} counts = {dc_red * PA_PER_COUNT / 1e6:.2f} µA of photocurrent")
```

WHAT TO WRITE `fs`, the sample rate in samples per second, from the time column `t`: $f_s = (N-1)/(t_{\text{last}} - t_{\text{first}})$. `dc_ir` and `dc_red`, the DC level (the mean) of each channel, in counts.

HINT N samples span $N-1$ intervals. `len(t)` is N, and `t[0]` and `t[-1]` are the first and last times.

DONE WHEN the cell prints a sample rate of 120 to 135 samples per second (one real board gave 124) and DC levels of about 100 000 to 120 000 counts in each channel, which it converts to photocurrent at 62.5 pA per count (about 6 to 7.5 µA).

C3 ANSWER AC swing by eye

C3 · ANSWER in the notebook

```
ac_ir_eye = ...      # ✎ IR swing of one beat (counts)
ac_red_eye = ...     # ✎ red swing of one beat (counts)
print(f"AC/DC by eye: IR  {100 * ac_ir_eye / dc_ir:.2f} %")
print(f"              red {100 * ac_red_eye / dc_red:.2f} %")
```

TYPE IN `ac_ir_eye` and `ac_red_eye`: the peak-to-peak swing of one beat in each channel, from the bottom of a dip to the top of the recovery. The cell prints AC/DC in percent.

UNIT ADC counts.

WHERE TO FIND IT the plot of C1. Zoom in on a few beats and read the swing on the vertical axis; ignore the slow wander from breathing.

YOU SHOULD SEE an IR swing of several hundred to about 2000 counts and a red swing a third to a half as large: AC/DC of 0.3 to 2 % for IR and 0.13 to 0.8 % for red, IR always the larger (Question 1). One real board at rest gave 1.8 and 0.8 %.

Part D: Filter, find the beats, heart rate two ways

GOAL Filter the PPG, find every beat, and get your heart rate two ways. about 35 min

The raw signal has three parts: the slow wander from breathing, finger pressure, and temperature (below about 0.5 Hz), the pulse (about 1 to 3 Hz, plus harmonics), and fast electronic noise. A band-pass filter keeps the pulse and removes the rest.

D1 EDIT Choose the cutoffs and write `bandpass(x, fs)`

D1 · EDIT in the notebook

```
F_LOW, F_HIGH = ..., ...    # ✎ your cutoffs in Hz

def bandpass(x, fs):
    """Remove the trend of x, then keep F_LOW to F_HIGH."""
    # ✎ 1. detrend   2. butter(...)   3. sosfiltfilt(...)
    return detrend(x)    # ✎ replace with your filter
```

```
7
8  y_ir = bandpass(ir, fs)
9  y_red = bandpass(red, fs)
```

WHAT TO WRITE the cutoffs `F_LOW` and `F_HIGH` in hertz (line 1). In `bandpass`, replace lines 5 and 6 by the three steps: remove the straight-line trend with `detrend(x)`; design a second-order Butterworth band-pass with `butter(2, [F_LOW, F_HIGH], btype="bandpass", fs=fs, output="sos")`; apply it to the trend-free signal with `sosfiltfilt(sos, ...)` and return the result. `sosfiltfilt` runs the filter forward and then backward, so the beats do not shift in time.

HINT 60 bpm is 1 Hz. The low cutoff sits above breathing and drift but below the slowest heart rate you expect; the high cutoff passes the pulse and its second harmonic. Question 2 asks you to justify your choice.

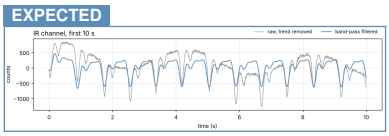
DONE WHEN the cell runs without an error and **D2** shows a filtered trace that keeps every dip.

D2 ▸ RUN Compare the signals before and after filtering

D2 plots the first 10 s of each channel before (gray, trend removed) and after the filter, and the spectrum of the IR channel with the pass band shaded.

YOU SHOULD SEE the filtered traces keep every dip and lose the slow wander. In the spectrum, the heart-rate line and its harmonics stand above a flat floor; the filter removes the large part below `F_LOW` and the floor above `F_HIGH` (Question 2).

EXPECTED



D2 The top panel of three. The filtered trace (blue) keeps every dip of the raw one (gray) and loses the slow wander: it swings about zero.

D3 EDIT Write `find_beats(y, fs)` and compute the heart rate

D3 · EDIT in the notebook

```
1  def find_beats(y, fs):
2      """Indices of the beats (the dips) of the filtered y."""
3      dips, _ = find_peaks(-y, distance=..., prominence=...)  # ✎
4      return dips
5
6  beats = find_beats(y_ir, fs)
7  ibi = ...          # ✎ inter-beat intervals (s), from t
8  hr_peaks = ...     # ✎ average heart rate in bpm
9  print(f"{len(beats)} beats found; heart rate from the peaks =
       {hr_peaks:.1f} bpm")
```

WHAT TO WRITE in `find_beats`, the settings of `find_peaks` (line 3). It returns the sample index of every beat, every sharp dip of the filtered signal; `find_peaks` finds maxima, so it searches `-y`. `distance`: no two beats closer than 0.4 s (150 bpm). `prominence`: about 0.3 times the typical swing of `y`, so that small bumps (the dicrotic notch, noise) do not count. Then `ibi`, the inter-beat intervals in seconds, and `hr_peaks`, the average heart rate HR $= 60/\overline{\mathrm{IBI}}$ in bpm (Eq. (3.34)).

HINT 0.4 s is `int(0.4 * fs)` samples. For the swing, take the 95th minus the 5th percentile of `y` (`np.percentile`) in a line of its own before `find_peaks`. `t[beats]` are the beat times, and `np.diff` gives the intervals. Average the intervals first, then take 60 over the mean.

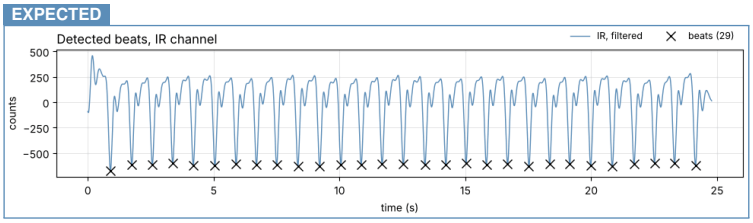
DONE WHEN the cell prints about one beat per second and a heart rate of 50 to 90 bpm at rest (77 on one real board).

D4 ▸ RUN Check the detected beats

D4 marks every detected beat with an × in both filtered channels and plots the instantaneous heart rate 60/IBI. Check the marks by eye: every beat, and nothing else, should be marked in *both* channels. If a beat is missed or a bump is counted, change `find_beats` in D3 (the prominence first) and run D3 and D4 again.

YOU SHOULD SEE one mark under every dip in both channels, as in Fig. 3.36, and an instantaneous heart rate that changes by a few bpm from beat to beat, often in step with your breathing (Question 3).

EXPECTED



D4 The top panel of three: one × under every dip, about one per second, and none anywhere else. The red panel below it must show the same marks.

D5 ✎ EDIT Write `fft_heart_rate(x, fs)`

D5 · ✎ EDIT in the notebook

```
1  def fft_heart_rate(x, fs):
2      """Heart rate (bpm): tallest FFT line in 0.7 to 3.5 Hz."""
3      f, X = spectrum(detrend(x), fs)   # Hz, FFT magnitude
4      return np.nan     # ✎ replace: heart rate in bpm
5
6  hr_fft = fft_heart_rate(ir, fs)
7  T = t[-1] - t[0]
8  print(f"heart rate from the peaks: {hr_peaks:.1f} bpm")
9  print(f"heart rate from the FFT:   {hr_fft:.1f} bpm   (bin
       spacing 1/T = {1 / T:.3f} Hz = {60 / T:.1f} bpm)")
10 if abs(hr_peaks - hr_fft) <= 60 / T:
11     print("✓ The two heart rates agree within the bin spacing. If
           every beat is marked in both channels, "
12           "that is Checkpoint 2.")
```

WHAT TO WRITE the return line (line 4): 60 times the frequency of the tallest FFT line between 0.7 and 3.5 Hz (42 to 210 bpm). Line 3 already computes `f`, the frequencies in hertz, and `X`, the FFT magnitude of the trend-free signal.

HINT select the band with `(f >= 0.7) & (f <= 3.5)`; `np.argmax` gives the index of the largest value of `X` in it.

DONE WHEN the cell prints both heart rates and the line `✓ The two heart rates agree within the bin spacing`. The bins of a 25 s record are $1/T = 0.04$ Hz, or 2.4 bpm, apart (Eq. (3.26)).

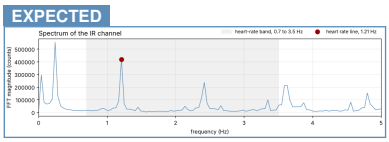
EXPECTED

```
heart rate from the peaks: 72.3 bpm
heart rate from the FFT:   72.6 bpm
(bin spacing 1/T = 0.040 Hz = 2.4 bpm)
✓ The two heart rates agree within the
bin spacing. If every beat is marked in
both channels, that is Checkpoint 2.
```

D5 Two heart rates that differ by less than the bin spacing, then the line with the check mark. Your numbers will be your own.

D6 ▸ RUN Rebuild the signal from the heart-rate line

D6 marks your heart-rate line in the spectrum and rebuilds the signal from that single FFT component with the inverse FFT: a pure sine that ticks once per beat (Fig. 3.39).



D6 The top panel of two. The marked line is the tallest one inside the shaded band, 0.8 to 1.5 Hz at rest; its harmonics stand at twice and three times that frequency. The line below 0.5 Hz is breathing.

✓CHECK **Checkpoint 2** Every beat is marked in both channels (**D4**), and the heart rates from the peaks and from the FFT agree within the FFT bin spacing (**D5** prints ✓). Two independent methods, one heart rate: yours.

Part E: AC, DC, ratio of ratios, SpO_2

GOAL Estimate your blood oxygen saturation from the two colors.

about 25 min

For each wavelength, AC/DC is the fraction of the light that the pulsing arterial blood removes; LED brightness and detector gain cancel in it. The ratio of ratios R of Eq. (3.30) compares the two colors, and the lab calibration $\mathrm{SpO}_2 = 110 - 25R$, Eq. (3.33), turns it into a saturation (Fig. 3.35). Measure AC and DC *beat by beat*: one beat runs from one dip to the next. In that window AC is the peak-to-peak size of the filtered signal and DC is the mean of the raw signal. Use the same windows, the IR beats, for both colors.

E1 ✎EDIT Write `ac_dc_per_beat(raw, y, beats)`

E1 · ✎EDIT in the notebook

```
def ac_dc_per_beat(raw, y, beats):
    """AC and DC of each beat, from one dip to the next."""
    ac, dc = [], []
    for a, b in zip(beats[:-1], beats[1:]):   # dip a to dip b
        pass    # ✎ replace: append this beat's AC and DC
    return np.array(ac), np.array(dc)

ac_ir, dc_ir_b = ac_dc_per_beat(ir, y_ir, beats)
ac_red, dc_red_b = ac_dc_per_beat(red, y_red, beats)
summary = pd.DataFrame({"AC (counts)": [ac_red.mean(),
    ac_ir.mean()],
                        "DC (counts)": [dc_red_b.mean(),
                            dc_ir_b.mean()],
                        "AC/DC (%)": [100 * np.mean(ac_red /
                            dc_red_b), 100 * np.mean(ac_ir /
                            dc_ir_b)]},
                       index=["red", "IR"])
summary.round(3)
```

WHAT TO WRITE replace the `pass` of line 5: for the beat from dip `a` to the next dip `b`, append its AC, from the filtered `y[a:b]`, to `ac`, and its DC, from the raw `raw[a:b]`, to `dc`. The function returns two arrays, one value per beat.

HINT AC is the peak-to-peak size, the largest minus the smallest value; DC is the mean.

DONE WHEN the table shows AC, DC, and AC/DC for red and IR, with the IR AC/DC the larger.

E2 EDIT Ratio of ratios R and SpO_2

E2 · EDIT in the notebook

```
R_beats = ...     # ratio of ratios for each beat
R = ...           # one value for the recording
spo2 = ...        # SpO2 in percent

def window_ratio(x):
    """Whole-window AC/DC: (95th - 5th percentile) / mean."""
    return (np.percentile(x, 95) - np.percentile(x, 5)) /
        np.mean(x)

R_window = window_ratio(red) / window_ratio(ir)
print(f"R = {R:.3f} (beat to beat: {R_beats.min():.3f} to
    {R_beats.max():.3f})")
print(f"SpO2 = 110 - 25 R = {spo2:.1f} %")
print(f"whole-window method on the raw signals: R =
    {R_window:.3f}, SpO2 = {110 - 25 * R_window:.1f} %")
```

WHAT TO WRITE `R_beats`, the ratio of ratios of every beat, from the arrays of **E1** (`ac_red`, `dc_red_b`, `ac_ir`, `dc_ir_b`); `R`, one value for the recording, the median of the beats, which ignores a single bad beat (worked example "One bad beat"); and `spo2` in percent from Eq. (3.33).

HINT the arrays divide element by element: `ac_red / dc_red_b` is the red AC/DC of every beat. Use `np.median`.

DONE WHEN the cell prints R, the range of the beat values, and SpO_2.

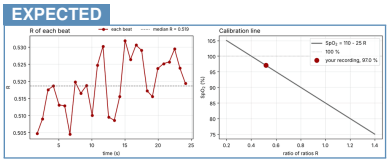
The cell also prints the whole-window R, computed from the 5th to 95th percentile span of each raw signal over its mean: a quick estimate that needs no beat finding.

E3 RUN Plot R and the calibration line

E3 plots R for every beat with its median, and your point on the calibration line $110 - 25R$.

YOU SHOULD SEE beat values of R that agree within a few percent, a median R of 0.36 to 0.45, and SpO_2 of about 99 to 101 %; the whole-window R differs, usually larger. A value above 100 % comes from the generic calibration, not from your blood (Question 4).

EXPECTED



E3 Left: the beat values of R scatter by a few percent about their median. Right: your point on the line. The simulated finger has $R = 0.52$; a real one sits further left, near 0.4.

E4 RUN Compare your recording with `ppg_backup.csv`

`analyze(df)` runs your functions of Parts C to E on any recording. E4 shows your recording next to `ppg_backup.csv`, which a real demo board recorded from a healthy volunteer.

YOU SHOULD SEE for `ppg_backup.csv`: 8 beats, 62.5 bpm from the peaks and 62.4 bpm from the FFT, AC/DC of 1.25 % IR and 0.45 % red, $R =$

0.36, and SpO_2 = 101 %. Your own numbers fall in the ranges of Table 3.3.

✓CHECK **Checkpoint 3** You have the four AC and DC values, R, and SpO_2, and can say in one sentence whether the result is physically reasonable. You built a pulse oximeter: the same physics as the clip on a hospital patient's finger.

Part F: Experiment: what changes the PPG?

GOAL Find out what pressure, exercise, and an empty sensor do to the signal. about 25 min

F1 ▸ RUN Set up the recorder

F1 defines `record_condition(name)`: it records 15 s, saves the data as `ppg_<name>.csv`, and analyzes them with your functions. Run it once.

F2 ◆ DO Record each condition

For each condition below, set it up, put its name in the line marked ✎, `record_condition("resting")`, and run F2. Change only the finger; keep your hand still and cupped as in **B1**. Before each run, predict whether DC and AC/DC go up or down.

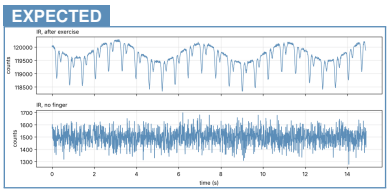
- `"resting"`: as in **C1**.
- `"pressing hard"`: press your fingertip firmly onto the sensor.
- `"barely touching"`: the lightest touch that still covers the window.
- `"no finger"`: nothing on the sensor.
- Only if you are comfortable doing it, one of `"after exercise"` (30 s of jumping jacks or a flight of stairs, then sit and record at once) or `"holding breath"` (start recording after about 15 s of holding your breath). Stop if you feel dizzy.

YOU SHOULD SEE one summary line per condition, and a file `ppg_<name>.csv` for each. With no finger the DC drops to a few thousand counts, yet the code still reports a heart rate and an SpO_2 (Question 6).

TRY IT ON THE BOARD Change how hard you press. With a feather-light touch the DC level drops and the pulse can disappear in noise; with firm pressure the tissue is squeezed, the arteries are partly blocked, and the AC component shrinks. Somewhere in between the pulse is largest. Also try cupping your hand over the sensor to block room light, and watch the DC level change.

F3 ▸ RUN Compare the conditions

F3 plots the raw IR signal of every condition, each panel with its own vertical scale, and tabulates the results of `analyze` for all of them.

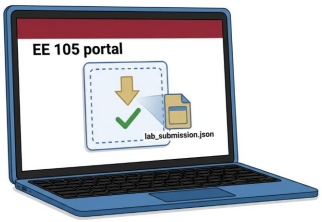


F3 The last two of five panels. After exercise the dips come faster. With no finger the level falls from about 120 000 to a few thousand counts and only noise is left.

Save and upload

Save ▸ RUN Write `lab3.2_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab3.2_submission.json` in the lab folder: your numbers, your **C1** recording (every second sample, so the portal can recompute your heart rate), and the Part F summary. Log in at `ee105.org/portal`, open Lab 3.2, and upload that file. Your recordings stay in the lab folder as `ppg_*.csv`.



Save Upload `lab3.2_submission.json` on the course portal under Lab 3.2.

YOU SHOULD SEE `[ok] Saved lab3.2_submission.json`, and then the portal shows your grade. A run on the simulated board counts as practice and earns no credit.

FINISH THE LAB

You built a pulse oximeter: a live two-color PPG, a filter and beat finder, your heart rate two ways, and an SpO_2 estimate. The lab is done when every box is ticked.

☐	**C1**	◆ DO	your own 25 s recording, `ppg_rest.csv`, with `USE_BACKUP = False`
☐	**C2**	✎ EDIT	`fs`, `dc_ir`, `dc_red`
☐	**C3**	⊡ ANSWER	`ac_ir_eye`, `ac_red_eye`
☐	**D1**	✎ EDIT	`F_LOW`, `F_HIGH`, and `bandpass`
☐	**D3**	✎ EDIT	`find_beats`, `ibi`, `hr_peaks`
☐	**D5**	✎ EDIT	`fft_heart_rate`
☐	**E1**	✎ EDIT	`ac_dc_per_beat`
☐	**E2**	✎ EDIT	`R_beats`, `R`, `spo2`
☐	**F2**	◆ DO	at least three conditions, one of them `"no finger"`
☐		✓ CHECK	Checkpoint 1: a clear live pulse in both channels
☐		✓ CHECK	Checkpoint 2: every beat marked; peaks and FFT agree
☐		✓ CHECK	Checkpoint 3: AC, DC, R, SpO_2, and a verdict on them
☐			Questions 1 to 7 answered in the notebook
☐	**Save**	⭳ SAVE	`lab3.2_submission.json` uploaded on the portal

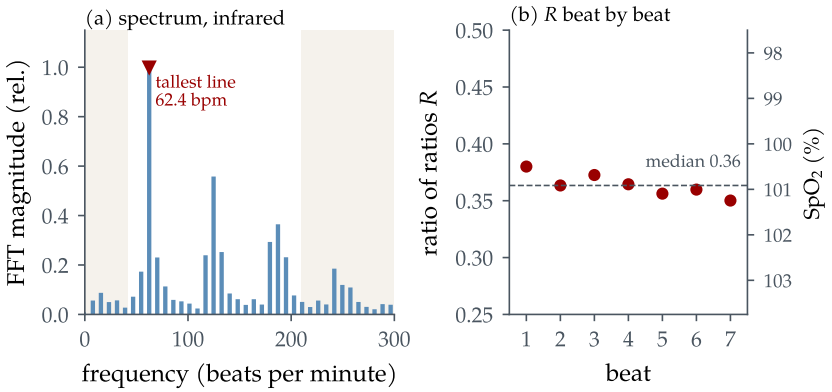


Figure 3.46: Parts D and E on the real recording `ppg_backup.csv`. (a) Spectrum of the infrared signal. The heart rate is the tallest line in the search band from 42 to 210 bpm (unshaded), here 62.4 bpm; the lines near 125 and 187 bpm are its harmonics. With only 7.7 s of data the lines are 7.8 bpm apart. (b) R for each of the seven beats. The values agree within a few percent; the median, 0.36, gives $SpO_2 = 101\,\%$ on the generic calibration line (right axis).

Typical results

Quantity	Typical result on a real board
Sampling rate	120 to 135 samples per second; the 3250 samples of C1 take 24 to 27 s
DC with a finger	about 100 000 to 120 000 counts in each channel (6 to 7.5 µA)
DC with no finger	a few thousand counts at most
AC/DC per beat	infrared 0.3 to 2 %, red 0.13 to 0.8 %; infrared always the larger
R (median of the beats)	0.36 to 0.45
$SpO_2 = 110 - 25R$	about 99 to 101 %
Heart rate	50 to 90 bpm at rest (77 on one real board); peaks and FFT agree within $60/T$ (2.4 bpm for 25 s)

Table 3.3: Expected results for Lab 3.2. The file `ppg_backup.csv` gives 62.5 bpm from the peaks, 62.4 bpm from the FFT, and a median R near 0.36.

Table 3.3 lists typical numbers, Fig. 3.36 shows a real recording, and Fig. 3.46 shows what Parts D and E make of it. Many students get an SpO_2 slightly above 100 %. That is expected: $110 - 25R$ is a generic calibration that was never fit to this sensor, and healthy fingers on this board give R near 0.4, the 100 % point (worked example "A real recording from the board"). The whole-window R often differs from the beat-by-beat value: breathing and drift change both colors by about the same fraction, add to both AC values, and pull R toward 1, so the beat-by-beat median is the better number. With no finger, the code still reports a heart rate and an SpO_2, computed from noise; a real oximeter first checks that a finger is present. Pressing hard usually raises the DC level (less blood in the fingertip absorbs less light) and shrinks AC/DC; barely touching lowers the DC and lets motion and room light in. After exercise the heart rate rises but R and SpO_2 barely change, and holding the breath for half a minute barely changes SpO_2 in a healthy person.

Troubleshooting

- *Only noise, no pulse:* press more lightly, keep still and quiet, warm your hand, and block room light. Pressing too hard is the most common mistake.
- *DC of only a few thousand counts:* no finger on the window, or the selector is not on Module 3 (**A1** would have warned).
- *A trace flat at* 262 143 *counts:* the ADC is saturated. Record with a lower LED current: in **C1**, use `board.ppg(seconds=SECONDS, amplitude=0x40)`.
- *Missed or extra beats:* change the prominence in `find_beats` (**D3**), and check that you search the negative of the signal.
- *R above 1 with a clean pulse, on every try:* the `ir` and `red` columns may be swapped in your firmware. With a healthy, still finger the `ir` column must have the larger AC/DC. Report it to the teaching staff.
- *"No reply" from the board:* rerun **A1**.
- *No board:* set `SIMULATE = True` in **A1** to practice, and `USE_BACKUP = True` in **C1** to analyze real data.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. What were the mean infrared and red counts with and without a finger (**B1**)? From your Part C plot, estimate the AC swing of each channel and compute AC/DC in percent (**C3**). Which channel has the larger ratio? Convert your DC levels to photocurrent (62.5 pA per count).
2. Which cutoffs did you choose for the band-pass filter (**D1**), and why? At which frequencies is the part that the filter removed (**D2**)?
3. Report your heart rate from the peaks and from the FFT. Do they agree within $60/T$ bpm? How much does the instantaneous rate vary from beat to beat (**D4**), and does it follow your breathing?
4. Report the four AC and DC values, R, and SpO_2. Is the result physically reasonable? If it is above 100 %, explain why. Do you trust the beat-by-beat R or the whole-window R more?
5. How did DC, AC/DC, and R change when you pressed hard, barely touched, and (if you did it) after exercise or holding your breath? Explain with the picture of light in tissue (Fig. 3.31).

6. With no finger on the sensor your code still reports a heart rate and an SpO_2. Why? What simple check should an oximeter make before it shows a number?
7. Why does R not depend on the brightness of the LEDs or the gain of the photodiode?

Summary

- A sensor is a transducer from a physical quantity to an electrical signal. A sensing system has five stages: sensor, amplifiers and filters, ADC, processing and transmission, display.
- Sensitivity is the slope $\Delta S_{\text{out}}/\Delta S_{\text{in}}$ of the transfer curve. Range, linearity, and resolution complete the description. Noise is measured by its RMS value, and $\text{SNR}_{\text{dB}} = 20\log_{10}(\text{signal}/x_{\text{rms}})$.
- An NTC thermistor follows $R_T = R_0 e^{\beta(1/T - 1/T_0)}$; a Pt100 RTD follows $R_0(1 + AT + BT^2)$. The board's divider (thermistor on top, 10 kΩ below) gives $V_{\text{out}} = V_S R/(R + R_T)$ and 36.7 mV/K at 25 °C; a Wheatstone bridge outputs $V_S\,\delta R/(4R)$ for a small change and zero at balance.
- The measurement current heats the sensor by P_T/δ (0.27 mW, about 0.14 °C on the board). The bead responds as a first-order system with $\tau = R_{\text{th}}C_{\text{th}}$, seconds to warm and tens of seconds to cool. A reference thermometer turns a precise reading into an accurate one (one-point offset or two-point line).
- An N-bit ADC has $\text{LSB} = V_{\text{ref}}/2^N$ (3.22 mV for the board's 10 bits at 3.3 V); divided by the sensitivity it gives the temperature resolution, about 0.09 °C. Sample faster than twice the highest frequency present, or high frequencies alias to low ones.
- Op amps (Chapter 2) with feedback: non-inverting gain $1 + R_f/R_i$ (a follower buffers a sensor); inverting gain $-R_f/R_i$; difference amplifier $V_{\text{ref}} + (R_f/R_i)(V_2 - V_1)$ for bridges on a single supply; TIA $V_{\text{out}} = \mp I_S R_f$, positive on the board (3 MΩ, full scale 1.1 µA). Remove the DC before amplifying a small AC signal.
- The Fourier transform splits a signal into sine components; filters keep or remove frequency bands. DFT bins are $1/T_{\text{record}}$ apart.
- Beer-Lambert: $A = -\log_{10}(I/I_0) = \varepsilon c \ell$. Hb absorbs more at 660 nm, HbO_2 more at 880 nm to 940 nm, so two wavelengths separate them.
- A PPG has a large DC part (tissue, venous blood) and a small AC part (pulsing arterial blood). $R = (\text{AC}_{\text{red}}/\text{DC}_{\text{red}})/(\text{AC}_{\text{IR}}/\text{DC}_{\text{IR}})$ and, with the lab calibration, $SpO_2 = 110 - 25R$.

- Heart rate: smooth, remove drift, find peaks, HR $= 60/\text{IBI}$; or take the strongest FFT line in 0.7 to 3.5 Hz and multiply by 60.
- On the board (Labs 3.1 and 3.2): about 130 PPG samples per second, 62.5 pA per count, R near 0.4 for a healthy finger, so the generic line often reads just above 100 %.

3.12 *Problems*

3.1 ◦ For each sensor, name the physical input, the electrical quantity that changes, and how the demo board (or a similar system) would read it:

(a) an NTC thermistor;

(b) a photodiode;

(c) the MAX86916 optical sensor;

(d) a foil strain gauge.

3.2 ◦ A pressure sensor outputs 0.5 V at 0 kPa and 2.9 V at 120 kPa, and it is linear in between.

(a) Find its sensitivity in mV/kPa.

(b) What output do you expect at 45 kPa?

3.3 • Eight readings of a sensor held at a constant input are 2.012, 2.009, 2.015, 2.010, 2.013, 2.008, 2.014, and 2.011 V.

(a) Find the mean and the RMS noise.

(b) A change in the input moves the mean output by 120 mV. Find the SNR as a ratio and in decibels.

3.4 ◦ A 10K 3950 NTC thermistor ($R_0 = 10\,\text{k}\Omega$ at 25 °C, $\beta = 3950\,\text{K}$):

(a) What is its resistance at 37 °C?

(b) What temperature corresponds to 15 kΩ?

3.5 • *(Demo board.)* The board's divider has the 10K 3950 thermistor on top and 10 kΩ on the bottom (Fig. 3.9), read by a 10-bit ADC with $V_{\text{ref}} = 3.3\,\text{V}$.

(a) Find V_{out} and the ADC code at 25 °C and at a skin temperature of 34 °C.

(b) Find the sensitivity $\mathrm{d}V_{\text{out}}/\mathrm{d}T$ at 34 °C and the temperature change that corresponds to one LSB.

3.6 •• Show that for a divider with a thermistor R_T on top and a fixed resistor R on the bottom, the sensitivity $|\mathrm{d}V_{\text{out}}/\mathrm{d}T|$ at a chosen temperature T^* is largest when $R = R_T(T^*)$. Then find the best R for measuring skin at 34 °C with the 10K 3950 thermistor, and by what percentage a standard 6.8 kΩ resistor improves the sensitivity over 10 kΩ.

3.7 • The board's divider (10K 3950 thermistor on top, 10 kΩ on the bottom) is powered from 5.0 V instead of 3.3 V.

(a) Find the measurement current and the power in the thermistor at 25 °C.

(b) With a dissipation constant of 2.0 mW/K, how much warmer than its surroundings is the bead?

(c) Back at 3.3 V, a student uses a 2.2 kΩ bottom resistor. Find the thermistor power, the self-heating, and the sensitivity at 25 °C. Is this a good trade?

3.8 •• You pinch the thermistor at $t = 0$, when it reads $T_0 = 23.0\,^\circ\text{C}$. It reads $T_5 = 28.7\,^\circ\text{C}$ at $t = 5\,\text{s}$ and $T_{10} = 30.8\,^\circ\text{C}$ at $t = 10\,\text{s}$. Assume the first-order response of Eq. (3.14).

(a) Show that $T_{\text{final}} = (T_5^2 - T_0 T_{10})/(2T_5 - T_0 - T_{10})$.

(b) Find T_{final} and τ.

(c) When is the reading within 0.1 °C of T_{final}?

3.9 • Calibration and tolerance.

(a) The board thermometer reads 23.1 °C while a reference thermometer beside it reads 22.4 °C. What offset do you store?

(b) The bottom resistor of the divider is really 9.90 kΩ (1 % low), but the code assumes 10.0 kΩ. At a true 25.0 °C, what temperature does the code report? (Ignore the rounding of the ADC.)

(c) Show that this resistor error multiplies the computed R_T by the same factor at every temperature. What is the reporting error at 37.0 °C? Would a one-point offset measured at 25 °C remove it?

3.10 ◦ An old recording of A1, one reading per millisecond, has a mean of 0.075 V. Of the readings, 54 % are exactly 0, and the rest form bumps that repeat every 16 to 17 readings.

(a) What is the frequency of the bumps?

(b) What was wrong with the setup, and what would you check?

(c) What resistance and temperature do Eqs. (3.9) and (3.6) give for the mean voltage? Why is that number meaningless?

3.11 • A 350 Ω strain gauge with gauge factor 2.1 sits in the R_1 position of a Wheatstone bridge (Fig. 3.14) whose other arms are 350 Ω, powered by $V_S = 5\,\text{V}$. The beam it is glued to is strained by 1000 microstrain.

(a) Find δR and the bridge output V_o.

(b) What amplifier gain brings V_o up to 1 V?

3.12 • A Pt100 bridge ($V_S = 3.3\,\text{V}$, the other three arms 100 Ω, the Pt100 in the R_1 position) drives the difference amplifier of Fig. 3.19 with $R_i = 10\,\text{k}\Omega$, $R_f = 100\,\text{k}\Omega$, and $V_{\text{ref}} = 0.50\,\text{V}$, on a 3.3 V supply.

(a) Find V_{out} at 0, 25, and 37 °C.

(b) Up to what temperature does the output stay below 3.3 V?

(c) Find the temperature change per ADC step (3.22 mV) near 37 °C.

3.13 ◦ A TMP36 IC sensor outputs 750 mV at 25 °C and 10 mV per degree. It is read by the board's 10-bit ADC ($V_{\text{ref}} = 3.3\,\text{V}$).

(a) Find its output and the ADC code at 40 °C.

(b) What temperature step is one LSB? Compare with the thermistor divider.

(c) A non-inverting amplifier with a gain of 3.0 is added. What is the new temperature step per LSB, and up to what temperature does the output stay below 3.3 V?

3.14 ◦ A 12-bit ADC has $V_{\text{ref}} = 3.3\,\text{V}$.

(a) Find the LSB and the code for a 1.000 V input.

(b) What is the largest quantization error if each code is read as the center of its step?

(c) A PPG must capture heart rates up to 200 bpm including harmonics up to the fifth. What is the minimum sampling rate?

3.15 • Power-line interference at 60 Hz leaks into a sensor signal that is sampled without an anti-aliasing filter.

(a) At what apparent frequency does the hum appear if the sampling rate is 50 Hz, 100 Hz, 59 Hz, or 1 kHz?

(b) Which of these rates would make the hum look like a resting heartbeat in a PPG, and what heart rate would it suggest?

3.16 • Room lights flicker at 120 Hz.

(a) At what frequency does the flicker appear in a PPG sampled at 130 and at 125 samples per second?

(b) For which sampling rates between 100 and 140 samples per second would the alias land inside the heart-rate band, 0.7 to 3.5 Hz?

(c) An RC low-pass filter with a cutoff of 5 Hz is placed before the ADC. By what factor, and by how many decibels, does it reduce the flicker?

3.17 ◦ For the non-inverting amplifier of Fig. 3.47:

(a) Find the gain.

(b) Find V_{out} for $V_{\text{in}} = 50\,\text{mV}$.

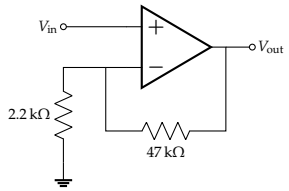


Figure 3.47: Problem 3.17.

3.18 • The circuit of Fig. 3.48 cascades a non-inverting stage and an inverting stage. The op amps run from ±5 V and their outputs can reach the rails.

(a) Find the overall gain $V_{\text{out}}/V_{\text{in}}$.

(b) Find V_{out} for $V_{\text{in}} = 0.1\,\text{V}$.

(c) What is the largest input amplitude that avoids clipping?

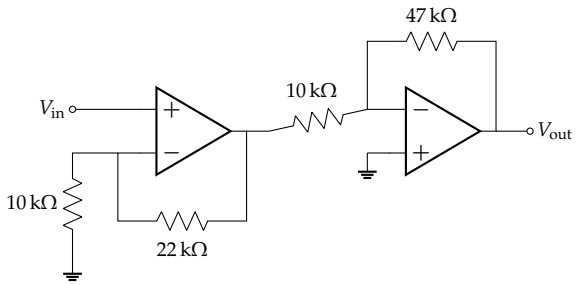


Figure 3.48: Problem 3.18: a non-inverting stage followed by an inverting stage.

3.19 • A photodiode in a TIA (Fig. 3.20) delivers at most 2 µA.

(a) Choose R_f so the output reaches −3 V at maximum light.

(b) Choose C_f for a 10 Hz bandwidth.

(c) What is the output at 0.5 µA?

(d) The board runs from a single 3.3 V supply. What simple change makes the output positive?

3.20 • *(Demo board.)* With the selector on Module 3 and J5 on P1–A, A6 reads code 22 with the pair-1 LED off and code 640 with it on (Fig. 3.21, 3 MΩ feedback).

(a) Find the two photocurrents and the part added by the LED.

(b) The LED current is doubled, which doubles the light it adds. What code do you expect?

(c) What feedback resistor would put that doubled signal at about 3.0 V?

3.21 •• *(Readout design.)* Design the two-stage readout of Fig. 3.22 for a photodiode whose current is 1 µA DC with a pulsatile ripple of 1 % peak to peak. Orient the photodiode so that stage 1 gives $V_1 = +I_{PD}R_f$, and use Eq. (3.23) for stage 2.

(a) Choose R_f so that V_1 has a DC level of 1.0 V. What is the AC swing of V_1?

(b) Choose V_{DAC} and the ratio R_{2f}/R_2 so that the output has a 1 V peak-to-peak AC swing. Give one resistor pair.

(c) Between what voltages does the output swing? Does it fit a 0 V to 3.3 V ADC?

3.22 • A 10 µF capacitor is placed from A1 to ground on the board's divider (Fig. 3.27).

(a) Find the cutoff frequency at 25 °C.

(b) By what factor, and by how many decibels, does it reduce 60 Hz hum?

(c) Find the RC time constant and compare it with the bead's thermal time constant of about 6 s. Does the capacitor slow the thermometer noticeably?

3.23 ◦ Transmittance and absorbance.

(a) A sample passes 50 % of the light. What is its absorbance? What absorbance corresponds to 1 % transmission?

(b) A sample has $A = 0.50$. What fraction of the light gets through?

(c) The concentration in (b) triples. What fraction gets through now?

3.24 • A dye has a molar absorption coefficient of $2.0 \times 10^4\,\mathrm{M^{-1}\,cm^{-1}}$ at the wavelength used. A 1.0 cm cuvette transmits 25 % of the light.

(a) Find the absorbance and the dye concentration.

(b) What fraction of the light would get through a 3.0 cm path of the same solution?

3.25 • With the coefficients of the example "Two absorbers, two wavelengths" and a 1.00 cm path, a diluted blood sample has an absorbance of 0.0305 at 660 nm and 0.0556 at 880 nm. Find the concentrations of HbO_2 and Hb and the saturation.

3.26 •• The AC/DC approximation.

(a) Starting from $I = I_{DC}\,10^{-\Delta A}$, show that $\mathrm{AC/DC} \approx (\ln 10)\,\Delta A$ for a small pulse.

(b) For AC/DC = 1.00 % exactly (the light drops to $0.99\,I_{DC}$), find ΔA exactly and with the approximation. How large is the relative error?

(c) Show that the factor $\ln 10$ cancels in R, so that $R = \Delta A_{red}/\Delta A_{IR}$ to the same accuracy.

3.27 • Two people are measured with the board's oximeter.

(a) Person 1: $\mathrm{AC_{red}} = 12$, $\mathrm{DC_{red}} = 800$, $\mathrm{AC_{IR}} = 30$, $\mathrm{DC_{IR}} = 1000$ (all in mV). Find R and SpO_2.

(b) Person 2: $\mathrm{AC_{red}} = 24$, $\mathrm{DC_{red}} = 600$, $\mathrm{AC_{IR}} = 20$, $\mathrm{DC_{IR}} = 500$ (mV). Find R and SpO_2. Should this person seek medical attention?

(c) Why does pressing the finger very hard on the sensor make the reading unreliable?

3.28 •• Using the approximate extinction coefficients at 660 nm ($\varepsilon_{Hb} = 3227$, $\varepsilon_{HbO_2} = 320$) and 880 nm ($\varepsilon_{Hb} = 726$, $\varepsilon_{HbO_2} = 1154$, all in $\mathrm{M^{-1}\,cm^{-1}}$):

(a) Solve Eq. (3.32) for the saturation S as a function of R.

(b) Find the values of R for which the theory predicts 100 % and 0 %.

(c) Compare the theoretical and empirical $(110 - 25R)$ SpO_2 at $R = 0.6$, and give a physical reason for the difference.

3.29 • *(Demo board.)* Beat-by-beat analysis.

(a) One beat of a board recording has AC = 470 and DC = 109 020 counts in the red channel, and AC = 1180 and DC = 120 650 counts in the infrared. Find AC/DC for each channel, R, and SpO_2.

(b) The beat-by-beat values of R in another recording are 0.42, 0.44, 0.41, 1.30, 0.43, and 0.45. Find SpO_2 from the mean and from the median of R. Which would you report, and why?

3.30 ◦ A 25 s PPG recording has its tallest FFT line in the heart-rate band in bin 34 (bin 0 is DC).

(a) What are the frequency of that bin and the heart rate?

(b) What heart rates do the two neighboring bins represent?

(c) How long must a recording be for a bin spacing of 1 bpm?

3.31 • *(Python.)* A PPG recorded at 100 samples per second has beats at sample indices 57, 148, 236, 327, 420, and 509.

(a) Write a few lines of Python that compute the IBIs, the instantaneous heart rates, and the average heart rate, and give the results.

(b) An FFT is taken of an 8 s window of the same recording. What is the bin spacing in hertz and in bpm?

3.32 •• *(Design.)* You want a fever thermometer built from a 10K 3950 thermistor in the board's arrangement (thermistor on top, a fixed resistor R below, 3.3 V, 10-bit ADC).

(a) Which R gives the largest sensitivity at 37 °C? Choose the nearest of the standard values 5.6, 6.2, and 6.8 kΩ.

(b) With $R = 6.2\,\text{k}\Omega$, find V_{out} and the ADC code at 35 and 41 °C, the sensitivity at 37 °C, and the temperature step per LSB.

(c) A difference amplifier (Eq. (3.21)) subtracts $V_{\text{out}}(35\,^\circ\text{C})$ and uses $V_{\text{ref}} = 0.30\,\text{V}$. What gain maps 41 °C to 3.00 V, and what is the temperature step per LSB after the amplifier? What now limits the resolution?

3.33 • *(Python, demo board.)* Write a Python function `code_to_celsius` that converts a 10-bit ADC code from the board's thermistor divider (Fig. 3.9, 10K 3950 part) to degrees Celsius. Use it to convert the codes 512, 560, and 610.

4
Machine Learning

In this chapter you will learn to

- Explain when a learned model beats a hand-written rule, and turn raw sensor data into a short feature vector.
- Compute dot products and matrix-vector products, and count the multiply-accumulate operations (MACs) they need.
- Evaluate an artificial neuron with step, sigmoid, tanh, and ReLU activations, and design perceptrons and their decision boundaries.
- Run the forward pass of a small network, and count the weights, biases, and MACs of fully connected and convolutional layers.
- Train a network by gradient descent, and backpropagate gradients by hand through a small network.
- Judge a classifier with training and test curves, a confusion matrix, sensitivity, precision, and the F_1 score.
- Estimate memory, latency, and energy of a network on a microcontroller, quantize it to 8 bits, and explain how a resistive crossbar multiplies a matrix by a vector with Ohm's law and KCL.
- Classify handwritten digits in Python and from the demo-board camera.

Hold a handwritten 7 in front of the small camera on the demo board. A fraction of a second later the notebook on your laptop prints `prediction: 7 (97.0 %)`. Between the paper and that line of text, the OV7675 camera turned light into 76 800 pixel values, the Arduino Nano 33 BLE Sense kept one pixel in four and sent the 160×120 pic-

ture over USB, the laptop cut out the digit and shrank it to $28 \times 28 = 784$ numbers, and a neural network multiplied those numbers by tables of numbers it had learned from sixty thousand examples of other people's handwriting (Figure 4.1). Nobody wrote a rule that says what a 7 looks like. The rule was learned.

This chapter explains how. We start with why learning is needed at all, then build a neural network from its smallest part, the neuron, using nothing more than vectors, dot products, and one nonlinear function. We then show how a network learns its weights by walking downhill on a cost function, and how to tell whether what it learned is any good. The last part brings us back to hardware: how many multiply-accumulates and bytes a network needs, how 8-bit numbers shrink it to fit a microcontroller, and how a grid of resistors can do a matrix multiplication in one step using the two circuit laws of Chapter 1. The two labs train a digit classifier in Python and then run it on live pictures from the board's camera.

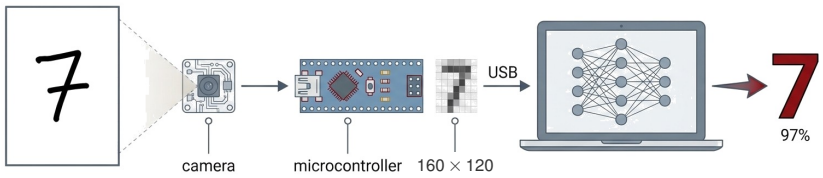


Figure 4.1: The machine-vision system of this chapter. The camera on the demo board captures a handwritten digit, the Arduino sends a 160×120 picture over USB, and a neural network on the laptop turns it into a label with a confidence.

4.1 *From Sensors to Decisions*

In Chapter 3 a thermistor voltage became a temperature through a formula. The relationship between the measured voltage and the temperature is fixed by physics, so a few lines of code that apply the formula are all the "intelligence" the system needs. The same input always gives the same output. We call such a mapping *deterministic*.

Now consider a microphone that should turn on a machine when someone says "stop." The waveform for that word changes with the speaker's accent, pitch, speed, distance from the microphone, and the noise in the room. There is no formula from samples to the word, and a list of hand-written rules would never cover every case. A self-driving car faces the same problem at a larger scale: it must fuse camera images, lidar, radar, and inertial measurements under changing light and weather. For these *probabilistic* problems we collect many labeled examples and let an algorithm find the mapping. A neural network is one such algorithm, and it is the one we study here.

KEY IDEA Use a formula when physics fixes the input-output relation. Learn a model from labeled data when the relation varies too much to write down.

Until recently, running a neural network meant a desktop processor or a single-board computer drawing hundreds of milliwatts. Micro-

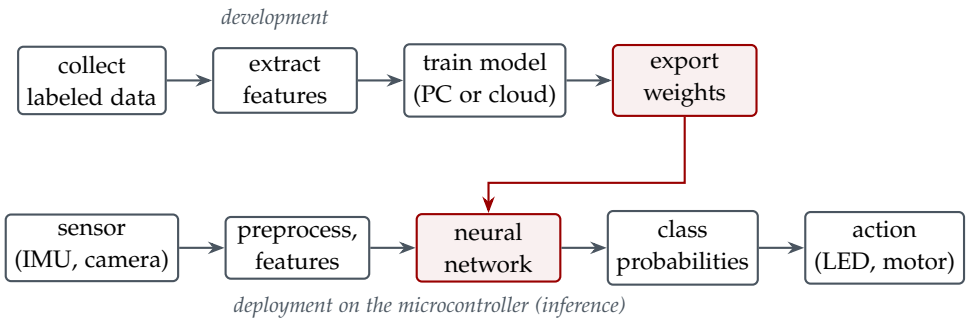


Figure 4.2: The TinyML workflow. A model is trained once on a computer from labeled data; its weights are copied to the microcontroller, which then runs the same feature extraction and the network on every new window of sensor data.

controllers have caught up. The Nano 33 BLE Sense on the demo board uses an nRF52840 chip built around a 32-bit ARM Cortex-M4 core with a floating-point unit, clocked at 64 MHz, with 1 MB of flash and 256 kB of RAM. That is enough to run small networks at a power of a milliwatt or less, which is what makes always-on intelligence in a wearable, a vibration monitor on a bridge, or a pest detector in a field possible. Running machine learning on such a device is called *TinyML*.

The TinyML workflow

Figure 4.2 shows the steps. During development you *collect* labeled data from the sensor, *extract features* from each recording, and *train* a model on a computer. You then *deploy* the trained weights to the microcontroller, where the same preprocessing and feature extraction run on live data and the model makes a prediction. Using a trained model on new data is called *inference*. The final step is an action: the code checks the output probabilities and does something, for example `if (p_circle > 0.8)` turn on an LED.

Features

A *feature* is a measurable property of what you observe. Raw data are often a poor input to a small model simply because there are too many numbers. Our running example classifies four hand gestures (circle, idle, left-right, up-down) from the Nano's 3-axis accelerometer. A 2-second window sampled at 62.5 Hz holds 125 samples per axis, or 375 numbers. Feeding all 375 into a network makes it large and slow, and it needs a lot of training data to avoid memorizing noise.

A better plan is to compute a handful of numbers that describe the motion. The root-mean-square (RMS) value of each axis,

$$x_{\mathrm{rms}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x[n]^2}, \tag{4.1}$$

measures how strongly the board moved along that axis. Three RMS values already separate the gestures surprisingly well when you plot them as points in 3D: each gesture forms its own cloud. Features need not be in the time domain. The power spectral density from the Fourier analysis of Chapter 3 tells you at which frequencies the motion has energy, so the peak frequencies, peak heights, and the power in a few frequency bands make good features too. The final model uses 11 features per axis (the RMS, three spectral peak frequencies, three peak heights, and four band powers), so 375 raw values become a feature vector of 33 numbers.

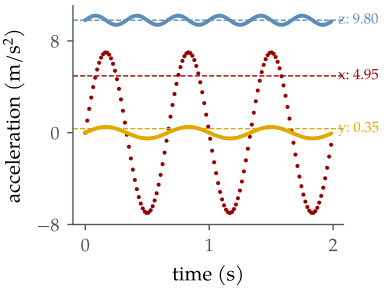


Figure 4.3: One 2-second window of a left-right gesture, 125 samples per axis. Dashed lines mark the RMS of each axis (Worked Example 4.1).

Worked Example 4.1 *Three numbers instead of 375*

A left-right gesture is recorded for 2 s at 62.5 Hz. Model the three axes as $a_x = 7.0\sin(2\pi\,1.5\,t)$, $a_y = 0.5\sin(2\pi\,1.5\,t)$, and $a_z = 9.8 + 0.4\sin(2\pi\,3\,t)$ (all in $\mathrm{m/s^2}$; these are representative values, Figure 4.3). Compute the RMS feature of each axis and the reduction in input size.

Solution. The window holds $N = 2 \times 62.5 = 125$ samples per axis. A sine of amplitude A sampled over whole periods has RMS $A/\sqrt{2}$ (the window holds exactly 3 periods of 1.5 Hz and 6 periods of 3 Hz). So $a_{x,\mathrm{rms}} = 7.0/\sqrt{2} = 4.95\,\mathrm{m/s^2}$ and $a_{y,\mathrm{rms}} = 0.5/\sqrt{2} = 0.354\,\mathrm{m/s^2}$. For z, the constant and the sine add in power: $a_{z,\mathrm{rms}} = \sqrt{9.8^2 + 0.4^2/2} = 9.80\,\mathrm{m/s^2}$, dominated by gravity. The model input shrinks from $3 \times 125 = 375$ numbers to 3, a factor of 125. A direct evaluation of Eq. (4.1) on the 125 samples in Python gives the same three values. ■

4.2 *Vectors, Matrices, and the Multiply-Accumulate*

Everything a neural network computes is built from one operation: multiply two numbers and add the result to a running sum. To see why, we need three ideas from linear algebra.

A *vector* is an ordered list of numbers, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. The three RMS features above form a vector with $n = 3$. A grayscale image is a vector too: read the pixels row by row and you get one long list (Figure 4.4). A 28×28 image becomes a vector with $n = 784$ entries. The network does not know the numbers came from a picture; it only sees the list.

The *dot product* of two vectors of equal length multiplies matching entries and adds them up:

$$\mathbf{w} \cdot \mathbf{x} = \sum_{i=1}^{n} w_i x_i = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n. \tag{4.2}$$

Each term is one multiply and one add, a *multiply-accumulate* or MAC,

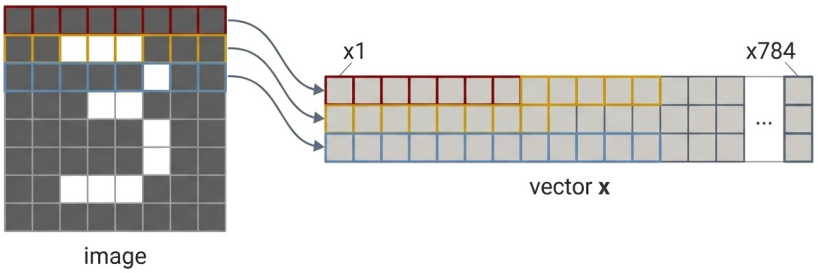


Figure 4.4: A small image is a grid of brightness values. Reading the grid row by row lays it out as one long vector, the form a neural network takes as input.

so a dot product of length n costs n MACs. The dot product also has a useful meaning. If $\mathbf{w}$ is a template, say a pattern of pixel weights, then $\mathbf{w} \cdot \mathbf{x}$ is large when the image $\mathbf{x}$ has bright pixels where $\mathbf{w}$ is positive and dark pixels where $\mathbf{w}$ is negative. It measures how much the input looks like the template.

A *matrix* is a rectangular table of numbers. An $m \times n$ matrix $\mathbf{W}$ has m rows and n columns. Multiplying it by a vector of length n gives a vector of length m whose jth entry is the dot product of row j with $\mathbf{x}$:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}, \qquad z_j = \sum_{i=1}^{n} W_{ji}\, x_i + b_j. \tag{4.3}$$

We added a vector $\mathbf{b}$ of offsets; in a neural network these are the biases. The product needs $m \times n$ MACs. Equation (4.3) is the heart of every layer in this chapter, and it is also exactly what the resistor crossbar of Section 4.9 computes with currents.

KEY EQUATION

$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$

One layer: $m \times n$ MACs for n inputs and m outputs.

WORKED EXAMPLE 4.2 *A matrix-vector product and its cost*

Compute $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$ for

$$\mathbf{W} = \begin{pmatrix} 1 & -2 & 0.5 \\ 0 & 3 & -1 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} 2 \\ 1 \\ 4 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0.5 \\ -1 \end{pmatrix},$$

and count the MACs.

SOLUTION. Row 1: $1(2) + (-2)(1) + 0.5(4) + 0.5 = 2 - 2 + 2 + 0.5 = 2.5$. Row 2: $0(2) + 3(1) + (-1)(4) - 1 = 0 + 3 - 4 - 1 = -2$. So $\mathbf{z} = (2.5, -2)$. $\mathbf{W}$ is 2×3, so the product takes $2 \times 3 = 6$ MACs; adding the bias costs one more addition per row. ■

WORKED EXAMPLE 4.3 *Dot products and the cost of a layer*

(a) Compute $\mathbf{w} \cdot \mathbf{x}$ for $\mathbf{w} = (0.5, -1, 2, 0)$ and $\mathbf{x} = (4, 1, 0.5, 3)$, and count the MACs. (b) A layer of 64 neurons takes a 28×28 image as input. How many MACs does one image need? (c) How

many would it need if it took the whole 160×120 camera picture of Lab 4.2 instead?

SOLUTION. (a) $0.5(4) + (-1)(1) + 2(0.5) + 0(3) = 2 - 1 + 1 + 0 = 2$, in 4 MACs. The zero weight still costs a MAC unless the code skips it. (b) Each neuron computes one dot product of length 784, so the layer needs $64 \times 784 = 50\,176$ MACs. (c) The picture has $160 \times 120 = 19\,200$ pixels, so the layer would need $64 \times 19\,200 = 1\,228\,800$ MACs, 24.5 times more. Shrinking the image before the network is the cheapest way to save computation, which is why Lab 4.2 reduces every picture to 28×28 first. ■

4.3 *The Artificial Neuron*

A BIOLOGICAL NEURON collects signals from thousands of other neurons through synapses of different strengths, and fires when the combined input crosses a threshold (Figure 4.5). In 1943 Warren McCulloch and Walter Pitts proposed a mathematical cartoon of this behavior.[1] The artificial neuron keeps only the arithmetic:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) = f(\mathbf{w} \cdot \mathbf{x} + b) = f(z). \tag{4.4}$$

The inputs x_i are features or pixels. Each input is scaled by a *weight* w_i (the synapse strength). The *bias* b shifts the sum: a large positive bias makes a neuron that fires for almost any input, a large negative bias makes one that rarely fires. The weighted sum $z = \mathbf{w} \cdot \mathbf{x} + b$ is called the pre-activation, and the *activation function* f turns it into the output y. Figure 4.6 shows the picture we will use throughout.

[1] A short history: McCulloch and Pitts' neuron (1943); Rosenblatt's perceptron, which learned its own weights (1958); Minsky and Papert's proof that a single-layer perceptron cannot compute XOR (1969); Hopfield's energy-based networks (1982); backpropagation for training multilayer networks (1986).

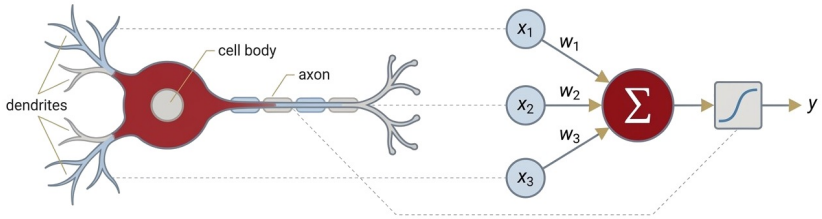


Figure 4.5: A biological neuron (left) and the artificial neuron it inspired (right). Dendrites become weighted inputs, the cell body becomes a sum and a threshold, and the axon becomes the output.

Activation functions

Four activation functions appear in this course (Figure 4.7). The *step* function outputs 1 if $z > 0$ and 0 otherwise; it is the original perceptron

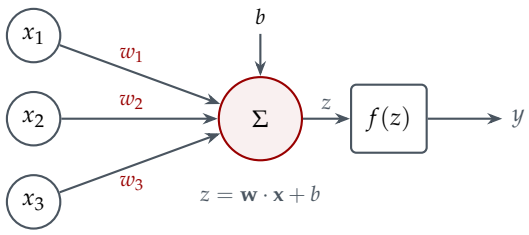


Figure 4.6: An artificial neuron with three inputs. Each input is multiplied by its weight, the products and the bias are summed, and the activation function gives the output, Eq. (4.4).

and gives a clean yes or no. The *sigmoid*

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4.5}$$

is a smoothed step that runs from 0 to 1, passing through 0.5 at $z = 0$. The hyperbolic tangent $\tanh(z) = 2\sigma(2z) - 1$ has the same S shape but runs from -1 to 1 and is centered on zero. The *rectified linear unit* $\mathrm{ReLU}(z) = \max(0, z)$ passes positive inputs unchanged and blocks negative ones. It is the cheapest to compute (one comparison) and is the default for hidden layers in modern networks, including the ones in the labs.

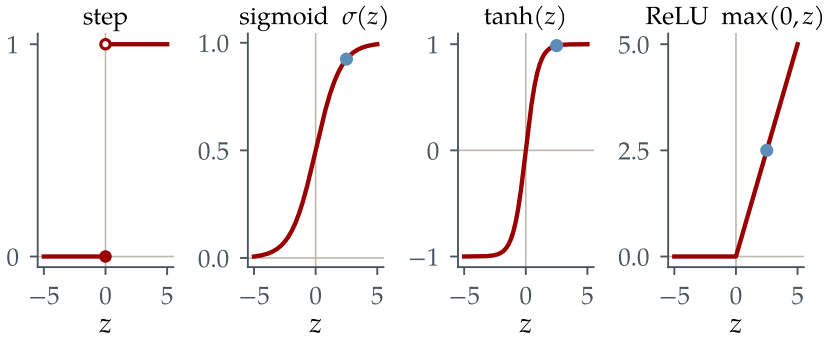


Figure 4.7: The four activation functions. The blue dots mark the outputs at $z = 2.5$ used in Worked Example 4.5.

Why not keep the step? Learning means nudging the weights so the output moves a little toward the right answer. With a step, a tiny weight change either does nothing or flips the output from 0 to 1, and that flip ripples through the rest of the network in ways that are hard to predict: you fix the network for one image of a 3 and break it for others. A smooth activation makes a small change in any weight produce a small change in the output,

$$\Delta y \approx \sum_j \frac{\partial y}{\partial w_j} \Delta w_j + \frac{\partial y}{\partial b} \Delta b, \tag{4.6}$$

so for small steps the output responds almost linearly to each weight. That is what gradient descent in Section 4.6 relies on. It also helps that the derivatives are cheap: $\sigma'(z) = \sigma(z)\,[1 - \sigma(z)]$, $\tanh'(z) = 1 -$

$\tanh^2(z)$, and the ReLU slope is 1 for $z > 0$ and 0 for $z < 0$. The sigmoid has one drawback: for $|z|$ larger than about 5 it is flat (it *saturates*), $\sigma'(5) = 0.00665$, and learning slows down. ReLU does not saturate for positive inputs.

Worked Example 4.4 *A perceptron decision*

A perceptron has weights $w_1 = 2$, $w_2 = -3$, $w_3 = 1.5$ and inputs $x_1 = 1$, $x_2 = 0$, $x_3 = 1$. It outputs 1 if the weighted sum is greater than the threshold 0, otherwise 0. Find the output.

Solution. The weighted sum is $\mathbf{w} \cdot \mathbf{x} = 2(1) + (-3)(0) + 1.5(1) = 3.5$. Since $3.5 > 0$, the output is $y = 1$. Note that $x_2 = 0$ removes the large negative weight from the sum; had x_2 been 1, the sum would be 0.5 and the output still 1. ■

Worked Example 4.5 *Three activations of one input*

A neuron has input $x = 1.0$, weight $w = 2.5$, and no bias. Compute its output with a sigmoid, a ReLU, and a tanh activation.

Solution. The pre-activation is $z = wx = 2.5$. Sigmoid: $\sigma(2.5) = 1/(1 + e^{-2.5}) = 1/(1 + 0.0821) = 0.924$. ReLU: $\max(0, 2.5) = 2.5$. Tanh: $\tanh(2.5) = (e^{2.5} - e^{-2.5})/(e^{2.5} + e^{-2.5}) = 0.987$, which agrees with $2\sigma(5) - 1 = 2(0.99331) - 1 = 0.987$. The sigmoid and tanh squash the input into a bounded range; ReLU passes it through unchanged because it is positive (Figure 4.7). ■

Worked Example 4.6 *How steep is each activation?*

Find the slopes of the sigmoid, tanh, and ReLU at $z = 0$, 2, and 5. A signal passes backward through four sigmoid neurons in a chain. By what factor can their slopes multiply it, at most?

Solution. Use $\sigma'(z) = \sigma(z)[1 - \sigma(z)]$ and $\tanh'(z) = 1 - \tanh^2 z$ (Figure 4.8). At $z = 0$: $\sigma' = 0.5 \times 0.5 = 0.25$ and $\tanh' = 1$. At $z = 2$: $\sigma(2) = 0.8808$, so $\sigma' = 0.8808 \times 0.1192 = 0.105$; $\tanh 2 = 0.9640$, so $\tanh' = 1 - 0.9293 = 0.0707$. At $z = 5$: $\sigma' = 0.00665$ and $\tanh' = 0.000182$. The ReLU slope is 1 at $z = 2$ and $z = 5$ (and 0 for any negative z); at $z = 0$ it has no single slope, and PyTorch uses 0 there. The sigmoid slope is never larger than 0.25, so four sigmoids in a chain multiply a backward signal by at most $0.25^4 = 0.0039$, and by much less when they saturate. ReLU passes the signal unchanged wherever it is active, which is why deep networks use it in their hidden layers (Section 4.7). ■

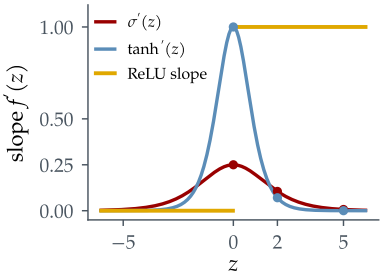


Figure 4.8: Slopes of the sigmoid, tanh, and ReLU. Dots mark $z = 0$, 2, and 5 (Worked Example 4.6).

4.4 *Perceptrons and Decision Boundaries*

A neuron with a step activation is a *perceptron*. Writing the threshold as a negative bias, $b = -\text{threshold}$, its rule is

$$y = \begin{cases} 1, & \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0, & \mathbf{w} \cdot \mathbf{x} + b \leq 0. \end{cases} \tag{4.7}$$

Ask a perceptron whether to go to Coachella based on three yes/no inputs: good weather (x_1), friends going (x_2), and having tickets (x_3), with $w_1 = w_2 = 5$ and $w_3 = 11$. With a threshold of 10, the tickets alone decide: $11 > 10$ goes, while weather and friends together give 10, which is not greater than 10. Raise the threshold to 12 and the rule becomes "tickets plus at least one of the other two." The decision is entirely in the weights and the threshold. Choosing them by hand is easy for three inputs; for 784 pixels it is hopeless, which is why we will let the network learn them.

WORKED EXAMPLE 4.7 *Should I go to the music festival?*

Design a neural network that decides whether a person attends a music festival from three inputs: weather forecast (good or bad), tickets available (yes or no), and friends attending (yes or no). Give the number of inputs, weights, the activation function, and the output.

SOLUTION. Encode each input as a bit: $x_1 = 1$ for good weather, $x_2 = 1$ if tickets are available, $x_3 = 1$ if friends attend. A single neuron is enough: 3 inputs, 3 weights, and 1 bias (4 parameters), and 1 output, $y = 1$ for "attend." Use a step activation for a yes/no decision, or a sigmoid if you want a probability of attending.

To pick values, decide what matters. Without tickets you cannot go, so give tickets the largest weight; then require good weather or friends. One choice is $w_1 = 2$, $w_2 = 6$, $w_3 = 3$, $b = -7$:

weather	tickets	friends	z	step	$\sigma(z)$
any	0	any	≤ -2	0	≤ 0.12
0	1	0	-1	0	0.27
1	1	0	1	1	0.73
0	1	1	2	1	0.88
1	1	1	4	1	0.98

The person attends only with tickets and at least one of good weather or friends. In practice you would not hand-pick the weights: you would record past decisions and train them. ■

Logic gates and the decision line

With two binary inputs, a perceptron can act as a logic gate. Consider AND. The output must be 1 only for $(1,1)$, so with the rule in Eq. (4.7) we need

$$w_1 + w_2 + b > 0, \qquad w_1 + b \le 0, \qquad w_2 + b \le 0, \qquad b \le 0.$$

The equation $w_1 x_1 + w_2 x_2 + b = 0$ is a straight line in the (x_1, x_2) plane, the *decision boundary*. Points on one side give 1, points on the other give 0. A perceptron works when a single line separates the 1s from the 0s, as in Figure 4.9; such a problem is called *linearly separable*.

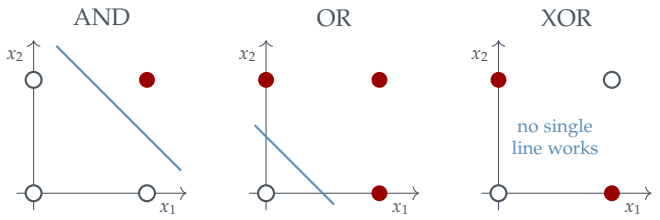


Figure 4.9: Decision boundaries for two-input gates. Filled dots are inputs whose output is 1. AND ($x_1 + x_2 - 1.5 = 0$) and OR ($x_1 + x_2 - 0.5 = 0$) are separated by one line; XOR is not.

Worked Example 4.8 *Perceptrons for AND, OR, NAND, and NOR*

Using Eq. (4.7) with $w_1 = w_2 = 1$, find the range of bias that gives AND, and the range that gives OR. Then find parameters for NAND and NOR. (This is step B4 of Lab 4.1.)

Solution. *AND.* The four inequalities above become $2 + b > 0$, $1 + b \le 0$, $1 + b \le 0$, $b \le 0$, so $-2 < b \le -1$. Pick the middle, $b = -1.5$: the sums for $(0,0), (0,1), (1,0), (1,1)$ are $-1.5, -0.5, -0.5, 0.5$, giving $0, 0, 0, 1$.

OR. Now $(0,0)$ must give 0 and the other three 1: $b \le 0$ and $1 + b > 0$, so $-1 < b \le 0$. Pick $b = -0.5$: sums $-0.5, 0.5, 0.5, 1.5$, giving $0, 1, 1, 1$.

NAND and NOR. Flipping the sign of every weight and of the bias moves each point to the other side of the line, so it inverts the output (for points not exactly on the line). NAND: $w_1 = w_2 = -1$, $b = 1.5$, sums $1.5, 0.5, 0.5, -0.5$, output $1, 1, 1, 0$. NOR: $w_1 = w_2 = -1$, $b = 0.5$, sums $0.5, -0.5, -0.5, -1.5$, output $1, 0, 0, 0$.

Because NAND alone can build any logic function, a network of perceptrons can compute anything a digital circuit can. The lab notebook's AND uses $b = -1$, which sits on the edge of the allowed range: it works, but only because $(1,0)$ and $(0,1)$ land exactly on the boundary, where $z = 0$ gives 0. ■

Decision boundaries are not limited to logic. Any two features, for example two sensor readings, form a plane, and a neuron draws a

straight line across it.

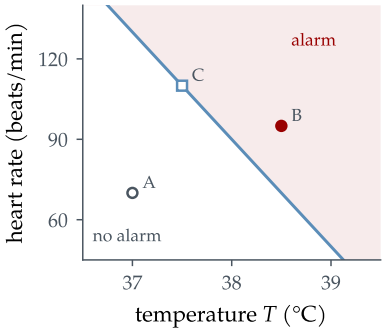


Figure 4.10: The decision line $2T + 0.05R = 80.5$ of Worked Example 4.9. Readings in the shaded region raise the alarm; point C lies on the line.

Worked Example 4.9 *A fever alarm from two sensor readings*

A wearable measures skin temperature T (in °C) and heart rate R (in beats per minute). A neuron computes $z = 2T + 0.05R - 80.5$ and raises an alarm when $z > 0$. (a) Find z and the sigmoid output $\sigma(z)$ for the readings A $= (37.0\,°\text{C}, 70)$, B $= (38.5\,°\text{C}, 95)$, and C $= (37.5\,°\text{C}, 110)$. (b) Find the temperature at which the alarm starts for heart rates of 60 and 120 beats per minute, and describe the decision boundary.

Solution. (a) A: $z = 74 + 3.5 - 80.5 = -3$, $\sigma = 0.0474$, no alarm. B: $z = 77 + 4.75 - 80.5 = 1.25$, $\sigma = 0.777$, alarm. C: $z = 75 + 5.5 - 80.5 = 0$, $\sigma = 0.5$; C sits exactly on the boundary, and the rule $z > 0$ gives no alarm.

(b) Setting $z = 0$ gives $T = (80.5 - 0.05R)/2$. At $R = 60$ the alarm starts at $T = 38.75\,°\text{C}$; at $R = 120$ it starts at $T = 37.25\,°\text{C}$. The boundary is a straight line in the (T, R) plane (Figure 4.10): a fast heart rate lowers the temperature needed for an alarm. The weights also carry units: 2 per degree and 0.05 per beat per minute make 1 °C count as much as 40 beats per minute. In practice each input is first scaled to a similar range, so that training does not have to discover such factors. ■

XOR needs a hidden layer

XOR outputs 1 when exactly one input is 1. Its 1s sit on opposite corners of the square (Figure 4.9), and no straight line separates them from the 0s. To see this algebraically, the conditions would be $b \le 0$, $w_2 + b > 0$, $w_1 + b > 0$, and $w_1 + w_2 + b \le 0$. Adding the middle two gives $w_1 + w_2 + 2b > 0$, so $w_1 + w_2 + b > -b \ge 0$, which contradicts the last condition. Minsky and Papert's 1969 proof of this limit stalled the field for more than a decade. The way out is to stack perceptrons in layers.

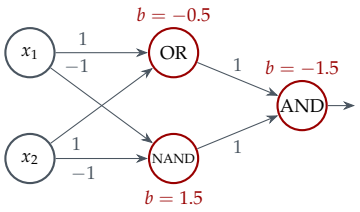


Figure 4.11: A two-layer perceptron network that computes XOR (Worked Example 4.10).

Worked Example 4.10 *XOR from three perceptrons*

Verify that the network of Figure 4.11 computes XOR.

Solution. The hidden neuron h_1 is the OR perceptron and h_2 is the NAND perceptron from Worked Example 4.8. The output neuron is an AND of h_1 and h_2. For the inputs $(0,0), (0,1), (1,0), (1,1)$: $h_1 = 0, 1, 1, 1$ and $h_2 = 1, 1, 1, 0$, so $h_1 + h_2 - 1.5 = -0.5, 0.5, 0.5, -0.5$ and $y = 0, 1, 1, 0$, which is XOR. The hidden layer moved the four points to new coordinates (h_1, h_2) in which one line does separate them. ■

That last sentence is the key to everything that follows. A hidden layer re-describes the input in new features, and the next layer draws a simple boundary in that new description. With enough hidden neurons, networks can approximate any continuous function (the universal approximation theorem).

> **Worked Example 4.11** *A neuron that looks for a pattern*
>
> A tiny image has four pixels, x_1 (top left) to x_4 (bottom right), each from 0 (white) to 1 (black). Design a sigmoid neuron whose output is high only when the top-left pixel is dark and the other three are light. Evaluate it for a perfect match, a blank image, an all-black image, and the noisy image $(0.8, 0.1, 0.1, 0)$.
>
> Solution. Weight the pixel we want heavily and positively, and the others negatively: $\mathbf{w} = (4, -4, -4, -4)$. Choose $b = -2$ so that a blank image is pushed toward 0. The outputs $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$ are: perfect match $(1, 0, 0, 0)$: $z = 2$, $\sigma = 0.881$; blank: $z = -2$, $\sigma = 0.119$; all black: $z = 4 - 12 - 2 = -10$, $\sigma = 0.0000454 \approx 0$; noisy $(0.8, 0.1, 0.1, 0)$: $z = 3.2 - 0.4 - 0.4 - 2 = 0.4$, $\sigma = 0.599$. The neuron is a template detector: its weight vector *is* the pattern. In an MNIST network each hidden neuron has 784 weights, and after training many of them look like strokes or loops; an output neuron for "0" can then fire when the neurons for the four arcs of a zero all fire. ■

4.5 *Multilayer Networks and the Forward Pass*

A neural network arranges neurons in layers. The *input layer* holds the features (it does no computation). Each *hidden layer* takes the outputs of the previous layer, and the *output layer* gives the answer. When every neuron in a layer connects to every neuron in the next, the layer is *fully connected* and is exactly Eq. (4.3) followed by an activation applied to each entry:

$$\mathbf{h} = f\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right), \qquad \mathbf{a} = g\left(\mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}\right). \tag{4.8}$$

Computing the output from the input this way, one layer at a time, is the *forward pass*. A network with several hidden layers is called *deep*; the "deep" in deep learning refers only to this number of layers. Data move strictly forward in these *feedforward* networks, also called multilayer perceptrons. (A *recurrent* network feeds outputs back as inputs at the next time step, which gives it memory for speech or other sequences; we do not use them here.)

Why the activation f? Without it the two layers collapse into one: $\mathbf{W}^{(2)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}$ is just another matrix times $\mathbf{x}$ plus another

vector. However many linear layers you stack, you still get a single linear map, which cannot do XOR. The nonlinearity is what lets depth add power.

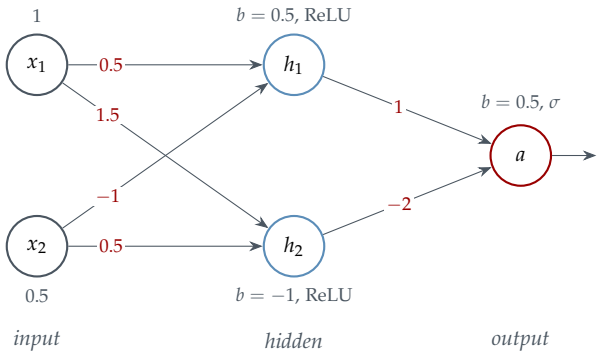


Figure 4.12: A 2-2-1 network used in Worked Examples 4.12 and 4.22. Red numbers on the arrows are weights; the input values are shown above and below the input nodes.

Worked Example 4.12 *A forward pass by hand*

For the network in Figure 4.12 with input $\mathbf{x} = (1, 0.5)$, compute the hidden activations and the output.

Solution. The first layer has $\mathbf{W}^{(1)} = \begin{pmatrix} 0.5 & -1 \\ 1.5 & 0.5 \end{pmatrix}$ and $\mathbf{b}^{(1)} = (0.5, -1)$. Then $z_1 = 0.5(1) - 1(0.5) + 0.5 = 0.5$ and $z_2 = 1.5(1) + 0.5(0.5) - 1 = 0.75$. Both are positive, so ReLU passes them: $\mathbf{h} = (0.5, 0.75)$. The output neuron has $z = 1(0.5) - 2(0.75) + 0.5 = -0.5$ and $a = \sigma(-0.5) = 1/(1 + e^{0.5}) = 0.378$. If the output means "probability of class 1," this input is classified as class 0. The pass took $4 + 2 = 6$ MACs. ■

Worked Example 4.13 *XOR with two ReLU neurons*

A 2-2-1 network has the hidden neurons $h_1 = \mathrm{ReLU}(x_1 + x_2)$ and $h_2 = \mathrm{ReLU}(x_1 + x_2 - 1)$ and the linear output $y = h_1 - 2h_2$. Evaluate it for the four binary inputs, and count its parameters and MACs.

Solution. $(0,0)$: $\mathbf{h} = (0,0)$, $y = 0$. $(0,1)$ and $(1,0)$: $\mathbf{h} = (1,0)$, $y = 1$. $(1,1)$: $\mathbf{h} = (2,1)$, $y = 2 - 2 = 0$. The outputs $0, 1, 1, 0$ are XOR. The second neuron stays off until both inputs are on, and then cancels the extra output of the first. The network has $2 \cdot 2 + 2 = 6$ hidden parameters and $2 + 1 = 3$ output parameters, 9 in all, and needs $4 + 2 = 6$ MACs. Unlike the perceptron version of Worked Example 4.10, this network uses only ReLU and a linear output, whose slopes are usable, so gradient descent can find such weights by itself. ■

Designing the topology

How many neurons go in each layer? The input layer is fixed by the data: a 64×64 grayscale image gives 4096 inputs, one per pixel, each scaled to lie between 0 and 1. The output layer is fixed by the task: to recognize the digits 0 to 9, use 10 output neurons, one per digit, and pick the one with the largest output. You might ask why not 4 outputs, since 4 bits can encode 10 digits. With one neuron per digit, the "0" neuron only has to collect evidence for a zero from the hidden layer (for example, from the four hidden neurons that detect the four arcs of a zero). With a binary code, the network would have to recognize the digit and then convert it to binary, a harder job. The hidden layers are the art: there is no formula, so you try a few sizes and keep the one that generalizes best.

The classic MNIST network (Figure 4.13) has 784 inputs, one hidden layer of 15 sigmoid neurons, and 10 outputs. Figure 4.14 shows where the 784 numbers come from.

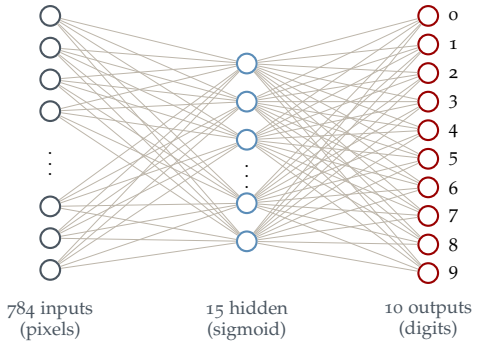


Figure 4.13: The MNIST network of this chapter: 784 inputs, 15 hidden neurons, 10 outputs. Only a few input and hidden nodes are drawn. It has 11 935 weights and biases (Worked Example 4.14).

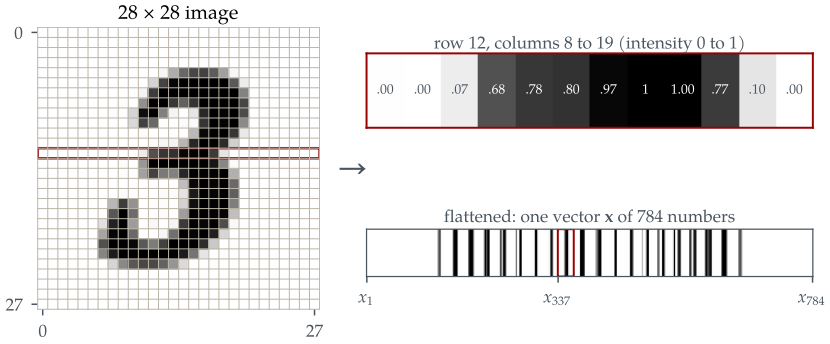


Figure 4.14: An MNIST-style digit. Each of the 28×28 pixels holds an intensity from 0 (background) to 1 (ink); MNIST stores white ink on black, and the figure draws the ink dark. One row is magnified with its values; reading all 28 rows in order gives the input vector $\mathbf{x}$ of 784 numbers.

Softmax: turning outputs into probabilities

For a classifier with K classes, the last layer usually computes K raw scores $z_1, \dots, z_K$ (called *logits*) and converts them to probabilities with the *softmax* function:

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}. \tag{4.9}$$

Every p_k is positive and they add up to 1. The largest score still wins, but now the output also says how sure the network is: the percentage that the Lab 4.2 notebook prints next to its prediction is the largest p_k.

Worked Example 4.14 *Counting weights and biases*

How many parameters does a fully connected network with 3 inputs, 4 hidden neurons, and 2 outputs have? How many does the 784-15-10 MNIST network have? Give the general formula for layer sizes $(n_0, n_1, \dots, n_L)$.

Solution. Each layer from n_{l-1} to n_l neurons has an $n_l \times n_{l-1}$ weight matrix and n_l biases. 3-4-2: $3 \cdot 4 + 4 = 16$ for the hidden layer and $4 \cdot 2 + 2 = 10$ for the output layer, a total of 26. 784-15-10: $784 \cdot 15 + 15 = 11\,775$ and $15 \cdot 10 + 10 = 160$, a total of 11 935. In general,

$$N_{\text{params}} = \sum_{l=1}^{L} \left(n_{l-1}\, n_l + n_l \right), \qquad N_{\text{MAC}} = \sum_{l=1}^{L} n_{l-1}\, n_l .$$

The MNIST network needs 11 910 MACs per image. Almost all the cost is in the first layer, because it has the most inputs. ■

Convolutional layers

A fully connected first layer gives every hidden neuron its own weight for every pixel. If the digit moves by three pixels, a different set of weights sees it. A *convolutional* layer instead slides a small filter, typically 3×3 weights, across the whole image and computes one dot product plus a bias at every position. The same nine weights are used everywhere, so a filter that responds to a short vertical stroke finds that stroke wherever it is. A layer with C_{out} filters produces C_{out} output images, called *channels*; when its input already has C_{in} channels, each filter has $C_{\text{in}} \times 3 \times 3$ weights. With a one-pixel border of zeros around the input (*padding*), the output has the same width and height as the input. A *max pooling* step then keeps the largest value in each 2×2 block, halving the width and height. For $k \times k$ filters and an output of $H \times W$ pixels per channel,

$$N_{\text{params}} = C_{\text{out}} \left(C_{\text{in}} k^2 + 1 \right), \qquad N_{\text{MAC}} = H\, W\, C_{\text{out}}\, C_{\text{in}}\, k^2 . \tag{4.10}$$

Sharing weights makes a convolutional layer small in memory but large in MACs: it has few weights, and each is used hundreds of times. Figure 4.15 shows `SmallCNN`, the network that Lab 4.1 trains and Lab 4.2 runs on camera pictures.

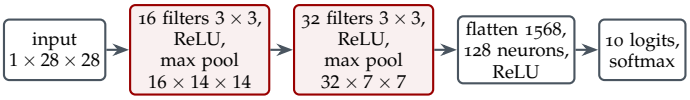


Figure 4.15: `SmallCNN`, the convolutional network of Labs 4.1 and 4.2: two convolutional layers, each followed by ReLU and 2×2 max pooling, then two fully connected layers. It has 206 922 parameters and needs 1.22 million MACs per image (Worked Example 4.15).

Worked Example 4.15 *The cost of SmallCNN*

Count the parameters and MACs of `SmallCNN` (Figure 4.15) layer by layer, and compare with the 784-128-10 network, which has 101 770 parameters and needs 101 632 MACs.

Solution. Use Eq. (4.10) for the convolutions and the formulas of Worked Example 4.14 for the fully connected layers. First convolution ($C_{\text{in}} = 1$, $C_{\text{out}} = 16$, output 28×28): $16(9 + 1) = 160$ parameters and $28 \cdot 28 \cdot 16 \cdot 9 = 112\,896$ MACs. Second convolution ($C_{\text{in}} = 16$, $C_{\text{out}} = 32$, output 14×14 after the first pooling): $32(16 \cdot 9 + 1) = 4640$ parameters and $14 \cdot 14 \cdot 32 \cdot 144 = 903\,168$ MACs. After the second pooling the $32 \times 7 \times 7$ result is flattened into 1568 numbers. The 1568-128 layer has $1568 \cdot 128 + 128 = 200\,832$ parameters and 200 704 MACs; the 128-10 layer has 1290 parameters and 1280 MACs.

Totals: 206 922 parameters and 1 218 048 MACs per image. The CNN has about twice the parameters of 784-128-10 but 12 times the MACs. Its two convolutional layers hold only 2.3% of the parameters but do 83% of the MACs, because each filter weight is used at every pixel position. On the Nano, at 32 million MACs per second (Section 4.9), one image would take 38 ms. ■

4.6 *Learning: Cost Functions and Gradient Descent*

Training means choosing the weights and biases so that the network's output matches the correct answer on many labeled examples. For MNIST, an input $\mathbf{x}$ is a 784-long vector of pixel values, and its label is written as a 10-long *one-hot* vector $\mathbf{y}(\mathbf{x})$ with a 1 in the position of the correct digit: an image of a 1 has $\mathbf{y} = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)$.

The natural goal, "maximize the number of correctly classified images," is a poor thing to optimize. A small change in one weight usually changes no classification at all, so the count is flat almost everywhere and jumps suddenly; it gives no hint about which way to move the weights. We instead define a smooth *cost function* (also called

a loss) that measures how far the outputs are from the labels. The quadratic cost, or mean squared error (MSE), is

$$C(\mathbf{w},\mathbf{b}) = \frac{1}{2n}\sum_{\mathbf{x}} \|\mathbf{y}(\mathbf{x}) - \mathbf{a}(\mathbf{x})\|^2, \tag{4.11}$$

where the sum runs over the n training examples, $\mathbf{a}(\mathbf{x})$ is the network output, and $\|\cdot\|^2$ adds the squares of the entries. C is never negative, and it is zero only if every output equals its label. When C is large, the network is producing garbage.

For classification, the labs use the *cross-entropy* cost, which compares the softmax probabilities of Eq. (4.9) with the one-hot label (in PyTorch it is `nn.CrossEntropyLoss`):

$$C = -\sum_{k=1}^{K} y_k \ln p_k = -\ln p_{\text{correct}}. \tag{4.12}$$

It is 0 when the network puts probability 1 on the correct class and grows without bound as that probability goes to 0. Its advantage over MSE is that it keeps a strong gradient when the network is confidently wrong: with softmax outputs, $\partial C/\partial z_k = p_k - y_k$, whereas with MSE and a saturated sigmoid the gradient is multiplied by a tiny $\sigma'(z)$ and learning stalls. For two classes and a single sigmoid output a (the probability of class 1), the same cost is the *binary cross-entropy*

$$C = -\big[\, y \ln a + (1-y)\ln(1-a) \,\big], \qquad \frac{\partial C}{\partial z} = a - y, \tag{4.13}$$

with label $y = 0$ or 1 (Problem 4.15 asks you to derive the derivative).

Worked Example 4.16 *Softmax and cross-entropy*

A three-class network outputs logits $\mathbf{z} = (2.0,\ 1.0,\ 0.1)$. Find the softmax probabilities, the predicted class, and the cross-entropy cost if the correct class is the first one.

Solution. $e^{2.0} = 7.389$, $e^{1.0} = 2.718$, $e^{0.1} = 1.105$, with sum 11.213. So $\mathbf{p} = (0.659,\ 0.242,\ 0.099)$, which adds to 1. The prediction is class 1 with 65.9% confidence. The cost is $C = -\ln 0.659 = 0.417$. Had the correct class been the third, the cost would be $-\ln 0.099 = 2.32$, much larger, because the network gave that class little probability. ■

Worked Example 4.17 *Quadratic and cross-entropy cost for one image*

A three-class network classifies an image whose correct class is the second, $\mathbf{y} = (0,1,0)$. Compute the quadratic cost $\frac{1}{2}\|\mathbf{y}-\mathbf{a}\|^2$ and the cross-entropy for (a) a good output $\mathbf{a} = (0.1,\ 0.7,\ 0.2)$ and (b) a confidently wrong output $\mathbf{a} = (0.05,\ 0.05,\ 0.90)$.

SOLUTION. (a) Quadratic: $\frac{1}{2}(0.1^2 + 0.3^2 + 0.2^2) = \frac{1}{2}(0.14) = 0.070$. Cross-entropy: $-\ln 0.7 = 0.357$.

(b) Quadratic: $\frac{1}{2}(0.05^2 + 0.95^2 + 0.9^2) = \frac{1}{2}(1.715) = 0.858$. Cross-entropy: $-\ln 0.05 = 3.00$.

Both costs grow, but the quadratic cost can never exceed 1 when both vectors are probabilities, while the cross-entropy keeps growing as the probability of the correct class goes to 0. Its gradient with respect to the logits is simply $\mathbf{p} - \mathbf{y} = (0.05, -0.95, 0.90)$ in case (b): large exactly where the output is wrong, which is what drives learning. ■

Gradient descent

The cost depends on every weight and bias, so think of it as a surface over a space with one axis per parameter; even the small MNIST network has 11 935 axes. We want the lowest point. With two parameters you could plot the surface and look. With thousands, you cannot plot it, and setting all the derivatives to zero and solving is not practical either. But you can still feel the slope where you stand and take a step downhill, like a ball rolling into a valley (Figure 4.16).

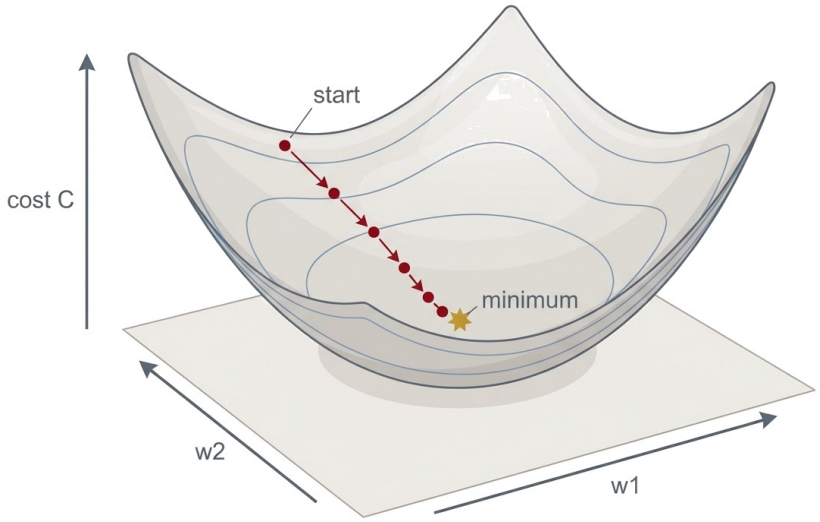


Figure 4.16: Learning as walking downhill. Each step of gradient descent moves the parameters against the local slope of the cost surface, toward a valley where the cost is low.

Collect all the parameters in one vector $\mathbf{v}$. The gradient ∇C is the vector of partial derivatives $\partial C/\partial v_k$; it points in the direction in which C increases fastest. So we step the other way:

$$\mathbf{v} \;\leftarrow\; \mathbf{v} - \gamma\,\nabla C, \tag{4.14}$$

where the *learning rate* γ (also written α or `lr`) sets the step size. For a small step, the change in cost is $\Delta C \approx \nabla C \cdot \Delta\mathbf{v} = -\gamma\|\nabla C\|^2$, which

is negative: each step goes downhill. Repeating the update is *gradient descent*.

KEY EQUATION

$\mathbf{v} \leftarrow \mathbf{v} - \gamma \nabla C$

Gradient descent. γ is the learning rate.

WORKED EXAMPLE 4.18 *Fitting a line by gradient descent*

Fit $a = wx + b$ to the data $x = (0, 1, 2, 3, 4)$, $y = (1, 2.2, 2.0, 3.6, 4.1)$ by minimizing $C = \frac{1}{2n}\sum(a_i - y_i)^2$ with batch gradient descent, starting from $(w, b) = (-0.5, 0.5)$ with learning rate $\gamma = 0.1$. Do the first update by hand and describe where the iteration ends.

SOLUTION. Differentiating the cost gives

$$\frac{\partial C}{\partial w} = \frac{1}{n}\sum_i (a_i - y_i)\, x_i, \qquad \frac{\partial C}{\partial b} = \frac{1}{n}\sum_i (a_i - y_i).$$

At the start, the predictions are $a = (0.5, 0, -0.5, -1, -1.5)$ and the errors $a - y = (-0.5, -2.2, -2.5, -4.6, -5.6)$. The cost is $C = (0.25 + 4.84 + 6.25 + 21.16 + 31.36)/10 = 6.386$. The gradients are $\partial C/\partial w = (0 - 2.2 - 5.0 - 13.8 - 22.4)/5 = -8.68$ and $\partial C/\partial b = -15.4/5 = -3.08$. The update is

$$w = -0.5 - 0.1(-8.68) = 0.368, \qquad b = 0.5 - 0.1(-3.08) = 0.808,$$

and the cost drops to 0.746. The next two steps give $(0.654, 0.912)$ with $C = 0.132$ and $(0.747, 0.948)$ with $C = 0.065$. After about 150 steps the parameters settle at $(w, b) = (0.76, 1.06)$ with $C = 0.0552$, the same line that a least-squares fit gives (Figure 4.17). The last part is slow: after three steps the cost is within 18% of its minimum, but b needs about a hundred further steps to creep from 0.95 to 1.06, because the cost surface is a long, shallow valley in that direction. ■

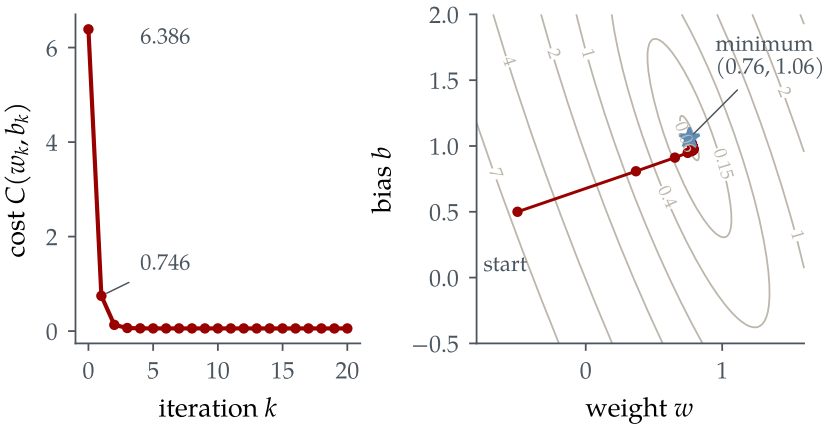


Figure 4.17: Gradient descent for Worked Example 4.18. Left: the cost falls by almost a factor of ten in the first step and then levels off. Right: the path of (w, b) on the contours of C, heading for the minimum.

Choosing the learning rate

The learning rate is the most important setting you choose by hand. Too small, and training crawls. Too large, and each step overshoots the valley, so the parameters bounce from side to side or even fly off. Figure 4.18 shows the three cases for the one-parameter cost $C(w) = w^2 + 4w + 4$. A setting you choose by hand rather than learn, such as γ, the number of hidden neurons, or the number of epochs, is called a *hyperparameter*.

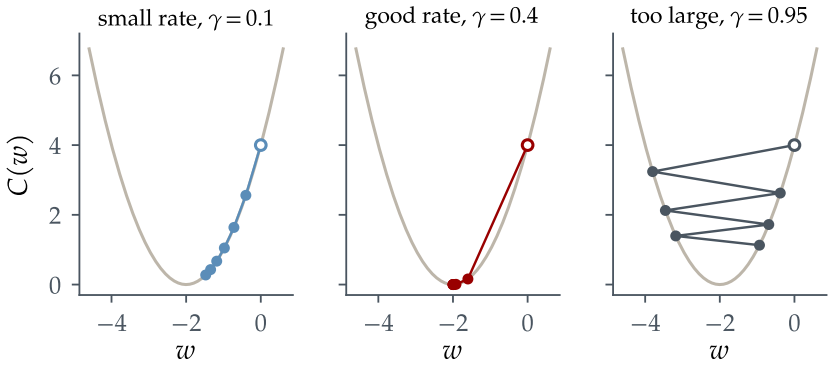


Figure 4.18: Six gradient-descent steps on $C(w) = (w+2)^2$ from $w_0 = 0$ (open circle). A small rate creeps toward the minimum at $w = -2$, a good rate is within 0.1 of it after two steps, and a rate near the stability limit $\gamma = 1$ oscillates across the valley.

Worked Example 4.19 *One gradient-descent step*

For $C(w) = w^2 + 4w + 4$, perform one gradient-descent update from $w_0 = 0$ with $\gamma = 0.1$.

Solution. $dC/dw = 2w + 4$, which is 4 at $w = 0$. So $w_1 = 0 - 0.1 \times 4 = -0.4$. The cost drops from $C(0) = 4$ to $C(-0.4) = 2.56$. The minimum is at $w = -2$, where the derivative is zero. In general each step multiplies the distance to the minimum by $(1 - 2\gamma)$: $\gamma = 0.1$ shrinks it by 0.8 per step, $\gamma = 0.5$ jumps straight to the minimum, and $\gamma > 1$ makes the error grow. ■

Stochastic gradient descent, batches, and epochs

Equation (4.11) averages over all 60 000 training images, so one exact gradient needs a forward and backward pass through every image. That is slow. *Stochastic gradient descent* (SGD) estimates the gradient from a small random *mini-batch* of m examples,

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{\mathbf{x}_j}, \tag{4.15}$$

and updates after each batch. One pass through the whole training set is an *epoch*. Each estimate is noisy, but on average it points downhill,

and the noise can even shake the parameters out of shallow dips in the cost surface. Larger batches give smoother estimates and use the parallel hardware of a GPU better, at the cost of more memory per step.

> **WORKED EXAMPLE 4.20** *How many updates is an epoch?*
>
> Lab 4.1 trains the 784-10 network for 5 epochs with mini-batches of 128, and the 784-128-10 network for 10 epochs with mini-batches of 64, on the 60 000 MNIST training images. How many gradient-descent updates does each run make?
>
> SOLUTION. $60\,000/128 = 468.75$, so an epoch has 469 batches: 468 full batches and a last one of $60\,000 - 468 \times 128 = 96$ images. Five epochs make $5 \times 469 = 2345$ updates. With batches of 64, $60\,000/64 = 937.5$, so an epoch has 938 updates and 10 epochs make 9380. Full-batch gradient descent would make only one update per epoch for the same amount of computation. ■

4.7 *Backpropagation*

Gradient descent needs $\partial C/\partial w$ for every weight in the network. *Backpropagation* computes all of them with one forward pass and one backward pass, using nothing but the chain rule of calculus. The idea: after the forward pass we know the output error. We assign part of that error to each weight feeding the output layer, then pass the error backward to the previous layer in proportion to the weights it went through, and repeat.

For a fully connected network with pre-activations $\mathbf{z}^{(l)}$ and activations $\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)})$, define the error of layer l as $\boldsymbol{\delta}^{(l)} = \partial C/\partial \mathbf{z}^{(l)}$. Then, with $\odot$ meaning entry-by-entry multiplication,

$$\boldsymbol{\delta}^{(L)} = \nabla_{\mathbf{a}} C \odot f'(\mathbf{z}^{(L)}), \tag{4.16}$$

$$\boldsymbol{\delta}^{(l)} = \left(\mathbf{W}^{(l+1)}\right)^{\mathsf{T}} \boldsymbol{\delta}^{(l+1)} \odot f'(\mathbf{z}^{(l)}), \tag{4.17}$$

$$\frac{\partial C}{\partial W^{(l)}_{jk}} = \delta^{(l)}_j \, a^{(l-1)}_k, \qquad \frac{\partial C}{\partial b^{(l)}_j} = \delta^{(l)}_j. \tag{4.18}$$

Equation (4.17) runs the error backward through the transpose of the same weight matrix used in the forward pass. For each layer the backward pass needs that one product with the transposed matrix and one outer product for the weight gradients, each with as many MACs as the forward product, so a training step costs about three times the MACs of an inference. That is why training is affordable even for networks with millions of weights. The gradient for a weight is always "error at its output end times activation at its input end." With a soft-

max output and the cross-entropy cost, or a sigmoid output and the binary cross-entropy, Eq. (4.16) reduces to $\delta^{(L)} = \mathbf{a} - \mathbf{y}$.

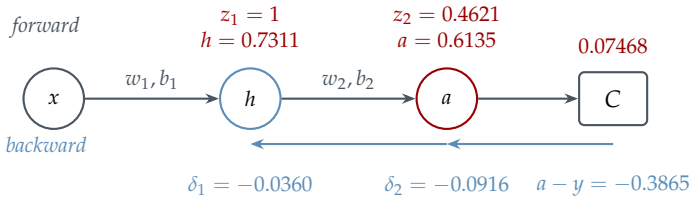


Figure 4.19: The chain network of Worked Example 4.21 with $x = 1$, $y = 1$, $w_1 = 1$, $b_1 = 0$, $w_2 = 2$, $b_2 = -1$. Forward values in red, backward errors in blue.

Worked Example 4.21 *Backpropagation by hand*

A network has one input x, a hidden sigmoid neuron $h = \sigma(w_1 x + b_1)$, and an output sigmoid neuron $a = \sigma(w_2 h + b_2)$, with cost $C = \frac{1}{2}(a - y)^2$ (Figure 4.19). For the training example $x = 1$, $y = 1$ and the parameters $w_1 = 1$, $b_1 = 0$, $w_2 = 2$, $b_2 = -1$, compute all four gradients, then take one gradient-descent step with $\gamma = 1$.

Solution. *Forward pass.* $z_1 = 1(1) + 0 = 1$, $h = \sigma(1) = 0.7311$. $z_2 = 2(0.7311) - 1 = 0.4621$, $a = \sigma(0.4621) = 0.6135$. $C = \frac{1}{2}(0.6135 - 1)^2 = 0.07468$.

Output error. $\partial C/\partial a = a - y = -0.3865$ and $\sigma'(z_2) = a(1 - a) = 0.6135 \times 0.3865 = 0.2371$, so $\delta_2 = -0.3865 \times 0.2371 = -0.09164$. Then

$$\frac{\partial C}{\partial w_2} = \delta_2 \, h = -0.06699, \qquad \frac{\partial C}{\partial b_2} = \delta_2 = -0.09164.$$

Back to the hidden layer. The error reaching h is $\delta_2 w_2 = -0.18328$. Multiply by $\sigma'(z_1) = h(1 - h) = 0.1966$: $\delta_1 = -0.03604$. Then

$$\frac{\partial C}{\partial w_1} = \delta_1 \, x = -0.03604, \qquad \frac{\partial C}{\partial b_1} = \delta_1 = -0.03604.$$

Update with $\gamma = 1$: $w_1 = 1.036$, $b_1 = 0.036$, $w_2 = 2.067$, $b_2 = -0.908$. A new forward pass gives $a = 0.6528$ and $C = 0.06026$, lower than before. All gradients are negative because the output is too small: every parameter moves in the direction that raises a. The gradient of w_1 is smaller than that of w_2 because the error was multiplied by one more sigmoid slope (at most 0.25) on its way back. In deep sigmoid networks this shrinking is the *vanishing gradient* problem, one reason ReLU replaced the sigmoid in hidden layers. ■

Hand calculations are easy to get wrong, so always check them. A gradient can be checked numerically: nudge one parameter by a small

ϵ in each direction and compute $[C(w+\epsilon) - C(w-\epsilon)]/2\epsilon$. The short script below repeats Worked Example 4.21 and checks $\partial C/\partial w_1$ this way; both methods give -0.03604.

```
import numpy as np
sig = lambda z: 1/(1 + np.exp(-z))
x, y = 1.0, 1.0
w1, b1, w2, b2 = 1.0, 0.0, 2.0, -1.0

z1 = w1*x + b1;  h = sig(z1)          # forward pass
z2 = w2*h + b2;  a = sig(z2)
d2 = (a - y) * a*(1 - a)              # dC/dz2
d1 = d2 * w2 * h*(1 - h)              # dC/dz1
print(d2*h, d2, d1*x, d1)             # dC/dw2, dC/db2, dC/dw1, dC/db1

def cost(w1):                         # numerical check of dC/dw1
    return 0.5*(sig(w2*sig(w1*x + b1) + b2) - y)**2
eps = 1e-6
print((cost(w1 + eps) - cost(w1 - eps))/(2*eps))
```

Worked Example 4.22 *Backpropagation through the 2-2-1 network*

The network of Figure 4.12 (ReLU hidden layer, sigmoid output) is trained with the binary cross-entropy of Eq. (4.13). For the input $\mathbf{x} = (1, 0.5)$ with label $y = 1$, find all the gradients with Eqs. (4.17) and (4.18), and take one step with $\gamma = 0.1$.

Solution. *Forward pass* (Worked Example 4.12): $\mathbf{z}^{(1)} = (0.5, 0.75)$, $\mathbf{h} = (0.5, 0.75)$, $z^{(2)} = -0.5$, $a = 0.3775$, so $C = -\ln 0.3775 = 0.974$.

Output layer. $\delta^{(2)} = a - y = -0.6225$. Then $\partial C/\partial \mathbf{W}^{(2)} = \delta^{(2)}\mathbf{h}^{\mathsf{T}} = (-0.3112, -0.4668)$ and $\partial C/\partial b^{(2)} = -0.6225$.

Hidden layer. Send the error back through the transposed weights, $(\mathbf{W}^{(2)})^{\mathsf{T}}\delta^{(2)} = (1, -2)^{\mathsf{T}}(-0.6225) = (-0.6225, 1.2449)$, and multiply by the ReLU slopes, both 1 because both $z_j^{(1)} > 0$: $\boldsymbol{\delta}^{(1)} = (-0.6225, 1.2449)$. The weight gradient is the outer product with the input,

$$\frac{\partial C}{\partial \mathbf{W}^{(1)}} = \boldsymbol{\delta}^{(1)}\mathbf{x}^{\mathsf{T}} = \begin{pmatrix} -0.6225 & -0.3112 \\ 1.2449 & 0.6225 \end{pmatrix}, \qquad \frac{\partial C}{\partial \mathbf{b}^{(1)}} = \boldsymbol{\delta}^{(1)}.$$

The second hidden neuron gets a positive gradient: it reaches a through the negative weight -2, so making it smaller raises a.

Update with $\gamma = 0.1$: $\mathbf{W}^{(1)} = \begin{pmatrix} 0.5622 & -0.9689 \\ 1.3755 & 0.4378 \end{pmatrix}$, $\mathbf{b}^{(1)} = (0.5622, -1.1245)$, $\mathbf{W}^{(2)} = (1.0311, -1.9533)$, $b^{(2)} = 0.5622$. A new forward pass gives $\mathbf{h} = (0.6401, 0.4699)$, $a = 0.5755$, and $C = 0.552$: one step moved the output from 0.38 to 0.58, toward the label. ■

In PyTorch you never write these equations yourself. Three lines in the lab's training loop do all the work for every batch:

`optimizer.zero_grad()` clears the old gradients, `loss.backward()` runs backpropagation to fill in $\partial C/\partial w$ for every parameter, and `optimizer.step()` applies Eq. (4.14).

4.8 Training, Testing, and Judging a Classifier

The MNIST dataset of handwritten digits is the standard first problem for neural networks. It has 60 000 training images and 10 000 test images, each 28×28 pixels in grayscale (Figure 4.20), scanned from samples written by a few hundred people (Census Bureau employees and high-school students). The test images come from different writers than the training images, so a good test score means the network learned digits, not people.



Figure 4.20: The MNIST format, shown for the ten digits: white strokes on black, each digit scaled into a 20×20 box and centered in a 28×28 frame.

Training, validation, and test sets

A model that has seen an image during training will usually classify it correctly, so its accuracy on training data says little about new data. The *holdout method* splits the dataset at random into three parts (Figure 4.21). The *training set* (about 60%) is used to update the weights. The *validation set* (about 20%) checks the model on unseen data while you adjust hyperparameters such as the learning rate or the number of hidden neurons. The *test set* (the last 20%) is used once, at the end, for the final score. If you tune your design by looking at the test set, it quietly becomes a second validation set and your reported accuracy is too optimistic. When data are scarce, k-fold cross-validation splits the data into k equal parts, trains k times, each time validating on a different part and training on the other $k-1$, and averages the results.

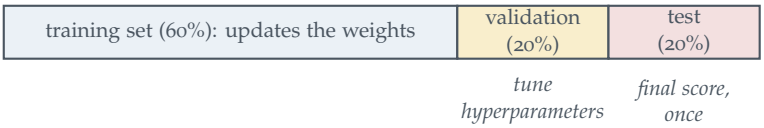


Figure 4.21: The holdout method: one random split of the labeled data into three sets with different jobs.

Underfitting and overfitting

Plot the accuracy (or the cost) on the training and validation sets after every epoch. The two curves tell you what is wrong with a model. An *underfit* model is too simple to capture the patterns: both accuracies stay low and close together. A model with a *good fit* reaches high accuracy on both. An *overfit* model has memorized details of the training set, including noise: its training accuracy keeps rising while the validation accuracy stalls or falls, and the gap between them grows.

Figure 4.22 shows both behaviors in two training runs on all 60 000 MNIST images. The 784-10 network with no hidden layer stops at about 91% on both sets: there is a limit to what a single linear layer can do. The 784-512-512-10 network trained for 50 epochs reaches 99.6% on the training set but stays near 97.9% on the test set; after about epoch 20, more training only widens the gap. To fix underfitting, train longer, add features, or use a larger model (more neurons or layers). To fix overfitting, get more data, stop training when the validation cost or accuracy stops improving (*early stopping*), use a smaller model, or add regularization such as *dropout*, which switches off random neurons during training so that no single neuron can memorize an example. Lab 4.1 shows the same effect faster by training the large network on only 5000 images, where the gap is wider.

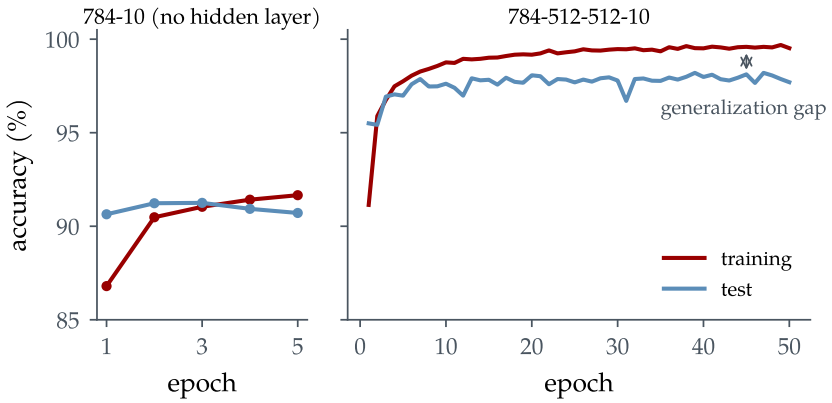


Figure 4.22: Training and test accuracy of two networks trained on the full MNIST training set. Left: a model too simple to improve past about 91%. Right: a large model whose training accuracy approaches 100% while test accuracy levels off near 98%.

Worked Example 4.23 *Choosing the epoch to stop*

The 784-512-512-10 network of Lab 4.1, Part F (Adam, learning rate 0.001, batches of 128) is trained for 30 epochs on 5000 MNIST images. The table gives one seeded run of the lab notebook at some epochs: the training accuracy (counted during each epoch) and the accuracy and cross-entropy cost on the 10 000 test images, which play the role of the validation set, as in the lab. Runs differ a little, so your numbers may be off by up to a point.

epoch	training (%)	test (%)	test cost
1	70.3	85.7	0.4625
3	91.1	89.0	0.3451
5	92.9	88.5	0.3646
8	95.8	91.4	0.2777
10	96.1	91.0	0.2942
14	98.7	94.0	0.2190
15	99.0	93.8	0.2195
20	99.6	93.9	0.2458
24	100	94.5	0.2372
30	100	94.6	0.2511

(a) Which epoch would early stopping keep if it watches the test cost, and which if it watches the test accuracy? (b) How large is the gap between training and test accuracy at the end? (c) What happens to the test cost after its best epoch, and what does it mean?

SOLUTION. (a) The test cost is lowest, 0.2190, at epoch 14, where the test accuracy is 94.0%. Early stopping on the cost keeps the weights from epoch 14 and saves the last 16 of 30 epochs, 53% of the training time. The test accuracy keeps creeping up instead and peaks at the last epoch, 94.6% at epoch 30: only 0.6 points more for twice the training. The two criteria pick different epochs. This example uses the cost, the usual choice, because it responds to every change in the network's confidence, while the accuracy changes only when a prediction flips.

(b) The training accuracy reaches 100% at epoch 24. At the end the gap is $100 - 94.6 = 5.4$ percentage points: the network has memorized its 5000 training images.

(c) It rises from 0.2190 to 0.2511, 15% higher, while the test accuracy gains only 0.6 points. The network keeps getting more confident on the training images, and it also grows more confident in its wrong answers on new images, which the cross-entropy punishes. The cost shows overfitting before the accuracy does. ■

The confusion matrix

A single accuracy number hides which classes get mixed up. A *confusion matrix* has one row per actual class and one column per predicted class; the entry in row i, column j counts the validation examples of class i that the model called class j. Correct answers lie on the diagonal. Figure 4.23 shows the gesture classifier on 1000 validation windows. Its total accuracy is the diagonal sum over the total, $(205 + 199 + 223 + 186)/1000 = 0.813$.

For one class at a time, call that class *positive* and all others *negative*. A *true positive* (TP) is a positive predicted positive; a *true negative* (TN)

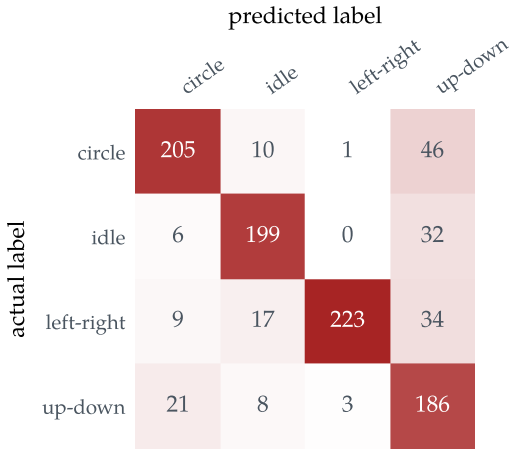


Figure 4.23: Confusion matrix of the 4-class gesture classifier (1000 validation windows). The idle class is used in Worked Example 4.24.

a negative predicted negative; a *false positive* (FP, a "type I error") a negative predicted positive; a *false negative* (FN, "type II") a positive predicted negative. From these four counts,

$$\text{sensitivity} = \frac{TP}{TP+FN}, \qquad \text{specificity} = \frac{TN}{TN+FP},$$
$$\text{precision} = \frac{TP}{TP+FP}, \qquad F_1 = \frac{2\,TP}{2\,TP+FP+FN}. \tag{4.19}$$

Sensitivity is also called the true positive rate (TPR), recall, or hit rate; specificity is the true negative rate (TNR), or selectivity; precision is the positive predictive value (PPV). Sensitivity asks "of the real positives, how many did we catch?"; precision asks "of the ones we flagged, how many were right?". The F_1 score is the harmonic mean of the two, $F_1 = 2PR/(P+R)$, and is high only if both are.

Worked Example 4.24 *Metrics for one class*

From Figure 4.23, compute the accuracy, sensitivity, specificity, precision, and F_1 score for the class *idle*.

Solution. Row "idle" holds the 237 actual idle windows: $TP = 199$ and $FN = 6+0+32 = 38$. Column "idle" holds 234 idle predictions, so $FP = 10+17+8 = 35$. The rest are true negatives: $TN = 1000-199-38-35 = 728$.

$$\begin{aligned}
\text{accuracy} &= (199+728)/1000 = 0.927,\\
\text{sensitivity} &= 199/(199+38) = 0.840,\\
\text{specificity} &= 728/(728+35) = 0.954,\\
\text{precision} &= 199/(199+35) = 0.850,\\
F_1 &= 2(199)/(2(199)+35+38) = 0.845.
\end{aligned}$$

The same steps give $F_1 = 0.815$, 0.875, and 0.721 for circle, left-right,

and up-down. Up-down has the worst score because many circles (46) and left-right gestures (34) were predicted as up-down, so its precision is only $186/298 = 0.624$. ■

Worked Example 4.25 *When 97% accuracy is bad*

A second gesture dataset has 6 circle, 973 idle, 11 left-right, and 10 up-down windows. A "model" predicts idle for every input. Find its accuracy and its F_1 scores.

Solution. It is right on all 973 idle windows and wrong on the other 27, so its accuracy is $973/1000 = 0.973$. For idle, $TP = 973$, $FP = 27$, $FN = 0$, so $F_1 = 1946/(1946 + 27) = 0.986$. For the other three classes $TP = 0$, so $F_1 = 0$. The average F_1 over the four classes is 0.247. The high accuracy only reflects the imbalance of the data. An image example is the same: a classifier that always says "field" is 99% accurate on a dataset that is 98.9% field photos and useless for finding dogs. Low per-class F_1 scores expose this. Fixes include collecting more minority examples, resampling, or weighting the cost of minority-class errors more heavily. ■

Worked Example 4.26 *A fall detector*

A wrist-worn fall detector is tested on 10 000 two-second windows of recorded activity, 50 of which contain a real fall. It flags 75 windows, and 45 of those are real falls. Find TP, FP, FN, TN, the accuracy, sensitivity, specificity, precision, and F_1. How many false alarms would a user see per day of wear?

Solution. $TP = 45$, $FP = 75 - 45 = 30$, $FN = 50 - 45 = 5$, and $TN = 10\,000 - 45 - 30 - 5 = 9920$. Accuracy $= (45 + 9920)/10\,000 = 0.9965$; sensitivity $= 45/50 = 0.90$; specificity $= 9920/9950 = 0.9970$; precision $= 45/75 = 0.60$; $F_1 = 90/(90 + 30 + 5) = 0.72$.

The detector misses one fall in ten, and two of every five alarms are false. The specificity sounds excellent, but a day has $86\,400/2 = 43\,200$ two-second windows, and a false-alarm rate of $30/9950 = 0.30\%$ per non-fall window gives about 130 false alarms per day. For rare events, precision and the false-alarm rate per day matter far more than accuracy. ■

4.9 *Hardware for Neural Networks*

A TRAINED NETWORK IS A LIST OF NUMBERS and a recipe of multiply-accumulates. Whether it fits on a microcontroller comes down to three questions: how many bytes the weights need, how many MACs one inference needs, and how much energy those MACs and memory accesses cost.

Memory, MACs, and latency

Each parameter stored as a 32-bit floating-point number takes 4 bytes.[2] The MAC count follows from Worked Example 4.14. The Cortex-M4 in the Nano takes about 2 clock cycles per MAC in a simple loop, so at 64 MHz it does roughly 32 million MACs per second. Table 4.1 puts the networks of this chapter side by side.

[2] Here 1 kB = 1024 bytes. The nRF52840 has 1024 kB of flash for code and constant weights and 256 kB of RAM for variables.

Table 4.1: Cost of one inference for the networks used in this chapter. Memory is for the weights and biases only.

network	parameters	MACs	float32 (kB)	int8 (kB)	time at 32 M MAC/s
33-20-10-4 (gestures)	934	900	3.6	0.9	28 µs
784-10	7 850	7 840	30.7	7.7	0.25 ms
784-15-10	11 935	11 910	46.6	11.7	0.37 ms
784-128-10	101 770	101 632	397.5	99.4	3.2 ms
784-512-512-10	669 706	668 672	2616	654	21 ms
`SmallCNN`	206 922	1 218 048	808.3	202.1	38 ms

Real-time sensing usually asks for an answer within 5 to 10 ms, so the small fully connected networks are fast enough, while the largest one and the CNN are not. Memory is the tighter limit: the 784-128-10 network needs 398 kB in floating point, more than the whole RAM, and the CNN needs twice that. This is why the camera lab sends pictures to the laptop for classification and why models for microcontrollers are made small and stored as 8-bit integers.

> **WORKED EXAMPLE 4.27** *The smallest MNIST network*
>
> Design the smallest network with one hidden layer that classifies the MNIST test set with at least 85% accuracy, and count the floating-point operations (multiplications and additions) needed for one prediction.
>
> SOLUTION. The input (784 pixels) and output (10 digits) are fixed, so the only choice is the hidden size H. In Lab 4.1 (Part F) you train `MLP([784, H, 10])` for $H = 2, 4, 8, 16, 32$ with ReLU, the Adam optimizer at learning rate 0.001, batches of 64, and 5 epochs, and record the test accuracy of each. A representative sweep on 20 000 training images gives about 28%, 57%, 86%, 91%, and 91% (your numbers will differ by a few percent from run to run). The smallest

hidden layer that reaches 85% is $H = 8$, which reaches about 90% when trained on all 60 000 images. Its margin is small; $H = 16$ is the safer design at about twice the cost. The 784-8-10 network, with ReLU and 10 linear outputs (the softmax is only inside the cost), has $784 \times 8 + 8 + 8 \times 10 + 10 = 6370$ parameters.

Operation count. Each output of a layer with n inputs needs n multiplications and n additions ($n - 1$ to add the products plus one for the bias). Hidden layer: $784 \times 8 = 6272$ multiplications and 6272 additions. Output layer: $8 \times 10 = 80$ of each. Total: 6352 multiplications plus 6352 additions, or 12 704 floating-point operations per image (plus 8 ReLU comparisons). For comparison, the lab's 784-128-10 network needs $2 \times 101\,632 = 203\,264$, sixteen times more, for about six more points of accuracy. ■

Worked Example 4.28 *Will it fit on the Nano?*

Lab 4.1 trains a 784-128-10 network in float32. Find its weight memory, MAC count, and inference time on the Nano's Cortex-M4 (64 MHz, 2 cycles per MAC), and decide whether it fits in the 256 kB of RAM.

Solution. Parameters: $784 \times 128 + 128 + 128 \times 10 + 10 = 101\,770$. In float32: $4 \times 101\,770 = 407\,080$ bytes $= 397.5$ kB, too large for 256 kB of RAM (it could live in the 1 MB flash, which is slower to read). MACs: $784 \times 128 + 128 \times 10 = 101\,632$. Throughput is $64 \times 10^6/2 = 32 \times 10^6$ MAC/s, so one image takes $101\,632/(32 \times 10^6) = 3.18$ ms, fast enough. Quantized to int8 (next subsection) the weights take 101 770 bytes = 99.4 kB and fit. ■

Quantization to 8 bits

Neural networks tolerate small errors in their weights remarkably well, so we can store each weight in fewer bits. In symmetric 8-bit quantization, choose a scale s and store the integer

$$q = \mathrm{round}(w/s), \qquad -128 \le q \le 127, \qquad w \approx s\,q. \tag{4.20}$$

A common choice is $s = \max|w|/127$, so the largest weight maps to ± 127. The rounding error is at most $s/2$. Quantization cuts memory by a factor of 4 compared with float32, and integer MACs are cheaper in time and energy than floating-point ones. Frameworks such as TensorFlow Lite for Microcontrollers run such 8-bit models on chips like the nRF52840. A related trick, *pruning*, removes weights near zero: deleting 75% of the parameters cuts memory, and roughly the energy per inference, by about 75% (a little less, because the positions of the remaining weights must be stored too).

WORKED EXAMPLE 4.29 *Quantizing three weights*

Quantize $\mathbf{w} = (0.10, -0.50, 0.75)$ to signed 8-bit integers using a full-scale range of ± 1 (that is, $s = 1/127$), then dequantize. Repeat with $s = \max|w|/127$.

SOLUTION. With $s = 1/127 = 0.007874$: $w/s = (12.7, -63.5, 95.25)$, which rounds to $q = (13, -64, 95)$. Dequantizing gives $sq = (0.1024, -0.5039, 0.7480)$, errors of at most $0.0039 \approx s/2$. With $s = 0.75/127 = 0.005906$: $w/s = (16.93, -84.67, 127)$, so $q = (17, -85, 127)$ and $sq = (0.1004, -0.5020, 0.7500)$. The errors are now at most 0.002 because the 256 levels cover only the range the weights actually use (Figure 4.24). Each weight now takes 1 byte instead of 4. ■

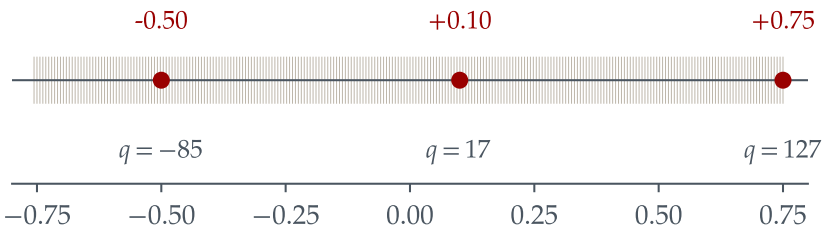


Figure 4.24: Symmetric 8-bit quantization with $s = 0.75/127$. Every gray tick is a representable value; each weight (red) is stored as the integer of its nearest tick.

WORKED EXAMPLE 4.30 *An 8-bit dot product*

A neuron has weights $\mathbf{w} = (0.50, -0.25, 0.75)$ and inputs $\mathbf{x} = (0.2, 0.6, -0.4)$. Quantize both to 8 bits with Eq. (4.20), each with its own scale $s = \max|\cdot|/127$, compute the dot product in integers, and compare with the exact value.

SOLUTION. Weights: $s_w = 0.75/127 = 0.005906$; $\mathbf{w}/s_w = (84.67, -42.33, 127)$, so $\mathbf{q}_w = (85, -42, 127)$. Inputs: $s_x = 0.6/127 = 0.004724$; $\mathbf{x}/s_x = (42.33, 127, -84.67)$, so $\mathbf{q}_x = (42, 127, -85)$. The processor now does three integer MACs: $85(42) + (-42)(127) + 127(-85) = 3570 - 5334 - 10\,795 = -12\,559$. Each product fits in 16 bits, and the running sum is kept in a 32-bit register so that it cannot overflow. One multiply at the end restores the scale: $s_w s_x = 2.790 \times 10^{-5}$, and $-12\,559 \times 2.790 \times 10^{-5} = -0.3504$. The exact value is $0.1 - 0.15 - 0.3 = -0.35$, an error of 0.11%. Only the final rescaling needs floating point; the hundreds of MACs of a real neuron are all integer operations. ■

Energy per operation

In CMOS logic, every time a node of capacitance C is charged to the supply V and discharged, the energy CV^2 is spent. A digital MAC switches many such nodes, and moving each weight from memory to the multiplier costs even more than the arithmetic (Table 4.2).[3] This is why low-power designs lower the supply voltage (near-threshold operation), shrink the data (quantization and pruning), keep the weights on the chip, or wake the processor only when there is something to classify (duty cycling).

[3] These widely quoted figures are from M. Horowitz, "Computing's energy problem (and what we can do about it)," ISSCC 2014; the memory rows are the 32-bit values derived from it by S. Han et al. (ISCA 2016). Newer chips use less energy per operation, but the ratios are similar.

Table 4.2: Approximate energy per operation in a 45-nanometer chip. A MAC is a multiply plus an add to the running sum: about 0.3 pJ for an 8-bit multiply with a 32-bit integer add (the sum is kept in 32 bits) and 4.6 pJ in 32-bit floating point.

operation	energy (pJ)
8-bit integer add	0.03
32-bit integer add	0.1
8-bit integer multiply	0.2
32-bit floating-point add	0.9
32-bit floating-point multiply	3.7
read 32 bits from a small on-chip SRAM cache	5
read 32 bits from off-chip DRAM	640

Worked Example 4.31 *Where the energy of an inference goes*

Use Table 4.2 to estimate the energy of one inference of the 784-128-10 network (101 632 MACs, 101 770 parameters) when (a) everything is float32 and each parameter is read once from on-chip SRAM, (b) everything is int8, with four weights packed in each 32-bit read, and (c) the float32 weights must come from off-chip DRAM.

Solution. (a) Arithmetic: $101\,632 \times 4.6\,\text{pJ} = 467.5\,\text{nJ}$. Memory: $101\,770 \times 5\,\text{pJ} = 508.9\,\text{nJ}$. The total is 976 nJ, about 1 µJ, and more than half of it is spent moving weights.

(b) Arithmetic, with 8-bit multiplies and 32-bit sums: $101\,632 \times 0.3\,\text{pJ} = 30.5\,\text{nJ}$. Memory: $101\,770/4 = 25\,443$ reads of 5 pJ, 127.2 nJ. The total is 157.7 nJ, 6.2 times less than (a), and memory is now 81% of it.

(c) The reads alone take $101\,770 \times 640\,\text{pJ} = 65.1\,\mu\text{J}$, 67 times the whole of (a).

Quantization saves energy mostly by shrinking the data that must move, and keeping the weights on the chip matters even more. At 10 inferences per second, design (a) averages 9.8 µW and design (b) 1.6 µW. ■

Worked Example 4.32 *Energy per MAC and duty cycling*

(a) A digital MAC has an effective switched capacitance of 1 pF and an analog MAC 0.1 pF, both at 1 V. Find the energy per MAC

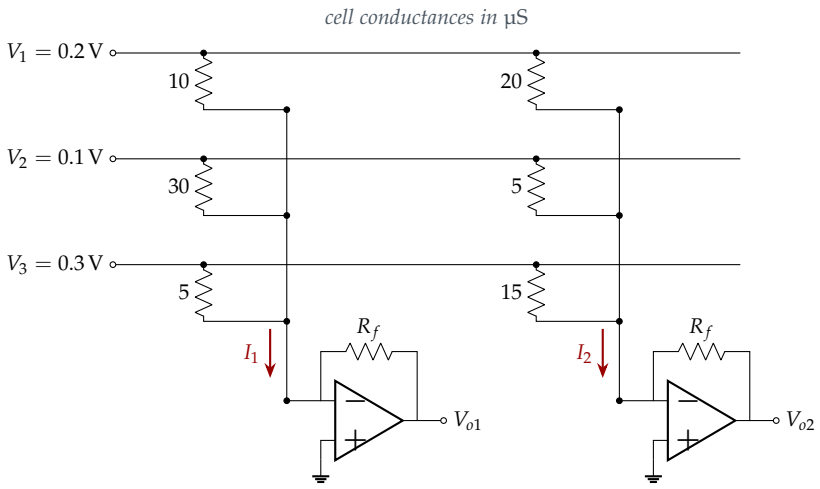


Figure 4.25: A 3×2 resistive crossbar. Input voltages drive the rows; each cell's conductance is a weight. Each column current is the dot product of the inputs with that column's conductances, and a transimpedance amplifier ($R_f = 10\,\text{k}\Omega$) holds the column at virtual ground and converts the current to a voltage. Values for Worked Example 4.33.

and per inference of the 784-128-10 network. (b) An edge sensor wakes for 5 ms every 100 ms, drawing 20 mW when active and 0.1 mW asleep. Find its average power.

SOLUTION. (a) $E = CV^2$: digital $10^{-12} \times 1^2 = 1\,\text{pJ}$, analog 0.1 pJ. One inference has 101 632 MACs, so $101\,632 \times 1\,\text{pJ} = 102\,\text{nJ}$ digital and 10.2 nJ analog. (b) The average weights each power by the fraction of time spent in it:

$$P_{\text{avg}} = \frac{(5\,\text{ms})(20\,\text{mW}) + (95\,\text{ms})(0.1\,\text{mW})}{100\,\text{ms}} = 1.095\,\text{mW},$$

about 18 times less than staying awake. ■

Analog in-memory computing with a resistive crossbar

A digital processor fetches each weight from memory, multiplies, and adds, one MAC at a time. A resistive crossbar does all the MACs of a layer at once, inside the memory, using the circuit laws of Chapter 1 (Figure 4.25). Each weight is stored as the conductance $G = 1/R$ of a programmable resistor (a memristor, a phase-change cell, or a flash transistor) sitting where a horizontal row wire crosses a vertical column wire. The inputs are applied as voltages V_i on the rows. Each column is held at 0 V (a virtual ground) by a transimpedance amplifier, the same op-amp current-to-voltage converter that reads the photodiode in Chapter 3.

Ohm's law gives the current through the cell in row i and column j as $G_{ji}V_i$, the product of a weight and an input. Kirchhoff's current law at the column wire says the currents add:

$$I_j = \sum_i G_{ji}\,V_i \qquad \Longleftrightarrow \qquad \mathbf{I} = \mathbf{G}\,\mathbf{V}. \tag{4.21}$$

This is Eq. (4.3) computed by physics: every multiply happens in a resistor and every add happens on a wire, all at the same time, and no weight ever moves. The amplifier output $V_{oj} = -R_f I_j$ is then digitized by an ADC, or passed to a circuit that applies the activation function. Conductances cannot be negative, so a signed weight uses two cells on two columns, $W = (G^+ - G^-)/G_0$, and the difference of the two column currents is the result.

KEY IDEA A crossbar computes $\mathbf{I} = \mathbf{GV}$ in one step: Ohm's law multiplies, KCL adds.

The catch is precision. Device conductances vary from cell to cell and drift with time and temperature, wires have resistance, and the ADCs at the column ends cost area and power. Analog arrays therefore need calibration and work best for inference with a few bits of precision. A related approach, *neuromorphic* or *spiking* hardware, imitates the brain's event-driven style: neurons communicate with brief pulses and circuits switch only when an input changes, so a quiet sensor costs almost no dynamic power. These approaches aim at energies well below a picojoule per MAC (Figure 4.26).

WORKED EXAMPLE 4.33 *Matrix-vector product with a crossbar*

For the crossbar of Figure 4.25 (conductances in µS), find the column currents, the amplifier outputs with $R_f = 10\,\text{k}\Omega$, and the power dissipated in the array.

SOLUTION. Column 1 has conductances $(10, 30, 5)$ and column 2 has $(20, 5, 15)$, from top to bottom. With $\mathbf{V} = (0.2, 0.1, 0.3)$ V and the columns at virtual ground, each cell sees its full row voltage:

$$I_1 = 0.2(10) + 0.1(30) + 0.3(5) = 2 + 3 + 1.5 = 6.5\,\mu\text{A},$$
$$I_2 = 0.2(20) + 0.1(5) + 0.3(15) = 4 + 0.5 + 4.5 = 9.0\,\mu\text{A}.$$

(Volts times microsiemens gives microamperes.) The outputs are $V_{o1} = -(10\,\text{k}\Omega)(6.5\,\mu\text{A}) = -65\,\text{mV}$ and $V_{o2} = -90\,\text{mV}$. The power is $\sum GV^2$ over the six cells: $0.2^2(30) + 0.1^2(35) + 0.3^2(20) = 1.2 + 0.35 + 1.8 = 3.35\,\mu\text{W}$. The array did six MACs in a single settling time of the circuit. ■

WORKED EXAMPLE 4.34 *Sizing a crossbar*

A 128×128 crossbar stores weights as conductances from 1 to 50 µS and applies inputs from 0 to 0.2 V. (a) Find the largest current one column can carry, and the largest feedback resistor R_f that keeps the amplifier output within 1 V. (b) If one read of the whole array takes 100 ns, how many MACs per second is that, compared with the Nano? (c) Find the power and the energy per MAC in the worst case, every cell at 50 µS and every input at 0.2 V.

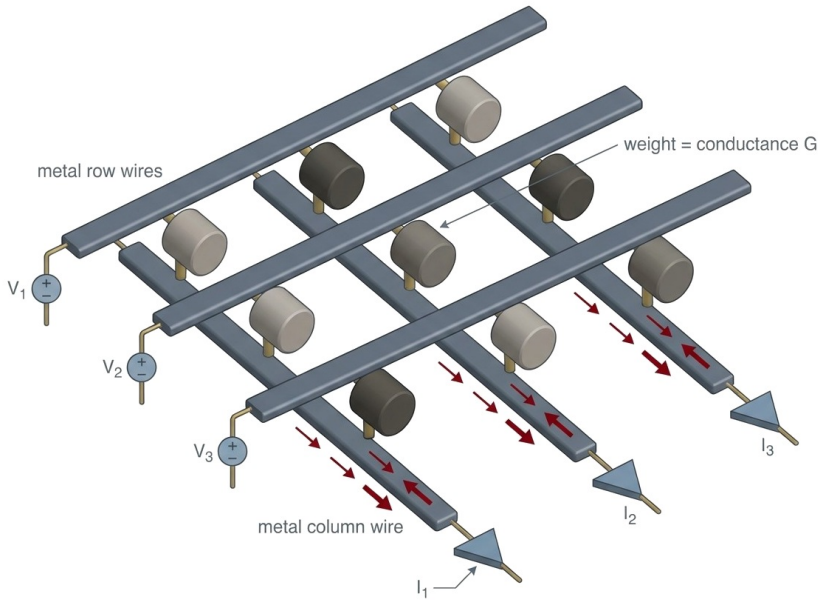


Figure 4.26: A crossbar array in three dimensions. Input voltages on the rows, one resistive memory cell at every crossing, and column currents that sum the products and flow into the sense amplifiers.

SOLUTION. (a) With all 128 cells of a column at full conductance and full voltage, $I_{\max} = 128 \times 50\,\mu\text{S} \times 0.2\,\text{V} = 1.28\,\text{mA}$. The output magnitude is $R_f I$, so $R_f \le 1\,\text{V}/1.28\,\text{mA} = 781\,\Omega$.

(b) One read does $128 \times 128 = 16\,384$ MACs, so $16\,384/100\,\text{ns} = 1.64 \times 10^{11}$ MACs per second, 5120 times the 32 million of the Nano's processor.

(c) $P = \sum GV^2 = 16\,384 \times 50\,\mu\text{S} \times (0.2\,\text{V})^2 = 32.8\,\text{mW}$. Over 100 ns that is 3.28 nJ, or $3.28\,\text{nJ}/16\,384 = 0.20\,\text{pJ}$ per MAC, below the 0.3 pJ of an 8-bit digital MAC with a 32-bit sum (Table 4.2) even in this worst case, and with no weight ever moved. The amplifiers and ADCs at the column ends, not counted here, often cost more than the array itself. ■

4.10 *Laboratory: Neural Networks in Python*

Lab 4.1 at a glance

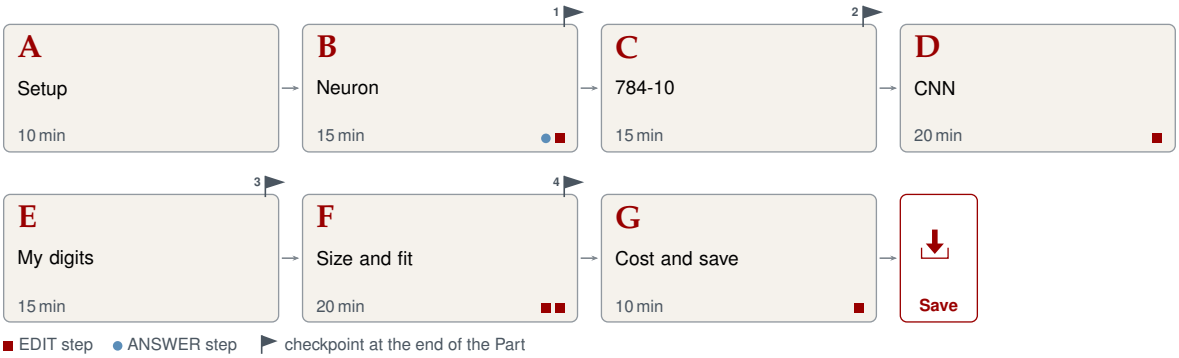
Goal: build a neuron and perceptrons in PyTorch, train three networks that read handwritten digits, test them on your own handwriting, watch a network overfit, and save the best network for Lab 4.2.
Hardware: laptop with Python and PyTorch; a thick dark pen, white paper, and a phone camera. No demo board.
Notebook: `lab4.1-neural-networks.ipynb` in the folder `lab4.1-neural-networks` of the lab kit, with two sample photos of a handwritten 5. The MNIST digits come with the kit (folder `data/mnist`); nothing is downloaded.
Before the lab: run the first cell (**A1**) once: it checks that PyTorch is installed. Write at least three digits with a thick dark pen on white paper and photograph them as described in **E2**.
Time: about 1 h 45 min. All the training in the notebook takes 2 to 3 minutes on a laptop CPU.
You hand in: `lab4.1_submission.json` on the course portal; keep your notebook with your answers, and keep the saved network `data/models/mnist_cnn.pt` for Lab 4.2.

Key takeaways

- One perceptron draws one straight line: it can be AND, OR, NAND, or NOR, never XOR. Two layers of perceptrons make XOR.
- Training repeats four lines per mini-batch of 128: zero the gradients, compute the cost, backpropagate, step (469 per epoch).
- Accuracy costs computation: 784-10 reads about 91 % of digits with 7840 MACs, the CNN 98.5 % with 1.2 million.
- A network only works on inputs prepared like its training data: unframed, a photo of a 5 is read as a 7 with 98 % confidence.
- On 5000 images a large network reaches 100 % training but about 94.6 % test accuracy. Stop at the lowest test cost.

Board for Lab 4.1

No board today. This lab runs entirely on your laptop. Leave the demo board unplugged; its selector and jumpers do not matter. Lab 4.2 uses the board with the camera.



Objectives

- Implement a neuron with trainable weights, and build logic gates from perceptrons.
- Train a 784-10 network, a 784-128-10 network, and a small convolutional network on MNIST with the cross-entropy cost.
- Classify photos of your own handwritten digits, and read a confusion matrix.
- Find the smallest hidden layer that works, tell underfitting from overfitting, and count the parameters and MACs of each network.

Background

The notebook follows this chapter: the neuron and its activations (Section 4.3), perceptrons (Section 4.4), mini-batches, cross-entropy, and gradient descent (Section 4.6), the confusion matrix and overfitting (Section 4.8), and the cost of each network (Section 4.9). The function `ee105.datasets.load_mnist` reads the 60 000 training and 10 000 test digits from the kit. Each pixel is scaled to $[0, 1]$ and then normalized with $(x - 0.5)/0.5$, which maps it to $[-1, 1]$. Every image you give these networks later, your photos today and the camera pictures in Lab 4.2, must get exactly the same scaling.

All networks are trained by one helper, `train()`: the four-line training step of this chapter in a loop over batches and epochs, plus a test on the 10 000 test images after every epoch. The hidden-layer networks are built by a small class `MLP` from a list of layer sizes; its `forward` method first lays out each picture as a vector of 784 inputs (Fig. 4.27). The key lines are:

```
class MLP(nn.Module):                # MLP([784, 128, 10]): 784-128-10
    def __init__(self, sizes):
        super().__init__()
        self.layers = nn.ModuleList([nn.Linear(a, b)
                          for a, b in zip(sizes[:-1], sizes[1:])])
```

No download, no torchvision. The notebook reads MNIST from the kit. Every training run is sized for a laptop CPU, and each long run says how to train longer if you have time.

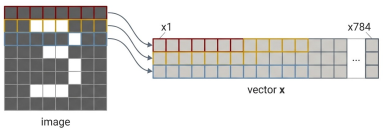


Figure 4.27: What `x.view(-1, 28*28)` does in `forward`: the rows of each 28 × 28 picture are laid end to end as one vector of 784 inputs (Figure 4.4).

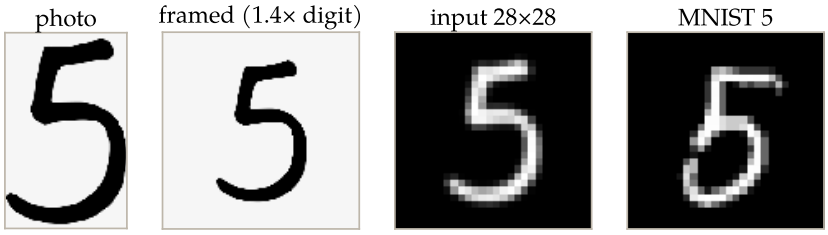


Figure 4.28: A photo becomes an MNIST-style input: the photo (`picture5.png`), the square that `frame_digit` cuts around the ink, and the 28×28 input that `to_digit` makes of it. Right: a real MNIST test digit for comparison.

```
    def forward(self, x):
        x = x.view(-1, 28*28)              # flatten 28x28 -> 784
        for layer in self.layers[:-1]:
            x = torch.relu(layer(x))       # hidden layers
        return self.layers[-1](x)          # 10 logits

model = MLP([784, 128, 10])
opt = optim.SGD(model.parameters(), lr=0.02)
criterion = nn.CrossEntropyLoss()          # softmax + cross-entropy
for images, labels in loader:              # one epoch
    opt.zero_grad()                        # clear old gradients
    loss = criterion(model(images), labels)   # forward pass and cost
    loss.backward()                        # backpropagation
    opt.step()                             # gradient-descent update
```

A network trained on MNIST has only seen centered white digits on a black background. A phone photo of ink on paper is the opposite, lit unevenly, and framed differently. This mismatch between training data and real data is called a *distribution shift*, and it is the main reason a 98% network can fail on your handwriting. The notebook reduces it in four steps (Fig. 4.28): `load_gray` opens the photo in grayscale and turns it upright; `frame_digit` finds the ink and cuts a square around it 1.4 times the digit's size (MNIST digits fill a 20×20 box inside 28×28 pixels, and $28/20 = 1.4$); `to_digit` resizes the square to 28×28, inverts it to a white digit on black, and stretches the contrast so the paper is 0 and the strokes are near 1; and `normalize` applies $(x - 0.5)/0.5$. Lab 4.2 uses the same `to_digit` for camera pictures.

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **A2**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT or ANSWER needs your input in the cell first (Section 0.11 explains the tags). Until you fill them in, the EDIT and ANSWER cells hold placeholders (0 or `nan`) so that the whole notebook runs.

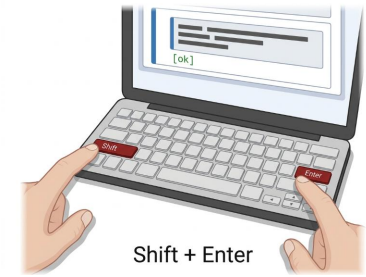
Part A: Setup

GOAL Load PyTorch and the 70 000 MNIST digits. about 10 min

A1 ▸ RUN Load PyTorch and the toolkit

Open `lab4.1-neural-networks.ipynb`, click the first code cell, and press Shift+Enter. The cell checks that PyTorch is installed (the setup file of the kit installs it in the `ee105` environment) and loads the toolkit and the libraries. If it stops with "PyTorch is not installed", see Troubleshooting.

YOU SHOULD SEE `[ok] PyTorch` with its version and the number of CPU threads, for example `[ok] PyTorch 2.5.1 on the CPU (4 threads)`.



A1 Click the first code cell and press Shift+Enter.

A2 ▸ RUN Load and normalize the MNIST digits

A2 reads the digits with `ee105.datasets.load_mnist`. Each image is 28×28 pixels, from 0 (black) to 255 (white), a white digit on a black background (Fig. 4.20). The function `normalize` scales the pixels to $[0, 1]$ and maps them to $[-1, 1]$; Part E and Lab 4.2 use the same function.

YOU SHOULD SEE `training set (60000, 1, 28, 28), test set (10000, 1, 28, 28)`, pixel values from -1.0 to 1.0, and about 6000 training images of each digit (5421 to 6742).

Who wrote MNIST? A few hundred people, Census Bureau employees and high-school students. The 10 000 test digits come from other writers than the 60 000 training digits, so a good test score means the network learned digits, not people.

Part B: A neuron and perceptrons

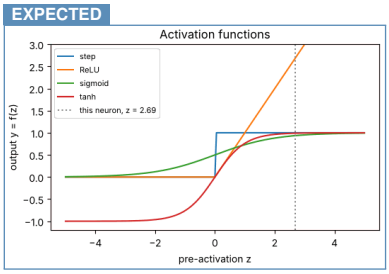
GOAL Compute a neuron's activations, and build logic gates and XOR from perceptrons. about 15 min

B1 ▸ RUN One neuron and four activation functions

A neuron computes $z = \mathbf{w} \cdot \mathbf{x} + b$ and outputs $y = f(z)$ (Eq. (4.4), Fig. 4.6). In PyTorch, `nn.Parameter` marks the three weights and the bias as values the optimizer may change during training. B1 makes a neuron with random weights (seed 105, so everyone gets the same ones), feeds it $\mathbf{x} = (1, 2, 3)$, applies the step, ReLU, sigmoid, and tanh to the same z, and plots the four functions (Fig. 4.7).

YOU SHOULD SEE the weights -0.2055, 0.0008, and 1.0546 (NumPy prints them as `-2.0550e-01` and so on), the bias -0.2668, $z = 2.6932$, and step 1.0000, ReLU 2.6932, sigmoid 0.9366, tanh 0.9909.

EXPECTED



B1 The four activation functions on one plot, with a dotted line at this neuron's z, near 2.7: step and sigmoid end at 1, ReLU keeps rising.

B2 ANSWER Activations of $z = 2.5$ by hand

B2 · ANSWER in the notebook

```
1 hw2 = {"sigmoid": 0.0,    # ✎ your values for z = 2.5
2        "ReLU": 0.0,       # ✎
3        "tanh": 0.0}       # ✎
4
5 for name, value in hw2.items():
6     pt = activations[name](torch.tensor(2.5)).item()
7     print(f"{name:>7s}(2.5): by hand {value:.3f}, PyTorch
          {pt:.3f}")
```

TYPE IN the three values of `hw2` (lines 1 to 3): the sigmoid, ReLU, and tanh outputs of a neuron with input $x = 1.0$, weight $w = 2.5$, and no bias, so $z = 2.5$ (Worked Example 4.5).

UNIT none.

WHERE TO FIND IT compute them with `1 / (1 + math.exp(-2.5))`, `max(0.0, 2.5)`, and `math.tanh(2.5)`, or on a calculator. The cell prints PyTorch's value next to each of yours.

YOU SHOULD SEE sigmoid 0.924, ReLU 2.500, tanh 0.987, each equal to PyTorch's value.

B3 ▸ RUN Perceptrons: AND and OR gates

A perceptron is a neuron with a step activation: it outputs 1 if $\mathbf{w} \cdot \mathbf{x} + b > 0$ and 0 otherwise, with the bias $b = -\text{threshold}$ (Eq. (4.7)). With two inputs it can act as a logic gate. `gate_table` prints a gate's truth table and checks it. AND ($\mathbf{w} = (1, 1)$, $b = -1$) and OR ($b = -0.5$) are given; both lines end with `correct`.

B4 ✎ EDIT NAND, NOR, and XOR

B4 · ✎ EDIT in the notebook

```
1 nand_w, nand_b = [1, 1], -1    # ✎ AND values: change them
2 nor_w, nor_b = [1, 1], -1      # ✎ AND values: change them
3 ok = [gate_table("NAND", nand_w, nand_b),
4       gate_table("NOR", nor_w, nor_b)]
5 xor_table(nand_w, nand_b)          # XOR = AND(OR, NAND)
6 if all(ok):
7     print("✓ Checkpoint 1 passed: NAND, NOR and XOR are correct")
```

WHAT TO WRITE new weights and biases for NAND (line 1) and NOR (line 2). They start with the AND parameters, so they print the wrong truth tables; change them until both lines end with `correct`. One perceptron cannot do XOR (Fig. 4.9), but two layers can: the cell feeds an OR and your NAND into an AND (Fig. 4.11, Worked Example 4.10).

HINT flipping the sign of every weight and of the bias turns $\mathbf{w} \cdot \mathbf{x} + b > 0$ into $\mathbf{w} \cdot \mathbf{x} + b < 0$, so the output flips; then check the $(0, 0)$ row, where only the bias counts (Worked Example 4.8).

DONE WHEN the cell prints NAND `(0, 0)->1 (0, 1)->1 (1, 0)->1 (1, 1)->0` and NOR `1, 0, 0, 0`, both `correct`, then XOR `0, 1, 1, 0` and `✓Checkpoint 1 passed`.

A short history. McCulloch and Pitts proposed the artificial neuron in 1943, Rosenblatt's perceptron learned its own weights in 1958, Minsky and Papert proved in 1969 that one perceptron cannot compute XOR, and backpropagation trained multilayer networks in 1986.

✓CHECK **Checkpoint 1** Your NAND and NOR truth tables both say `correct`, and the XOR table is right (**B4** prints ✓). You built XOR, the function that stalled neural networks in 1969, from three perceptrons.

Part C: A one-layer network (784-10)

GOAL Train your first digit reader and look inside its weights.

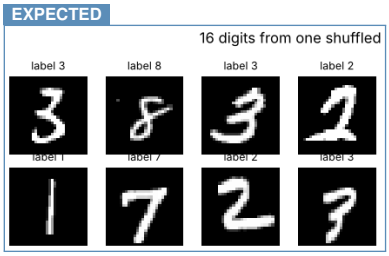
about 15 min

C1 ▸ RUN The training helpers and one mini-batch

C1 defines the helpers used for every network in this lab. `train()` is the four-line training step of the Background listing (`zero_grad`, forward pass and cost, `backward`, `step`) in a loop over batches and epochs; after each epoch it measures the cost and accuracy on the 10 000 test images with `evaluate()`. The training accuracy is counted during the epoch, while the weights are still changing. `plot_curves` draws the accuracy (and the cost) per epoch; `plot_sweep` is for Part F. Stochastic gradient descent updates the weights after every *mini-batch* (Eq. (4.15)): a `DataLoader` shuffles the training set and hands out batches of 128, and one pass over all 60 000 images is an *epoch*. Each step uses only 128 images, so the path down the cost surface wobbles but heads for the minimum (Fig. 4.16).

YOU SHOULD SEE `one batch: images (128, 1, 28, 28), labels (128,); 469 batches per epoch`, and 16 digits of the batch with their labels.

EXPECTED



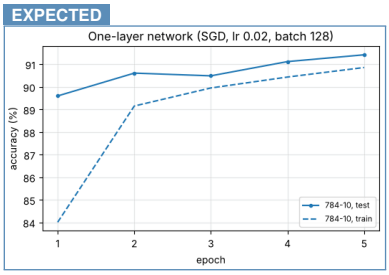
C1 Half of the 16 digits: white on black, each titled with its label. Yours are another shuffle.

C2 ▸ RUN Train the 784-10 network and look at its weights

`SimpleNet` flattens each image to 784 numbers (Fig. 4.27) and has one `nn.Linear(784, 10)` layer: a 10×784 weight matrix and 10 biases, one output (logit) per digit (Fig. 4.13). `nn.CrossEntropyLoss` applies the softmax and the cross-entropy cost in one step (Eqs. (4.9) and (4.12)), and `optim.SGD` does gradient descent. C2 trains it for 5 epochs with SGD, learning rate 0.02, and batches of 128 (a few seconds). Then it draws the 784 weights of each output neuron as a 28×28 image: red pixels push the score for that digit up, blue pixels push it down, so each neuron is a template for its digit (Worked Example 4.11).

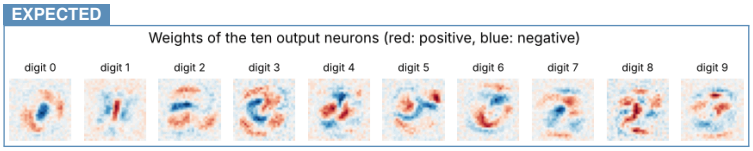
YOU SHOULD SEE five lines `epoch 1/5` to `5/5`, the test accuracy rising from about 89.6 % to 91.4 %, the accuracy and cost curves, ten blurry digit shapes, and ✓`Checkpoint 2 passed`. A test accuracy above the training accuracy in the first epochs is normal: the training accuracy is averaged over an epoch in which the weights were still improving.

EXPECTED



C2 The left panel of two. The test curve starts near 90 % and ends near 91 %, above the training curve.

EXPECTED



C2 The weights of the ten output neurons: each looks like a blurred version of its digit, red where ink raises the score.

✓CHECK **Checkpoint 2** The one-layer network reaches about 91 % on the test set (**C2** prints ✓ from 90 % on); show its accuracy and cost curves. About nine digits in ten, from a network with no hidden layer.

Part D: A hidden layer and a convolutional network

GOAL Train two deeper networks and read the CNN's confusion matrix.

about 20 min

D1 ▸ RUN Train the 784-128-10 network

`MLP` builds a fully connected network from a list of layer sizes, with ReLU after each hidden layer (the Background listing); `MLP([784, 128, 10])` has one hidden layer of 128 neurons. D1 trains it with SGD, learning rate 0.02, batches of 64, for 10 epochs (about 20 s).

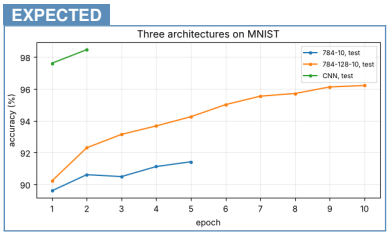
YOU SHOULD SEE ten epoch lines, ending near 96.2 % on the test set.

D2 ▸ RUN Train the CNN and draw its confusion matrix

A convolutional network slides small 3×3 filters over the image (Eq. (4.10)). The same nine weights are used at every position, so a filter that detects a stroke finds it anywhere in the picture. `SmallCNN` (Fig. 4.15) has 16 filters, a 2×2 max pooling (28×28 becomes 14×14), 32 filters, another pooling (7×7), and fully connected layers of 128 and 10 neurons. D2 trains it with the Adam optimizer, learning rate 0.001, for 2 epochs (about a minute on a laptop, longer on an older one; `epochs=5` adds a few tenths of a percent), and plots the test accuracy of the three networks. Lab 4.2 defines the same class to load the weights you save in **G3**. Last, D2 draws the confusion matrix (Section 4.8): row = true digit, column = predicted digit, correct answers on the diagonal.

YOU SHOULD SEE the CNN near 98.5 % after 2 epochs, above the 784-128-10 and the 784-10 networks; in the confusion matrix, about 1000 on the diagonal and a few small numbers off it, most often in the row of the digit 2.

EXPECTED



D2 Three test curves: 784-10 ends near 91, 784-128-10 near 96, and the CNN above 98 % after only 2 epochs.

D3 EDIT Sensitivity, precision and F1 of the worst digit

D3 · EDIT in the notebook

```
d = 0                          # ✎ true digit, most mistakes
sensitivity = float("nan")     # ✎ TP / (TP + FN)
precision = float("nan")       # ✎ TP / (TP + FP)
f1 = float("nan")              # ✎ 2 TP / (2 TP + FP + FN)
row = cm[d].copy(); row[d] = 0
print(f"digit {d}: {cm[d].sum() - cm[d, d]} mistakes, most often
    called {row.argmax()} ({row.max()} times)")
print(f"sensitivity {sensitivity:.3f}, precision {precision:.3f},
    F1 {f1:.3f}")
```

WHAT TO WRITE `d` (line 1), the true digit with the most mistakes, the largest row sum of `cm` off the diagonal. Treat it as the positive class and compute its `sensitivity` $TP/(TP+FN)$, `precision` $TP/(TP+FP)$, and `f1` $= 2TP/(2TP+FP+FN)$ (lines 2 to 4; Eq. (4.19), Worked Example 4.24). You may add lines above line 1 for `TP`, `FN`, and `FP`.

HINT the mistakes of a row are its sum minus its diagonal entry: `cm.sum(axis=1) - np.diag(cm)` gives all ten, and `np.argmax` picks the largest. FN comes from the row of d, FP from its column (Fig. 4.29).

DONE WHEN the cell prints the digit, its number of mistakes and the digit it is most often called (usually `digit 2: 27 mistakes`, mostly called 8 or 3), and a sensitivity, precision, and F_1 between 0.95 and 1.

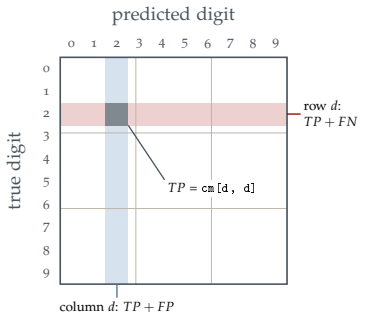


Figure 4.29: Reading `cm` for one digit d (here 2): the rest of its row are the false negatives, the rest of its column the false positives.

Part E: Your own handwritten digits

GOAL Test the 784-10 network and the CNN on digits you wrote yourself.

about 15 min

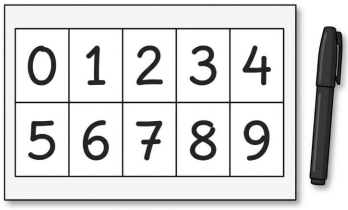
E1 ▸ RUN The photo pipeline, with and without framing

E1 defines the four steps of the Background (Fig. 4.28): `load_gray`, `frame_digit`, `to_digit` (from the toolkit; the camera of Lab 4.2 uses the same function), and `normalize`, and two helpers: `predict` gives the ten probabilities of one input, and `check_photos` runs networks on every photo in a dictionary {file name: digit you wrote}. Then it shows why framing matters: without `frame_digit`, `to_digit` only cuts the center square of the photo, and the digit ends up a different size and in a different place than in MNIST.

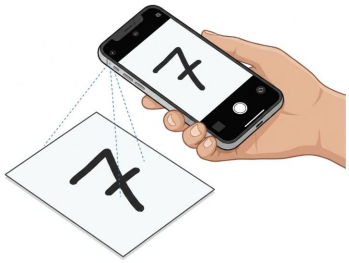
YOU SHOULD SEE two lines for `picture5.png`: with the center crop only, the 784-10 network reads a 7 with 98 % confidence while the CNN still reads a 5; framed, both read a 5 (Question 1).

E2 ◆DO Test your own digits

1. Write a digit with a *thick* dark pen on white paper. Do at least three digits.
2. Take a photo of each digit with your phone, in good light. Crop it so that it shows only the paper and the digit: shadows, fingers, or the table edge confuse the framing step.
3. Save the photos as JPEG or PNG in the lab folder, next to the notebook. iPhones save HEIC photos, which Python cannot open here: set Settings, Camera, Formats, Most Compatible before taking the photo, or use a screenshot.
4. In E2, below the line marked ✎, add one line per photo with its file name and the digit you wrote, for example `photos["my_digit_3.jpg"] = 3`. Keep the two sample photos of a 5 (`picture5.png`, `picture5_v2.png`) that come with the lab. Run E2.

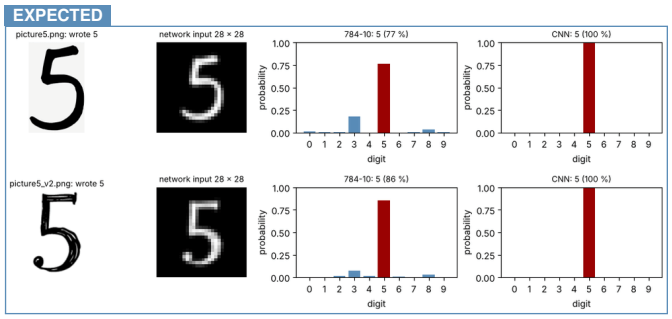


E2 Thick dark marker on white paper, one digit per square.



E2 Photograph each digit in good light, crop the photo to the paper around it, save it in the lab folder, and list it in `photos`.

YOU SHOULD SEE one row per photo: the photo, the 28 × 28 network input, and the ten probabilities of the 784-10 network and of the CNN, each titled with its prediction; then `N of M photos correct` for each network. The 784-10 network gets the two sample 5s right with only 77 % and 86 % confidence, the CNN with 99 % or more. A line `[!!] ... not found` names a photo that is not in the lab folder.



E2 The two sample photos of a 5. Both networks read a 5; the CNN is far more confident. Your own photos add one row each.

✓CHECK **Checkpoint 3** At least one digit you wrote is predicted correctly by the CNN (E2). A network trained on strangers' handwriting can read yours.

Part F: How big a network, and how long to train

GOAL Find the smallest network that works, and watch a large one memorize its training set. about 20 min

The number of hidden neurons, the learning rate, and the number of epochs are *hyperparameters*: you choose them, the network does not learn them (Section 4.6). This Part varies two of them, the size of the hidden layer and the number of epochs.

F1 EDIT The smallest network that works

F1 · EDIT in the notebook

```
1 N_FAST = 20000    # images for the sweep (None: all 60 000)
2 sweep = {}    # ✎ H -> final test accuracy (%)
3 # ✎ your loop over H = 2, 4, 8, 16, 32
4 plot_sweep(sweep)    # the plot; the smallest H with 85 %
```

WHAT TO WRITE a loop that fills `sweep[H]` for $H = 2, 4, 8, 16, 32$ (replace line 3): train `MLP([784, H, 10])` with Adam, learning rate 0.001, batches of 64, 5 epochs, `n_train=N_FAST`, and `verbose=False`, and store the final test accuracy of each (Worked Example 4.27). To keep it quick, `N_FAST` = 20 000 uses a third of the training images; `None` uses all of them.

HINT copy the `train(...)` call of **D1** and change its settings. `train()` returns the history; its last `test_acc`, `h["test_acc"][-1]`, is the final test accuracy. A `torch.manual_seed(0)` before each run makes it repeatable.

DONE WHEN after about a minute the cell prints five accuracies (in percent) near `{2: 28, 4: 57, 8: 86, 16: 91, 32: 91}`, plots them against the 85 % goal, and names the smallest H that reaches it with its floating-point operations per image (Question 2).

F2 ▸ RUN Train a large network on 5000 images

The 784-10 network of Part C is too simple: after 5 epochs its training and test accuracy are both near 91 % and close together, and more epochs barely help. F2 trains a large 784-512-512-10 network (669 706 parameters) with Adam, learning rate 0.001, for 30 epochs on only 5000 training images (`N_FIT`), and plots both networks. The small training set makes overfitting appear within a minute; the chapter's Fig. 4.22 used all 60 000 images and 50 epochs (about 5 min), so your gap opens sooner and wider. Here the test set plays the role of the validation set (Fig. 4.21); in a real project you would keep the test set for the very end.

YOU SHOULD SEE the training accuracy of the large network climbing to 100 % while its test accuracy levels off near 94.6 %, and its test cost lowest near epoch 13 to 15, then creeping up, as in Fig. 4.30.

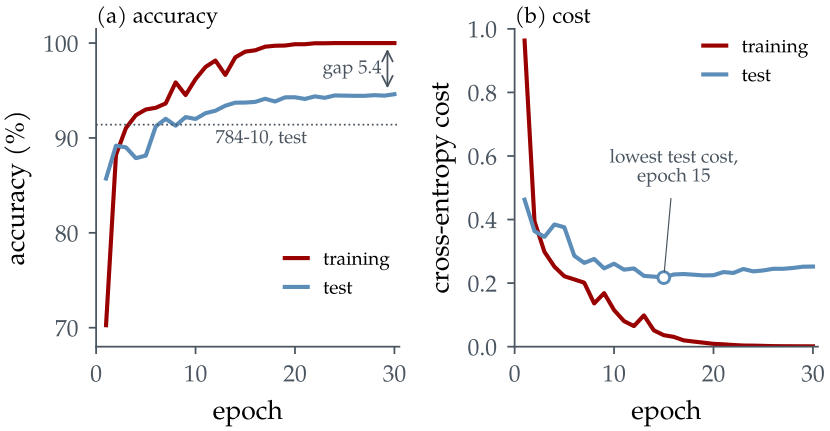


Figure 4.30: Part F, one run of the 784-512-512-10 network on 5000 images. (a) The training accuracy reaches 100% at epoch 24, while the test accuracy levels off near 94.6%, a gap of 5.4 points; the dotted line is the 784-10 network of Part C (91.4%). (b) The training cost keeps falling, but the test cost is lowest at epoch 15 in this run and then creeps up: the network is overfitting. Runs differ by an epoch or so; the run of Worked Example 4.23 has its lowest test cost at epoch 14.

F3 EDIT Early-stopping epochs

F3 · EDIT in the notebook

```
1 best_epoch = 0          # ✎ epoch of the highest test accuracy
2 min_loss_epoch = 0      # ✎ epoch of the lowest test cost
3 print(f"784-512-512-10: best test accuracy at epoch {best_epoch},
      lowest test cost at epoch {min_loss_epoch}")
4 for name, h in [("784-10", hist_simple), ("784-512-512-10",
      hist_over)]:
5     print(f"{name:>15s}: train {h['train_acc'][-1]:.2f} %, test
          {h['test_acc'][-1]:.2f} %, "
6           f"gap {h['train_acc'][-1] - h['test_acc'][-1]:.2f}
              points")
```

WHAT TO WRITE `best_epoch` (line 1), the epoch with the highest test accuracy, and `min_loss_epoch` (line 2), the epoch with the lowest test cost, both counted from 1, from `hist_over["test_acc"]` and `hist_over["test_loss"]`. Early stopping would keep the weights from one of these epochs (Worked Example 4.23).

HINT `np.argmax` and `np.argmin` count from 0, so add 1; `int(...)` makes a plain integer.

DONE WHEN the cell prints both epochs (the lowest test cost near epoch 13 to 15) and the final gap between training and test accuracy: near 0 for 784-10, about 5 points for 784-512-512-10 (Question 3).

CHECK **Checkpoint 4** The plots of F2 show underfitting (784-10) and overfitting (784-512-512-10), and you have an early-stopping epoch (F3). You have seen both ways a network can fail, and the cure for one of them.

Part G: What each network costs, and saving the CNN

GOAL Count the memory and time each network needs, and save the CNN for Lab 4.2. about 10 min

On a microcontroller the questions are memory (parameters) and time (multiply-accumulates, MACs) per image (Section 4.9). For a fully connected network with layer sizes $(n_0, n_1, \ldots, n_L)$,

$$N_{\text{params}} = \sum_{l=1}^{L} (n_{l-1} n_l + n_l), \qquad N_{\text{MAC}} = \sum_{l=1}^{L} n_{l-1} n_l.$$

G1 EDIT Write `fc_cost(sizes)`

G1 · EDIT in the notebook

```
1 def fc_cost(sizes):
2     """(parameters, MACs) of a fully connected network."""
3     params = 0    # sum of n_(l-1) * n_l + n_l
4     macs = 0      # sum of n_(l-1) * n_l
5     return params, macs
6
7
8 print(f"784-10: {fc_cost([784, 10])}, expected (7850, 7840)")
```

WHAT TO WRITE `params` (line 3) and `macs` (line 4) of a fully connected network with the two sums above; `sizes` is the list $[n_0, n_1, \ldots, n_L]$ (Worked Example 4.14).

HINT `zip(sizes[:-1], sizes[1:])` pairs each layer with the one before it, so `sum(a * b for a, b in zip(...))` is the MAC count.

DONE WHEN the cell prints `784-10: (7850, 7840), expected (7850, 7840).`

G2 ▸ RUN The cost table

G2 checks your counts against PyTorch's and adds the CNN, whose MACs `count_macs` counts layer by layer. Memory is 4 bytes per parameter in float32 and 1 byte in int8; the Nano's Cortex-M4 does about 32 million MACs per second, which gives the time per image.

YOU SHOULD SEE a table whose `fc_cost` columns equal PyTorch's: parameters and MACs 7850 and 7840 (784-10), 101 770 and 101 632 (784-128-10), 669 706 and 668 672 (784-512-512-10), and 206 922 and 1 218 048 for the CNN, about 38 ms per image on the Nano (Question 4).

G3 ▸ RUN Save the CNN for Lab 4.2

G3 saves the CNN as `data/models/mnist_cnn.pt` in the kit (the folder `..` of the notebook): the weights (`state_dict`), the constructor arguments of `SmallCNN`, the normalization, and the test accuracy, so that Lab 4.2 can rebuild exactly this network. If you changed the CNN or trained it longer, the file holds your version. Do not delete it.

YOU SHOULD SEE `[ok] saved the CNN (98.46 % on the MNIST test set) to .../data/models/mnist_cnn.pt`.

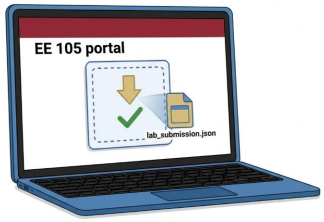
If you have time. Read a whole number: write four digits in a row with space between them, cut the photo at the blank columns between the digits, and run each piece through `frame_digit`, `to_digit`, and the CNN. Or train longer: `epochs=5` for the CNN in **D2**, or `N_FAST = None` in **F1**. Or train `MLP([784, 128, 10])` with Adam at learning rates 0.001 and 0.01 and compare the curves (Fig. 4.18).

Save and upload

Save ▸ RUN Write `lab4.1_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab4.1_submission.json` in the lab folder: your numbers, a summary of the CNN (its layers and their sizes), and its predictions for 100 fixed MNIST test images (every 100th one), which the portal checks against the true labels. Log in at `ee105.org/portal`, open Lab 4.1, and upload that file. Keep `data/models/mnist_cnn.pt`: Lab 4.2 uses it.

YOU SHOULD SEE `[ok] Saved lab4.1_submission.json (1.5 kB)` with its folder, and then the portal shows your grade.



Save Upload `lab4.1_submission.json` on the course portal under Lab 4.1.

FINISH THE LAB

You built a neuron, logic gates from perceptrons, and three digit readers, tested them on your own handwriting, watched a network overfit, and counted what each network costs. The lab is done when every box is ticked.

- ☐ **B2** ANSWER `hw2`: sigmoid, ReLU, tanh of $z = 2.5$
- ☐ **B4** EDIT `nand_w`, `nand_b`, `nor_w`, `nor_b`
- ☐ **D3** EDIT `d`, `sensitivity`, `precision`, `f1`
- ☐ **E2** DO at least three of your own digits in `photos`
- ☐ **F1** EDIT `sweep` for $H = 2, 4, 8, 16, 32$
- ☐ **F3** EDIT `best_epoch`, `min_loss_epoch`
- ☐ **G1** EDIT `fc_cost`
- ☐ **G3** ▸ RUN `data/models/mnist_cnn.pt` saved for Lab 4.2
- ☐ CHECK Checkpoint 1: NAND, NOR, and XOR correct
- ☐ CHECK Checkpoint 2: the 784-10 network near 91 %
- ☐ CHECK Checkpoint 3: one of your digits read correctly
- ☐ CHECK Checkpoint 4: underfitting, overfitting, an early-stopping epoch

☐	Questions 1 to 4 answered in the notebook
☐ **Save** SAVE	`lab4.1_submission.json` uploaded on the portal

Typical results

Table 4.3 lists typical results, and Figure 4.30 shows the curves of Part F. The random seeds are fixed, so your numbers should come within about a point of these (multithreading and library versions change the details); results on your own photos depend on your handwriting.

step	typical result
B1, B2	seed 105: $z = 2.693$. **B2**: sigmoid 0.924, ReLU 2.5, tanh 0.987.
C2	784-10 after 5 epochs: about 91.4% on the test set.
D1 to D3	784-128-10: about 96.2%. CNN after 2 epochs: about 98.5%. The digit with the most mistakes is usually 2 (about 27 errors, mostly called 8 or 3).
E1, E2	sample photos: the 784-10 network gets both 5s right only after framing (77% and 86% confidence); the center crop alone is read as a 7 with 98% confidence. The CNN reads the center crop as a 5 too and gives the framed 5s 99% or more.
F1	$H = 2$: 28%, 4: 57%, 8: 86%, 16: 91%, 32: 91%. Smallest H that reaches 85%: 8, with 12 704 floating-point operations per image.
F2, F3	784-512-512-10 on 5000 images: training 100%, test about 94.6%; the lowest test cost near epoch 13 to 15.
G1, G2	parameters/MACs: 7850/7840 (784-10), 101 770/101 632 (784-128-10), 669 706/668 672 (784-512-512-10), 206 922/1 218 048 (CNN, 38 ms on the Nano).

Table 4.3: Typical results of the Lab 4.1 notebook. Accuracies are on the 10 000 MNIST test images.

Troubleshooting

- *"PyTorch is not installed."* Run the setup file again, or run the `%pip` line that **A1** prints, in a new cell, then restart the kernel (Kernel, Restart) and run **A1** again. On Linux use the CPU-only line it gives; plain `pip install torch` downloads a build of more than 2 GB.
- *A photo is "not found"* (**E2**). The file must be in the same folder as the notebook, and the name must match exactly, including `.jpg` or `.png`. iPhone photos are HEIC and will not open: choose Settings, Camera, Formats, Most Compatible, or use a screenshot.
- *Your digit is misread.* `frame_digit` finds the ink from the darkest pixels, so shadows, fingers, or a dark table edge make the frame too large and the digit too small. Crop the photo tighter, use a thicker pen, and look at the network input in the figure of **E2**.
- *Training is slow or the laptop runs out of memory.* The notebook needs about 1 GB of memory at its peak, and 2 to 3 minutes of training in

all is normal; do not rerun a cell that is still running. Close other programs, or lower `N_FAST` in **F1**.

- *A printout shows 0 or `nan`.* The EDIT and ANSWER cells hold placeholders so that the whole notebook runs before you finish them; a 0 or `nan` means that step is still to be done.
- *Lab 4.2 cannot find the CNN.* Run **G3**: it saves `data/models/mnist_cnn.pt`.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. How many of your own digits did the 784-10 network and the CNN classify correctly (**E2**)? Explain one failure in terms of distribution shift. Why does the 784-10 network care so much about framing (**E1**), while the CNN does not?
2. What is the smallest hidden layer that reaches 85% (**F1**), and how many floating-point operations does one prediction take?
3. In Part F, at which epoch does the large network's test accuracy stop improving? What would early stopping choose (**F3**), and why does the training accuracy keep rising after that?
4. Use the table of **G2** to compare the test accuracy, parameters, and MACs of the 784-10, 784-128-10, and 784-512-512-10 networks and the CNN. Is the extra accuracy worth the extra cost? Which networks fit in the Nano's 256 kB of RAM in float32, and which in int8?

4.11 *Laboratory: Camera Classifier*

Lab 4.2 at a glance

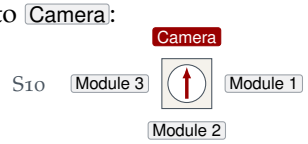
Goal: stream pictures from the demo-board camera into a notebook, classify handwritten digits live with the CNN from Lab 4.1, measure its accuracy on your own handwriting, and reduce the gap to MNIST.
Hardware: EE 105 demo board with the Nano 33 BLE Sense and the OV7675 camera module, USB data cable, laptop with PyTorch, white paper, and a thick black marker.
Notebook: `lab4.2-camera-classifier.ipynb` in the folder `lab4.2-camera-classifier` of the lab kit. It loads the network that Lab 4.1 saved (file `mnist_cnn.pt` in the kit's `data/models` folder).
Before the lab: finish Lab 4.1 through step G3, so that `data/models/mnist_cnn.pt` exists. (If it does not, the notebook trains a stand-in CNN in about 15 s, which reaches about 97% on MNIST.) Write the digits 0 to 9 with a thick black marker on white paper, one digit in each square of about 4 cm, with space around it.
Time: about 95 minutes.
You hand in: `lab4.2_submission.json` on the course portal; keep your notebook with your answers.

Key takeaways

- A 160×120 picture is 25 600 base64 characters, 2.2 s at 115 200 baud; over the Nano's native USB it takes a few hundred ms.
- Inputs must match training: a 28×28 white digit on black, scaled by $(x - 0.5)/0.5$. Without that scaling the accuracy collapses.
- The confidence is the largest softmax output: 95 % or more for a clear digit, 40 to 50 % for one split between two classes.
- A CNN at 98.5 % on MNIST typically gets 60 to 90 % of your camera digits right: a domain shift (smaller, thinner digits, other light).
- A crop chosen on the same 20 pictures it is scored on looks better than it is: judge any fix on pictures it never saw.

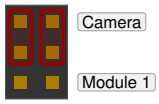
Set up the board for Lab 4.2

Selector. Turn S10 to Camera:



Jumpers. The camera's pixel clock and line-valid signals use Arduino pins A0 and A1, which the circuits block normally uses. Move both caps of J7 and J8, the two headers just left of the Arduino, from the Module 1 end to the Camera end (pins 1 and 2), so A0 and A1 reach the camera:

J7 and J8 caps on the Camera end



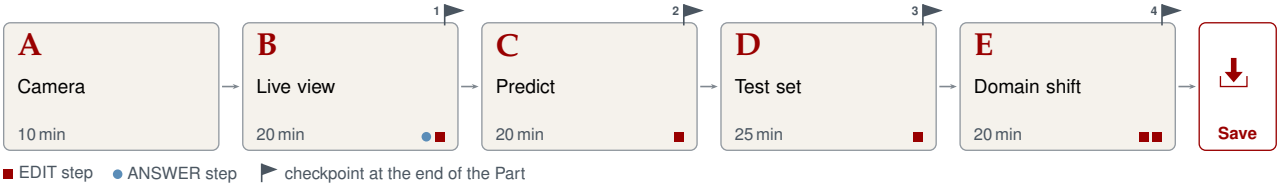
pin 1: camera
pin 2: Arduino `A0` (J7), `A1` (J8)
pin 3: touch panel (J7), divider (J8)

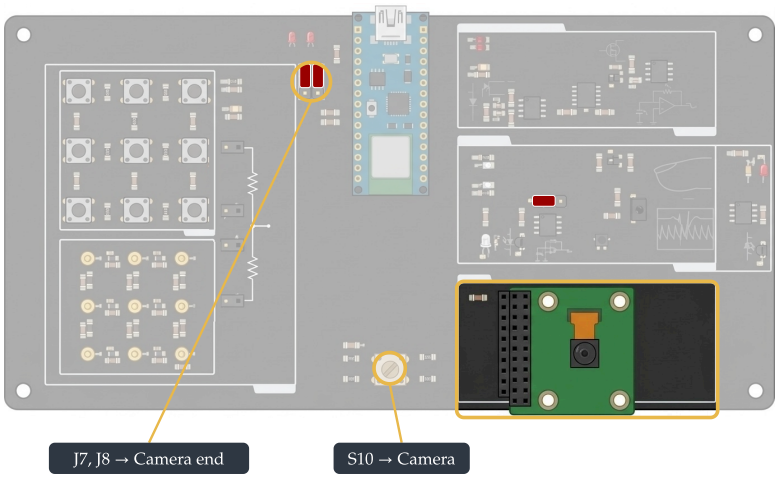
Camera. With the USB cable unplugged, check that the OV7675 camera module sits fully in header J6 (press it in if it is not there), lens facing out, away from the board, and every pin in its socket (the board map in Chapter 0 shows where J6, J7, and J8 are). Then plug in the USB cable. If another Arduino is connected to the laptop, unplug it: the notebook uses the first one it finds.

Check. The first notebook cell prints two lines that start with `[ok]`, the second saying that S10 is on `Camera`. If the reading does not fit Camera, it prints a warning; check S10 and carry on.

Afterwards. Move both caps back to the Module 1 end (pins 2 and 3). The circuits and sensors labs need them there.

Figure 4.31 shows the camera header, J7, J8, and S10 on the board, and Figure 4.32 shows the lab running.

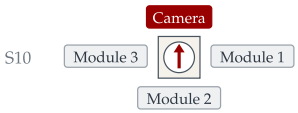




Lab 4.2: set up the board

Uses: camera on J6

Selector S10: turn it to **Camera**



Jumpers

J7 and J8 → both caps on the **Camera** end



pin 1: camera
pin 2: A0 (J7), A1 (J8)
pin 3: touch panel (J7), divider (J8)

Plug in and place

Camera module in J6, lens facing out.

Afterwards move both caps back to the Module 1 end.

Figure 4.31: The board for Lab 4.2. The camera module sits in header J6 (highlighted). S10 is on Camera, and the caps of J7 and J8 are on the Camera end, so A0 and A1 go to the camera. Move both caps back to the Module 1 end after the lab.

Objectives

- Capture grayscale pictures on the Arduino and stream them to the laptop over USB; relate the picture size to the data rate.
- Preprocess each picture to the MNIST format and classify it, with the softmax confidence.
- Measure the accuracy on a test set of your own digits, with a confusion matrix, and compare it with the MNIST test accuracy.
- Explain the drop as a distribution shift, and improve the result by changing the crop.

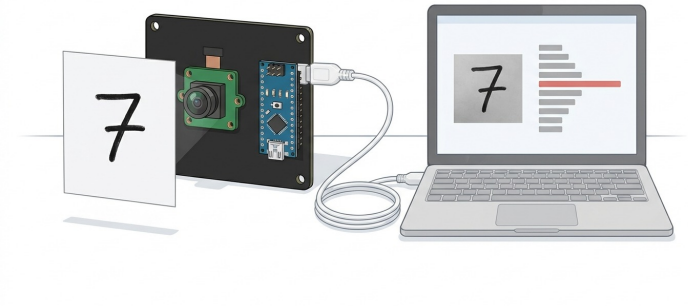


Figure 4.32: Lab 4.2 running. The camera on the board looks at a digit written on paper; the laptop shows the picture and the ten class probabilities, with the winning digit's bar highlighted.

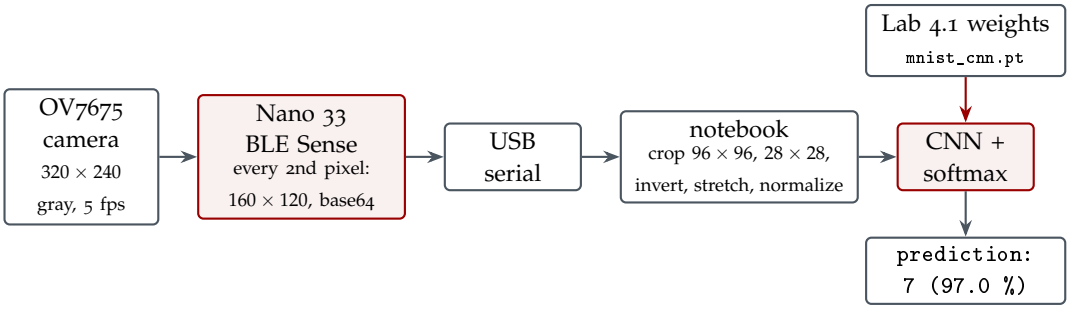


Figure 4.33: Data path of Lab 4.2. The Arduino captures and shrinks each picture; the laptop preprocesses it and runs the network trained in Lab 4.1.

Background

Figure 4.33 shows the data path. The OV7675 camera delivers a 320×240 grayscale frame at 5 frames per second, 76 800 pixel values. When the notebook asks for a picture with `cam.snap(160, 120)`, the camera firmware on the Nano reads one frame and keeps pixel $(320x/w,\ 240y/h)$ for each output pixel (x, y) of a $w \times h$ picture (integer division). For 160×120 that is every second pixel in each direction, one pixel in four. It then sends the wh bytes, one per pixel, as base64 text, 4 characters for every 3 bytes, on one line. The notebook uploads this firmware itself the first time; no Arduino IDE, camera library, separate camera window, or serial-port selection is needed.

> **Worked Example 4.35** *How long does a picture take?*
>
> How many characters does a 160×120 picture take, and how long would they take on a serial line at 115 200 baud with 10 bits per character? Repeat for 80×60.
>
> Solution. $160 \times 120 = 19\,200$ bytes. Base64 turns every 3 bytes into 4 characters: $19\,200 \times 4/3 = 25\,600$ characters, or 256 000 bits, which take $256\,000/115\,200 = 2.22$ s. For 80×60: 4800 bytes, 6400 characters, 0.556 s. In Part B expect a picture to take only a few hundred milliseconds instead (your board may differ; record your own value). The Nano's serial port is a native USB connection that moves data at USB speed whatever baud rate is set, so the wire is not the limit; waiting for the next camera frame (up to 200 ms at 5 frames per second) and the work on both ends are. ■

The notebook turns each picture into a network input with `to_digit`, the same function Lab 4.1 used for your photos (Figure 4.34). It cuts a center square whose side is a fraction, 0.8 by default, of the picture height (a 96×96 square from a 160×120 picture), resizes it to 28×28 by averaging neighboring pixels, inverts it if the paper is bright so the digit is white on black, and stretches

the contrast: the median pixel (the paper) becomes 0 and the brightest strokes 1. The network then needs the Lab 4.1 normalization $(x - 0.5)/0.5$. The prediction is the digit with the largest softmax output, and that largest p_k is the confidence.

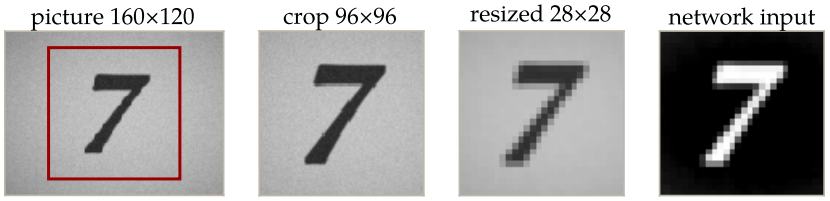


Figure 4.34: A 160×120 grayscale picture of a handwritten 7 and the Lab 4.2 steps that turn it into a 28×28 network input: the picture (red square: the center crop, 0.8 of the picture height), the 96×96 crop, the crop averaged down to 28×28, and the input after inversion and contrast stretching.

The key parts of the firmware and of the preprocessing are:

```
void snap(int w, int h) {                // command "snap 160 120"
  int W = Camera.width(), H = Camera.height();   // 320 x 240
  Camera.readFrame(frame);               // one grayscale frame
  Serial.print("#size "); Serial.print(w);
  Serial.print(' ');      Serial.println(h);
  for (int y = 0; y < h; y++) {
    int sy = (long)y * H / h;            // source row
    for (int x = 0; x < w; x++) {
      int sx = (long)x * W / w;          // source column
      b64_put(frame[sy * W + sx]);       // 3 bytes -> 4 characters
    }
  }
  b64_end();                             // pad and end the line
}
```

```
def to_digit(img, size=28, fraction=0.8):
    x = resize(center_square(img, fraction), size) / 255.0
    if x.mean() > 0.5:                   # bright paper: invert
        x = 1.0 - x
    lo = np.percentile(x, 50)            # background level
    hi = np.percentile(x, 99.5)          # brightest strokes
    return np.clip((x - lo) / max(hi - lo, 1e-3), 0, 1)
```

A network that scores 98.5% on MNIST usually does worse on your pictures. The MNIST digits were written by other people with other pens, scanned, and centered so they fill a 20×20 box. Your pictures differ in size, stroke width, lighting, and blur, a distribution shift (also called a *domain shift*). Part E measures it and tries one remedy: a different crop fraction rescales your digits toward the MNIST size.

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **B1**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT or ANSWER needs your input in the cell first (Section 0.11 explains the tags). Several steps ask you to hold a digit up to the camera: hold it 10 to 20 cm from the lens, the paper facing it, so that the digit fills about half of the picture height and sits in the middle.

Part A: Connect the camera

GOAL Seat the camera, move the jumpers, and get two `[ok]` lines.

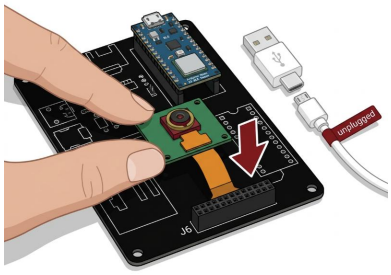
about 10 min

A1 DO Connect the camera

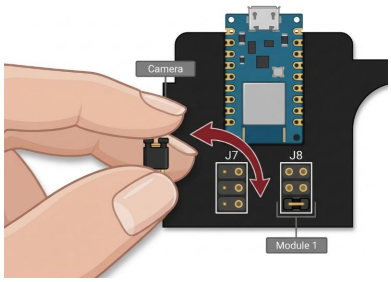
1. Unplug the USB cable. If the OV7675 camera is not already on header J6 of the demo board, plug it in, lens facing out, away from the board.
2. Move both caps of J7 and J8 (the two small headers just left of the Arduino) from the Module 1 end to the Camera end (pins 1 and 2), so A0 and A1 of the Arduino reach the camera.
3. Turn the selector S10 to Camera (Fig. 4.31).
4. Plug the USB cable back in, click the first code cell, and press Shift+Enter. The first time, `connect_camera` compiles and uploads the camera firmware (about a minute; it replaces the course firmware, and the next board lab puts that back by itself).

No board today? Set `SIMULATE = True` in this cell to practice the whole lab on a simulated camera.

YOU SHOULD SEE two lines that begin with `[ok]`: the camera board is connected (firmware `ee105cam`), and S10 is on `Camera`. A `[!!]` line about the selector is only a warning: if S10 points at Camera, carry on.



A1 USB cable unplugged: seat the camera fully in header J6, lens facing out.



A1 J7 and J8 are single rows of 3 pins: move each cap to the Camera end, covering pins 1 and 2. (The picture draws the headers two pins wide.)

Part B: Look through the camera

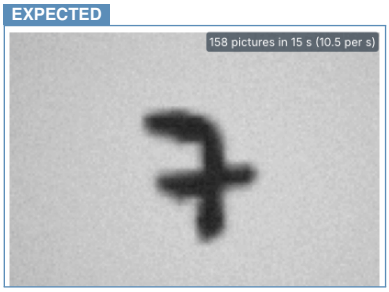
GOAL See what the camera sees, time the pictures, and turn one into a 28 × 28 network input. about 20 min

B1 DO Live view (15 s)

Run B1: `cam.live(15)` shows 15 seconds of live pictures (press the stop button to end early) and keeps the last one. Hold one of your digits in front of the camera, about 10 to 20 cm away, the paper facing the lens (Fig. 4.32), and move it until the digit fills about half of the picture height, in the middle. Watch the brightness: the paper should look light gray, not black and not burned out.

YOU SHOULD SEE your digit in the live view, dark on light-gray paper. On a slow laptop the live view may run at only 1 to 3 pictures per second, which is fine.

EXPECTED



B1 The live view: a dark digit on light-gray paper, about half the picture height.

B2 ANSWER Orientation of the picture

B2 · ANSWER in the notebook

```
1 FLIP_UD = False      # ✎ True if the paper appears upside down
2 FLIP_LR = False      # ✎ True if writing appears mirrored
```

TYPE IN `FLIP_UD = True` if the paper appears upside down in the live view, `FLIP_LR = True` if the writing appears mirrored; leave both `False` if the picture is the right way up. Every picture used below goes through `grab()`, which applies the setting.

UNIT none (`True` or `False`).

WHERE TO FIND IT the live view of **B1**; **B4** shows a picture with the setting applied. If you change a setting later, rerun B2.

B3 EDIT Picture time and characters at 115 200 baud

B3 times five pictures at 160 × 120 and at 80 × 60 (lines 1 to 6) and compares each time with a classic serial line. The Nano sends one byte per pixel as base64 text (Fig. 4.33).

B3 · EDIT in the notebook

```
1  timing = {}
2  for w, h in [(160, 120), (80, 60)]:
3      t0 = time.time()
4      for _ in range(5):
5          grab(w, h)
6      timing[(w, h)] = dt = (time.time() - t0) / 5
7      chars = 0        # ✎ base64 characters for w * h bytes
8      t_uart = 0       # ✎ seconds at 115 200 baud, 10 bits each
9      print(f"{w} x {h}: {chars:.0f} characters, "
10           f"{1000 * t_uart:.0f} ms at 115 200 baud; "
11           f"measured {1000 * dt:.0f} ms")
```

WHAT TO WRITE for each picture size `w` × `h`, `chars` (line 7), the number of base64 characters for the *wh* bytes, and `t_uart` (line 8), the time those characters would

take on a classic serial line at 115 200 baud with 10 bits per character (Worked Example 4.35).

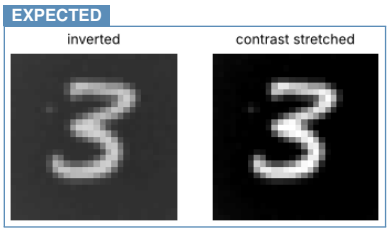
HINT one byte per pixel, 4 characters for every 3 bytes; a character takes 10/115 200 s.

DONE WHEN the cell prints `160 x 120: 25600 characters, 2222 ms at 115 200 baud` and `80 x 60: 6400 characters, 556 ms`, each next to your measured time: a few hundred milliseconds per 160×120 picture (roughly 250 to 450 ms), and an 80×60 picture only a little faster. Note your own values. Is the serial link what limits the picture rate (Question 3)?

B4 ◆DO The preprocessing steps

Hold up one of your digits as in **B1** and run B4: it takes a picture and shows every step of `to_digit` (Fig. 4.34), from the picture with the center crop (`FRACTION` = 0.8 of the picture height) marked in red to the inverted, contrast-stretched 28×28 input. If the picture is upside down or mirrored, fix **B2**.

YOU SHOULD SEE five pictures that end in a white digit on black filling the middle of the 28×28 input. The line below gives the paper level, roughly 150 to 220 (of 255) under good light, the darkest ink level, below 80, and the fraction of stroke pixels in the input (MNIST: about 12 %).



B4 The last two of five panels: the inverted crop and the network input, a white digit on black that fills the middle.

✓CHECK **Checkpoint 1** The live view shows your digit, and the preprocessing steps of **B4** end in a white digit on black that fills the middle of the 28×28 input. Your camera now speaks MNIST: 784 numbers, white on black.

Part C: Load the network and predict live

GOAL Load the CNN you trained in Lab 4.1 and read your digits live, with its confidence. about 20 min

C1 ▸RUN Load your Lab 4.1 CNN

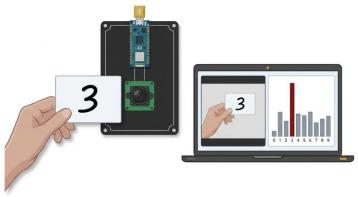
Lab 4.1 saved its CNN in `data/models/mnist_cnn.pt`. The class `SmallCNN` in C1 is the same as in Lab 4.1, so the saved weights fit it. If the file is missing (you did not finish Lab 4.1), the cell trains the same CNN on 20 000 MNIST images for 2 epochs (about 15 s) so that you can continue. C1 then applies the Lab 4.1 normalization $(x - 0.5)/0.5$ to the 10 000 MNIST test images and tests the network on them.

YOU SHOULD SEE `[ok] loaded your Lab 4.1 CNN` and an MNIST test accuracy near 98.5 % (the stand-in CNN: about 97 %, after a `[!!] ... not found` line).

TinyML. The Nano's nRF52840, a 64 MHz Cortex-M4 with 1 MB of flash and 256 kB of RAM, can run small networks at a power of a milliwatt or less. Running machine learning on such a chip is called TinyML (Section 4.9).

C2 DO Live prediction (15 s)

C2 defines `classify`, which preprocesses one picture and returns the predicted digit, the ten softmax probabilities, and the network input. The largest probability is the network's *confidence*. The live view then shows the prediction for every picture in its title. Hold up different digits, one at a time, as in **B1**.

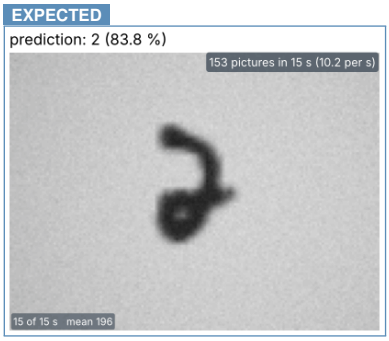


C2 Hold a digit 10 to 20 cm in front of the camera, the paper facing the lens; the title of the live view shows the prediction. (The picture shows the card beside the camera; hold it in front of the lens.)

YOU SHOULD SEE `prediction: 7 (97.0 %)` or similar in the title, changing with each digit. Thick, clear digits come out at 95 % or more; ambiguous ones lower.

C3 DO Three predictions for the record

Run C3. For each of three different digits, hold it up and press Enter in the box that appears. Each figure shows the picture, the network input, and all ten probabilities (Fig. 4.35).



C2 The prediction and its confidence in the title. The simulated camera sends about 10 pictures per second; a real board fewer.

YOU SHOULD SEE three figures, each with ten probabilities. A clear, thick digit usually gets more than 90 %; a sloppy one splits between two classes, like the open-top 3 of Fig. 4.35, whose numbers are an illustration, not a measurement (Question 1).

C4 EDIT Softmax by hand

C4 · EDIT in the notebook

```
img = grab()
with torch.no_grad():
    z = model(normalize(to_digit(img,
        fraction=FRACTION)))[0].numpy()     # the 10 logits
p_hand = np.zeros(10)    # ✎ softmax of z
p_torch = torch.softmax(torch.tensor(z), dim=0).numpy()
print("digit   " + "".join(f"{k:>8d}" for k in range(10)))
print("logit   " + "".join(f"{v:8.2f}" for v in z))
print("by hand " + "".join(f"{v:8.4f}" for v in p_hand))
print("PyTorch " + "".join(f"{v:8.4f}" for v in p_torch))
```

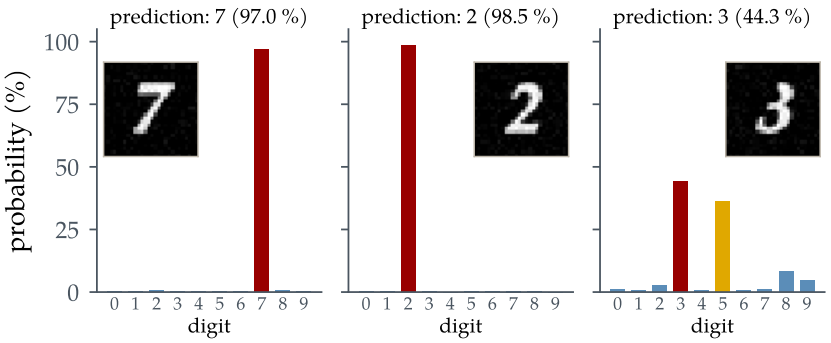


Figure 4.35: Three Part C predictions as the notebook shows them (an illustration made with representative values, not a measurement on the board): the network input (inset) and the ten softmax probabilities. A thick 7 and a 2 are recognized with 97.0 and 98.5% confidence. A 3 with an open top loop shares its probability between 3 (44.3%) and 5 (36.2%), a sign that the answer is uncertain.

WHAT TO WRITE `p_hand` (line 4), the softmax of the ten logits `z` (the outputs before the softmax): $p_k = e^{z_k} / \sum_j e^{z_j}$ (Eq. (4.9)). Lines 1 to 3 take a picture and compute `z`; hold up a digit before you run the cell.

HINT `np.exp(z)` works on the whole array at once, and `.sum()` adds up the ten values.

DONE WHEN the rows `by hand` and `PyTorch` agree in every digit printed, and the ten probabilities add up to 1.

✓CHECK **Checkpoint 2** The live view predicts your digits (**C2**), and you have the three prediction figures of **C3**. A network you trained reads the digits you hold up, live.

Part D: Your own test set

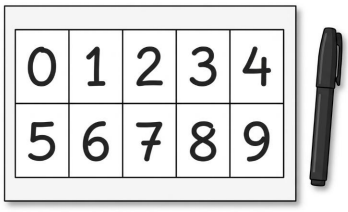
GOAL Measure the network's accuracy on 20 pictures of your handwriting.

about 25 min

D1 ◆DO Collect 20 pictures

How good is the network on your handwriting and this camera? D1 runs `collect`, which asks for each digit in turn. Hold the paper up and press Enter; the view runs for 2 s and keeps the last picture. It takes *two pictures of each digit*: move the paper a little after the first one, and hold still. That makes 20 labeled pictures, saved in `my_digits.npz` in the lab folder (`np.load` reads them back later).

YOU SHOULD SEE the live view with "digit 3, picture 1 of 2: hold still" and so on, then `[ok] 20 pictures saved in my_digits.npz`.



D1 Your ten digits: thick black marker, one per square of about 4 cm, with space around each. Hold each one up when the cell asks for it and press Enter.

D2 ▸RUN Contact sheet

D2 shows the network input of every picture with the true digit, the prediction, and the confidence in its title (green: correct, red: wrong), and saves the predictions in `my_digits_results.csv`, a table for your report. It also defines `plot_confusion` for **D3**.

YOU SHOULD SEE 20 centered white digits on black, most titles green, as in Fig. 4.36.

D3 ✎EDIT Accuracy and confusion matrix

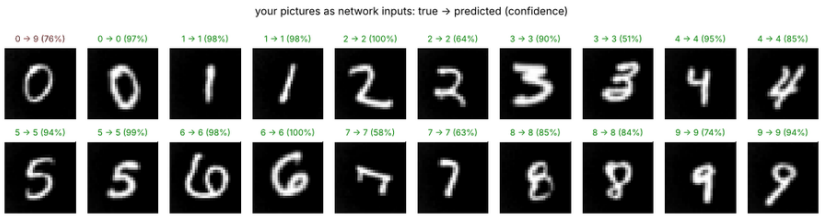


Figure 4.36: A typical contact sheet (D2): the 20 network inputs, two of each digit, titled true digit → prediction (confidence). Green titles are correct; the one red title marks a wrong prediction (a 0 read as a 9).

D3 · EDIT in the notebook

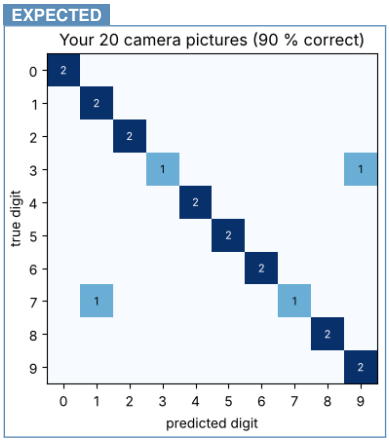
```
1  preds = np.array([classify(img)[0] for img in pictures])
2  cam_acc = 0.0                          # ✎ % of pictures correct
3  cm = np.zeros((10, 10), dtype=int)     # ✎ cm[true, pred]
4  print(f"your pictures: {cam_acc:.0f} % correct ({np.sum(preds ==
       labels)} of {len(labels)});  "
5        f"MNIST test set: {mnist_acc:.2f} %")
6  plot_confusion(cm, f"Your {len(labels)} camera pictures
       ({cam_acc:.0f} % correct)")
```

WHAT TO WRITE `cam_acc` (line 2), the percentage of your pictures classified correctly, and `cm` (line 3), the confusion matrix: `cm[true, predicted]` counts each picture once (row = true digit, column = predicted digit, Section 4.8). `preds` holds the network's answers and `labels` the true digits.

HINT `preds == labels` is an array of `True` and `False`, and its mean is the fraction correct. A loop over `zip(labels, preds)` adds one count per picture (or use `np.add.at`).

DONE WHEN the cell prints your accuracy next to the MNIST accuracy and draws a matrix whose counts add up to 20, mostly on the diagonal. Before any improvement expect roughly 60 to 90 % on your 20 pictures, against about 98.5 % on MNIST.

EXPECTED



D3 The counts add up to 20, almost all on the diagonal; each count off it is one confusion (row: true digit, column: prediction).

✓CHECK **Checkpoint 3** You have the contact sheet, your accuracy next to the MNIST accuracy, and the confusion matrix (D3). You have an honest number for how well the network reads your handwriting.

Part E: Domain shift and one improvement

GOAL Measure the gap to MNIST, and narrow it with the crop.

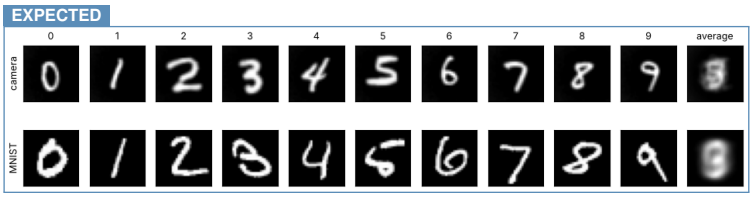
about 20 min

The network learned from MNIST, and your pictures come from a different *domain*: other writers, other pens, a camera with its own lighting and framing.

E1 ▸ RUN Your inputs next to MNIST

E1 puts one of your inputs of each digit next to an MNIST digit of the same class, and shows the average of each set (right).

YOU SHOULD SEE your digits smaller and with thinner, blurrier strokes than the MNIST digits; the average camera digit is a smaller blob than the MNIST average.



E1 Top: one input of each digit, here from the simulated camera; bottom: MNIST; right: the averages of each set.

E2 EDIT Stroke fraction and digit height

E2 · EDIT in the notebook

```
1  def digit_height(X):
2      """Mean number of rows with a pixel > 0.5, over X (N x 28 x
           28)."""
3      return 0.0    # ✎ rows with a stroke, mean over images
4
5
6  ink_cam, ink_mn = 0.0, 0.0    # ✎ fractions of pixels > 0.5
7  print(f"stroke pixels: camera {ink_cam:.1%}, MNIST {ink_mn:.1%}")
8  print(f"digit height:  camera {digit_height(X_cam):.1f} px, MNIST
       {digit_height(X_mn):.1f} px")
```

WHAT TO WRITE the return line of `digit_height` (line 3): the number of rows that contain at least one stroke pixel (value above 0.5), counted for each image and averaged over the images. Then `ink_cam` and `ink_mn` (line 6), the fraction of stroke pixels in `X_cam` (your inputs) and in `X_mn` (MNIST), both $N \times 28 \times 28$.

HINT `(X > 0.5).any(axis=2)` marks the rows that hold a stroke; `.sum(axis=1)` counts them per image. The mean of `X > 0.5` is the fraction of stroke pixels.

DONE WHEN the cell prints both numbers for both sets. MNIST: about 12 % stroke pixels and a height of 19.7 pixels. With a crop of 0.8 the simulated camera gives digits about 14 to 16 pixels tall with about 8 % stroke pixels; your values depend on your handwriting and distance (Question 2).

E3 ▸ RUN Without the normalization

The normalization is part of the match too. E3 feeds the $[0, 1]$ pixels directly, without $(x - 0.5)/0.5$, so the network sees inputs it never saw in training.

YOU SHOULD SEE `with normalization: N %, without: M %`; without it the accuracy drops sharply, to about 30 % on the simulated camera (Question 5).

E4 EDIT Choose the crop

If your digits are smaller or larger than MNIST's, a different crop rescales them. E4 scores your 20 stored pictures for six crop fractions (no new pictures needed), plots the result against the MNIST accuracy, and sets `FRACTION` to the best one; `classify()` uses it from then on.

E4 · EDIT in the notebook

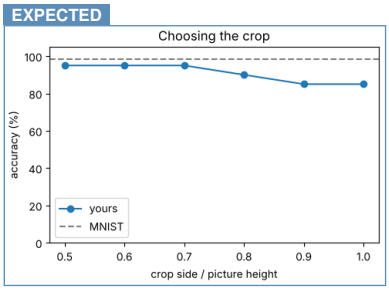
```
fractions = [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
sweep = []     # accuracy (%) for each fraction
if sweep:          # classify() uses the best crop from now on
    FRACTION = fractions[int(np.argmax(sweep))]
    print(f"best crop fraction: {FRACTION} ({max(sweep):.0f} %)")
fig, ax = plt.subplots(figsize=(5.5, 3.3))
ax.plot(fractions[:len(sweep)], sweep, "o-", label="yours")
ax.axhline(mnist_acc, color="gray", ls="--", label="MNIST")
ax.set(xlabel="crop side / picture height", ylim=(0, 105),
       ylabel="accuracy (%)", title="Choosing the crop")
ax.legend()
plt.show()
```

WHAT TO WRITE `sweep` (line 2), a list with the accuracy in percent on your stored pictures for each crop fraction in `fractions`.

HINT `classify(img, fraction=fr)[0]` is the predicted digit of one picture with crop `fr`; compare it with its label for every picture in `zip(pictures, labels)`.

DONE WHEN the plot shows six points and the cell prints the best one, such as `best crop fraction: 0.6 (95 %)`: on the simulated camera the best crop is usually 0.5 to 0.7. This number was chosen on the same pictures it was scored on (Question 6).

EXPECTED



E4 Six points below the dashed MNIST line, often highest at a tight crop (0.5 to 0.7 on the simulated camera).

E5 DO A digit written on the spot

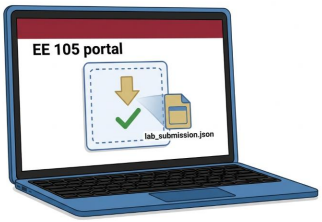
Write a new digit on the spot, run E5, and hold the digit up to the camera. E5 runs the live view once more with the crop of E4, on a picture that crop was not chosen on.

YOU SHOULD SEE the crop fraction in use, and the prediction for your new digit in the title of the live view.

CHECK **Checkpoint 4** You have the crop sweep and the best crop (E4), and a live view that reads a digit you wrote on the spot (E5). You found the domain gap, measured it, and narrowed it.

TRY IT ON THE BOARD If you have time, run E5 again with one thing changed: a thin pen instead of the marker, the paper twice as far away, the room lights dimmed, or the paper turned by about 30°. Change one thing at a time and watch the confidence: a thin pen and dim light usually hurt the most.

Save and upload



Save Upload `lab4.2_submission.json` on the course portal under Lab 4.2.

Save ▸ RUN Write `lab4.2_submission.json`

First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab4.2_submission.json` in the lab folder: your numbers, the labels and predictions of your 20 pictures, and their 20 network inputs (28×28 each, small). Log in at `ee105.org/portal`, open Lab 4.2, and upload that file.

After the lab, unplug the USB cable and move the caps of J7 and J8 back to the Module 1 end (pins 2 and 3, the picture of **A1** in reverse): the other labs read A0 and A1 from the circuits block.

YOU SHOULD SEE `[ok] Saved lab4.2_submission.json`, and then the portal shows your grade. A run on the simulated camera counts as practice and earns no credit.

FINISH THE LAB

You built a camera that reads handwriting: a live camera view in the notebook, one preprocessing for photos and camera pictures, a test set of your own digits with its confusion matrix, and a study of the domain shift. The lab is done when every box is ticked.

- ☐ **B2** ANSWER `FLIP_UD`, `FLIP_LR` (only if needed)
- ☐ **B3** EDIT `chars`, `t_uart`
- ☐ **C3** DO three prediction figures
- ☐ **C4** EDIT `p_hand`
- ☐ **D1** DO 20 pictures, two of each digit, in `my_digits.npz`
- ☐ **D3** EDIT `cam_acc`, `cm`
- ☐ **E2** EDIT `digit_height`, `ink_cam`, `ink_mn`
- ☐ **E4** EDIT `sweep`
- ☐ CHECK Checkpoint 1: your digit as a white-on-black network input
- ☐ CHECK Checkpoint 2: live predictions and three figures
- ☐ CHECK Checkpoint 3: contact sheet, accuracy, confusion matrix
- ☐ CHECK Checkpoint 4: the best crop and a digit read on the spot
- ☐ Questions 1 to 6 answered in the notebook
- ☐ **Save** SAVE `lab4.2_submission.json` uploaded on the portal
- ☐ caps of J7 and J8 back on the Module 1 end

Typical results

The numbers below are expected ranges to compare with; your board, light, and handwriting may differ, so record your own values. Expect thick-marker digits held at the right distance to be classified with 95% or more confidence, and ambiguous ones, such as a 3 with an open top, to come out near 40 to 50% (Figure 4.35). Before any improvement, expect roughly 60 to 90% correct on your 20 pictures, against about 98.5% on MNIST. Expect a 160×120 picture to take a few hundred milliseconds (roughly 250 to 450 ms) and an 80×60 picture to be only a little faster; the live view itself may run at only 1 to 3 pictures per second on a slow laptop, which is fine. Under good light the paper should read roughly 150 to 220 (of 255) and the ink below 80. The next numbers come from the simulated camera: with a crop of 0.8 the digits are roughly 14 to 16 pixels tall in the network input, against 19.7 for MNIST, with roughly 8% stroke pixels against 12%; leaving out the normalization drops the accuracy to roughly 30%; and the best crop fraction is usually 0.5 to 0.7. Figure 4.36 shows a typical contact sheet of **D2**.

Troubleshooting

- *"No Arduino found."* Use a USB data cable (some cables carry only power) and try another USB port.
- *The upload fails.* Double-tap the white RESET button on the Nano (Fig. 4.37), then rerun **A1**.
- *"Camera not found," or the camera sends no picture.* Check that the selector is on Camera, that the caps of J7 and J8 are on the Camera end, and that the camera sits fully in J6. Unplug the USB cable before reseating the camera, then plug it in, press RESET, and connect again (**A1**).
- *The picture is upside down or mirrored.* Set `FLIP_UD` or `FLIP_LR` to `True` in **B2** and rerun it.
- *Dark or washed-out pictures.* Face a window or a lamp rather than pointing the camera at the light, and avoid glare on glossy paper.
- *Wrong predictions.* Move closer or farther until the digit fills about half of the picture height, keep it inside the red square of **B4**, and use the marker, not a pen.
- *"mnist_cnn.pt not found."* The notebook trains a stand-in CNN so you can continue, but your Lab 4.1 network is better: finish Lab 4.1 through step G3 and rerun **C1**.

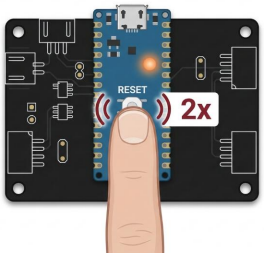


Figure 4.37: An upload that fails or hangs: double-tap the white RESET button on the Nano, then rerun **A1**.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. Report your three predictions of **C3** with their confidences. Explain the confidence values with the softmax. When is a low confidence useful?
2. Compare your camera accuracy (**D3**) with the MNIST test accuracy. Use the figure of **E1** and the numbers of **E2** to describe the distribution shift. What could data augmentation (random shifts, rotations, and scalings of the training digits) cover, and what not?
3. From **B3**: how many characters does one 160×120 picture take, how long would they take at 115 200 baud, and how long did a picture really take? What limits the picture rate?
4. The CNN has 206 922 parameters and needs about 1.22 million MACs per image (Worked Example 4.15). How much memory does it need in float32 and in int8, and how long would one image take on the Nano at 32 million MACs per second? Why does this lab run the network on the laptop?
5. What happened when the normalization was left out (**E3**)? What else must match between training (Lab 4.1) and use (this lab)?
6. The best crop fraction in **E4** was chosen on the same 20 pictures it was scored on. Why is that accuracy too optimistic (Section 4.8)? How could the two pictures of each digit give an honest number?

Summary

- Learned models replace hand-written rules when the input-output relation varies (voice, images, gestures). Features such as RMS and band power shrink raw sensor windows to a few informative numbers.
- A layer computes $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, costing $m \times n$ multiply-accumulates. A neuron applies an activation f (step, sigmoid, tanh, ReLU) to $z = \mathbf{w} \cdot \mathbf{x} + b$.
- A perceptron draws one straight decision boundary: it can do AND, OR, NAND, and NOR but not XOR. A hidden layer with a nonlinear activation re-describes the input so that later layers can separate it.
- A convolutional layer reuses a few small filters at every position: $C_{\text{out}}(C_{\text{in}}k^2 + 1)$ parameters but $HWC_{\text{out}}C_{\text{in}}k^2$ MACs.
- A network is trained by minimizing a cost (MSE or cross-entropy with softmax) with gradient descent, $\mathbf{v} \leftarrow \mathbf{v} - \gamma\nabla C$, in mini-

batches. Backpropagation computes every gradient with one backward pass through the transposed weights.

- Judge a model on data it never trained on: training, validation, and test sets; learning curves for under- and overfitting; the confusion matrix, sensitivity, specificity, precision, and F_1.
- On hardware, count parameters (memory) and MACs (time and energy). Moving weights costs more energy than arithmetic; 8-bit quantization cuts memory by 4. A resistive crossbar computes $\mathbf{I} = \mathbf{G}\mathbf{V}$ in one step through Ohm's law and KCL.
- The camera lab chains the whole system: capture, keep every second pixel, stream, crop, shrink to 28×28, invert, normalize, classify, and report a softmax confidence.

4.12 *Problems*

4.1 ◦ For $\mathbf{W} = \begin{pmatrix} 2 & -1 & 0 \\ 1 & 1 & -3 \end{pmatrix}$, $\mathbf{x} = (3,\, 1,\, 0.5)$, and $\mathbf{b} = (-1,\, 2)$:

(a) compute $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$;

(b) apply ReLU and sigmoid to each entry;

(c) how many MACs did the product take?

4.2 ◦ Dot products and the cost of a layer.

(a) Compute the dot product of $\mathbf{w} = (1,\, 0,\, -2,\, 0.5,\, 3)$ and $\mathbf{x} = (2,\, 7,\, 1,\, -4,\, 0.5)$. How many MACs does it take?

(b) A layer of 32 neurons takes a 64×64 grayscale image as input. How many weights, biases, and MACs per image does it have?

(c) The image is first averaged down to 16×16. By what factor do the MACs of the layer drop?

4.3 ◦ Show that $\sigma'(z) = \sigma(z)[1 - \sigma(z)]$ for the sigmoid of Eq. (4.5). Evaluate $\sigma(0)$, $\sigma'(0)$, $\sigma(4)$, and $\sigma'(4)$, and show that $\sigma(-z) = 1 - \sigma(z)$.

4.4 ◦ A neuron has the inputs $\mathbf{x} = (0.5,\, -1,\, 2)$, the weights $\mathbf{w} = (0.4,\, 0.3,\, -0.2)$, and the bias $b = 0.1$. Find its output with a step, ReLU, sigmoid, and tanh activation.

4.5 ◦ A perceptron uses the rule of Eq. (4.7) with $\mathbf{w} = (2,\, -1)$ and $b = -1.5$.

(a) Find its output for the inputs $(0, 0)$, $(1, 0)$, $(1, 1)$, $(2, 0)$, and $(2, 2)$.

(b) Where does the decision boundary cross the x_1 and x_2 axes? Sketch it and shade the side where the output is 1.

(c) Multiply $\mathbf{w}$ and b by 10. Does the boundary move? What changes if the step is replaced by a sigmoid?

4.6 • A perceptron with three binary inputs and weights $w_1 = w_2 = w_3 = 1$ should output 1 when at least two inputs are 1 (a majority gate).

(a) Find the range of bias b that works with the rule of Eq. (4.7).

(b) Find weights and a bias for a two-input perceptron that computes x_1 AND (NOT x_2).

4.7 • Two layers without activations: $\mathbf{W}^{(1)} = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$, $\mathbf{b}^{(1)} = (1, 0)$, $\mathbf{W}^{(2)} = (1 \quad -1)$, $b^{(2)} = 2$.

(a) Show that the network equals a single layer $y = \mathbf{w} \cdot \mathbf{x} + c$, and find $\mathbf{w}$ and c.

(b) Evaluate the network at $\mathbf{x} = (-4, 1)$ with and without a ReLU after the first layer. What does the difference show?

4.8 • A network has two inputs, three ReLU hidden neurons, and two outputs with a softmax, with

$$\mathbf{W}^{(1)} = \begin{pmatrix} 1 & 0.5 \\ -1 & 2 \\ 0.5 & 1 \end{pmatrix}, \qquad \mathbf{b}^{(1)} = \begin{pmatrix} 0 \\ 1 \\ -0.5 \end{pmatrix},$$
$$\mathbf{W}^{(2)} = \begin{pmatrix} 1 & -1 & 2 \\ -0.5 & 1 & 1 \end{pmatrix}, \qquad \mathbf{b}^{(2)} = \begin{pmatrix} 0.2 \\ 0.4 \end{pmatrix}.$$

For $\mathbf{x} = (2, -1)$:

(a) find the hidden pre-activations and activations;

(b) find the two logits, the softmax probabilities, and the predicted class;

(c) find the cross-entropy cost if the true class is the second;

(d) count the parameters and the MACs.

4.9 • A digit classifier takes a 30×30 grayscale image directly as input, with hidden layers of 32 and 16 ReLU neurons and 10 outputs.

(a) How many weights and biases does it have, and how many MACs per image?

(b) How much memory do the parameters need in float32 and in int8 (in kB, 1 kB = 1024 bytes)?

(c) What is the inference time on the Nano at 32 million MACs per second?

4.10 •• A smaller version of `SmallCNN` (Figure 4.15) has 8 and 16 filters and a hidden layer of 64 neurons, with the same 3×3 filters, padding, and pooling.

(a) Use Eq. (4.10) to find the parameters and MACs of each layer and the totals.

(b) Find the weight memory in float32 and in int8 (1 kB = 1024 bytes), and the inference time at 32 million MACs per second.

(c) The largest intermediate result is the output of the first convolution. How many kB does it need in float32? Do the float32 weights and this buffer fit together in the Nano's 256 kB of RAM?

(d) By what factors are the parameters and the MACs smaller than those of `SmallCNN` (Worked Example 4.15)?

4.11 ◦ The gesture network outputs the logits $(1.0, 3.0, 0.5, 2.0)$ for (circle, idle, left-right, up-down).

(a) Compute the softmax probabilities and the predicted class.

(b) Compute the cross-entropy cost if the true class is up-down, and if it is idle.

4.12 • A sigmoid output neuron has $z = -4$ for an example whose label is $y = 1$: a confident mistake.

(a) Find a.

(b) With the quadratic cost $C = \frac{1}{2}(a - y)^2$, find C and $\partial C/\partial z$.

(c) With the binary cross-entropy of Eq. (4.13), find C and $\partial C/\partial z$.

(d) Take one gradient-descent step on the bias with $\gamma = 1$ for each cost. How far does z move? Which cost learns faster from a confident mistake?

4.13 • For the cost $C(w) = (w - 3)^2$, start at $w_0 = 0$ and take three gradient-descent steps with $\gamma = 0.25$, 0.5, 0.75, and 1.25. For each rate, list w_1, w_2, w_3 and describe the behavior. Show that each step multiplies $w - 3$ by $(1 - 2\gamma)$, and find the range of γ for which the iteration converges.

4.14 ◦ A gesture dataset has 12 000 labeled windows.

(a) Split it 70/15/15 into training, validation, and test sets.

(b) With mini-batches of 32, how many updates make one epoch, and how many windows are in the last batch?

(c) How many updates do 20 epochs take? How many would full-batch gradient descent make in 20 epochs?

4.15 • A single sigmoid neuron $a = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$ is trained with the binary cross-entropy $C = -[y \ln a + (1-y)\ln(1-a)]$.

(a) Show that $\partial C/\partial z = a - y$, so that $\partial C/\partial \mathbf{w} = (a-y)\mathbf{x}$.

(b) For $\mathbf{x} = (1, 2)$, $\mathbf{w} = (0.5, -0.25)$, $b = 0.1$, and $y = 1$, compute a, C, and the gradients.

(c) Take one step with $\gamma = 0.5$ and compute the new a and C.

4.16 • A network computes $\hat{y} = w_2\,\mathrm{ReLU}(w_1 x + b_1) + b_2$ (one ReLU hidden neuron and a linear output) with the cost $C = \tfrac{1}{2}(\hat{y} - y)^2$. Take $x = 2$, $y = 3$, $w_1 = 0.5$, $b_1 = 0.5$, $w_2 = 1.5$, and $b_2 = 0$.

(a) Do the forward pass and find C.

(b) Find all four gradients by backpropagation.

(c) Take one step with $\gamma = 0.1$ and find the new $\hat{y}$ and C.

(d) Repeat (b) with $b_1 = -2$. Why can a ReLU neuron stop learning?

4.17 •• A 2-2-2 network has ReLU hidden neurons, a softmax output, and the cross-entropy cost, with $\mathbf{x} = (1,\, 2)$,

$$\mathbf{W}^{(1)} = \begin{pmatrix} 0.5 & -0.5 \\ 1 & 0.25 \end{pmatrix}, \qquad \mathbf{b}^{(1)} = \begin{pmatrix} 0.75 \\ -1 \end{pmatrix},$$

$$\mathbf{W}^{(2)} = \begin{pmatrix} 1 & -1 \\ -0.5 & 2 \end{pmatrix}, \qquad \mathbf{b}^{(2)} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

and the true class is the second.

(a) Do the forward pass: $\mathbf{h}$, the logits, $\mathbf{p}$, and C.

(b) Using $\boldsymbol{\delta}^{(2)} = \mathbf{p} - \mathbf{y}$ and Eqs. (4.17) and (4.18), find $\partial C/\partial \mathbf{W}^{(2)}$, $\partial C/\partial \mathbf{b}^{(2)}$, $\boldsymbol{\delta}^{(1)}$, $\partial C/\partial \mathbf{W}^{(1)}$, and $\partial C/\partial \mathbf{b}^{(1)}$.

(c) Take one step with $\gamma = 0.2$ and recompute C.

(d) *Python.* Check $\partial C/\partial W^{(1)}_{22}$ numerically as in Section 4.7.

4.18 ◦ A model reaches 99.6% training accuracy and 97.9% test accuracy after 50 epochs, but its test accuracy was already 98.0% at epoch 20 and has not improved since. Is it underfitting, a good fit, or overfitting? Name three changes that would help, and one change that would make it worse.

4.19 ◦ Honest testing.

(a) A student tries 40 combinations of learning rate and hidden-layer size, checks the test accuracy after each, and reports the best, 98.4%. Why is this number too optimistic, and how should the work have been organized?

(b) With only 5000 labeled images for tuning (a separate test set is kept aside), the tuning is done by 5-fold cross-validation. How many images are in each fold, how many are used for training in each round, and how many networks are trained for each setting?

4.20 • A digit classifier for the classes "0," "1," and "7" gives this confusion matrix on 300 test images (rows actual, columns predicted, in the order 0, 1, 7):

$$\begin{pmatrix} 90 & 6 & 4 \\ 10 & 75 & 15 \\ 5 & 20 & 75 \end{pmatrix}.$$

(a) Find the total accuracy.

(b) For class "1," find TP, FN, FP, TN, sensitivity, specificity, precision, and F_1.

(c) Which pair of classes is confused most, and why might that be?

4.21 • A wearable arrhythmia detector outputs a probability for each of 2000 ten-second windows, 100 of which contain an arrhythmia. With a threshold of 0.5 it flags 120 windows, 80 of them correctly. With a threshold of 0.8 it flags 75 windows, 65 of them correctly.

(a) For each threshold find TP, FP, FN, TN, the accuracy, sensitivity, specificity, precision, and F_1.

(b) Which threshold would you use for an alarm that wakes a nurse at night, and which for a screening tool whose positives a cardiologist reviews later? Why?

4.22 • Quantize $\mathbf{w} = (-0.30, 0.08, 0.64, -0.40)$ to signed 8-bit integers with $s = \max|w|/127$. Give the scale, the integers, the dequantized values, and the largest error. Is the error within $s/2$?

4.23 ◦ Memory of a quantized network.

(a) A network has 250 000 parameters. How many kB (1 kB = 1024 bytes) do they take in float32, in int8, and at 4 bits per weight?

(b) The program also needs 40 kB of RAM for activations and buffers. Which of the three versions fit in the Nano's 256 kB

of RAM?

(c) How many kB does the 784-128-10 network need at 4 bits per weight?

4.24 • Quantize $\mathbf{w} = (0.45, -0.80, 0.20)$ and $\mathbf{x} = (1.5, 0.5, 1.0)$ to signed 8-bit integers, each with its own scale $s = \max|\cdot|/127$. Compute the integer dot product, convert it back with $s_w s_x$, and compare with the exact $\mathbf{w} \cdot \mathbf{x}$. What is the largest integer product here, and why is the running sum kept in a 32-bit register for a 784-input neuron?

4.25 •• A wearable runs the 784-15-10 network 4 times per second on the Nano (32 million MACs per second). The processor draws 10 mW while it computes and 5 µW asleep.

(a) Find the time and the energy of one inference.

(b) Find the average power.

(c) How long does a 100 mAh battery at 3.7 V last, counting only the processor?

(d) Find the energy per MAC and compare it with the 0.3 pJ of an 8-bit MAC with a 32-bit sum in Table 4.2. Where does the difference go?

4.26 • The 2×2 crossbar in Figure 4.38 has row voltages $V_1 = 0.15$ V and $V_2 = 0.05$ V. Its columns are held at virtual ground by transimpedance amplifiers with $R_f = 20$ kΩ.

(a) Find the column currents I_1 and I_2.

(b) Find the amplifier output voltages.

(c) Find the total power dissipated in the four cells.

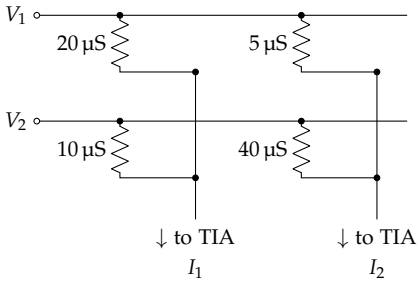


Figure 4.38: Crossbar for Problem 4.26. Each cell is labeled with its conductance.

4.27 •• A layer has signed weights $\mathbf{W} = \begin{pmatrix} 0.5 & -1.0 \\ -0.25 & 0.75 \end{pmatrix}$ (output j in row j, input i in column i). Map it onto a differential crossbar

with $W = (G^{+} - G^{-})/G_0$ and $G_0 = 40\,\mu\text{S}$ per unit weight, using $G = 0$ wherever possible.

(a) Give the G^{+} and G^{-} matrices.

(b) For inputs $\mathbf{V} = (0.1,\ 0.2)$ V, find the two differential currents $I_j^{+} - I_j^{-}$ and check them against $G_0\,\mathbf{WV}$.

(c) Real cells vary by about $\pm 10\%$. Explain qualitatively how this limits the precision of the weights, and one way to compensate.

4.28 • A 64×32 crossbar (64 inputs, 32 outputs) uses row voltages from 0 to 0.1 V and cell conductances from 0 to 20 µS.

(a) Find the largest possible column current.

(b) Choose R_f so that the transimpedance amplifier output never exceeds 1 V in magnitude.

(c) How many MACs does one read of the array perform, and how many MACs per second if a read takes 200 ns?

(d) Each column current is digitized by an 8-bit ADC (256 steps) whose full scale is the current of (a). What current is one ADC step, and what conductance change of a single cell at 0.1 V does it equal? What does this mean for the precision of the weights?

4.29 • The Lab 4.2 firmware reads a 320×240 frame and sends a $w \times h$ picture by keeping frame pixel $(320x/w,\ 240y/h)$ (integer division), one byte per pixel, as base64 text.

(a) For a 160×120 picture, which frame pixels are kept, and what fraction of the frame is sent?

(b) How many base64 characters do a 160×120 and a 120×90 picture take, and how long would each take at 115 200 baud with 10 bits per character?

(c) The notebook crops a center square whose side is 0.8 of the picture height and resizes it to 28×28. Find the crop size and the number of picture pixels per network pixel along each side, for both picture sizes.

(d) A picture actually arrives in a few hundred milliseconds. Explain.

4.30 •• The gesture classifier of Section 4.1 runs on the Nano. It extracts 33 features from each 2-second window and uses a 33-20-10-4 network.

(a) Find its parameters and MACs per window. Repeat for a network that takes the 375 raw samples directly (375-20-10-4). By what factor do features reduce the MACs?

(b) Find the inference time of each at 32 million MACs per second.

(c) The processor draws 15 mW for 2 ms per window to compute features and run the network, and 20 µW asleep the rest of the time. With one window every 2 s (the windows do not overlap), find the average power.

4.31 • *Python.* Implement the perceptron learning rule $\mathbf{w} \leftarrow \mathbf{w} + (y - \hat{y})\mathbf{x}$, $b \leftarrow b + (y - \hat{y})$, with $\hat{y}$ from Eq. (4.7), starting from $\mathbf{w} = (0, 0)$, $b = 0$ and presenting the AND examples in the order $(0,0), (0,1), (1,0), (1,1)$ in every epoch. Report the final weights and bias, and in which epoch the first error-free pass happens. Plot the final decision line with the four points.

4.32 •• *Design.* You may spend at most 16 kB (16 384 bytes) of memory on the parameters of a 784-H-10 network stored in int8 (1 byte per weight and per bias).

(a) Find the largest H that fits, and the memory it uses.

(b) Find its MACs per image and its inference time at 32 million MACs per second.

(c) Using the `MLP` class and `train()` of Lab 4.1, train this network and the 784-8-10 network of Worked Example 4.27 with the Part F settings (Adam, learning rate 0.001, batches of 64, 5 epochs) on all 60 000 training images. Is the accuracy gain worth the memory? Report your measured test accuracies.

5
Communications

In this chapter you will learn to

- Describe a link with Shannon's block model and map each block onto the demo board.
- Encode text as bits with ASCII and UTF-8, and tell bit rate apart from symbol rate (baud).
- Draw on-off keying, Manchester, and PAM-4 waveforms for a byte.
- Estimate received optical power, photocurrent, and the speed of a photodiode receiver built around a transimpedance amplifier.
- Calculate thermal, shot, and dark-current noise and find the signal-to-noise ratio.
- Relate SNR to bit error rate, read an eye diagram, and apply the Shannon capacity limit.
- Frame a message, find its start by correlation, and protect it with parity, a checksum, a repetition code, or a Hamming code.
- Combine imperfect sensors with weighted averaging, a complementary filter, or a Kalman filter.

Put two demo boards edge to edge so that the LED at the edge of one faces the photodiode at the edge of the other. On the first board type the command `communications send` followed by a line of text, such as "This is a test transmission!"; on the second type `communications receive`, and the line of text appears on the second screen. No wire joins the boards and no radio is involved. The LED blinks a few hundred times per second, too fast for your eye to follow, and the photodiode turns each blink back into a voltage that the

Arduino reads as a 1 or a 0.

This is free-space optical communication, often called LiFi (for *light fidelity*). Every piece of it is something you have met before. The LED and photodiode came from Chapter 2, the photodiode readout and the op amp from Chapter 3, and the Python data handling from Chapters 0 and 4. What is new is the question you ask of them. Instead of measuring a quantity, you now want to move information from one place to another, as fast as possible, with as few mistakes as possible. That question leads to bits and symbols, modulation, noise, error rates, and a hard upper limit set by Claude Shannon in 1948.

The last part of the chapter turns the problem around. Several sensors each tell you something imperfect about the world, and you want the single best estimate. That is *sensor fusion*. Your instructor demonstrates it in class with a link that sends one message over a light channel and a sound channel at once.

5.1 *Communicating with Light*

People signaled with light long before they signaled with electricity. Smoke columns carried warnings between hilltops. Ships flashed signal lamps at each other in Morse code. The heliograph, a mirror on a tripod, aimed flashes of sunlight at a receiver many kilometers away. Today almost every long-distance phone call and web page travels as light pulses inside optical fibers. Figure 5.1 lines these up. All four share the idea you will use in the lab: switch the light between states that the receiver can tell apart, and agree in advance on what each state means.

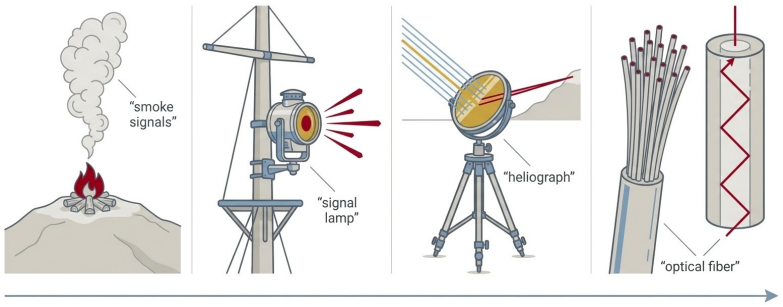


Figure 5.1: Optical communication through history: smoke signals, signal lamps, the heliograph, and optical fiber. Each one switches light between states that a receiver can distinguish.

Worked Example 5.1 *How fast is a smoke signal?*

A smoke signaler sends a 1 as a long puff and a 0 as a short puff. A short puff lasts 0.5 s, a long puff lasts 1.5 s, and the observer needs a gap of at least 0.5 s between puffs to see them as separate. If 1s and 0s are equally likely, what is the average bit rate?

SOLUTION. Each bit is a puff plus a gap. A 0 takes $0.5 + 0.5 = 1.0\,\text{s}$ and a 1 takes $1.5 + 0.5 = 2.0\,\text{s}$. With equal numbers of each, the average time per bit is

$$\bar{T}_b = \tfrac{1}{2}(1.0 + 2.0) = 1.5\,\text{s},$$

so the bit rate is $R_b = 1/\bar{T}_b = 0.67\,\text{bit/s}$. For comparison, the firmware's own link between two boards in Lab 5.1 sends about 267 bit/s, 400 times faster, and a research fiber link in 2024 carried 402 Tbit/s. ■

Guiding light in a fiber

Light travels at $c = 3.00 \times 10^8\,\text{m/s}$ in vacuum and more slowly in matter. The ratio is the *refractive index*,

$$n = \frac{c}{v}. \tag{5.1}$$

Light slows down because its field sets the electrons in the material oscillating, and these tiny dipoles re-radiate waves that combine with the original into a slower one. The response depends on the material and on the light's frequency, so n depends on both. Glass has $n \approx 1.45$ to 1.5, water 1.33, and silicon about 3.5 in the near infrared.

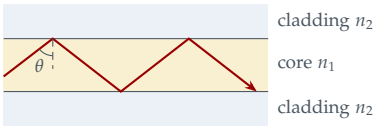


Figure 5.2: Total internal reflection traps light in the core of a fiber when $n_1 > n_2$ and the ray meets the wall at $\theta > \theta_c$.

When light crosses from one material to another it bends according to Snell's law, $n_1 \sin\theta_1 = n_2 \sin\theta_2$, with angles measured from the normal to the surface. Going from a high index to a low index, the ray bends away from the normal. Past the *critical angle*

$$\theta_c = \arcsin\frac{n_2}{n_1} \tag{5.2}$$

no light gets out at all: the interface reflects all of it. An optical fiber exploits this *total internal reflection*. Its core has a slightly higher index than the cladding around it, so a ray that enters nearly parallel to the axis bounces from wall to wall without loss (Figure 5.2). For a core with $n_1 = 1.48$ and cladding with $n_2 = 1.46$, $\theta_c = \arcsin(1.46/1.48) = 80.6°$, so rays within about 9° of the fiber axis stay trapped. Inside the core, light moves at $c/1.48 \approx 2.0 \times 10^8\,\text{m/s}$.

Fiber links reach enormous rates by running many wavelengths side by side in one strand; the 2024 record of 402 Tbit/s used 1505 wavelength channels.

LiFi: dropping the fiber

A fiber needs a cable. Free-space optical communication sends the light through air instead. Indoor LiFi uses the LED lamps that already light a room: each lamp is modulated faster than the eye can see (well

above the 100 Hz to 200 Hz range where flicker becomes visible), and a photodetector in a laptop or phone receives the data (Figure 5.3). The return link usually uses infrared.

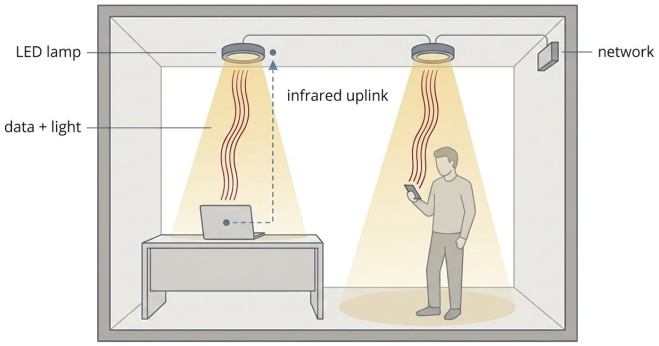


Figure 5.3: LiFi in a room. Ceiling LED lamps light the room and send data at the same time; each cone of light carries a separate data stream to the devices below it.

LiFi complements WiFi rather than replacing it (Table 5.1).[1] Its strengths and weaknesses share one cause: light does not pass through walls. That keeps the signal inside one room, which helps security and lets every room reuse the channel, but the receiver needs a clear line of sight to the lamp.

[1] LiFi was popularized by Harald Haas in a 2011 TED talk. In 2023 the IEEE added light communication to the WiFi family of standards as IEEE 802.11bb.

	WiFi	LiFi
Carrier	radio, 2.4, 5, and 6 GHz bands	visible light, 400 THz to 790 THz
Spectrum	shared, regulated	unregulated, very wide
Walls	passes through	blocked
Coverage	whole home or office	one lit area per lamp
Interference	crowded bands	none with radio; ambient light
Line of sight	not needed	needed (or a reflection)
Where useful	everywhere	hospitals, aircraft, secure rooms

Table 5.1: WiFi and LiFi compared. LiFi trades range for a huge, unregulated spectrum that stays inside one room.

5.2 *The Anatomy of a Communication System*

In 1948 Claude Shannon drew a diagram that still organizes every communication system, from smoke signals to 5G.[2] Figure 5.4 shows a version of it. An *information source* produces a message: text, audio, a picture. The *transmitter* turns the message into a signal that suits the channel. It has two jobs, *encoding* (message to bits) and *modulation* (bits to a physical waveform). The *channel* carries the signal and adds *noise*. The *receiver* undoes the transmitter's work: it *demodulates* the waveform back to bits and *decodes* the bits back to a message for the *destination*.

[2] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, 1948. The same paper defined the bit as the unit of information.

Figure 5.5 maps Shannon's blocks onto the hardware you will use. On the sender, the computer holds the message as numbers. Inside a computer a signal is just an array: a double-precision number can represent every integer from -2^{53} to 2^{53} exactly, so the digital side has far more resolution than any analog stage after it. A *digital-to-analog*

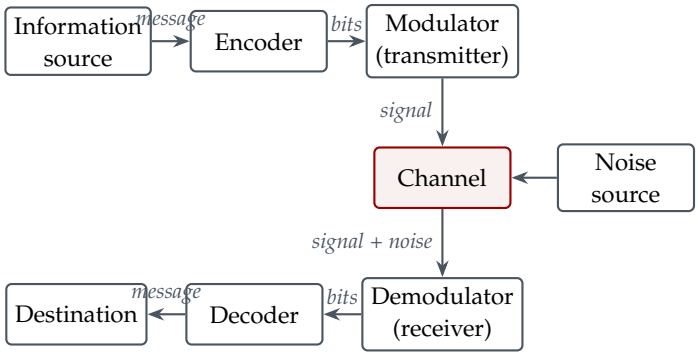


Figure 5.4: Shannon's model of a communication system. The transmitter encodes and modulates, the channel adds noise, and the receiver demodulates and decodes.

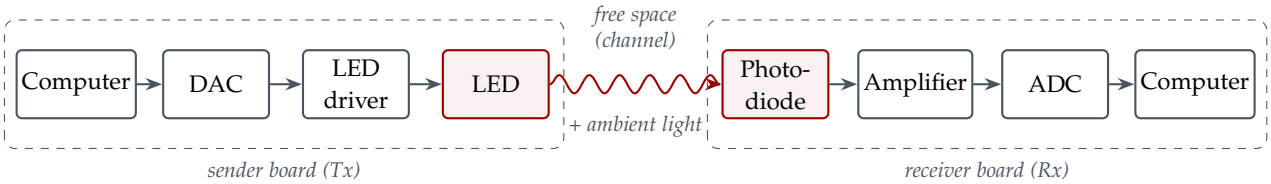


Figure 5.5: The LiFi link on the demo boards. Python or Arduino code produces numbers, a DAC turns them into a voltage, the driver turns the voltage into LED current, and light crosses the gap. The photodiode turns light back into current, the amplifier turns current into voltage, and the ADC turns voltage back into numbers. Every block adds some noise.

converter (DAC) turns each number into a voltage. The demo board uses a 12-bit DAC chip (AD5593R) that the Arduino controls over the I²C bus; the firmware uses about 1.65 mV per DAC code, so code 2000 is about 3.3 V, and higher codes give the same 3.3 V. The LED driver turns that voltage into LED current, and the LED turns current into light. On the demo board the communication LED is *LED pair 2* of the biosensing block, a small side-view white LED driven by DAC channel 4.

On the receiver, the photodiode turns light into current, an amplifier (or simply a resistor) turns current into voltage, and the Arduino's *analog-to-digital converter* (ADC) turns voltage into a number from 0 to 1023. On the demo board the pair 2 photodiode feeds a transimpedance amplifier with a 3 MΩ feedback resistor, and jumper J5 routes its output to pin A6. Each conversion step is a chance to lose information, either by rounding (quantization) or by adding noise.

How many levels does a converter have?

Chapter 3 introduced the ADC; here we need one result from it, the noise that rounding adds. An N-bit converter has 2^N output levels. Spread over a full-scale range V_{FS}, the step between levels, called one *least significant bit* (LSB), is

$$\Delta = \frac{V_{\mathrm{FS}}}{2^N}. \tag{5.3}$$

A converter rounds every value to the nearest level, so the output differs from the true value by up to $\pm\Delta/2$. If the true value is equally

likely to fall anywhere within a step, this *quantization error* behaves like noise with RMS value

$$v_{q,\mathrm{rms}} = \frac{\Delta}{\sqrt{12}}. \tag{5.4}$$

For a full-scale sine wave, the best possible SNR of an ideal N-bit converter works out to $\mathrm{SNR}_{\mathrm{dB}} \approx 6.02N + 1.76$: every extra bit buys about 6 dB.

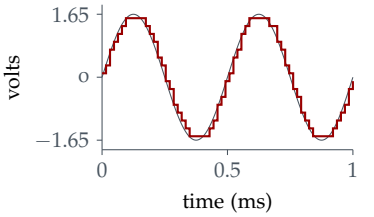


Figure 5.6: A 2 kHz sine wave (thin) and its 4-bit version (thick) on a 3.3 V range: 16 levels 206 mV apart.

Worked Example 5.2 *Resolution of an 8-bit and a 4-bit DAC*

A sine wave is generated with a DAC whose output spans 3.3 V. Compare an 8-bit and a 4-bit DAC: number of levels, step size, and RMS quantization noise.

Solution. The 8-bit DAC has $2^8 = 256$ levels, so

$$\Delta_8 = \frac{3.3\,\mathrm{V}}{256} = 12.9\,\mathrm{mV}, \quad v_q = \frac{12.9\,\mathrm{mV}}{\sqrt{12}} = 3.72\,\mathrm{mV}.$$

The 4-bit DAC has only $2^4 = 16$ levels:

$$\Delta_4 = \frac{3.3\,\mathrm{V}}{16} = 206\,\mathrm{mV}, \quad v_q = \frac{206\,\mathrm{mV}}{\sqrt{12}} = 59.5\,\mathrm{mV}.$$

Four fewer bits make the noise 16 times larger. The 4-bit staircase in Figure 5.6 is plain to see, and its ideal SNR is only $6.02(4) + 1.76 = 25.8$ dB, against 49.9 dB for 8 bits. The same arithmetic applies to the Arduino's 10-bit ADC with a 3.3 V reference: one count is 3.22 mV and the quantization noise is 0.93 mV RMS. ■

5.3 *Information, Bits, and Symbols*

A bit is the smallest unit of information: the answer to one yes-or-no question whose two answers are equally likely. Eight bits make a *byte*, which can take $2^8 = 256$ values. To send text you first need an agreed mapping from characters to bytes.

ASCII and UTF-8

The American Standard Code for Information Interchange (ASCII) assigns a 7-bit number to each English letter, digit, punctuation mark, and a few control codes, 128 codes in all. The printable characters run from 32 (space) to 126 (~). Code 10 is the newline. Table 5.2 lists a few. Two patterns are handy: uppercase letters start at 65 (A), and lowercase letters are exactly 32 higher, which flips one bit.

Char.	Dec.	Hex	Binary
space	32	0x20	00100000
!	33	0x21	00100001
0	48	0x30	00110000
A	65	0x41	01000001
E	69	0x45	01000101
H	72	0x48	01001000
a	97	0x61	01100001
i	105	0x69	01101001
~	126	0x7E	01111110

Table 5.2: A few ASCII codes. Each fits in 7 bits; the leading bit of the byte is 0.

ASCII has no room for *é*, *ñ*, Chinese characters, or emoji. UTF-8 solves this without breaking ASCII. It is the encoding of almost every

web page and the default of Python's `encode()` method. Unicode gives every character a number, its *code point*, written `U+` and four or more hexadecimal digits; *é* is `U+00E9`. UTF-8 stores each code point in one to four bytes (Table 5.3). Every ASCII character keeps its single byte. For other characters the leading 1s of the first byte say how many bytes belong together, every following byte starts with `10`, and the code point's bits fill the places marked *x*. For example, *é* becomes the two bytes `0xC3 0xA9`, so `"café".encode()` is five bytes long even though the word has four characters. The lab notebook turns its message into bytes with exactly this call.

Code points	Bytes	Bit pattern
`U+0000` to `U+007F` (ASCII)	1	`0xxxxxxx`
`U+0080` to `U+07FF`	2	`110xxxxx 10xxxxxx`
`U+0800` to `U+FFFF`	3	`1110xxxx 10xxxxxx 10xxxxxx`
`U+10000` to `U+10FFFF`	4	`11110xxx 10xxxxxx 10xxxxxx 10xxxxxx`

Table 5.3: UTF-8. The code point's bits, most significant first, fill the *x* places; unused high places are 0. Most accented Latin letters need two bytes, most Chinese characters and symbols such as € need three, and emoji need four.

Worked Example 5.3 *Sending "Hi" as bits*

Convert the string `"Hi"` into the bit sequence that a transmitter sends, most significant bit (MSB) first. Then do the same in Python.

Solution. From Table 5.2, `H` is 72 and `i` is 105. Write each in binary with 8 digits:

$$72 = 64 + 8 = 01001000_2, \qquad 105 = 64 + 32 + 8 + 1 = 01101001_2.$$

The transmitted stream is `01001000 01101001`, 16 bits. In Python:

```
bits = ''.join(f'{b:08b}' for b in "Hi".encode())
print(bits)   # 0100100001101001
```

The format `08b` pads each byte to 8 binary digits, so leading zeros are not lost. ■

Worked Example 5.4 *UTF-8 by hand*

(a) Encode *é* (`U+00E9`) and the sun symbol ☀ (`U+2600`) in UTF-8. (b) You send the text `"Café ☀ EE 105"` over the lab's link. How many characters and how many bytes is it? How many bits does the lab's frame hold if 32 marker bits are added (Section 5.8), and how long does it take at 2 ms per bit?

Solution. (a) `U+00E9` lies between `U+0080` and `U+07FF`, so it takes two bytes, which have 11 places for bits. In binary, `0xE9` is `1110 1001`; padded to 11 bits it is `00011 101001`. The first five bits go into `110xxxxx` and the last six into `10xxxxxx`:

$$110\,00011 = \texttt{0xC3}, \qquad 10\,101001 = \texttt{0xA9}.$$

U+2600 lies between U+0800 and U+FFFF, so it takes three bytes with $4+6+6=16$ places. Its 16 bits are 0010 011000 000000, giving

$$1110\,0010 = \texttt{0xE2}, \quad 10\,011000 = \texttt{0x98}, \quad 10\,000000 = \texttt{0x80}.$$

(b) The text has 13 characters: 11 ASCII characters (C, a, f, three spaces, E, E, 1, 0, 5) at one byte each, *é* at two bytes, and ☀ at three, so $11+2+3=16$ bytes, or 128 bits. With the 32 marker bits the frame holds 160 bits, which take $160\times 2\,\text{ms} = 0.32\,\text{s}$. In Python, `len(text)` returns 13 and `len(text.encode())` returns 16. ■

The order of bits inside a byte is a convention, and both ends must agree on it. The lab notebook sends the MSB first. The firmware's own text link, which the lab calls *Task 1*, tests `b & (0x01 << i)` for $i = 0, 1, \ldots, 7$, so it sends the *least* significant bit first. A receiver that assumes the wrong order reads H (01001000) as 00010010, which is a control code, not a letter.

Bit rate and baud

The *bit rate* R_b counts bits per second. The *symbol rate* R_s, measured in *baud*, counts how many times per second the transmitter changes to a new symbol, that is, to a new light level, tone, or pulse. If each symbol is one of M possible levels, it carries $\log_2 M$ bits and

$$R_b = R_s \log_2 M. \tag{5.5}$$

With two levels (on and off), $M=2$ and bit rate equals baud. With four levels, each symbol carries two bits and the bit rate doubles at the same baud. Faster symbols need more bandwidth; more levels need less noise. Section 5.7 makes that trade-off precise.

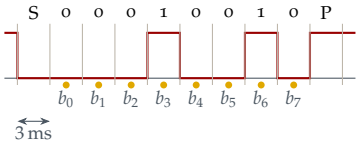


Figure 5.7: One Task 1 frame for H (0x48), sent LSB first: start slot S (LED off), bits b_0 to b_7, stop slot P (LED on). Dots mark where the receiver samples.

The Task 1 link lives in the course firmware: `communications send` drives LED pair 2, and `communications receive` watches the photodiode on A6. The heart of it is two short functions. `PERIOD` is 3 ms, code 2000 is full drive (3.3 V), and `light_on()` returns 1 when `analogRead(A6)` exceeds a fixed threshold of 200 counts, about 0.64 V:

```
void send_byte(char b){
  setDACVal(dac_led2, 0);              // start slot: LED off
  delay(PERIOD);
  for(int i = 0; i < 8; i++){          // LSB first
    setDACVal(dac_led2, (b & (0x01 << i)) ? 2000 : 0);
    delay(PERIOD);
  }
  setDACVal(dac_led2, 2000);           // stop slot: LED on
  delay(PERIOD);
}
char get_byte(){                       // on an on-to-off edge
  char ret = 0;
  delay(PERIOD*1.5);                   // to the middle of bit 0
```

```
  for(int i = 0; i < 8; i++){
    ret = ret | (light_on() << i);
    delay(PERIOD);
  }
  return ret;
}
```

WORKED EXAMPLE 5.5 *Timing of the Task 1 link*

The Task 1 firmware sends each character as a frame of ten slots, each lasting `PERIOD` = 3 ms: one start slot with the LED off, eight data bits, and one stop slot with the LED on (Figure 5.7). It sends the test string `"This is a test transmission!\n"`, 29 characters including the final newline. Find the symbol rate, the useful bit rate, and the time to send the string once.

SOLUTION. Every slot is one symbol, so the symbol rate is

$$R_s = \frac{1}{3\,\text{ms}} = 333\,\text{baud}.$$

Each character takes $10 \times 3\,\text{ms} = 30\,\text{ms}$ but carries only 8 data bits, so the useful bit rate is

$$R_b = \frac{8\ \text{bits}}{30\,\text{ms}} = 267\,\text{bit/s},$$

or 33 characters per second. The framing costs 2 of every 10 slots, an efficiency of 80%. The whole string takes $29 \times 30\,\text{ms} = 0.87\,\text{s}$; the firmware then pauses 1 s with the LED on before repeating it. ■

WORKED EXAMPLE 5.6 *Bit rate and baud of the lab link*

In Parts D to F of Lab 5.1 the notebook sends the 45-byte message file as one frame of 392 bits: 16 marker bits, 360 data bits, and 16 more marker bits. Find the baud, the bit rate, the time to send the frame, and the useful bit rate (data bits per second) for (a) OOK at 4 ms per symbol, (b) PAM-4 (two bits per symbol, Section 5.4) at 4 ms per symbol, and (c) OOK at 0.5 ms per symbol, the fastest the lab tries.

SOLUTION. (a) $R_s = 1/4\,\text{ms} = 250\,\text{baud}$, and with one bit per symbol $R_b = 250\,\text{bit/s}$. The frame takes $392 \times 4\,\text{ms} = 1.568\,\text{s}$, so the useful rate is $360/1.568\,\text{s} = 230\,\text{bit/s}$, the same as $250 \times 360/392$.

(b) Still 250 baud, but $R_b = 250 \log_2 4 = 500\,\text{bit/s}$. The frame needs only 196 symbols and takes 0.784 s; the useful rate is 459 bit/s.

(c) $R_s = 2000\,\text{baud}$ and $R_b = 2000\,\text{bit/s}$. The frame takes 0.196 s, for a useful rate of 1837 bit/s, seven times the Task 1 link. Whether the receiver can keep up at this speed is the subject of Section 5.5.

5.4 Putting Bits on Light: Modulation

Modulation is the act of changing some property of a *carrier* so that it follows the message. A radio changes the amplitude, frequency, or phase of an electromagnetic wave. An LED gives you one property that is easy to control and easy to detect: its brightness. Changing the optical power to carry data is called *intensity modulation*, and receiving it with a photodiode, which responds to power and ignores the phase of the light, is called *direct detection*. Every scheme below is a way of choosing brightness levels over time.

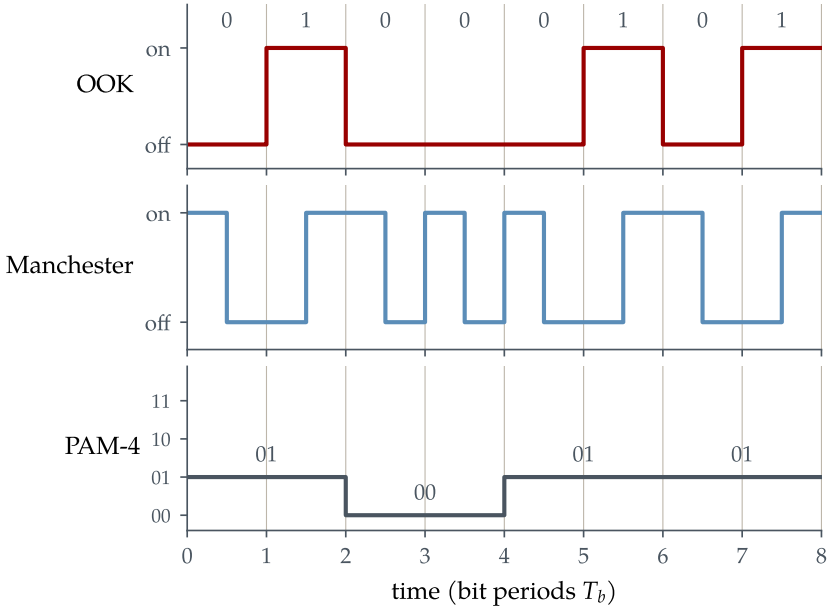


Figure 5.8: The letter `E` (`01000101`, MSB first) in three line codes. On-off keying uses one bit period per bit. Manchester puts a transition in the middle of every bit (here a 1 rises and a 0 falls). PAM-4 sends the bits in pairs, one of four levels per symbol, so it needs half as many symbols.

On-off keying

The simplest scheme turns the LED on for a 1 and off for a 0, holding each level for one bit period T_b. This is *on-off keying* (OOK), the two-level case of *amplitude-shift keying* (ASK). Because the level stays put for the whole bit, it is also called *non-return-to-zero* (NRZ). Task 1 and Part D of the lab use OOK with the LED fully off for a 0 and fully on for a 1. The "off" level does not need to be completely dark: an earlier version of the lab switched between DAC codes 1750 and 2000, so the LED was dim for a 0 and bright for a 1. What matters is that the receiver can tell the two levels apart.

OOK has two weaknesses with long runs of identical bits. A run has no transitions, so a receiver whose clock runs slightly fast or slow gradually loses track of where bits begin. And the average brightness depends on the data, which shows up as flicker in a lamp and as a shifting baseline at the receiver.

Manchester coding

Manchester coding fixes both weaknesses by splitting every bit into two halves with opposite levels. In the convention of Figure 5.8 (used by Ethernet), a 1 is low then high and a 0 is high then low.[3] Every bit therefore has a transition in its middle that the receiver can lock onto, and the LED is on exactly half the time no matter what the data are. The price is bandwidth: the waveform changes up to twice per bit, so the symbol rate is $2R_b$ and the link needs about twice the bandwidth of OOK for the same bit rate.

[3] Some textbooks use the opposite convention. Only the agreement between sender and receiver matters.

Multilevel ASK (PAM-4)

Instead of two brightness levels you can use M levels, so each symbol carries $\log_2 M$ bits. This is *pulse-amplitude modulation* (PAM-M), also called M-level ASK. Part F of the lab uses PAM-4 and maps bit pairs to four brightness levels, numbered from dark to bright:

$$\texttt{00} \to 0, \quad \texttt{01} \to 1, \quad \texttt{10} \to 2, \quad \texttt{11} \to 3.$$

Level 0 is the LED off and level 3 is full drive; the two middle levels are drive voltages chosen so that the *received* levels are evenly spaced (Worked Example 5.8). The earlier version of the lab used DAC codes 1750, 1800, 1850, and 2000 for the four levels. At the same symbol rate, PAM-4 doubles the bit rate. The catch is that the levels are now closer together for the same total brightness swing, so less noise is needed to push a symbol across a decision boundary.

Noise usually moves a symbol only to a neighboring level. With the natural binary mapping above, the step from `01` to `10` flips both bits. *Gray coding* orders the levels `00`, `01`, `11`, `10` instead, so that neighbors differ in exactly one bit and a typical symbol error costs one bit. With Gray coding and $\log_2 M$ bits per symbol, BER $\approx P_s / \log_2 M$, where P_s is the symbol error probability.

Pulse-width, pulse-position, and frequency coding

Information can also ride on *timing*. In *pulse-width* coding a short pulse means 0 and a long pulse means 1; the sound channel of the fusion demo shown in class sends a short beep (a “dot”) for 0 and a beep three times longer (a “dash”) for 1, borrowing from Morse code. In

pulse-position modulation (PPM) every symbol is a fixed-length slot with one short pulse, and the pulse's position inside the slot carries the bits; with four possible positions, each pulse carries two bits. PPM keeps the average power low, which is why infrared remote controls and the old IrDA laptop ports used it. Finally, *frequency-shift keying* (FSK) assigns a different tone to each symbol. A multi-tone mode of the class demo sends 4 bits per symbol by combining two beep lengths with eight buzzer frequencies ($2 \times 8 = 16 = 2^4$ symbols).

> **Worked Example 5.7** *Line codes for the letter* `E`
>
> Write the waveforms that send `E` in OOK, Manchester, and PAM-4, MSB first, using the lab's PAM-4 mapping. How many symbols does each need, and what is the bit rate of each if every symbol (or half-bit, for Manchester) lasts 4 ms?
>
> Solution. `E` is $69 = \texttt{01000101}_2$.
>
> *OOK*: one symbol per bit, levels off, on, off, off, off, on, off, on. That is 8 symbols, and $R_b = 1/4\,\text{ms} = 250\,\text{bit/s}$.
>
> *Manchester*: each bit becomes two half-bit levels, $0 \to$ high-low and $1 \to$ low-high:
>
> $$\underbrace{\text{HL}}_{0}\ \underbrace{\text{LH}}_{1}\ \underbrace{\text{HL}}_{0}\ \underbrace{\text{HL}}_{0}\ \underbrace{\text{HL}}_{0}\ \underbrace{\text{LH}}_{1}\ \underbrace{\text{HL}}_{0}\ \underbrace{\text{LH}}_{1}.$$
>
> That is 16 half-bit symbols. If each half-bit lasts 4 ms, a full bit takes 8 ms and $R_b = 125\,\text{bit/s}$.
>
> *PAM-4*: split into pairs `01`, `00`, `01`, `01`, which map to levels 1, 0, 1, 1. That is 4 symbols, and $R_b = 2/4\,\text{ms} = 500\,\text{bit/s}$.
>
> Figure 5.8 draws all three at the same *bit* rate instead, so the Manchester levels last half a bit and the PAM-4 levels two bits. For the same symbol duration, PAM-4 is four times faster than Manchester, but it needs the cleanest channel. ■

5.5 The Optical Link

Now follow the light from the sender's LED to the receiver's ADC (Figure 5.9). Three questions decide whether the link works: how much light arrives, how much current it makes, and how quickly the receiver can follow changes.

The transmitter

Recall from Chapter 2 that an LED's current rises exponentially with its voltage once the voltage passes a turn-on value, and its light output

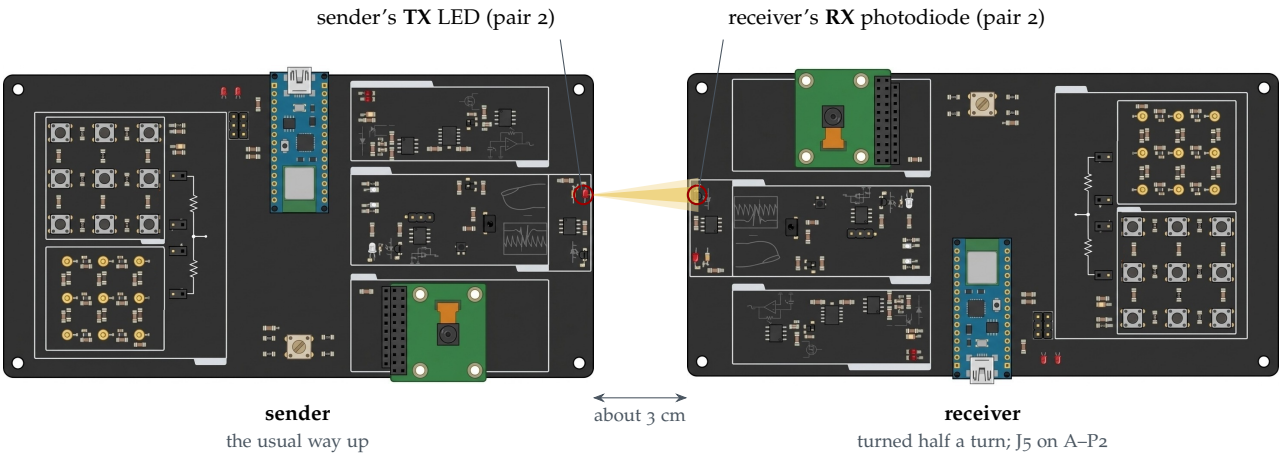


Figure 5.9: The Lab 5.1 setup seen from above (the USB cables to the laptop are not shown). The boards lie with their right edges facing each other, the receiver turned half a turn, and the TX LED (pair 2) at the edge of the sender shines on the RX photodiode at the edge of the receiver a few centimeters away. In this position the receiver's TX also faces the sender's RX, so the two boards can swap roles without moving.

is roughly proportional to its current. A small change in drive voltage near turn-on therefore gives a large change in light, and the light-versus-voltage curve is not a straight line. Higher up, the resistance in series with the LED (including the DAC's own output resistance) and the 3.3 V limit of the DAC flatten it again. Figure 5.10 shows a measurement from the demo boards: DAC codes 1750, 1800, 1850, and 2000 (2.89, 2.97, 3.05, and 3.30 V) produced receiver readings of about 42, 76, 120, and 174 ADC counts, which are not evenly spaced. Light first appears near 2.75 V. Part C of the lab measures this curve, called the *staircase*, on your own boards.

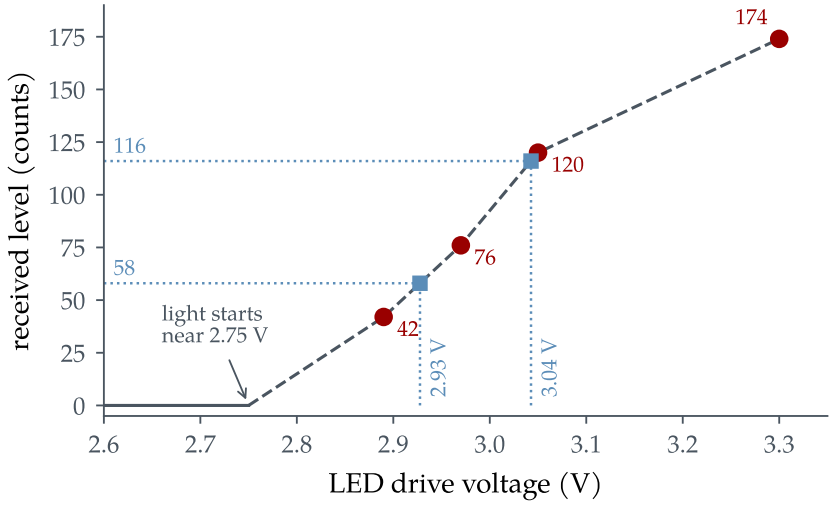


Figure 5.10: Received level versus LED drive voltage for two demo boards a few centimeters apart (dots, measured). Reading the curve backwards gives the drive voltages for evenly spaced received levels (squares, Worked Example 5.8).

Worked Example 5.8 *Choosing PAM-4 drive levels from the staircase*

Use the four measured points of Figure 5.10 to choose PAM-4 drive voltages whose received levels are evenly spaced between the LED off (0 counts) and full drive (174 counts at 3.30 V). Where do the decision thresholds go, and how much larger is the smallest gap than with the four DAC codes 1750 to 2000?

Solution. Evenly spaced levels from 0 to 174 counts are 0, 58, 116, and 174 counts. Level 0 is the LED off (0 V) and level 3 is 3.30 V. For the middle levels, interpolate linearly between neighboring measured points. 58 counts lies between $(2.89\,\text{V}, 42)$ and $(2.97\,\text{V}, 76)$:

$$V_1 = 2.89 + 0.08 \times \frac{58 - 42}{76 - 42} = 2.93\,\text{V}.$$

116 counts lies between $(2.97\,\text{V}, 76)$ and $(3.05\,\text{V}, 120)$:

$$V_2 = 2.97 + 0.08 \times \frac{116 - 76}{120 - 76} = 3.04\,\text{V}.$$

The thresholds go halfway between neighboring levels, at 29, 87, and 145 counts. Every gap is now 58 counts. The four DAC codes gave levels 42, 76, 120, and 174, whose smallest gap is $76 - 42 = 34$ counts, so the new choice opens the tightest gap by a factor of $58/34 = 1.7$. The gain comes from using the full swing down to the LED off and from undoing the bend of the LED curve. Note that the drive voltages 0, 2.93, 3.04, and 3.30 V are far from evenly spaced; the lab notebook finds them the same way, with `np.interp`. ■

The color of the light matters as much as its amount. The demo board's LEDs are side-view white LEDs (ROHM CSL0416WBCW1): a blue LED chip coated with a yellow phosphor, which turns part of the blue light into a broad band of yellow and red. The board's photodiode, a Vishay VEMD2023SLX01, has a daylight-blocking filter and responds only from about 750 nm to 1050 nm, peaking at 940 nm; it is meant for infrared emitters at 830 to 950 nm. Only the red tail of the white LED's phosphor light falls inside that band, so the optical coupling between the boards is weak. That is one reason the receiver needs a transimpedance gain as large as 3 MΩ.

An LED also has a speed limit set by carrier recombination and its capacitance. Indicator LEDs respond in tens of nanoseconds. White LEDs are slower, because their phosphor keeps glowing for roughly a hundred nanoseconds after the blue light changes, which limits them to a few megahertz; LiFi receivers often filter for the fast blue light. On the demo board the photodiode sees only the phosphor's light,

but even that is far faster than the receiver's own time constant of a few hundred microseconds (Worked Example 5.12).

The channel: spreading and pointing

An LED emits into a wide cone. Most LEDs are close to *Lambertian*: their intensity falls off as $\cos^m \phi$ away from the axis, where the order m is set by the half-power angle $\phi_{1/2}$ through $m = -\ln 2 / \ln(\cos \phi_{1/2})$. A plain Lambertian emitter with $\phi_{1/2} = 60^\circ$ has $m = 1$. The power falling on a detector of area A at distance d (Figure 5.11) is then

$$P_r = P_t \, \frac{m+1}{2\pi d^2} \, \cos^m \phi \; A \cos \psi. \tag{5.6}$$

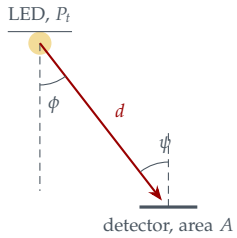


Figure 5.11: Geometry of a line-of-sight optical link: emission angle ϕ at the LED, incidence angle ψ at the detector, distance d.

Two features dominate. The $1/d^2$ factor is the inverse-square law: the same power spreads over a sphere whose area grows as d^2, so doubling the distance quarters the received power. The cosine factors punish misalignment: tilting either board reduces the signal, and outside the photodiode's field of view the signal vanishes.

The channel also adds light you did not send. Sunlight and room lights put a large, steady *ambient* photocurrent on top of the signal, and lamps running from the 60 Hz mains flicker at 120 Hz. Ambient light shifts the zero level, so a fixed threshold that worked in a dark room fails near a window. It can also saturate the receiver, and as you will see in Section 5.6, it adds shot noise even though it carries no data. The photodiode on the demo board (Chapter 2) fights this with its daylight-blocking filter, which passes only near-infrared light from about 750 nm to 1050 nm. It rejects much of the visible light of fluorescent and LED room lamps. Sunlight and incandescent bulbs, though, contain plenty of near infrared, so a sunny window or a desk lamp with a filament still reaches the receiver. The photodiode's sensitive area is 0.23 mm^2 and its half-sensitivity angle is $\pm 35^\circ$.

Worked Example 5.9 *Link budget for an infrared link*

An infrared LED with $m = 1$ emits $P_t = 5$ mW of optical power toward a photodiode of area $A = 0.23$ mm^2 and responsivity $\mathcal{R} = 0.6$ A/W at the LED's wavelength, 3 cm away and on axis. (These are representative values for an infrared pair, not data for the demo board, whose white LED puts only a small part of its light into the photodiode's band.) Find the received power and photocurrent. What happens at 6 cm, and at 3 cm if both boards are tilted so that $\phi = \psi = 30^\circ$?

Solution. With $m = 1$ and $\phi = \psi = 0$, Eq. (5.6) becomes $P_r =$

$P_t A/(\pi d^2)$:

$$P_r = \frac{(5\times10^{-3}\,\mathrm{W})(0.23\times10^{-6}\,\mathrm{m^2})}{\pi\,(0.03\,\mathrm{m})^2} = 4.07\times10^{-7}\,\mathrm{W} = 0.407\,\mu\mathrm{W}.$$

The photocurrent is $I_{ph} = \mathcal{R}P_r = (0.6)(0.407\,\mu\mathrm{W}) = 0.244\,\mu\mathrm{A}$. Only about one part in 12,000 of the emitted light reaches the detector.

At 6 cm the distance doubles, so the power drops by 4: $P_r = 0.102\,\mu\mathrm{W}$ and $I_{ph} = 0.061\,\mu\mathrm{A}$.

Tilted by 30° at both ends, the factor $\cos\phi\cos\psi = (0.866)^2 = 0.75$ applies: $P_r = 0.305\,\mu\mathrm{W}$ and $I_{ph} = 0.183\,\mu\mathrm{A}$. Moving the boards apart hurts more than tilting them a little. ■

The photodiode

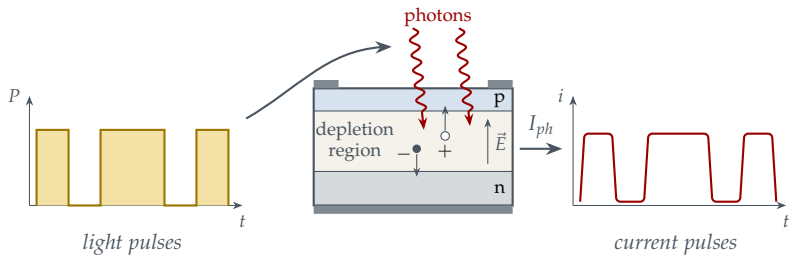


Figure 5.12: A photodiode turns light pulses into current pulses. Each absorbed photon creates an electron-hole pair in the depletion region. The junction's field sweeps the electron to the n side and the hole to the p side, so the external circuit sees a current that follows the light.

A photodiode is a PN or PIN junction operated with zero or reverse bias (Chapter 2). Each absorbed photon can create an electron-hole pair, and the electric field in the depletion region sweeps the pair apart, producing a current proportional to the optical power (Figure 5.12):

$$I_{ph} = \mathcal{R}\,P_r, \tag{5.7}$$

where the *responsivity* $\mathcal{R}$ is typically 0.4 A/W to 0.65 A/W for silicon in the red and near infrared. A small *dark current* I_d flows even in complete darkness, from carriers generated by heat; it grows with temperature and with reverse bias. Reverse bias widens the depletion region, which lowers the junction capacitance and makes the diode faster.

The receiver front end: from current to voltage

The ADC measures voltage, so the photocurrent must be converted. The simplest converter is a resistor: $V = I_{ph}R$. A bigger resistor gives a bigger voltage, but it also makes the receiver slower, because the photodiode's capacitance C_{pd} must charge through R with time constant $\tau = RC_{pd}$. Gain and speed fight each other.

The *transimpedance amplifier* (TIA) in Figure 5.13 breaks that link. It is the inverting op-amp circuit from Chapter 3 with the input resistor replaced by the photodiode. Negative feedback holds the inverting input at virtual ground, so the voltage across the photodiode never changes and its capacitance never has to charge. All of the photocurrent flows through the feedback resistor, and

$$v_{\text{out}} = I_{ph} R_f. \tag{5.8}$$

The gain, in volts per ampere (ohms), is R_f; hence the name *trans*-impedance. The small feedback capacitor C_f keeps the amplifier stable and sets its bandwidth.

A first-order RC low-pass response with time constant $\tau = R_f C_f$ has

$$f_{3\text{dB}} = \frac{1}{2\pi\tau}, \qquad t_r = 2.2\,\tau = \frac{0.35}{f_{3\text{dB}}}, \tag{5.9}$$

where t_r is the 10%-to-90% rise time of the response to a step of light. An OOK bit must last long enough for the signal to settle near its final value, so a useful rule of thumb is $T_b \gtrsim 2t_r$, or

$$R_{b,\max} \approx \frac{1}{2t_r} \approx 1.4\, f_{3\text{dB}}. \tag{5.10}$$

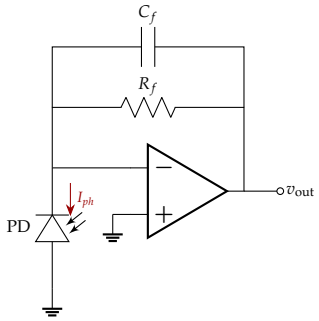


Figure 5.13: Transimpedance amplifier. The op amp holds the photodiode at virtual ground, so all of I_{ph} flows through R_f and $v_{\text{out}} = I_{ph} R_f$.

KEY IDEA The receiver's speed is set by an RC time constant. Shorter bits need a smaller $R_f C_f$, which means less gain.

WORKED EXAMPLE 5.10 *Designing a TIA for an OOK link*

A photodiode produces 2 µA when the sender's LED is on and 0.2 µA from ambient light alone. Choose R_f so the "on" level gives about 1 V more than the "off" level, then find the bandwidth, rise time, and approximate maximum OOK bit rate if $C_f = 1$ pF.

SOLUTION. The signal swing is $\Delta I = 2 - 0.2 = 1.8\,\mu\text{A}$, so $R_f = 1\,\text{V}/1.8\,\mu\text{A} = 556\,\text{k}\Omega$. The nearest standard value is 560 kΩ, giving

$$v_{\text{on}} = (2\,\mu\text{A})(560\,\text{k}\Omega) = 1.12\,\text{V},$$
$$v_{\text{off}} = (0.2\,\mu\text{A})(560\,\text{k}\Omega) = 0.112\,\text{V}.$$

The time constant is $\tau = (560\,\text{k}\Omega)(1\,\text{pF}) = 0.56\,\mu\text{s}$, so

$$f_{3\text{dB}} = \frac{1}{2\pi(0.56\,\mu\text{s})} = 284\,\text{kHz}, \qquad t_r = 2.2\tau = 1.23\,\mu\text{s},$$

and $R_{b,\max} \approx 1/(2t_r) \approx 406\,\text{kbit/s}$. As a check on noise, the thermal noise of R_f filtered by C_f integrates to $\sqrt{kT/C_f} = 64.3\,\mu\text{V}$ RMS, independent of R_f: a larger R_f raises the noise density but narrows the bandwidth by the same factor. ■

The demo board makes the opposite choice. Its receiver uses $R_f = 3\,\text{M}\Omega$, so a photocurrent of only 1.1 µA already drives the output to

3.3 V. That large gain lets two boards talk across several centimeters with a weak LED, but it costs speed, as the next two examples show.

Worked Example 5.11 *From LED power to ADC counts*

A sender's LED puts $P_t = 2\,\text{mW}$ into the photodiode's band, with a half-power angle of 30°. The receiver's photodiode ($A = 0.23\,\text{mm}^2$, $\mathcal{R} = 0.6\,\text{A/W}$ in that band) sits 5 cm away on the LED's axis and feeds a TIA with $R_f = 3\,\text{M}\Omega$, read by a 10-bit ADC over 3.3 V, as on the demo board. (The LED numbers are representative of an infrared LED, not measured for the board's white LED.) Find the Lambertian order m, the received power, the photocurrent, the TIA output, and the ADC reading. How close must the boards be for the reading to pass the 200-count threshold of the Task 1 receiver?

Solution. The half-power angle sets the order:

$$m = -\frac{\ln 2}{\ln(\cos 30^\circ)} = -\frac{0.693}{\ln 0.866} = \frac{0.693}{0.144} = 4.82.$$

A narrower beam has a larger m and puts more power on the axis. With $\phi = \psi = 0$, Eq. (5.6) gives

$$\begin{aligned} P_r &= P_t \,\frac{m+1}{2\pi d^2}\, A \\ &= (2\times 10^{-3}\,\text{W})\,\frac{5.82}{2\pi(0.05\,\text{m})^2}\,(0.23\times 10^{-6}\,\text{m}^2) = 0.170\,\mu\text{W}. \end{aligned}$$

Then $I_{ph} = \mathcal{R}P_r = 0.102\,\mu\text{A}$, $v_{\text{out}} = I_{ph}R_f = 0.307\,\text{V}$, and with 3.22 mV per count the ADC reads about $0.307/0.00322 = 95$ counts. A plain Lambertian LED ($m = 1$) would give $(m+1) = 2$ instead of 5.82, only a third as much.

The reading scales as $1/d^2$, so it reaches 200 counts at

$$d = 5\,\text{cm} \times \sqrt{95.2/200} = 3.45\,\text{cm}.$$

The real boards couple even less light, because only the red tail of the white LED gets through the photodiode's filter, and they reach about 150 to 250 counts only when a few centimeters apart. Either way, Task 1, with its fixed threshold, works only when the boards are close. ■

Worked Example 5.12 *How fast is the demo board's receiver?*

Step E2 of Lab 5.1 records the falling edge when the LED switches off. Suppose the received level takes 0.65 ms to fall from 90% to 10% of the step. (a) Find the time constant, the −3 dB bandwidth, and

the rule-of-thumb maximum OOK bit rate. (b) With $R_f = 3\,\mathrm{M\Omega}$, what feedback capacitance (including stray capacitance) does this imply? (c) The lab sends bits of 4, 2, 1, and 0.5 ms. Which satisfy $T_b \geq 2t_r$, and how far does the output get through a full step by the end of one bit?

SOLUTION. (a) For a first-order response the fall time equals the rise time, so $\tau = t_r/2.2 = 0.65\,\mathrm{ms}/2.2 = 0.295\,\mathrm{ms}$. Then

$$f_{3\mathrm{dB}} = \frac{1}{2\pi\tau} = 539\,\mathrm{Hz}, \qquad R_{b,\max} \approx \frac{1}{2t_r} = \frac{1}{2(0.65\,\mathrm{ms})} = 769\,\mathrm{bit/s}.$$

(b) $C_f = \tau/R_f = 0.295\,\mathrm{ms}/3\,\mathrm{M\Omega} = 98\,\mathrm{pF}$, about 100 pF.

(c) $2t_r = 1.3\,\mathrm{ms}$, so only the 4 and 2 ms bits pass the rule. After one bit the output has covered $1 - e^{-T_b/\tau}$ of a step: practically all of it at 4 ms, 0.999 at 2 ms, 0.966 at 1 ms, and 0.816 at 0.5 ms. Even at 0.5 ms a 1 that follows a 0 gets 82% of the way up, so OOK still works if the receiver samples late in the bit. The rule of thumb is conservative; the eye diagram of Section 5.7 shows exactly how much margin remains. ■

5.6 Noise

NOISE IS ANYTHING IN YOUR SIGNAL THAT YOU DID NOT WANT. In a measurement it blurs the value you read. In a communication link it can push a received level across a decision threshold and turn a 1 into a 0. Some noise comes from outside the circuit (ambient light, power-line hum, a nearby motor). Some comes from the physics of the components themselves and cannot be removed, only managed. This section covers the second kind.

Three fundamental mechanisms

Thermal noise (also called Johnson-Nyquist noise) comes from the random thermal motion of charge carriers in any resistor. The carriers jiggle even when no current flows, so the voltage across the resistor fluctuates around zero. Over a bandwidth B its RMS value is

KEY EQUATION
$v_{\mathrm{th}} = \sqrt{4kTRB}$
$i_{\mathrm{sh}} = \sqrt{2qIB}$
thermal and shot noise over bandwidth B

$$v_{\mathrm{th}} = \sqrt{4kTRB}, \tag{5.11}$$

where $k = 1.38 \times 10^{-23}\,\mathrm{J/K}$ is Boltzmann's constant and T is the absolute temperature in kelvin. Thermal noise grows with T, R, and B. It is *white*: its power is spread evenly over frequency, with density $4kTR$ in $\mathrm{V^2/Hz}$. The same noise can be written as a current source $i_{\mathrm{th}} = \sqrt{4kTB/R}$ in parallel with R.

Shot noise arises because current is made of discrete charges. When electrons cross a junction, or photons arrive at a detector, they arrive at random times, like raindrops on a roof. Even a perfectly steady average current I fluctuates, with RMS value

$$i_{\text{sh}} = \sqrt{2qIB}, \tag{5.12}$$

where $q = 1.602 \times 10^{-19}\,\text{C}$. Shot noise grows with the square root of the current and does not depend on temperature. The number of arrivals in a time window follows Poisson statistics, which is where the square root comes from. In a photodiode, every source of current contributes: the signal photocurrent, the ambient photocurrent, and the dark current.

Flicker noise, or $1/f$ noise, comes from defects and traps in materials that capture and release carriers at random. Its power density falls roughly as $1/f^{\alpha}$ with α between 0.8 and 1.2, so it dominates at low frequencies (often below a few kilohertz) and fades at high frequencies. It is one reason why modulating a signal to a higher frequency, away from DC, makes it easier to measure.

Noise along the link

Each block in Figure 5.5 adds its own mix of noise, and Worked Example 5.13 walks the chain block by block. Besides the three mechanisms above, you will meet quantization noise (Eq. (5.4)), *clock jitter* (sampling or updating at slightly wrong times, which matters most for fast-changing signals), power-supply noise coupling into analog outputs, and nonlinearity, which distorts the signal and creates harmonics.

Which bandwidth?

Every noise formula has a B in it, and choosing it correctly is half the battle. B is the range of frequencies that actually reaches your measurement, which is set by the narrowest element in the chain. That can be the signal bandwidth (a message occupying 10 kHz), an amplifier's $-3\,\text{dB}$ bandwidth, a filter's cutoff, or the bandwidth of your instrument (an oscilloscope with 20 MHz bandwidth sees noise only up to 20 MHz). For a signal sampled at rate f_s, the samples can represent frequencies only up to the Nyquist limit $f_s/2$, so sampling at 200 kHz means $B = 100\,\text{kHz}$. Because noise power is proportional to B, narrowing the bandwidth to just what the signal needs is one of the most effective ways to reduce noise.

A real filter does not cut off sharply, so a little noise above $f_{3\text{dB}}$ still gets through. For white noise passing a first-order low-pass filter with

time constant τ, the total is the same as for an ideal filter of width

$$B_n = \frac{\pi}{2} f_{3\text{dB}} = \frac{1}{4\tau}, \tag{5.13}$$

called the *noise bandwidth*. Use B_n when a TIA's $R_f C_f$ sets the bandwidth. (Thermal noise of R_f over $B_n = 1/(4R_fC_f)$ gives $\sqrt{4kTR_fB_n} = \sqrt{kT/C_f}$, the result quoted in the TIA example above.)

Adding noise sources

Independent noise sources do not add like ordinary voltages. Their instantaneous values are unrelated, sometimes adding and sometimes canceling, so their *powers* add. Since power is proportional to the square of the RMS value,

$$v_{n,\text{total}} = \sqrt{v_{n1}^2 + v_{n2}^2 + \cdots + v_{nk}^2}. \tag{5.14}$$

For example, 5 mV of thermal noise, 3 mV of shot noise, and 2 mV of amplifier noise combine to $\sqrt{25+9+4} = 6.16$ mV, not 10 mV. The rule has a useful consequence: the largest source dominates. A source three times smaller than the biggest one raises the total by only about 5%.

Amplifier data sheets quote a *noise factor* $F = \text{SNR}_{\text{in}}/\text{SNR}_{\text{out}}$ ($F = 1$ for a noiseless stage), or the *noise figure* $10\log_{10} F$. For a chain of stages with power gains $G_1, G_2, \ldots$, Friis' formula gives

$$F_{\text{total}} = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \cdots \tag{5.15}$$

Later stages are divided by the gain ahead of them, so the first amplifier after a photodiode must be the quietest. Stages with $G = 10, 5, 4$ and $F = 2, 3, 4$ give $F_{\text{total}} = 2 + 2/10 + 3/50 = 2.26$.

Worked Example 5.13 *Noise sources in an optical link*

Explain the types of noise in an optical communication system, and specifically the noise in a photodetector readout circuit.

Solution. Walk along Figure 5.5. The *DAC* adds quantization noise ($\Delta/\sqrt{12}$), thermal noise, nonlinearity (harmonic distortion), clock jitter, and power-supply noise. The *LED driver and LED* add thermal noise from driver resistors, shot noise from the LED current, and flicker noise. The *channel* adds ambient light (an offset with its own shot noise, plus 120 Hz lamp flicker) and loses signal to spreading and misalignment. The *ADC* adds quantization, thermal, aperture-jitter, reference, and clock noise.

In the *photodetector readout circuit* specifically:

- *Shot noise* of the photocurrent, $\sqrt{2qI_{ph}B}$, from random photon arrivals. Ambient light adds shot noise without adding signal.
- *Dark current* and its shot noise, $\sqrt{2qI_dB}$; dark current grows with temperature.
- *Thermal noise* of the load or feedback resistor, $\sqrt{4kTRB}$, present even with no light.
- *Op-amp voltage and current noise*, amplified by the TIA's gain.
- *Flicker noise* in the diode and op amp, important at low frequencies.

All of these are independent, so their powers add (Eq. (5.14)). ■

Worked Example 5.14 *Thermal noise of a 1 kΩ resistor*

What is the RMS thermal noise voltage of a 1000 Ω resistor at 300 K over a bandwidth of 100 kHz?

Solution. From Eq. (5.11),

$$v_{\text{th}} = \sqrt{4(1.38\times10^{-23})(300)(1000)(1\times10^{5})}\ \text{V}$$
$$= \sqrt{1.656\times10^{-12}\,\text{V}^2} = 1.29\,\mu\text{V}.$$

A microvolt is tiny next to the millivolt-sized steps of the Arduino's ADC, which is why resistor noise rarely matters in a simple circuit. It matters when the signal itself is tiny, as in a weakly lit photodiode. ■

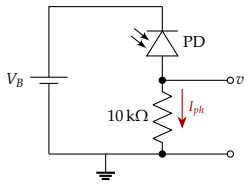


Figure 5.14: A reverse-biased photodiode with a 10 kΩ load resistor. The photocurrent develops $v = I_{ph}R$ across the resistor.

Worked Example 5.15 *Photodiode with a load resistor*

A reverse-biased photodiode with light shining on it drives a 10 kΩ resistor at 300 K (Figure 5.14). (a) What noise sources affect the voltage across the resistor? (b) With 1 mA of photocurrent, what is the RMS noise voltage if the voltage is sampled at 200 kHz?

Solution. (a) The shot noise of the photocurrent (plus that of any dark current) flowing through the resistor, and the thermal noise of the resistor itself. Flicker noise also exists but matters mostly at low frequencies.

(b) Sampling at 200 kHz gives a bandwidth $B = 100\,\text{kHz}$.

$$\begin{aligned} i_{\text{sh}} &= \sqrt{2(1.602\times10^{-19})(1\times10^{-3})(1\times10^{5})}\ \text{A} \\ &= \sqrt{3.20\times10^{-17}\,\text{A}^2} = 5.66\,\text{nA}, \\ v_{\text{sh}} &= (5.66\,\text{nA})(10\,\text{k}\Omega) = 56.6\,\mu\text{V}, \\ v_{\text{th}} &= \sqrt{4(1.38\times10^{-23})(300)(1\times10^{4})(1\times10^{5})}\ \text{V} \\ &= \sqrt{1.656\times10^{-11}\,\text{V}^2} = 4.07\,\mu\text{V}, \\ v_n &= \sqrt{(56.6)^2+(4.07)^2}\ \mu\text{V} = 56.7\,\mu\text{V}. \end{aligned}$$

The total is about 57 µV.

Shot noise dominates because the photocurrent is large. The signal itself is $v = (1\,\text{mA})(10\,\text{k}\Omega) = 10\,\text{V}$, so the SNR is enormous ($v^2/v_n^2 \approx 3\times10^{10}$, or 105 dB). In practice you would need a reverse bias above 10 V to keep the diode reverse-biased at this current. ■

Worked Example 5.16 *Shrinking the noise bandwidth*

Using the setup of the previous example, the signal now sits at 30 kHz with a bandwidth of 10 kHz, so it occupies only 25 kHz to 35 kHz. Sampling at 200 kHz still lets in noise from the whole 100 kHz band. What can you do, and what is the RMS noise afterwards?

Solution. Keep only the band that holds the signal. Options:

- An analog *band-pass filter* (25 kHz to 35 kHz) before the ADC.
- *Digital filtering* after sampling: a digital band-pass, or an FFT that keeps only the bins between 25 and 35 kHz, removes the out-of-band white noise.
- *Lock-in* (synchronous) detection, which multiplies by a copy of the 30 kHz carrier and low-pass filters, giving an even narrower effective bandwidth if the data are slow.
- Reducing unwanted DC photocurrent (shielding from ambient light), since shot noise grows as $\sqrt{I}$.

With $B = 10\,\text{kHz}$, both noise sources shrink by $\sqrt{100/10} = \sqrt{10} = 3.16$:

$$\begin{aligned} v_{\text{sh}} &= 56.6\,\mu\text{V}/3.16 = 17.9\,\mu\text{V}, \\ v_{\text{th}} &= 4.07\,\mu\text{V}/3.16 = 1.29\,\mu\text{V}, \\ v_n &= \sqrt{17.9^2+1.29^2}\ \mu\text{V} = 17.9\,\mu\text{V}. \end{aligned}$$

A tenfold narrower bandwidth cuts the RMS noise by a factor of 3.16 and the noise power by 10. ■

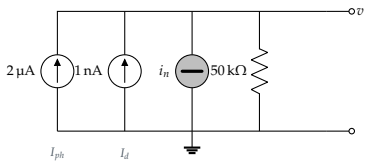


Figure 5.15: Noise model of the next worked example: photocurrent, dark current, all noise currents lumped as i_n, and the 50 kΩ conversion resistor.

Worked Example 5.17 *Detector noise and SNR*

A photodiode at 5 V reverse bias drives a 50 kΩ resistor. Light produces 2 µA of photocurrent and the dark current is 1 nA. The bandwidth is 1 MHz and $T = 300$ K. Find (a) the signal voltage, (b) the RMS noise voltage, counting thermal noise, photocurrent shot noise, and dark-current shot noise, and (c) the SNR, defined as $v_{\text{signal}}^2/v_n^2$. The noise model is Figure 5.15.

Solution. (a) $v = I_{ph}R = (2\,\mu\text{A})(50\,\text{k}\Omega) = 0.100\,\text{V}$; the dark current adds a negligible 50 µV.

(b) Work with currents, since all three sources sit in parallel with R:

$$\begin{aligned} i_{\text{th}}^2 &= \frac{4kTB}{R} = \frac{4(1.38\times10^{-23})(300)(1\times10^{6})}{5\times10^{4}} \\ &= 3.31\times10^{-19}\,\text{A}^2, \\ i_{\text{sh}}^2 &= 2qI_{ph}B = 2(1.602\times10^{-19})(2\times10^{-6})(1\times10^{6}) \\ &= 6.41\times10^{-19}\,\text{A}^2, \\ i_d^2 &= 2qI_dB = 2(1.602\times10^{-19})(1\times10^{-9})(1\times10^{6}) \\ &= 3.2\times10^{-22}\,\text{A}^2. \end{aligned}$$

The RMS values are 0.575 nA, 0.800 nA, and 0.018 nA. The total is

$$\begin{aligned} i_n &= \sqrt{(3.31+6.41+0.0032)\times10^{-19}}\ \text{A} = 0.986\,\text{nA}, \\ v_n &= i_nR = (0.986\,\text{nA})(50\,\text{k}\Omega) = 49.3\,\mu\text{V}. \end{aligned}$$

Shot noise of the photocurrent is the largest term, thermal noise is close behind, and the dark current is negligible at this light level.

(c) The signal-to-noise ratio is

$$\text{SNR} = \frac{(0.100\,\text{V})^2}{(49.3\times10^{-6}\,\text{V})^2} = 4.11\times10^{6},$$

or $10\log_{10}(4.11\times10^{6}) = 66.1$ dB. ■

Worked Example 5.18 *Which noise dominates on the demo board?*

The board's receiver has $R_f = 3\,\text{M}\Omega$ and a time constant of about 0.30 ms (Worked Example 5.12). With the sender's LED on, it reads 174 counts on a 10-bit ADC over 3.3 V. Estimate the thermal noise of R_f, the shot noise of the photocurrent, and the quantization noise, all at the ADC input. Compare their total with the standard deviation of about 2 counts measured on real boards.

SOLUTION. From Eq. (5.13), the noise bandwidth is $B_n = 1/(4\tau) = 1/(4 \times 0.30\,\text{ms})$, which is 833 Hz. One count is 3.3 V/1024, or 3.22 mV, so 174 counts is 0.561 V, and the photocurrent is 0.561 V divided by 3 MΩ, or 0.187 µA.

$$\begin{aligned} v_{\text{th}} &= \sqrt{4kTR_f B_n} \\ &= \sqrt{4(1.38\times10^{-23})(300)(3\times10^{6})(833)}\ \text{V} = 6.43\,\mu\text{V}, \\ v_{\text{sh}} &= R_f\sqrt{2qI_{ph}B_n} \\ &= (3\,\text{M}\Omega)\sqrt{2(1.602\times10^{-19})(1.87\times10^{-7})(833)}\ \text{A} \\ &= 21.2\,\mu\text{V}, \\ v_q &= \frac{3.22\,\text{mV}}{\sqrt{12}} = 0.93\,\text{mV}. \end{aligned}$$

Their root sum of squares is 0.93 mV, or 0.29 counts: quantization dominates, and the two fundamental sources are 40 to 150 times smaller. Yet real boards show $\sigma \approx 2$ counts (6.4 mV), seven times more. The extra noise comes from sources this chapter only named: the op amp's own noise, ripple on the 3.3 V supply, the ADC's internal noise (its effective resolution is below 10 bits), and the 120 Hz flicker of room lights reaching the photodiode. Shading the receiver and averaging several samples per bit attack the last two. ■

5.7 *Decisions, Bit Errors, and Eye Diagrams*

THE RECEIVER'S JOB IS TO DECIDE. After all the analog processing, the decoder looks at a voltage and has to say "1" or "0". Figure 5.16 shows what it faces: a slowed-down, noisy copy of the transmitted waveform sitting on top of an ambient-light baseline. The simplest decoder compares one sample per bit, taken near the middle of the bit, with a *threshold* halfway between the two levels.

Two details make this work on the demo boards. The first is *where* to sample: right after a transition the signal is still moving, so the receivers sample mid-bit. The Task 1 receiver waits 1.5 bit periods after the falling edge of the start slot, which lands in the middle of the first data bit (Figure 5.7). The second is *where to put the threshold*. A fixed value (Task 1 uses 200 ADC counts) works only if the received levels stay put. The lab notebook instead measures the dark level and the received level at full drive first (Part C) and puts the threshold halfway between them; the light-and-sound demo shown in class averages its light sensor for 2 s at startup and adds a margin.

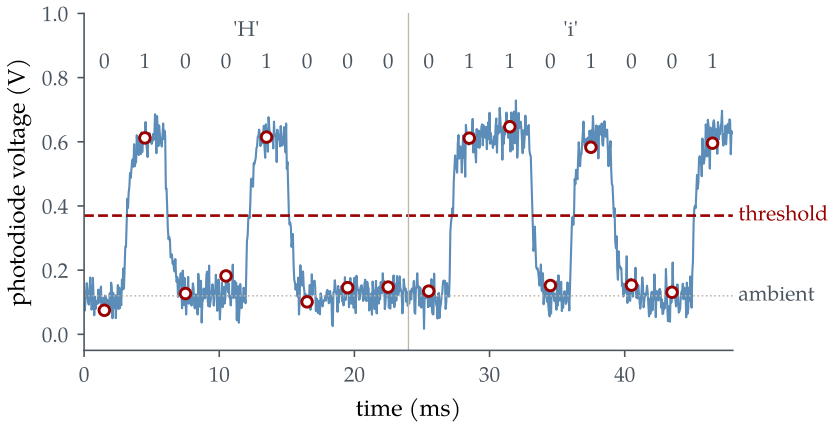


Figure 5.16: Simulated photodiode voltage for `"Hi"` sent with OOK at 3 ms per bit. The receiver's RC response rounds the edges and noise rides on both levels. Circles mark one sample per bit, taken mid-bit; comparing each with the threshold recovers `01001000 01101001`.

Signal-to-noise ratio and bit errors

The *signal-to-noise ratio* compares signal power with noise power,

$$\mathrm{SNR} = \frac{P_{\text{signal}}}{P_{\text{noise}}}, \qquad \mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} \mathrm{SNR}. \tag{5.16}$$

For voltages across the same resistance, power goes as voltage squared, so $\mathrm{SNR} = (v_s/v_n)^2$ and $\mathrm{SNR}_{\mathrm{dB}} = 20 \log_{10}(v_s/v_n)$. A few values are worth remembering: a power ratio of 2 is 3 dB, 10 is 10 dB, and 1000 is 30 dB; a voltage ratio of 2 is 6 dB and 10 is 20 dB. Decibels add where ratios multiply, so a link that loses 6 dB and then gains 20 dB ends 14 dB up. One trap belongs to optical links: the photocurrent is proportional to optical *power*, so the electrical signal power goes as the *square* of the optical power. Losing half the light (3 dB optical) costs 6 dB of electrical SNR when the noise stays the same.

The *bit error rate* is the fraction of bits received wrongly:

$$\mathrm{BER} = \frac{\text{number of bits in error}}{\text{total number of bits sent}}. \tag{5.17}$$

It is a probability, usually written in scientific notation. A good optical link aims for 10^{-9} or better; a link with 10^{-3} gets one character in about every 125 wrong.

To connect the two, suppose OOK levels 0 and A, a threshold at $A/2$, and Gaussian noise with RMS value σ added to every sample. A 0 is misread when the noise exceeds $+A/2$, and a 1 when it falls below $-A/2$. Both have the same probability, so

$$\mathrm{BER} = Q\left(\frac{A}{2\sigma}\right), \qquad Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-u^2/2}\, du. \tag{5.18}$$

$Q(x)$ is the probability that a standard Gaussian variable exceeds x; in Python it is `scipy.stats.norm.sf(x)`. It falls off very fast: $Q(3.09) =$

10^{-3}, $Q(4.75) = 10^{-6}$, $Q(6.0) = 10^{-9}$. What matters is the ratio of the *half-gap* between levels to the noise.

If we define the SNR of the OOK signal as $(A/2)^2/\sigma^2$, Eq. (5.18) becomes BER $= Q(\sqrt{\text{SNR}})$, plotted in Figure 5.17. A BER of 10^{-3} needs an SNR of 9.8 dB, and 10^{-9} needs 15.6 dB. Just 6 dB of extra SNR improves the BER a million times. PAM-4 with the same total swing has level spacing $A/3$ instead of A, so its half-gap is three times smaller. To get the same BER, it needs nine times the SNR, a 9.5 dB penalty. That is the cost of doubling the bit rate without speeding up the symbols. For M equally spaced levels with spacing d and thresholds at the midpoints, the two outer levels can err only one way and the inner ones both ways, so the symbol error probability is

$$P_s = 2\left(1 - \frac{1}{M}\right) Q\left(\frac{d}{2\sigma}\right). \tag{5.19}$$

For $M = 2$ this reduces to Eq. (5.18); for $M = 4$ it is $1.5\,Q(d/2\sigma)$.

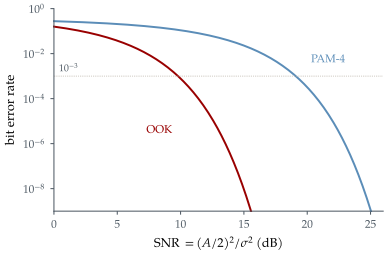


Figure 5.17: BER versus SNR for OOK, BER $= Q(\sqrt{\text{SNR}})$, and for PAM-4 with the same total swing. PAM-4 needs about 9.5 dB more SNR for the same BER.

Worked Example 5.19 *When the boards move apart*

Close together, an OOK link reads 4 counts for a 0 and 224 counts for a 1, with noise $\sigma = 2$ counts on both levels and the threshold at the midpoint. (a) Find the margin in units of σ and the SNR in dB. (b) The receiver moves to four times the distance; the noise stays the same. Find the new swing, the BER, and the chance that the 360-bit lab message has at least one error. (c) What happens to PAM-4 at the larger distance? Use $Q(3.44) = 2.9 \times 10^{-4}$ and $Q(1.15) = 0.125$.

Solution. (a) The half-gap is $(224 - 4)/2 = 110$ counts $= 55\sigma$. $Q(55)$ is far below anything you could ever measure. The SNR is $(A/2)^2/\sigma^2 = 55^2 = 3025$, or $10\log_{10} 3025 = 34.8$ dB.

(b) By the inverse-square law the optical power, and with it the swing, drops by $4^2 = 16$: $A = 220/16 = 13.75$ counts. The half-gap is 6.875 counts $= 3.44\sigma$ (SNR 10.7 dB, a loss of 24 dB), so

$$\text{BER} = Q(3.44) = 2.9 \times 10^{-4}.$$

In 360 bits you expect $360 \times 2.9 \times 10^{-4} = 0.104$ errors, and the chance of at least one is $1 - (1 - 2.9 \times 10^{-4})^{360} = 0.099$: one message in ten arrives with a wrong character.

(c) PAM-4 splits the same 13.75 counts into three gaps of 4.58 counts. The half-gap is 2.29 counts $= 1.15\sigma$, so $P_s = 1.5\,Q(1.15) = 0.188$. Almost one symbol in five is wrong, and the text is unreadable. PAM-4 fails long before OOK when the light gets weak. ■

Averaging helps. If the receiver takes N_s independent samples per bit and averages them, the noise RMS drops by $\sqrt{N_s}$. The lab receiver

records about 16 samples per symbol but decides on one; step D6 of Lab 5.1 averages the samples in the middle half of each symbol and compares the improvement with $\sqrt{N_s}$. The improvement falls short when part of the noise changes slowly compared with a bit, as room-light flicker at 120 Hz does, because all the samples of one bit then share it.

Eye diagrams

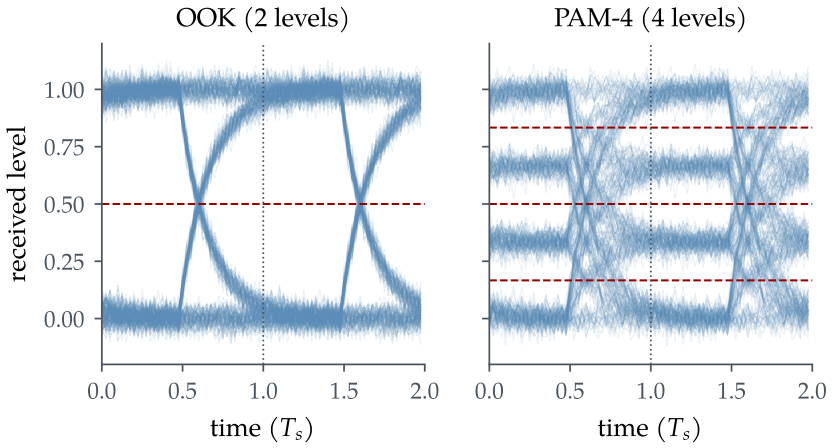


Figure 5.18: Simulated eye diagrams with the same noise and the same total swing. The OOK eye (left) is wide open around the threshold. The PAM-4 eye (right) has three smaller eyes stacked between four levels, each with less room for noise.

An *eye diagram* is a picture of a link's health. Cut the received waveform into pieces two symbols long, each starting half a symbol before a symbol boundary, and draw all of them on top of each other. The many possible bit patterns overlap into a shape like an open eye (Figure 5.18). The Lab 5.1 notebook draws exactly this.

The *eye height* is the clear vertical opening at the sampling instant; noise closes it from above and below. The *eye width* is the clear horizontal opening, the range of sampling instants that give no errors; timing *jitter* and slow edges close it from the sides. The thickness of the top and bottom bands shows the noise on the one and zero levels. The ratio of the one level to the zero level (in optical power) is the *extinction ratio*; an LED that is fully off for a 0 has a very large one. Sample where the eye is most open, with the threshold through the middle of the opening. A wide-open eye means a low BER; a closed eye means errors wherever you sample. The PAM-4 eye stacks three eyes, each a third as tall, which is the picture behind the 9.5 dB penalty.

Worked Example 5.20 *Reading an eye diagram*

Figure 5.19 shows the eye of an OOK link at 1 ms per bit, with levels of 5 and 220 counts, a first-order receiver with $\tau = 0.30$ ms,

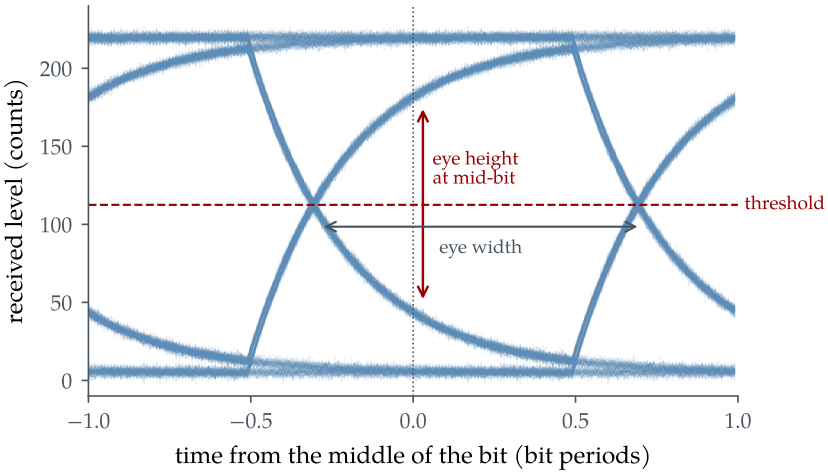


Figure 5.19: Simulated OOK eye at 1 ms per bit: levels 5 and 220 counts, receiver time constant 0.30 ms, noise 2 counts. Each trace is two bits long and centered on the middle of a bit. The slow receiver makes the eye lopsided: it opens late and is tallest at the end of the bit.

and noise $\sigma = 2$ counts. (a) Estimate the eye height at mid-bit, and check it with the RC model. (b) Where do the traces cross the threshold, and how wide is the eye? (c) Where should the receiver sample?

SOLUTION. (a) In the figure, the lowest 1 trace at mid-bit is near 177 counts and the highest 0 trace near 49 counts, an opening of about 128 counts. The RC model explains this. After a step at the start of the bit the output has covered $1 - e^{-t/\tau}$ of the swing $A = 215$ counts. A 1 that follows a long run of 0s has reached $5 + 215(1 - e^{-0.5/0.30}) = 5 + 215(0.811) = 179.4$ counts at mid-bit, and a 0 that follows a long run of 1s has fallen to $5 + 215(0.189) = 45.6$ counts. Noise widens each band by about $\pm 3\sigma = \pm 6$ counts, leaving a clear opening of about $179.4 - 45.6 - 12 = 122$ counts. The figure's opening is a little larger because its largest noise excursions happen to stay under 3σ.

(b) The threshold is at $(5 + 220)/2 = 112.5$ counts, halfway up the swing. A trace crosses it when $1 - e^{-t/\tau} = 1/2$, at $t = \tau \ln 2 = 0.208$ ms after a bit boundary. Measured from mid-bit, the crossings sit at $-0.5 + 0.208 = -0.292$ and $+0.5 + 0.208 = +0.708$ bit periods. The eye is therefore about one bit (1 ms) wide, a little less with noise, but it is shifted 0.208 ms late.

(c) The opening is tallest just before the next transition. At the end of the bit the worst 1 has reached $5 + 215(1 - e^{-1/0.30}) = 212.3$ counts and the worst 0 has fallen to $5 + 215\,e^{-1/0.30} = 12.7$ counts, an opening of about $212.3 - 12.7 - 12 = 188$ counts. Sampling at about 70 to 80% of the way through the bit gives a tall eye and still leaves timing margin on both sides. A receiver that finds the start

of the frame from the received signal itself, as the lab's correlation does (Section 5.8), shifts its sampling instants late by about the receiver's delay automatically. ■

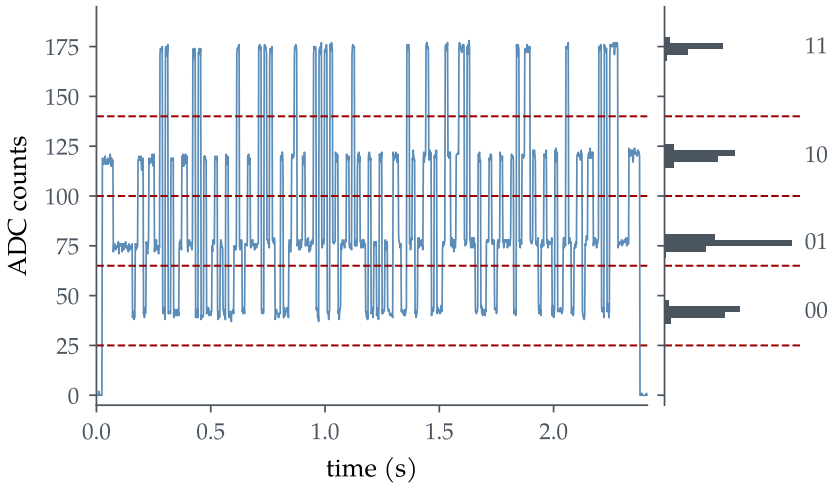


Figure 5.20: Real receiver data from an earlier version of the lab: 1205 ADC samples taken every 2 ms while a PAM-4 message arrived at 12 ms per symbol, sent with DAC codes 1750, 1800, 1850, and 2000. Dashed lines are the thresholds used then (25, 65, 100, 140 counts). The histogram at the right shows four tight clusters, one per symbol value.

Worked Example 5.21 *Reading a real PAM-4 recording*

Figure 5.20 shows samples recorded while the 45-byte message file arrived in PAM-4 at 12 ms per symbol. The four clusters have mean values of 41.5, 76.0, 119.6, and 174.2 ADC counts, each with a standard deviation between 1.7 and 2.0 counts; we use $\sigma = 1.9$ counts for every level. The receiver used thresholds of 25, 65, 100, and 140 counts (the first one separates "no signal" from the lowest level). (a) Where would midpoint thresholds sit? (b) Estimate the symbol error probability caused by noise with those thresholds and with midpoint thresholds. (c) The recorded message decoded with zero errors in 360 bits. Is that surprising?

Solution. (a) Midpoints between neighboring levels: $(41.5 + 76.0)/2 = 58.8$, $(76.0 + 119.6)/2 = 97.8$, and $(119.6 + 174.2)/2 = 146.9$ counts.

(b) The risk is set by the closest approach of a level to a threshold, measured in units of σ. With the thresholds used then, the tightest margin is between the 01 level (76.0) and the threshold at 65: 11.0 counts, or $11.0/1.9 = 5.8\sigma$. The chance that a 01 symbol falls below 65 is $Q(5.8) \approx 3.3 \times 10^{-9}$; the other levels have wider margins, so averaged over the four equally likely levels the symbol error probability is about 8×10^{-10}. With midpoint thresholds, the tightest margin is half the smallest gap, $(76.0 - 41.5)/2 = 17.3$

counts $= 9.1\sigma$, and $Q(9.1) \approx 5 \times 10^{-20}$ for the worst level, about 3×10^{-20} averaged over the levels. The hand-set thresholds were good but not centered.

(c) No. With only 196 symbols, and noise margins of more than 5σ, the expected number of noise errors is far below one. In this link, errors come from other causes: a sample taken too close to a transition when the sender's and receiver's clocks disagree, ambient light shifting the levels, or a missed start. The levels are also unevenly spaced (gaps of 34.5, 43.6, and 54.6 counts) because the LED's light output is a nonlinear function of DAC code. ■

The ultimate speed limit: Shannon capacity

Faster symbols need more bandwidth, and more levels need more SNR. Is there a limit to how far this can go? Shannon answered that in 1948. A channel of bandwidth B with Gaussian noise and signal-to-noise ratio SNR can carry at most

$$C = B \log_2(1 + \mathrm{SNR}) \quad \text{bit/s} \tag{5.20}$$

with an error rate as low as you like, provided you use clever enough coding. Above C, errors are unavoidable no matter what you do. The formula says bandwidth buys capacity linearly, while SNR buys it only logarithmically: doubling the SNR adds at most one bit per second per hertz.

KEY EQUATION

$C = B \log_2(1 + \mathrm{SNR})$

Shannon capacity of a channel with bandwidth B

Worked Example 5.22 *Capacity of a photodetector link*

(a) The detector of Worked Example 5.17 has an SNR of 4.11×10^6 over 1 MHz. What is its Shannon capacity? (b) The board link of Figure 5.20 has an OOK swing of $174.2 - 41.5 = 132.7$ counts and noise $\sigma = 1.9$ counts, and sends symbols every 12 ms, which needs a bandwidth of at least $1/(2 \times 12\,\mathrm{ms}) = 41.7\,\mathrm{Hz}$. Compare its capacity with what it actually achieves.

Solution. (a) From Eq. (5.20),

$$C = (1 \times 10^6\,\mathrm{Hz}) \log_2(1 + 4.11 \times 10^6) = 22.0\,\mathrm{Mbit/s}.$$

The detector could in principle carry about 22 bits per second per hertz.

(b) Using $\mathrm{SNR} = (A/2)^2/\sigma^2 = (66.35/1.9)^2 = 1219$ (30.9 dB),

$$C = (41.7\,\mathrm{Hz}) \log_2(1220) = (41.7)(10.25) = 427\,\mathrm{bit/s}.$$

That link sent 83 bit/s with OOK and 167 bit/s with PAM-4, so even PAM-4 used less than half of the capacity at this symbol rate. More levels (PAM-8, see Problem 5.19) or faster symbols move it

closer. The current lab takes the second route: the board plays the symbols from its memory, so they can be 24 times shorter (0.5 ms). The usable bandwidth grows in proportion, and the capacity grows almost as fast (the wider band also admits more noise, which lowers the SNR a little), until the receiver's own bandwidth of a few hundred hertz becomes the limit. ■

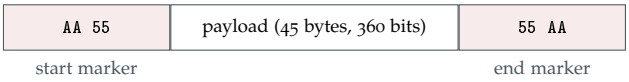
5.8 Framing, Synchronization, and Error Control

A STREAM OF BITS IS USELESS until the receiver knows where each bit, each byte, and each message starts. It also helps to know when something went wrong, and better still to fix it. This section covers both problems.

Finding the start

Task 1 uses *asynchronous* framing, the same scheme as the serial port (UART) that connects your Arduino to your laptop. The line idles in one state (LED on). A start slot in the other state (LED off) announces a new byte, the receiver then times the eight data bits from that falling edge, and a stop slot returns the line to idle so the next start edge is visible. Because the receiver resynchronizes on every byte, its clock only has to stay accurate for ten slots.

Parts D to F of the lab frame the whole message instead. The notebook sends the two *start bytes* `0xAA 0x55` before the payload and `0x55 0xAA` after it:



In binary the start marker is `10101010 01010101`, a run of alternating bits that is easy to recognize and that gives the receiver many edges to check its timing against. A known pattern placed at the start of a transmission for this purpose is called a *preamble* or *sync word*. The framing costs 32 bits on a 360-bit payload, an efficiency of $360/392 = 92\%$.

The receiving board starts recording when the light first rises above a trigger level, which is close to the start of the frame but not exactly at it. To find the start precisely, the notebook slides the ideal preamble waveform p along the recording x and computes, for every offset k, the *correlation*

$$c[k] = \sum_{j} \big(x[k+j] - \bar{x}\big)\big(p[j] - \bar{p}\big), \tag{5.21}$$

where the bars are averages. Where the recording looks like the preamble, the products are mostly positive and $c[k]$ peaks; the offset of the peak is the start of the frame. Each symbol k is then sampled at its middle and compared with the threshold, the 16 marker bits at each end are stripped, and the rest is decoded. Because the peak is found in the received signal, every fixed delay, including the receiver's own response time, is absorbed automatically.

Worked Example 5.23 *Finding the frame start by correlation*

A receiver has one sample per bit, already converted to $+1$ for a bit above the threshold and -1 below it. Its recording starts three bits before the frame, so it holds three idle 0s, the 16-bit preamble `10101010 01010101`, and then the first payload bits `01010100` (`T`). With p the preamble as ± 1 values, compute $c[k] = \sum_{j=0}^{15} x[k+j]\,p[j]$ for $k = 0$ to 5 and find the start. Why is `0xAA 0x55` a better preamble than `0xAA 0xAA`?

Solution. The preamble has eight 1s and eight 0s, so $\bar{p} = 0$ and the means drop out of Eq. (5.21). Each product $x[k+j]p[j]$ is $+1$ when the received bit agrees with the preamble bit and -1 when it disagrees, so $c[k] = (\text{agreements}) - (\text{disagreements}) = 16 - 2 \times (\text{disagreements})$. Comparing the 16 received bits that start at each offset with the preamble:

offset k	0	1	2	3	4	5
disagreements	12	3	15	0	15	2
$c[k]$	-8	10	-14	**16**	-14	12

The peak, $c[3] = 16$, says the frame starts at bit 3, as it should. The runner-up, $c[5] = 12$, comes from the alternating pattern lining up with itself two bits late. With `0xAA 0xAA` the preamble would be 16 alternating bits, and the same computation gives $c[1] = 14$ next to $c[3] = 16$: a shift of two bits now disagrees in only one place, so a single wrong bit could make the false peak win. The `00` in the middle of `10101010 01010101` breaks the alternation and holds every shifted match to 12 or less. ■

Worked Example 5.24 *Ada and Bob synchronize*

Ada and Bob live 30 km apart and are joined by an optical fiber with core index 1.48. (a) Ada sends a single bit, and Bob receives it at time t. Ignoring electronic delays, when did Ada send it? (b) Ada then sends 1000 bits at 1 kbit/s, and Bob assumes that whatever he receives at time t was sent at time t. What goes wrong? (c) Unknown delays in the DACs and ADCs appear. Ada proposes measuring the delay by recording the send time t_1 and receive time

t_2 in global (UTC) time; Bob proposes that Ada start every message with `Hello` and that he treat the arrival of `Hello` as time zero. Which approach is more practical?

SOLUTION. (a) Light in the core travels at $v = c/n = (3.00 \times 10^8\,\mathrm{m/s})/1.48 = 2.03 \times 10^8\,\mathrm{m/s}$, so the delay is

$$t_{\text{delay}} = \frac{30 \times 10^3\,\mathrm{m}}{2.03 \times 10^8\,\mathrm{m/s}} = 148\,\mu\mathrm{s}.$$

Ada sent the bit at $t - 148\,\mu\mathrm{s}$.

(b) Each bit lasts 1 ms, so Bob's bit boundaries sit 148 µs (15% of a bit) after Ada's. If he samples near the start of his bit periods, a little extra delay makes him read the *previous* bit. Worse, two clocks never run at exactly the same rate: a 100 parts-per-million difference slips another 100 µs over 1000 bits.

(c) Ada's method needs two clocks that agree to a small fraction of a millisecond, which network-synchronized computer clocks do not guarantee, and the delay can change between messages. Bob's method needs no shared clock: the known pattern marks time zero in the received signal itself and absorbs every fixed delay. This is the preamble idea that the lab uses with `0xAA 0x55`. Bob must still sample mid-bit and keep correcting his timing on edges. ■

Detecting errors: parity and checksums

The ASCII codes for printable characters fit in 7 bits, which leaves the eighth bit of each byte free. A *parity bit* uses it to make the number of 1s in the byte even (even parity) or odd (odd parity). With even parity, `E` (`1000101`, three 1s) gets a parity bit of 1 and is sent as `11000101`. If one bit flips on the way, the receiver counts an odd number of 1s and knows the byte is bad. If *two* bits flip, the count is even again and the error slips through. Parity catches every odd number of flipped bits and misses every even number.

WORKED EXAMPLE 5.25 *How often does parity miss an error?*

Bits flip independently with probability $p = 10^{-3}$. A byte of 7 data bits and one even-parity bit is sent. (a) What is the probability that the byte arrives with at least one error? (b) What is the probability of an error that parity misses? (c) What fraction of the bad bytes does parity catch?

SOLUTION. (a) The byte is perfect with probability $(1-p)^8 = 0.99203$, so it has at least one error with probability $1 - 0.99203 = 7.97 \times 10^{-3}$.

(b) Parity misses an even number of flips. Two flips is by far the most likely case:

$$P_2 = \binom{8}{2} p^2 (1-p)^6 = 28\,(10^{-6})(0.994) = 2.78 \times 10^{-5}.$$

Four or more flips add only about $70p^4 = 7 \times 10^{-11}$.

(c) Parity catches $1 - 2.78 \times 10^{-5} / 7.97 \times 10^{-3} = 0.9965$, or 99.65% of the bad bytes. One extra bit per byte buys a lot when errors are rare; it helps far less when errors come in bursts that flip neighboring bits together. ■

A *checksum* protects a whole message. The sender computes a number from all the bytes and sends it at the end; the receiver repeats the computation and compares. The simplest version adds all the bytes and keeps the sum modulo 256, so the checksum fits in one byte. It catches any single wrong byte, but two changes that cancel in the sum slip through, and so does a swap of two bytes, because addition does not care about order. A random corruption passes an 8-bit checksum about once in 256 tries. Commercial links use a *cyclic redundancy check* (CRC), a checksum based on polynomial division that is sensitive to order and catches every burst of errors no longer than the CRC itself.

WORKED EXAMPLE 5.26 *An 8-bit checksum*

The sender transmits `"Hi!"` followed by the sum of its bytes modulo 256. (a) Find the checksum. (b) Bit 1 of `i` flips in transit. Is the error caught? (c) Bit 0 flips in both `H` and `i`. Is it caught, and would a parity bit on each byte have caught it? (d) The first two bytes arrive swapped.

SOLUTION. (a) The bytes are 72, 105, and 33. Their sum is 210, which is less than 256, so the checksum is $210 =$ `0xD2`.

(b) `i` $=$ `01101001` becomes `01101011` $=$ 107 (`k`). The receiver sums $72 + 107 + 33 = 212 \neq 210$ and rejects the message.

(c) `H` (72) becomes `I` (73) and `i` (105) becomes `h` (104). The sum is $73 + 104 + 33 = 210$, the same as before, so `"Ih!"` passes. A parity bit per byte would catch it, because each byte has one flipped bit.

(d) `"iH!"` also sums to 210 and passes. A CRC would catch both (c) and (d). ■

Correcting errors: repetition and Hamming codes

Detection tells you to ask for the data again. Sometimes you cannot ask: the link is one-way, or the delay is too long. Then you add enough redundancy to *correct* errors. The simplest way is a *repetition code*: send

every bit three times and let the receiver take a majority vote. Any single flip in a group of three is outvoted. The price is steep: the *code rate*, data bits per transmitted bit, is only $1/3$.

Worked Example 5.27 *A triple-repetition code*

Bits flip independently with probability $p = 0.01$. (a) What is the bit error probability after majority decoding of a triple-repetition code? (b) How many errors do you expect in the lab's 360-bit message with and without the code, and how long does the coded message take at 250 bit/s on the line?

Solution. (a) The vote is wrong when two or three of the copies flip:

$$P_e = 3p^2(1-p) + p^3 = 3(10^{-4})(0.99) + 10^{-6} = 2.98 \times 10^{-4}.$$

(b) Uncoded: $360 \times 0.01 = 3.6$ errors. Coded: $360 \times 2.98 \times 10^{-4} = 0.107$ errors, a 34-fold improvement. But the coded message is $3 \times 360 = 1080$ bits, which take 4.32 s at 250 bit/s instead of 1.44 s. ■

In 1950 Richard Hamming, tired of weekend computer jobs at Bell Labs that stopped at the first detected error, found a far cheaper way to correct errors.[4] The smallest Hamming code, *Hamming (7,4)*, sends 4 data bits and 3 parity bits in 7 positions. The parity bits sit at the positions that are powers of two (1, 2, 4), and the data bits d_1 to d_4 fill positions 3, 5, 6, and 7. Each parity bit makes even parity over the positions whose binary index contains its bit (the first seven columns of Table 5.4): p_1 checks positions 1, 3, 5, 7; p_2 checks 2, 3, 6, 7; and p_4 checks 4, 5, 6, 7. Figure 5.21 draws the same rule as three overlapping circles.

At the receiver, recompute the parity checks. Each one that fails contributes its weight (1, 2, or 4), and the sum, called the *syndrome*, is the position of the flipped bit. A syndrome of 0 means no single error. Flip that bit back and the error is gone. The code rate is $4/7 = 57\%$, almost twice that of triple repetition, and it still corrects any single error in the block.

[4] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, 1950.

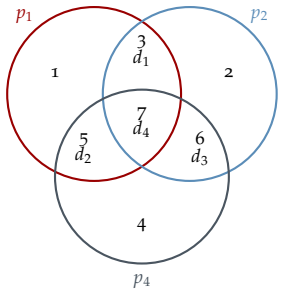


Figure 5.21: Hamming (7,4) as three circles. Each circle holds one parity bit and the data bits it checks, and each circle must contain an even number of 1s. A single flipped bit upsets exactly the circles that contain it, which pins down its position.

Worked Example 5.28 *Hamming (7,4) encoding and decoding*

Encode the data bits $d_1 d_2 d_3 d_4 = 1011$. Then position 5 flips in transit. Show how the receiver finds and fixes the error.

SOLUTION. With $d_1 = 1$, $d_2 = 0$, $d_3 = 1$, $d_4 = 1$:

$$p_1 = d_1 \oplus d_2 \oplus d_4 = 1 \oplus 0 \oplus 1 = 0,$$
$$p_2 = d_1 \oplus d_3 \oplus d_4 = 1 \oplus 1 \oplus 1 = 1,$$
$$p_4 = d_2 \oplus d_3 \oplus d_4 = 0 \oplus 1 \oplus 1 = 0.$$

In position order $p_1\, p_2\, d_1\, p_4\, d_2\, d_3\, d_4$ the codeword is `0110011`. Position 5 flips, so the receiver gets `0110111`. It recomputes the checks:

$$c_1 : \text{positions } 1,3,5,7 \to 0,1,1,1 \quad \text{(three 1s, fails)} \to 1,$$
$$c_2 : \text{positions } 2,3,6,7 \to 1,1,1,1 \quad \text{(four 1s, passes)} \to 0,$$
$$c_4 : \text{positions } 4,5,6,7 \to 0,1,1,1 \quad \text{(three 1s, fails)} \to 1.$$

The syndrome is $4c_4 + 2c_2 + c_1 = 4 + 1 = 5$. Flipping position 5 back restores `0110011`, and the data positions 3, 5, 6, 7 read 1011. ■

The same rule works for longer blocks. A 7-bit ASCII character needs four parity bits, at positions 1, 2, 4, and 8, which gives the *Hamming (11,7)* code: 7 data bits d_1 to d_7 fill positions 3, 5, 6, 7, 9, 10, 11 (Table 5.4). The light-and-sound demo shown in class uses this code.

Position	1	2	3	4	5	6	7	8	9	10	11
Binary	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011
Content	p_1	p_2	d_1	p_4	d_2	d_3	d_4	p_8	d_5	d_6	d_7
p_1 checks	•		•		•		•		•		•
p_2 checks		•	•			•	•			•	•
p_4 checks				•	•	•	•				
p_8 checks								•	•	•	•

Table 5.4: Hamming code layout. Each parity bit covers the positions whose binary index has a 1 in that parity bit's place. Hamming (7,4) uses positions 1 to 7 and parity bits p_1, p_2, p_4; Hamming (11,7) uses all eleven positions and adds p_8.

The receiver recomputes four checks, and failed checks now add weights 1, 2, 4, and 8, so the syndrome can name any of the 11 positions. Adding one more bit, p_0, with even parity over the whole 12-bit block gives the *extended* Hamming (12,7) code, known as SECDED (single-error correcting, double-error detecting). One error makes the overall parity fail and the syndrome point to the bad bit. Two errors leave the overall parity correct but give a nonzero syndrome, so the receiver knows something is wrong and does not try to "fix" it. The price is overhead: 12 bits sent for every 7 data bits, a code rate of $7/12 = 58\%$.

WORKED EXAMPLE 5.29 *Hamming-encode `E` and fix a flipped bit*

Encode the 7-bit ASCII code of `E` with Hamming (11,7), taking d_1 as the least significant bit. Then suppose position 10 flips in transit. Show how the receiver finds and fixes it.

SOLUTION. `E` is `1000101`, so from least significant up: $d_1 = 1$,

$d_2 = 0$, $d_3 = 1$, $d_4 = 0$, $d_5 = 0$, $d_6 = 0$, $d_7 = 1$. Place them at positions 3, 5, 6, 7, 9, 10, 11, then compute even parity (XOR, $\oplus$) over each group of Table 5.4:

$$p_1 = d_1 \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_7 = 1 \oplus 0 \oplus 0 \oplus 0 \oplus 1 = 0,$$
$$p_2 = d_1 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1,$$
$$p_4 = d_2 \oplus d_3 \oplus d_4 = 0 \oplus 1 \oplus 0 = 1,$$
$$p_8 = d_5 \oplus d_6 \oplus d_7 = 0 \oplus 0 \oplus 1 = 1.$$

The codeword, positions 1 to 11, is `0 1 1 1 0 1 0 1 0 0 1`. It has six 1s, so the extended code's overall parity bit is $p_0 = 0$.

Now position 10 flips: the receiver gets `0 1 1 1 0 1 0 1 0 1 1`. It recomputes each check over the received bits:

c_1 : positions $1,3,5,7,9,11 \to 0,1,0,0,0,1$ (two 1s, passes) $\to 0$,

c_2 : positions $2,3,6,7,10,11 \to 1,1,1,0,1,1$ (five 1s, fails) $\to 1$,

c_4 : positions $4,5,6,7 \to 1,0,1,0$ (passes) $\to 0$,

c_8 : positions $8,9,10,11 \to 1,0,1,1$ (fails) $\to 1$.

The syndrome is $8c_8 + 4c_4 + 2c_2 + c_1 = 8 + 2 = 10$. The overall parity now fails too (seven 1s), which confirms a single error. Flip position 10 back and read the data positions to recover `1000101` = E. ■

5.9 *Sensor Fusion*

EVERY SENSOR LIES A LITTLE. An accelerometer is noisy. A gyroscope drifts. GPS jumps around by several meters. A light sensor is fooled by a passing shadow. *Sensor fusion* is the art and science of combining several imperfect sensors into one estimate that is better than any of them alone: more accurate, more stable, more complete, and less likely to fail all at once (Figure 5.22).

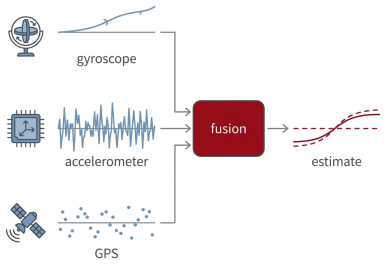


Figure 5.22: Sensor fusion: several sensors with different strengths and weaknesses feed one algorithm, which produces a single, better estimate.

You already carry examples. A phone combines its accelerometer, gyroscope, and magnetometer to know which way the screen faces. A car with driver assistance combines cameras (lanes and signs), radar (speed and distance, even in fog), lidar (3D shape), and ultrasonic sensors (parking). A smartwatch measuring heart rate optically, like the pulse oximeter of Chapter 3, uses its accelerometer to remove motion artifacts.

We fuse for *completeness* (no single sensor sees everything), to beat *noise and drift* (one sensor's weakness is another's strength), and for *redundancy* (if one sensor fails, the others keep working).

A mental model and the sensor's flaws

Think of a mug of hot coffee. Touching the mug is quick but rough; watching the steam is slow but steadier. A good guess blends both. Fusion formalizes that blend, and it works only as well as your understanding of each sensor's imperfections: *bias* (a constant offset), *noise* (random scatter), *drift* (an error that grows with time), resolution and range, bandwidth and latency, and sampling rate.

Fusion can happen at three levels. At the *raw-data level* you combine the measurements themselves (two temperature readings). At the *feature level* you combine quantities extracted from each sensor (a heart rate from the optical sensor and a step rate from the accelerometer). At the *decision level* you combine conclusions (each channel's decoded character, or each detector's "obstacle ahead"). First projects usually start at the raw level. Whatever the level, the data must be *aligned* first: expressed in the same coordinate frame, calibrated for scale and bias, and synchronized in time. Two sensors whose clocks disagree will "smear" any fast motion when you combine them.

Weighting by trust

Suppose two sensors measure the same quantity and report x_1 and x_2 with independent noise of standard deviation σ_1 and σ_2. The best linear combination weights each reading by the inverse of its noise variance:

$$\hat{x} = \frac{x_1/\sigma_1^2 + x_2/\sigma_2^2}{1/\sigma_1^2 + 1/\sigma_2^2}, \qquad \frac{1}{\hat{\sigma}^2} = \frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2}. \tag{5.22}$$

The noisier sensor gets less weight but is never ignored, and the fused uncertainty $\hat{\sigma}$ is smaller than either input's. The same result can be written as a correction:

$$\hat{x} = x_1 + K(x_2 - x_1), \qquad K = \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2}. \tag{5.23}$$

Start from the first reading and move toward the second by a fraction K, the *gain*. If sensor 2 is perfect ($\sigma_2 = 0$), $K = 1$ and you take its value. If it is useless ($\sigma_2 \to \infty$), $K = 0$ and you ignore it.

> **Worked Example 5.30** *Fusing the coffee estimates*
>
> Touching the mug suggests 70 °C with an uncertainty of $\sigma_1 = 6$ °C. The steam suggests 62 °C with $\sigma_2 = 3$ °C. What is the fused estimate and its uncertainty?
>
> Solution. The weights are proportional to $1/36$ and $1/9$, so the steam counts four times as much as the touch. Using Eq. (5.23),

$K = 36/(36+9) = 0.8$:

$$\hat{x} = 70 + 0.8\,(62 - 70) = 63.6\,^{\circ}\mathrm{C},$$
$$\hat{\sigma} = \left(\tfrac{1}{36} + \tfrac{1}{9}\right)^{-1/2} = \sqrt{7.2} = 2.68\,^{\circ}\mathrm{C}.$$

The fused uncertainty, 2.68 °C, beats even the better sensor's 3 °C. ■

The complementary filter

Weighting by noise works when both sensors are equally good at all speeds. Often they are not. Consider measuring the tilt angle of a board with the inertial measurement unit (IMU) on the Nano 33 BLE Sense. The *gyroscope* measures rotation rate ω; integrating it gives angle with almost no noise in the short term, but any small bias integrates into a drift that grows without limit. The *accelerometer* measures the direction of gravity, which gives angle directly with no drift, but it is noisy and is fooled whenever the board accelerates. The gyro is good at high frequencies and the accelerometer at low frequencies, so let each handle the part it is good at:

$$\hat{\theta}_k = \alpha\left(\hat{\theta}_{k-1} + \omega_k\,\Delta t\right) + (1-\alpha)\,\theta_{\mathrm{acc},k}, \tag{5.24}$$

with α close to 1 (often 0.95 to 0.99). The gyro path is effectively high-pass filtered and the accelerometer path low-pass filtered, with the two filters adding to exactly one, hence *complementary* (Figure 5.23). The crossover time constant is $\tau = \alpha\Delta t/(1-\alpha)$: over times shorter than τ you trust the gyro, and over longer times the accelerometer pulls out the drift.

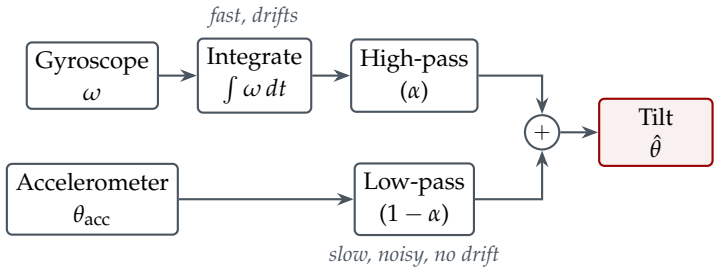


Figure 5.23: A complementary filter for tilt. The gyroscope supplies fast changes, the accelerometer supplies the long-term average, and the two weights sum to one.

Worked Example 5.31 *One step of a complementary filter*

A complementary filter runs every $\Delta t = 10\,\mathrm{ms}$ with $\alpha = 0.98$. The previous estimate is 10.0°, the gyroscope reads 20 °/s, and the accelerometer says the tilt is 12.0°. Find the new estimate and the filter's time constant.

SOLUTION. The gyro predicts $10.0 + (20)(0.01) = 10.2°$. Blend with the accelerometer:

$$\hat{\theta} = 0.98(10.2) + 0.02(12.0) = 9.996 + 0.240 = 10.24°.$$

The accelerometer nudges the estimate by only $0.04°$ per step, so a single noisy accelerometer reading cannot jerk it around. The time constant is $\tau = (0.98)(10\,\text{ms})/0.02 = 0.49\,\text{s}$. Over about half a second, any gyro drift is pulled back toward the accelerometer's answer. ■

WORKED EXAMPLE 5.32 *How much does gyro bias hurt?*

The gyroscope of the previous example has a bias: it reads $0.5\,°/\text{s}$ while the board sits still and level. (a) How far off is the angle after one minute if you only integrate the gyro? (b) With the complementary filter ($\Delta t = 10\,\text{ms}$, $\alpha = 0.98$), the estimate settles at a constant error. Find it. (c) What α keeps that error below $0.1°$, and what is the cost?

SOLUTION. (a) Integration turns the bias into a steady drift. After one minute it amounts to $(0.5\,°/\text{s})(60\,\text{s}) = 30°$, and it keeps growing.

(b) With the board still, the accelerometer reads 0 on average and the gyro reads $b = 0.5\,°/\text{s}$. In steady state the estimate stops changing, so $\hat{\theta}_k = \hat{\theta}_{k-1} = \hat{\theta}$ in Eq. (5.24):

$$\hat{\theta} = \alpha(\hat{\theta} + b\,\Delta t) + (1-\alpha)\cdot 0 \quad \Rightarrow \quad \hat{\theta} = \frac{\alpha\, b\,\Delta t}{1-\alpha} = b\,\tau.$$

With $\tau = 0.49\,\text{s}$ the error is $(0.5)(0.49) = 0.245°$, a constant instead of a drift.

(c) The error is $b\tau$, so it stays below $0.1°$ when $\tau < 0.1/0.5 = 0.2\,\text{s}$. Solving $\tau = \alpha\Delta t/(1-\alpha)$ for α gives $\alpha = \tau/(\tau + \Delta t) = 0.2/0.21 = 0.952$. The cost is noise: a smaller α hands more of each step to the noisy accelerometer, and the estimate follows its jitter and its mistakes during acceleration more closely. ■

The Kalman filter

The Kalman filter, also called linear quadratic estimation, combines the two ideas above and repeats them in time. It keeps an estimate $\hat{x}$ of the system's state (say a car's position) and a variance P that says how uncertain that estimate is. Each time step has two parts (Figure 5.24).

Predict. Use a model of how the state changes to move the estimate

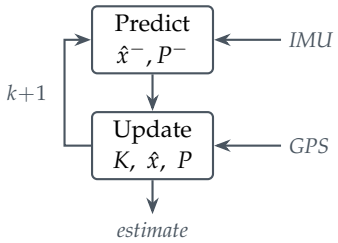


Figure 5.24: The Kalman filter loop: predict with a motion model (here driven by the IMU), then correct with a measurement (GPS).

forward. If the car's speed u comes from the wheel odometer or IMU,

$$\hat{x}^- = \hat{x} + u\,\Delta t, \qquad P^- = P + Q. \tag{5.25}$$

The process noise Q makes the uncertainty grow, because the model and the speed sensor are imperfect.

Update. When a measurement z with noise variance R arrives (say from GPS), blend it in exactly as in Eq. (5.23):

$$K = \frac{P^-}{P^- + R}, \qquad \hat{x} = \hat{x}^- + K\,(z - \hat{x}^-), \qquad P = (1 - K)\,P^-. \tag{5.26}$$

The uncertainty shrinks after every measurement and grows between them. The gain adjusts itself: when the prediction is uncertain, K is large and the filter trusts the measurement; when the measurement is noisy, K is small. In a self-driving example, the IMU measures relative motion quickly but drifts, while GPS gives absolute position but is noisy and slow to update. Figure 5.25 shows the result of fusing them: the IMU alone drifts away, GPS alone scatters by several meters, and the fused estimate stays within about a meter.

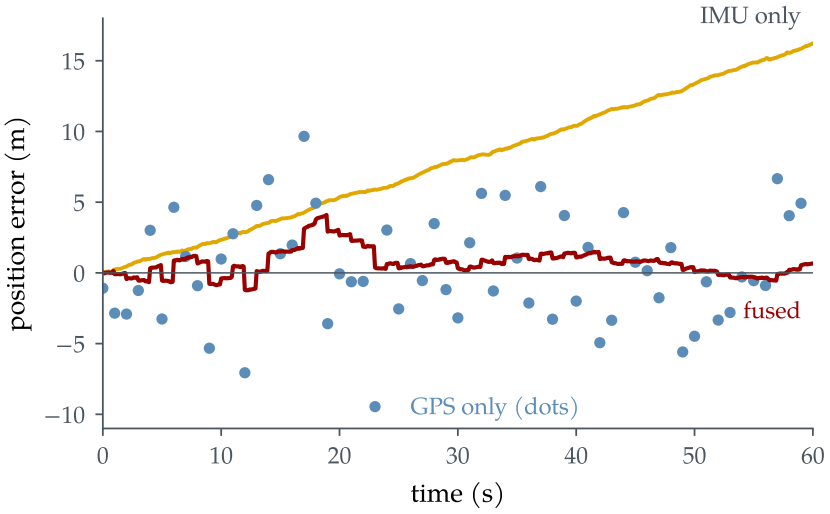


Figure 5.25: Simulated position error of a car over one minute. Dead reckoning from the IMU drifts steadily (gold). GPS fixes once per second scatter by several meters (dots). A Kalman filter that predicts with the IMU and corrects with GPS stays close to zero (cardinal). RMS errors: GPS 3.7 m, fused 1.2 m.

Worked Example 5.33 *One Kalman step for a car*

A car's position estimate is $\hat{x} = 100\,\text{m}$ with variance $P = 4\,\text{m}^2$. The odometer says the car moves at $10\,\text{m/s}$, and over the next $1\,\text{s}$ the process noise adds $Q = 1\,\text{m}^2$. A GPS fix then reads $z = 113\,\text{m}$ with variance $R = 20\,\text{m}^2$. Find the prediction, the gain, and the updated estimate and variance.

Solution. Predict: $\hat{x}^- = 100 + (10)(1) = 110\,\text{m}$ and $P^- = 4 + 1 = 5\,\text{m}^2$.

Update: $K = 5/(5+20) = 0.20$, so

$$\hat{x} = 110 + 0.20\,(113 - 110) = 110.6\,\text{m},$$
$$P = (1 - 0.20)(5) = 4.0\,\text{m}^2.$$

The GPS reading was 3 m ahead of the prediction, but GPS is four times noisier than the prediction, so the filter moves only a fifth of the way toward it. ■

Fusing communication channels

Fusion is not only for physical measurements. The fusion demo is shown in class rather than done as a lab: two Arduino boards send one message over light and sound at once (an LED to a light-dependent resistor, and a buzzer to a microphone) and fuse the decoded characters. The light channel frames each character with a start bit and a stop bit, like Task 1, but sends the data bits MSB first; the sound channel uses the dot-and-dash pulse-width code of Section 5.4. The demo's *OR-fusion* rule takes, for each position, the light channel's character if it is valid (printable, or passing a parity check) and otherwise the sound channel's. Cover the LED with a hand and the message still arrives by sound; clap near the microphone and it arrives by light. With three channels, the receiver can instead take a *majority vote*.

A few habits make any fusion system work better. Reject obvious outliers, and *gate* measurements, accepting one only if it is close enough to the prediction. Lower the weight of a sensor that freezes or saturates. Delay the faster signal to line up with a late one. Calibrate biases with a quick still measurement at startup, keep timestamps consistent, and check units and coordinate frames.

WORKED EXAMPLE 5.34 *How much does OR-fusion help?*

The light channel garbles 5% of characters and the sound channel 10%, independently. A parity check flags every garbled character. (a) With OR-fusion, what fraction of characters is lost? (b) Three channels each garble 10% of characters, independently, and the receiver takes a majority vote without any parity check. What fraction is lost, assuming a character is lost whenever two or more channels are wrong?

SOLUTION. (a) A character is lost only if both channels fail, with probability $0.05 \times 0.10 = 0.005$, or 0.5%. That is ten times better than the light channel alone.

(b) The vote fails if exactly two channels are wrong or all three

are:

$$P = 3(0.1)^2(0.9) + (0.1)^3 = 0.027 + 0.001 = 0.028.$$

Majority voting cuts the loss from 10% to 2.8%. It helps less than OR-fusion with error detection because it has no way to tell which channel is wrong; it can only count. ■

5.10 *Laboratory: LiFi, Sending Text over Light*

Lab 5.1 at a glance

Goal. Send text from one demo board to another with light: first with the firmware's own on-off keying link (Task 1), then with a link you build in Python, with framing, synchronization by correlation, decisions, bit error rate, and eye diagrams for OOK and PAM-4.

Hardware. Two demo boards and two USB data cables, on one laptop or on two partners' laptops. One board alone also works (loopback test).

Notebook. `lab5.1-lifi.ipynb` in the kit folder `lab5.1-lifi`, with the 45-byte message file `message_small_v2.txt`.

Before the lab. Read Sections 5.3 to 5.8. Run the notebook once at home with `SIMULATE = True` in **A1**; every part works with simulated boards. For the partner setup, decide who sends and who receives.

Time. About 100 minutes (Parts A to F: 10, 15, 15, 30, 10, and 20); the partner setup adds about 10. Work in pairs.

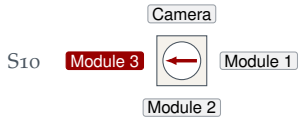
You hand in. `lab5.1_submission.json` on the course portal (in the partner setup both partners upload the receiver's file); keep your notebook with your answers.

Key takeaways

- A fixed threshold fails first: Task 1 decides at 200 counts, and the 1 level a few centimeters away is only 150 to 250 counts.
- A known 16-bit preamble, found by correlation, marks where the frame starts; each bit is then decided once, mid-symbol.
- Zero errors in N bits only shows BER $< 3/N$, 0.008 for the 360-bit message; noise margins of tens of σ predict far less.
- The receiver is a low-pass filter: its fall time, about 0.65 ms ($\tau = t_f/2.2$), shrinks the eye as the symbol period approaches it.
- PAM-4 doubles the bit rate at the same symbol period, but its three gaps make each eye a third as tall, so it fails first.

Set up the boards for Lab 5.1

Selector. On *both* boards turn S10 to Module 3 (biosensing block), which also powers pair 2, the communication LED and photodiode:



Jumper. On both boards put the cap of J5 on A–P2 (pins 2 and 3), so A6 reads the pair 2 photodiode, RX. The receiver needs it; with both boards on one laptop the notebook decides which board sends, so set both the same way:



cap on P1–A: A6 reads pair 1
cap on A–P2: A6 reads RX (pair 2)

Placement. The LED of pair 2 (printed TX, driven by DAC channel 4) and its photodiode (RX) sit at the right edge of each board. They are side-view parts that face out over the edge, along the table rather than up from it. Lay the two boards flat with their right edges facing each other, a few centimeters apart, so one board is turned half a turn relative to the other (Figure 5.9). Then slide one board along the gap until the sender's TX points straight at the receiver's RX. Shade the receiver from sunlight and filament lamps, whose near-infrared light passes its filter.

One board only (loopback). LED pair 1 (D3, DAC channel 3) shines across onto the same board's pair 1 photodiode (D4), so put J5 on P1–A (pins 1 and 2):



cap on P1–A: A6 reads pair 1
cap on A–P2: A6 reads RX (pair 2)

Check. The first notebook cell runs `module` on every board and confirms that its selector is on `Module 3` (biosensing block). If a board's reading does not fit, the cell prints a warning; check that S10 on that board points at Module 3 and carry on.

Afterwards. Put J5 back on P1–A, the position the pulse-oximeter lab (Lab 3.2) uses.

Figure 5.26 shows S10, J5, and TX and RX on the board, and Figure 5.9 shows two boards set up for the link.



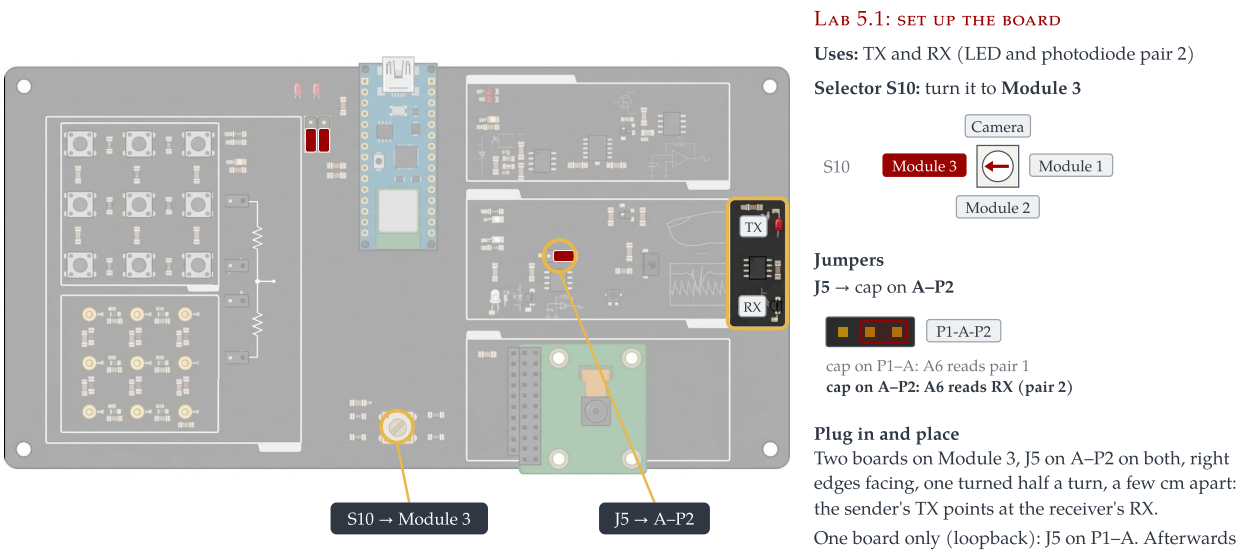


Figure 5.26: The board for Lab 5.1. The communication LED and photodiode, pair 2 (TX and RX, highlighted), carry the link. S10 is on Module 3 on both boards, and J5 is on A–P2 on both boards (P1–A for the one-board loopback test).

Objectives

- Receive the firmware's test message over light and find what breaks the link first.
- Measure the optical channel (dark level, noise, and received light versus LED drive) and choose signal levels and thresholds from the data.
- Frame a text file, find its start by correlation, decide each symbol, and compute the bit error rate for OOK and PAM-4.
- Draw eye diagrams and measure how the symbol period, the receiver's time constant, and the number of levels set the error rate.

Background

The theory is in Sections 5.3 to 5.8. Task 1 is the firmware link of Figure 5.7: ten 3 ms slots per character, least significant bit first, and a fixed receiver threshold of 200 counts.

For Parts C to F the notebook builds the whole symbol sequence in Python and loads it into the sending board, which plays it on LED pair 2 with exact timing, one symbol every 4 to 0.5 ms. The receiving board records its photodiode on A6, about 16 samples per symbol, starting the moment the light first rises above a trigger level. Python does everything else: it finds the start of the frame by correlation (Eq. (5.21)), samples each symbol at its middle, compares the samples with the thresholds, strips the markers, decodes the UTF-8 text, counts

LiFi. Harald Haas popularized the name in a 2011 TED talk, and in 2023 the IEEE added light communication to the WiFi family of standards as IEEE 802.11bb. Visible light spans 400 to 790 THz of unregulated spectrum.

SETUP	boards and J5	who runs what	differences
`"two"` (recommended)	two boards on one laptop; J5 on A–P2 on both	one notebook runs every step	the first board found sends, the second receives
`"partner"`	one board per laptop; J5 on A–P2 on both; `ROLE` set on each laptop	receiver: every step; sender: **A1** and the steps marked BOTH LAPTOPS	receiver first, then sender (**B1**: sender first); the sender types `OOK_V` (**D1**) and `PAM_V` (**F2**) from the receiver's printout; one submission file for both
`"one"`	one board; J5 on P1–A	one notebook	LED pair 1 shines on the board's own photodiode; **B1** is skipped

Table 5.5: The three setups of Lab 5.1. Steps marked BOTH LAPTOPS: **A2**, **B1**, **C1**, **D1**, **D2**, **E1**, **E2**, **F2**, **F3**, **F4**, and Save.

errors, and draws eye diagrams. Every transmission goes through one notebook function, `link()`, which returns the recording and the time between samples:

```
START_BITS = "1010101001010101"            # 0xAA 0x55
END_BITS   = "0101010110101010"            # 0x55 0xAA
frame = START_BITS + text_to_bits(MESSAGE) + END_BITS   # 392 bits
raw, dt = link(frame, 4, OOK_V)            # 4 ms per bit
```

The message file holds *This is a test message!!* and *Can you decode this?* on two lines, 45 bytes or 360 bits.

The notebook offers three setups, chosen with `SETUP` in **A1** (Table 5.5). In the partner setup the receiver's notebook does the analysis, and after **A1** the sender runs only the cells whose first line is `# BOTH LAPTOPS`, marked BOTH LAPTOPS below: the receiver runs each of them first (its board then waits up to 30 s for light) and the sender right after; in loops the sender waits 3 s before every transmission. **B1** goes the other way round. Set `SIMULATE = True` to practice at home on simulated boards.

Procedure

Each step below is one code cell of the notebook, with the same label above it (**A1**, **A2**, ...). Run the cells in order with Shift+Enter; a step tagged EDIT or ANSWER needs your input in the cell first (Section 0.11 explains the tags).

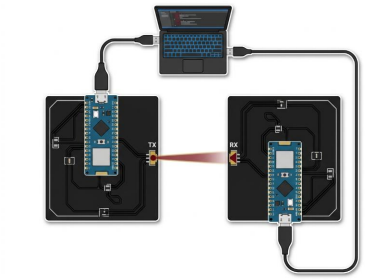
Part A: Choose your setup and connect

GOAL Pick a setup, align the boards, and connect. about 10 min

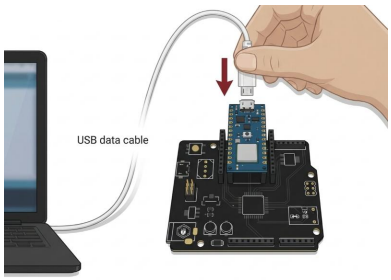
A1 ◆DO Choose your setup and connect

1. Turn S10 to Module 3 on every board you use, and put J5 on A–P2 on both boards (one-board setup: P1–A), as in the box above.
2. Lay the boards flat as in Fig. 5.9 and slide one along the gap until the sender's TX points at the receiver's RX (the receiver's TX then faces the sender's RX as well).
3. Plug in the boards with USB *data* cables.
4. In A1 set `SETUP` (line marked ✎) to `"two"`, `"partner"`, or `"one"` (Table 5.5); in the partner setup also set `ROLE` to `"sender"` on one laptop and `"receiver"` on the other. Run A1. It finds the boards, uploads the course firmware if needed, and checks the selectors.

YOU SHOULD SEE every connection line starting with `[ok]`, and a line that names your setup, for example `This notebook sends with tx and receives with rx.` With `"two"`, `Found 1 board(s)` means one cable is not a data cable.



A1 Right edges facing, a few centimeters apart, one board turned half a turn; slide one until TX points at RX.



A1 Plug both boards into this laptop (partner setup: one board per laptop), then run A1.

A2 ▸RUN The `link()` function

BOTH LAPTOPS. A2 defines `link()`, through which every transmission of the lab goes. It loads the symbols into the sending board and starts the receiving board's capture, which begins the moment the received light first rises above the level `TRIGGER` (set in Part C), and it returns the received samples and the time between them. You do not change it. The cell prints nothing.

Part B: Task 1, the firmware's own text link

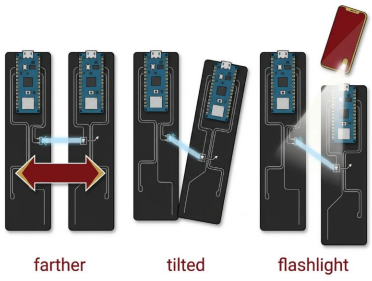
GOAL Send text from one board to the other with the firmware's own link. about 15 min

B1 ◆DO Task 1, then stress the link

BOTH LAPTOPS. The course firmware has a text link of its own, two commands you could also type into a serial terminal:

```
communications send
communications receive
```

`communications send <text>` repeats the text on LED pair 2 once per second; with no text it sends *This is a test transmission!* Each character is a frame of ten 3 ms slots: a start slot with the LED off, eight data bits least significant bit first, and a stop slot with the LED on



B1 Stress the link: move the boards apart, tilt one, shine a phone flashlight on the receiver. (The picture draws the boards as narrow strips; in the "farther" panel the arrow, not the gap, shows the move.)

(Fig. 5.7). `communications receive` compares the photodiode with a fixed threshold of 200 counts and, on each on-to-off edge, waits 1.5 slots and reads eight bits one slot apart. B1 gives the sender and the receiver these commands and shows what the receiver prints, like a Serial Monitor. The first line may be cut off. Partner setup: here the *sender* runs B1 first (it sends for 15 s) and the receiver right after; the receiver waits until the sender's cell has finished before going on to Part C.

1. Adjust the alignment until the message repeats cleanly.
2. Stress the link, rerunning B1 each time: move the boards apart, tilt one board, and shine a phone flashlight on the receiver. (The photodiode's filter blocks much of a white flashlight's light; sunlight or a filament lamp is a harder test.)
3. Write down what fails first and the longest distance that still works (Question 1).

YOU SHOULD SEE `This is a test transmission!` once per line, repeating cleanly while the boards are close and aligned. Because the threshold is fixed at 200 counts, Task 1 is usually the first thing to fail as the boards move apart, long before the light is too weak for Parts D to F. One-board setup: a line saying that Task 1 needs two boards.

✓CHECK **Checkpoint 1** The Task 1 message arrives cleanly (one-board setup: the staircase of **C1** shows light above the dark level instead). Text crossed the gap between two boards on a beam of light.

Part C: Channel check

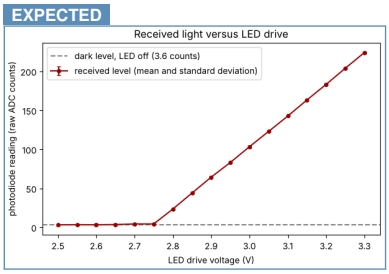
GOAL Measure the dark level and the light staircase, and choose your OOK levels. about 15 min

Before choosing light levels, measure what the receiver actually sees. Put the boards back in their best position from Part B and keep the setup still from now on.

C1 ▸ RUN Dark level and light staircase

BOTH LAPTOPS. With the LED off, the photodiode still sees room light and noise: C1 reads 1000 samples 0.2 ms apart with every LED off, prints the dark level and the noise σ (the standard deviation), and sets a first trigger level of dark $+ 8\sigma + 5$ counts. Then it steps the LED drive through 0 V and 2.50 to 3.30 V in 0.05 V steps, since the LED gives no light below its turn-on voltage and the DAC stops rising at 3.3 V. Each transmission is a bright marker symbol, which starts the receiver's capture, and three test levels of 20 ms each; the receiver averages the

EXPECTED



C1 No light up to the turn-on voltage, then a rise to the top step. The simulated curve is straight; a real one bends upward.

second half of every step. C1 plots the received level against the drive and saves it in `staircase.csv`.

YOU SHOULD SEE a dark level below about 20 counts with a noise of about 2 counts on a real board (the simulator: 3.5 and 1 count); above 8 counts of noise the cell warns. Then no light below the turn-on voltage, near 2.75 V, and a rise to roughly 150 to 250 counts at 3.3 V for boards a few centimeters apart, your own version of Fig. 5.10.

C2 ANSWER Choose the OOK levels

C2 · ANSWER in the notebook

```
v_turn_on = ...            # ✎ first drive (V) with light
OOK_V = [..., ...]         # ✎ drive (V) for bit 0, bit 1
OOK_LEVELS = [..., ...]    # ✎ expected levels (counts)
OOK_THRESHOLD = ...        # ✎ decision threshold (counts)

print(f"light starts at about {v_turn_on:.2f} V")
print(f"OOK_V = {OOK_V}   (partner setup: tell the sender)")
print(f"levels {OOK_LEVELS[0]:.1f} and {OOK_LEVELS[1]:.1f} counts, "
      f"threshold {OOK_THRESHOLD:.1f} counts")
# trigger for the data: a quarter of the way up
TRIGGER = OOK_LEVELS[0] + (OOK_LEVELS[1] - OOK_LEVELS[0]) / 4
print(f"trigger level for the data: {TRIGGER:.0f} counts")
```

TYPE IN `v_turn_on`, the turn-on voltage: the first drive voltage whose level is more than 5σ above the dark level. `OOK_V`, the on-off keying (OOK) drive voltages for bit 0 and bit 1: 0 V for a 0 and the brightest step for a 1, so that the received levels are as far apart as possible. `OOK_LEVELS`, the received level you expect for each, and `OOK_THRESHOLD`, halfway between them. The cell then sets the trigger for the data a quarter of the way from the 0 level to the 1 level.

UNIT volts for the drive, raw ADC counts for the levels and the threshold.

WHERE TO FIND IT the staircase plot of **C1**, the arrays `stair_v` and `stair`, and `DARK` and `NOISE`. You may type the numbers or compute them: `np.argmax` of a boolean array gives the first `True`.

YOU SHOULD SEE light from about 2.75 V, `OOK_V = [0.0, 3.3]`, and a threshold halfway between the two levels (simulator: 3.5 and 224 counts, threshold 114).

Part D: On-off keying

GOAL Send a whole text file with light, decode it, and count the bit errors. about 30 min

The message file `message_small_v2.txt` holds 45 bytes on two lines. Each byte is sent as 8 bits, most significant bit first. The frame starts with the two marker bytes `0xAA 0x55` (the preamble) and ends with `0x55 0xAA` (Section 5.8):

`10101010 01010101` + 360 payload bits + `01010101 10101010` = 392 bits.

D1 EDIT Write `text_to_bits(text)`

D1 · EDIT in the notebook

```
1  # BOTH LAPTOPS
2  if MODE == "sender":
3      OOK_V = [0.0, 3.3]     # ✎ the OOK_V the receiver printed
4  MESSAGE = open("message_small_v2.txt",
       encoding="utf-8").read().strip()
5  START_BITS = "1010101001010101"     # 0xAA 0x55
6  END_BITS = "0101010110101010"       # 0x55 0xAA
7
8
9  def text_to_bits(text):
10     """The UTF-8 bytes of `text` as a string of '0' and '1', most
           significant bit first."""
11     return ""    # ✎ replace with your code
12
13
14 def bits_to_text(bits):
15     """Inverse of text_to_bits(). Bytes that are not valid UTF-8
           show up as a replacement mark."""
16     data = bytes(int(bits[i:i + 8], 2) for i in range(0,
           len(bits) - 7, 8))
17     return data.decode("utf-8", errors="replace")
18
19
20 frame = START_BITS + text_to_bits(MESSAGE) + END_BITS
21 print(f"{len(MESSAGE.encode())} bytes -> frame of {len(frame)}
       bits")
22 print("first character:", repr(MESSAGE[0]),
       text_to_bits(MESSAGE[0]))
```

WHAT TO WRITE BOTH LAPTOPS. the return line of `text_to_bits` (line 11): encode the text with UTF-8 and write each byte as 8 binary digits, most significant bit first, all in one string of `0` and `1`. Partner setup, sender laptop only: also type the `OOK_V` that the receiver's notebook printed in C2 (line 3).

HINT `text.encode("utf-8")` gives the bytes, `f"{b:08b}"` writes one byte `b` as 8 digits, and `"".join(...)` glues the pieces together (Worked Example 5.4).

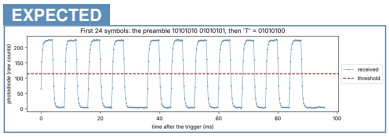
DONE WHEN the cell prints `45 bytes -> frame of 392 bits` and the first character `'T' 01010100`.

D2 ▸ RUN Send the frame at 4 ms per bit

BOTH LAPTOPS. D2 sends the frame with one bit per symbol, 4 ms per symbol (250 bit/s); the receiver records about 16 samples per symbol. It plots the whole capture and the first 24 symbols, and defines three helpers: `ideal_waveform`, the recording you would get with no noise and no delay, `sample_avg` for D6, and `show_decisions`, the plot of D5.

YOU SHOULD SEE two clean levels in the whole capture, and in the zoom the preamble `10101010 01010101` followed by `T = 01010100`, as in Fig. 5.28a.

EXPECTED



D2 The bottom panel of two: the preamble, then `T`; two flat levels far from the threshold, with short slanted edges.

D3 ✎ EDIT Write `find_start(x, template)`

D3 · ✎ EDIT in the notebook

```
1  def find_start(x, template):
2      """Index in x where the known waveform `template` fits best
           (largest correlation)."""
3      c = ...                      # ✎ correlate, means removed
4      return int(np.argmax(c))     # position of the best match
5
6
7  preamble = ideal_waveform(START_BITS, PERIOD_MS, dt, OOK_LEVELS)
8  pad = int(round(PERIOD_MS / 1000 / dt))   # samples per symbol
9  x = np.concatenate([np.full(pad, OOK_LEVELS[0]), raw])
10 search = x[:len(preamble) + 8 * pad]      # start within 8
       symbols
11 i0 = find_start(search, preamble) - pad
12 t0 = i0 * dt
13 print(f"the frame starts {t0 * 1000:.2f} ms after the capture
       started")
```

WHAT TO WRITE the line `c = ...` (line 3): the correlation of the recording `x` minus its mean with the ideal preamble waveform `template` minus its mean, $c[k] = \sum_j (x[k+j] - \bar{x})(p[j] - \bar{p})$ (Eq. (5.21)). The capture begins when the light first crosses the trigger, which is close to, but not exactly at, the start of the frame, and the amplifier delays everything a little; the best match gives the start precisely (Worked Example 5.23).

HINT `np.correlate(a, b, mode="valid")[k]` computes `sum(a[k:k+len(b)] * b)`. Subtract each mean first.

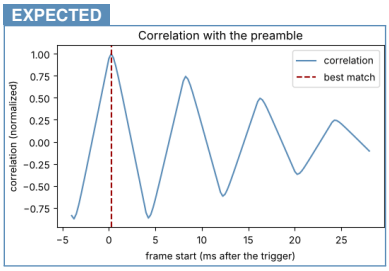
DONE WHEN the cell prints the frame start, a fraction of a millisecond to a few milliseconds after the capture started.

D4 ▸ RUN Correlation with the preamble

D4 plots the correlation against the frame start, with its peak marked, and the received preamble with the ideal one laid on top at the start found.

YOU SHOULD SEE one clear peak, and the ideal preamble lying on the received one.

EXPECTED



D4 The left panel of two: one tallest peak, at the frame start; the repeating preamble adds smaller side peaks.

D5 EDIT Write `sample_mid()` and count the errors

D5 · EDIT in the notebook

```
1  def sample_mid(x, dt, t0, period_ms, n_symbols):
2      """x at the middle of each of n_symbols symbols; symbol 0
           starts at t0 (s), samples are dt apart."""
3      T = period_ms / 1000
4      t = ...       # ✎ middles of symbols 0 .. n_symbols - 1 (s)
5      idx = ...     # ✎ nearest sample indices, in 0 .. len(x) - 1
6      return x[idx]
7
8
9  values = sample_mid(raw, dt, t0, PERIOD_MS, len(frame))
10 rx_bits = "".join("1" if v > OOK_THRESHOLD else "0" for v in
       values)
11 n_errors = ...       # ✎ number of payload bits received wrong
12 ber = ...            # ✎ bit error rate
13
14 n_payload = len(frame) - 32
15 print("sent    ", frame[:48])
16 print("received", rx_bits[:48])
17 print(f"{n_errors} errors in {n_payload} payload bits: BER =
       {ber:.2e}"
18       + ("" if n_errors else f"  (zero errors: BER very likely
             below 3/{n_payload} = {3 / n_payload:.1e})"))
19 print("received text:")
20 print(bits_to_text(rx_bits[16:-16]))
21 show_decisions(raw, dt, t0, PERIOD_MS, rx_bits)
```

WHAT TO WRITE in `sample_mid` (lines 4 and 5): `t`, the times of the middles of the symbols, and `idx`, the nearest sample indices. Right after a transition the signal is still moving, so take one sample in the *middle* of every symbol: symbol k is centered at $t_0 + (k + \frac{1}{2})T$ (Fig. 5.27). Then `n_errors` and `ber` (lines 11 and 12): the payload bits received wrong, leaving out the 16 marker bits at each end, and the bit error rate (Eq. (5.17)). The cell then plots the first 40 symbols with a dot on every decision sample and the decided bits on top, and draws the whole frame as a strip of received bits, errors framed in red.

HINT `np.arange(n_symbols)` numbers the symbols. Divide the times by `dt`, round with `np.round`, make integers with `.astype(int)`, and keep the indices inside the recording with `np.clip`. Compare `frame[16:-16]` with `rx_bits[16:-16]` bit by bit.

DONE WHEN the cell prints the sent and received bits, `0 errors in 360 payload bits` (with zero errors it reports BER $< 3/360 = 0.008$), and the decoded text: *This is a test message!!* and *Can you decode this?*, and every dot sits in the middle of a flat stretch.



D5 The strip of all 392 received bits: no square framed in red when every bit arrived.

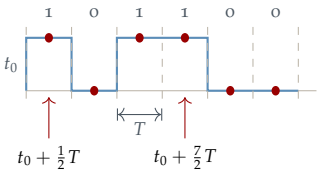


Figure 5.27: One sample (dot) in the middle of every symbol: symbol k is centered at $t_0 + (k + \frac{1}{2})T$, where t_0 is the frame start that D3 found.

D6 EDIT Noise margins and averaging

D6 · EDIT in the notebook

```
1  sent = np.array([int(b) for b in frame])   # the bits sent
2  v1 = values                                # one sample per bit
       (D5)
3  vavg, n_avg = sample_avg(raw, dt, t0, PERIOD_MS, len(frame))
4  mu0, mu1 = ..., ...    # ✎ means of the bit-0, bit-1 samples
5  s0, s1 = ..., ...      # ✎ their standard deviations
6  m0, m1 = ..., ...      # ✎ margins to the threshold (sigma)
```

```
ber_pred = ...          # ✎ predicted BER, Q(x) = norm.sf(x)
sa0, sa1 = ..., ...     # ✎ standard deviations of vavg

print(f"bit 0: mean {mu0:6.1f}, sd {s0:.2f} counts, margin
    {m0:6.1f} sigma")
print(f"bit 1: mean {mu1:6.1f}, sd {s1:.2f} counts, margin
    {m1:6.1f} sigma")
print(f"predicted BER {ber_pred:.1e}" + ("  (smaller than the
    computer can represent)" if ber_pred == 0 else ""))
print(f"averaging n = {np.median(n_avg):.0f} samples: sd shrinks
    by {s0 / sa0:.2f} (bit 0) and {s1 / sa1:.2f} (bit 1);"
      f" sqrt(n) = {np.sqrt(np.median(n_avg)):.2f}")
```

WHAT TO WRITE group the decision samples `v1` of **D5** by the bit that was sent (`sent`). `mu0`, `mu1` and `s0`, `s1`: the mean and standard deviation of each level; `m0`, `m1`: the distance from each level to the threshold in units of its standard deviation (the noise margins); `ber_pred`: the predicted BER $\frac{1}{2}\,[Q(m_0) + Q(m_1)]$ (Eq. (5.18)); and `sa0`, `sa1`: the standard deviations of `vavg`, the average of the samples in the middle half of each symbol.

HINT `v1[sent == 0]` picks the samples of the bits sent as 0; $Q(x)$ is `norm.sf(x)`.

DONE WHEN the cell prints both levels with margins of tens of σ, a predicted BER far below anything you can measure, and how much averaging n samples shrank σ, next to $\sqrt{n}$. With about 9 samples the simulator's σ shrinks by about 2 to 2.4 rather than 3 (Question 4).

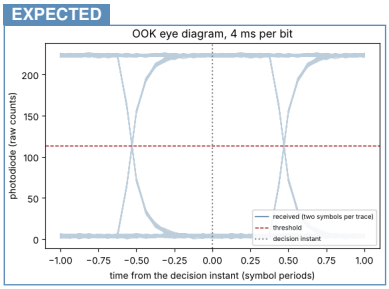
D7 ▸ RUN Eye diagram

`receive()` packs **D3** and **D5**, with your functions, into one call. It also looks for the end marker: if its position shows that the sender's symbols are clearly longer or shorter than set (more than 0.5 %), and the end marker then decodes correctly, it uses the measured period instead. `eye_diagram()` cuts the waveform into pieces two symbols long, centered on the decision instants, and draws them on top of each other (Section 5.7). The eye height is the vertical opening at the decision instant, the eye width the horizontal opening along the threshold (Fig. 5.19).

YOU SHOULD SEE the line `symbol period used: 4.000 ms (as set)` and a wide open eye with two tight bands, as in Fig. 5.28b.

✓CHECK **Checkpoint 2** The message is decoded with a small BER (**D5**), and the eye diagram of **D7** is open, with bands as thin as the noise of **D6**. A 45-byte file, sent bit by bit through the air and decoded by your own receiver.

EXPECTED



D7 Two thin bands far apart that cross the threshold only near the symbol edges: a wide open eye.

Part E: Speed

GOAL Push the bit rate until the eye closes, and measure why.

about 10 min

The photodiode amplifier is a low-pass filter with a time constant of a few hundred microseconds (Fig. 5.13, Eq. (5.9)). Shorter symbols leave it less time to settle, so the eye closes.

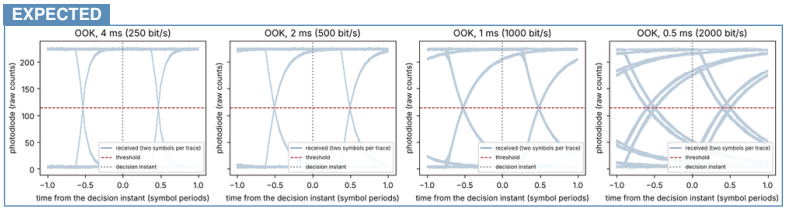
A white LED, an infrared detector. The board's LEDs are white (a blue chip under a yellow phosphor), but its photodiode responds only from about 750 to 1050 nm. Only the red tail of the LED's light reaches it, one reason the receiver needs a transimpedance gain as large as 3 MΩ.

E1 ▸ RUN Send at four speeds

BOTH LAPTOPS. E1 sends the same frame at 4, 2, 1, and 0.5 ms per bit; 0.5 ms is the fastest the firmware allows. The table lists for each period the bit rate, the sample spacing (`sample_us`), the period actually used (`period_used_ms`), the errors, the BER or its bound, and the *eye height*: the clear vertical opening at the decision instant, the lowest sample sent as the higher level minus the highest sample sent as the lower level (with four levels, the smallest of the three openings; negative means the eye is closed). E1 then plots the BER and the eye height against the bit rate and draws the four eyes.

YOU SHOULD SEE `sample_us` about 250, 125, 62, and 50, `period_used_ms` equal to the period you set, and an eye height that falls as the period shrinks (simulator: 215, 208, 175, and 119 counts at 4, 2, 1, and 0.5 ms). OOK usually stays error-free down to 1 or 0.5 ms per bit.

EXPECTED



E1 The four eyes. The bands stay thin, but the slanted edges eat into the opening as the symbol shortens; at 0.5 ms the eye is about half as tall as at 4 ms.

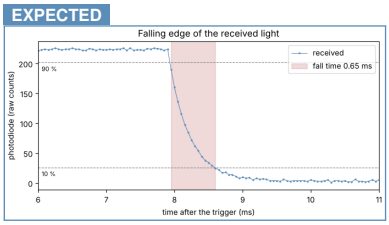
E2 EDIT Fall time and time constant

E2 · EDIT in the notebook

```
# BOTH LAPTOPS
raw_s, dt_s = link("11110000", 2, OOK_V, sps=40)

if raw_s is not None:
    t_s = np.arange(len(raw_s)) * dt_s * 1000   # ms
    hi = raw_s[(t_s > 4) & (t_s < 7)].mean()    # bright level
    lo = raw_s[(t_s > 12) & (t_s < 15)].mean()  # dark level
    after = t_s > 7                             # off at 8 ms
    t90 = ...       # ✎ first time (ms) below 90 % of the step
    t10 = ...       # ✎ the same for 10 %
    t_fall = ...    # ✎ fall time (ms)
    tau = ...       # ✎ time constant (ms)
```

EXPECTED



E2 The light falls along a smooth curve, not a step; the shaded band spans a few tenths of a millisecond.

```
13      rb_max = ...    # ✎ maximum OOK bit rate (bit/s)
14      print(f"fall time {t_fall:.2f} ms, time constant {tau:.2f}
            ms, R_b,max about {rb_max:.0f} bit/s")
15      show_edge(t_s, raw_s, lo, hi, t90, t10)
```

WHAT TO WRITE BOTH LAPTOPS. the cell sends four bright and four dark symbols, 2 ms each, and records the falling edge with fine time steps. Write `t90` and `t10` (lines 9 and 10): the first times after `after` begins where `raw_s` drops below 90% and below 10% of the way from `lo` to `hi`. Then the fall time `t_fall` $= t_{10} - t_{90}$, the time constant `tau` $= t_f/2.2$ (for a first-order response the fall time equals the rise time 2.2τ, Eq. (5.9)), and `rb_max` $\approx 1/(2t_f)$ in bit/s (lines 11 to 13) (Eq. (5.10), Worked Example 5.12). The cell then plots the falling edge with the 90% and 10% levels and shades the fall time.

HINT `np.argmax(condition)` finds the first sample where the condition is true: `t_s[after][np.argmax(raw_s[after] < lo + 0.9 * (hi - lo))]`. The times are in ms, so convert t_f to seconds for `rb_max`.

DONE WHEN the cell prints a fall time of a few tenths of a millisecond (simulator: 0.65 ms, $\tau = 0.30$ ms, $R_{b,\max}$ about 770 bit/s). Compare with the eye heights of **E1**.

Part F: PAM-4, two bits per symbol

GOAL Double the bit rate with four light levels, and compare with OOK. about 20 min

PAM-4 sends the bits in pairs, 00 → level 0, 01 → 1, 10 → 2, 11 → 3, so the frame needs only 196 symbols and the bit rate doubles at the same symbol period (Fig. 5.8). The price is that the levels sit closer together.

F1 ANSWER Choose the PAM-4 levels

F1 · ANSWER in the notebook

```
1  PAM_LEVELS = [..., ..., ..., ...]   # ✎ levels (counts), symbols
       0-3
2  PAM_V = [..., ..., ..., ...]        # ✎ drive voltages (V)
3  PAM_THRESHOLDS = [..., ..., ...]    # ✎ three thresholds (counts)
4
5  print(f"PAM_V = {PAM_V}   (partner setup: tell the sender)")
6  print("levels (counts):", np.round(PAM_LEVELS, 1), "
       thresholds:", np.round(PAM_THRESHOLDS, 1))
7  plot_staircase(PAM_LEVELS, PAM_V, PAM_THRESHOLDS)
```

TYPE IN `PAM_LEVELS`, four *received* levels evenly spaced from your OOK 0 level to your OOK 1 level; `PAM_V`, the four drive voltages that give them; and `PAM_THRESHOLDS`, the three thresholds halfway between neighboring levels. The cell plots them on the staircase.

UNIT raw counts for the levels and the thresholds, volts for the drive.

WHERE TO FIND IT your OOK levels and the staircase of **C1**. Levels 0 and 3 are the OOK levels; read the drive voltages of levels 1 and 2 off the staircase backwards with `np.interp(targets, levels, volts)`, passing only its rising part, from the turn-on voltage up (Worked Example 5.8).

YOU SHOULD SEE four evenly spaced levels on the staircase. On a real board the curve bends upward, so the two middle drive voltages bunch toward 3.3 V; on the straight simulated curve they are 2.93 and 3.12 V.

F2 EDIT Write `bits_to_pam()` and `pam_to_bits()`

F2 · EDIT in the notebook

```
1  # BOTH LAPTOPS
2  if MODE == "sender":
3      PAM_V = [0.0, 2.93, 3.12, 3.3]   # ✎ the PAM_V the receiver printed
4
5
6  def bits_to_pam(bits):
7      """'0'/'1' string -> string of PAM-4 symbols: '00'->'0', '01'->'1', '10'->'2', '11'->'3'."""
8      return ""    # ✎ replace with your code
9
10
11 def pam_to_bits(symbols):
12     """Inverse of bits_to_pam(): each symbol 0..3 back to two bits."""
13     return ""    # ✎ replace with your code
14
15
16 pam_frame = bits_to_pam(frame)
17 print(f"{len(frame)} bits -> {len(pam_frame)} PAM-4 symbols; preamble {pam_frame[:8]}")
```

WHAT TO WRITE BOTH LAPTOPS. the return lines of `bits_to_pam` (line 8) and of `pam_to_bits` (line 13), the conversions between a string of bits and a string of PAM-4 symbols, most significant bit first. Partner setup, sender laptop only: also type the `PAM_V` that the receiver's notebook printed in **F1** (line 3).

HINT `int("10", 2)` is 2, and `f"{2:02b}"` is `"10"`. Step through the bits two at a time with `range(0, len(bits), 2)`.

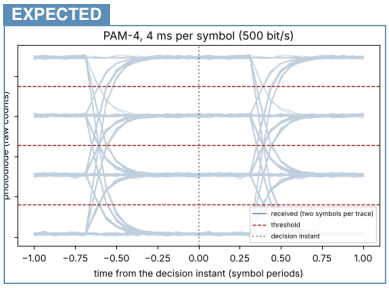
DONE WHEN the cell prints that the 392 bits became 196 PAM-4 symbols, and the preamble `22221111`.

F3 RUN Send PAM-4 at 4 ms per symbol

BOTH LAPTOPS. F3 sends the 196 symbols at 4 ms per symbol (500 bit/s), decodes them, prints the errors, the text, and a table of the four levels with their margins, and draws the OOK and PAM-4 eyes side by side.

YOU SHOULD SEE zero errors in 360 bits and the message. The PAM-4 eye has four levels and three openings, each about a third as tall as the OOK eye (simulator: about 68 against 215 counts), as in Fig. 5.28c.

EXPECTED



F3 The right panel of two: three open PAM-4 eyes, each about a third as tall as the OOK eye.

F4 ▸ RUN PAM-4 at four speeds, compared with OOK

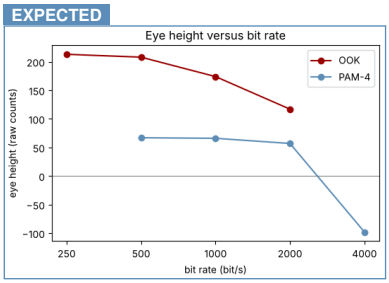
BOTH LAPTOPS. F4 repeats the speed test of **E1** with PAM-4, plots the BER and the eye height of both schemes against the *bit* rate, prints the fastest error-free bit rate of each, saves both tables in `lifi_speed.csv`, and switches the LEDs off.

YOU SHOULD SEE PAM-4 error-free down to about 1 ms per symbol (2000 bit/s); at 0.5 ms it usually fails, with a BER of 0.25 to 0.45. If your receiver is slower than 0.3 ms, OOK may also show errors at 0.5 ms and PAM-4 at 1 ms. The lab still works; your answers change.

✓CHECK **Checkpoint 3** The PAM-4 message is decoded (**F3**), and the BER and eye-height plot of **F4** compares OOK with PAM-4. Twice the bits per symbol, and you know exactly what it cost in noise margin.

If you have time (after the Save step). Move the receiving board to twice the distance, keep it aligned, and run **C1** again. By the inverse-square law (Section 5.5) the top of the staircase should rise only about a quarter as far above the dark level, though rarely exactly: the LED and the photodiode are not point sources, and moving a board changes its alignment.

EXPECTED

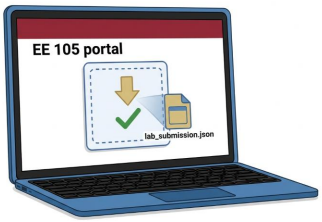


F4 The right panel of two: the PAM-4 eye height (blue) is about a third of the OOK one and drops below zero at the fastest rate.

Save and upload

Save ▸ RUN Write `lab5.1_submission.json`

BOTH LAPTOPS. First answer the Analysis questions below in the Questions cells at the end of the notebook (double-click a cell to edit it; Shift+Enter shows it formatted again). Then run the Save cell. It writes `lab5.1_submission.json` in the lab folder: your numbers, the bits you received, and the speed tables. Log in at `ee105.org/portal`, open Lab 5.1, and upload that file. In the partner setup only the receiver's notebook can write it (it holds the data): the receiver sends a copy to the sender (email or a shared folder), and each of you uploads it under your own login. Before you pack up, put J5 back on P1–A on every board, the position Lab 3.2 uses.



Save Upload `lab5.1_submission.json` on the course portal under Lab 5.1.

YOU SHOULD SEE `[ok] Saved lab5.1_submission.json`, and then the portal shows your grade. The sender's laptop in the partner setup prints that the receiver's notebook writes the file. A run on simulated boards counts as practice and earns no credit.

FINISH THE LAB

You built a light link: the firmware's text link tested against distance, tilt, and room light, a channel check, an OOK transmitter and receiver

with framing, correlation sync, BER, and eye diagrams, a speed test with the receiver's time constant, and a PAM-4 link that doubles the bit rate. The lab is done when every box is ticked.

☐	**A1**	DO	`SETUP` (and `ROLE`) chosen, boards aligned
☐	**B1**	DO	Task 1 received; what fails first, longest distance
☐	**C2**	ANSWER	`v_turn_on`, `OOK_V`, `OOK_LEVELS`, `OOK_THRESHOLD`
☐	**D1**	EDIT	`text_to_bits` (partner sender: also `OOK_V`)
☐	**D3**	EDIT	`find_start`
☐	**D5**	EDIT	`sample_mid`, `n_errors`, `ber`
☐	**D6**	EDIT	`mu0`, `s0`, `m0`, `...`, `ber_pred`, `sa0`, `sa1`
☐	**E2**	EDIT	`t90`, `t10`, `t_fall`, `tau`, `rb_max`
☐	**F1**	ANSWER	`PAM_LEVELS`, `PAM_V`, `PAM_THRESHOLDS`
☐	**F2**	EDIT	`bits_to_pam`, `pam_to_bits`
☐		CHECK	Checkpoint 1: the Task 1 message received cleanly
☐		CHECK	Checkpoint 2: OOK decoded, open eye
☐		CHECK	Checkpoint 3: PAM-4 decoded; OOK and PAM-4 compared
☐			Questions 1 to 6 answered in the notebook
☐	**Save**	SAVE	`lab5.1_submission.json` uploaded on the portal (both partners)
☐			J5 back on P1–A on every board

Typical results

Channel. On a real board the dark level is below about 20 counts with a noise of about 2 counts. Light appears near 2.75 V and reaches roughly 150 to 250 counts at 3.3 V for boards a few centimeters apart, a small signal even with 3 MΩ of gain, because only the red tail of the white LED's light passes the photodiode's infrared filter. The staircase bends upward (Figure 5.10), so the two middle PAM-4 drive voltages of **F1** are not evenly spaced, and Task 1 works only while the 1 level exceeds its fixed 200 counts.

OOK and PAM-4. At 4 ms per bit, zero errors in 360 bits is normal, with noise margins of tens of σ; averaging about 9 samples shrinks σ by about 2 to 2.4 rather than 3 (**D6**). The PAM-4 eye is about a third as tall as the OOK eye (Figure 5.28).

Speed. On the simulated boards the OOK eye height is 215, 208, 175, and 119 counts at 4, 2, 1, and 0.5 ms, with a fall time of 0.65 ms ($\tau = 0.30$ ms); the real receiver's time constant is a few hundred microseconds. Expect OOK to stay error-free down to about 1 or 0.5 ms per bit and PAM-4 down to about 1 ms per symbol; at 0.5 ms PAM-4 usually fails, with a BER of 0.25 to 0.45.

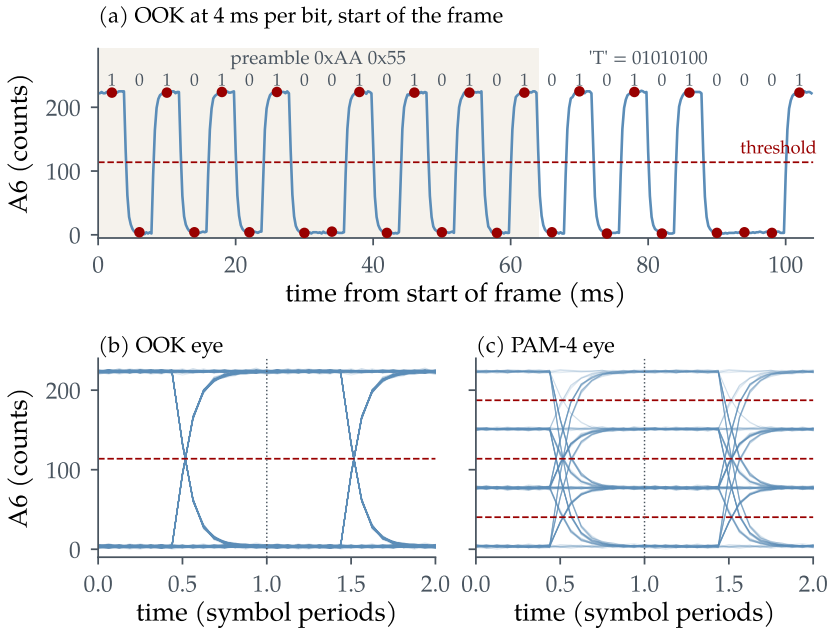


Figure 5.28: Typical Part D and F recordings (simulated boards) at 4 ms per symbol. (a) The start of the OOK frame: the preamble `10101010 01010101` and the first character, `T`. Each mid-symbol sample (dot) lies far above or below the threshold (dashed). (b) The OOK eye: levels about 220 counts apart and an eye height of about 214 counts. (c) The PAM-4 eye: four levels and three thresholds, each eye about 69 counts tall, about a third of the OOK eye. The short slanted edges come from the receiver's time constant.

Troubleshooting

- *"Found 1 board(s)."* One cable is a charge-only cable or is not plugged in. Use data cables, or switch to the partner or one-board setup (**A1**).
- *Task 1 prints nothing or junk* (**B1**). The 1 level is below the fixed 200-count threshold. Move the boards closer, line up the sender's TX with the receiver's RX, and check that J5 is on A–P2.
- *"No light reached the receiver."* No light rose above the trigger level for 30 s. Check the alignment, the selector on both boards, and J5 on the receiver. In the partner setup the receiver must run each shared cell first.
- *Noisy dark level* (**C1**). The other board's LED is still sending from **B1**, or a lamp or sunlight reaches the photodiode. Shade the receiver, wait a few seconds, and rerun the cell.
- *Garbled text at every period.* Room light can fire the trigger early, and the start search covers only the first 8 symbols after the trigger. Or a board moved. Shade the receiver and rerun Part C.
- *`period_used_ms` differs from the set period at* 0.5 ms. Each DAC update is an I^2C write of about 0.4 ms, so the sender may fall behind. The receiver then measures the true period from the end marker (it corrects differences up to 20%) and says so. Note it in your analysis.
- *One-board staircase flat at 1023 counts.* LED pair 1 can drive its own photodiode into saturation. Use as the 1 level in **C2** the lowest drive voltage on the flat top, so that the PAM-4 levels stay on the rising part of the curve.

Analysis questions

Answer in the Questions cells at the end of the notebook.

1. Report the dark level and its noise (**C1**), the turn-on voltage and the received 1 level (**C2**), the longest distance with a clean Task 1 message (**B1**), and what failed first. Why does Task 1 fail before your own receiver does?

2. Compute the symbol rate and the useful bit rate (payload bits per second, with the framing counted as overhead) of Task 1, of your OOK link at 4 ms, and of PAM-4 at 4 ms. Compare them with the earlier 12 ms link (83 and 167 bit/s).

3. Report the BER of every run in Parts D, E, and F. For runs with zero errors, give the bound $3/360$.

4. From **D6** and the PAM-4 level table of **F3**, give the mean and σ of each level, the noise margin of each threshold in units of σ, and the predicted BER from Eq. (5.18). Do your thresholds sit near the midpoints of the measured levels? By what factor did σ shrink when you averaged the middle half of each symbol? Compare with $\sqrt{n}$ and explain any difference.

5. Compare the OOK and PAM-4 eyes at 4 ms: eye height and level spacing. Which scheme would fail first if you moved the boards apart, and why (Worked Example 5.19)?

6. Why does the eye close at short symbol periods? Use τ and $R_{b,\max}$ from **E2** and the eye heights of **E1** (Worked Example 5.20), and name the scheme that gave the highest error-free bit rate on your link.

Summary

- Shannon's model: source, encoder, modulator, noisy channel, demodulator, decoder, destination. On the boards: DAC, LED pair 2, free space, photodiode, 3 MΩ transimpedance amplifier, ADC.
- Text becomes bytes through ASCII (7 bits) or UTF-8 (1 to 4 bytes per character); both ends must agree on bit order.
- $R_b = R_s \log_2 M$. OOK sends one bit per symbol; Manchester adds a mid-bit transition at the cost of twice the bandwidth; PAM-4 sends two bits per symbol at the cost of a 9.5 dB SNR penalty.
- Received power falls as $1/d^2$ and with misalignment (Eq. (5.6)); photocurrent is $I_{ph} = \mathcal{R} P_r$. A TIA gives $v_{\text{out}} = I_{ph} R_f$ with $f_{3\text{dB}} =$

$1/(2\pi R_f C_f)$ and $t_r = 2.2 R_f C_f$. The LED's light is not proportional to its drive voltage, so levels are chosen from a measured staircase.

- Thermal noise $\sqrt{4kTRB}$, shot noise $\sqrt{2qIB}$, flicker noise $\propto 1/f$, quantization noise $\Delta/\sqrt{12}$. Independent sources add as the root sum of squares; narrowing B reduces noise; a first-order filter has noise bandwidth $1/(4\tau)$.
- With a mid-level threshold and Gaussian noise, BER $= Q(A/2\sigma)$; M levels give $P_s = 2(1 - 1/M)Q(d/2\sigma)$. Eye diagrams show noise (eye height) and timing (eye width) at a glance.
- Shannon capacity $C = B \log_2(1 + \mathrm{SNR})$ caps the error-free bit rate.
- Framing (start and stop bits, preambles such as `0xAA 0x55`) and correlation let the receiver find bits and messages. Parity and checksums detect errors; repetition and Hamming codes correct single errors.
- Sensor fusion weights each source by its trust: inverse-variance averaging, complementary filters, and the Kalman filter's predict-and-update loop. Decision-level fusion (OR, majority vote) lets a link survive the loss of a channel.

5.11 *Problems*

Unless stated otherwise, use $k = 1.38 \times 10^{-23}\,\mathrm{J/K}$, $q = 1.602 \times 10^{-19}\,\mathrm{C}$, $T = 300\,\mathrm{K}$, and a 10-bit ADC over 3.3 V (3.22 mV per count).

5.1 ◦ Write the ASCII codes of `U`, `S`, and `C` in hexadecimal and 8-bit binary. In what order does the Task 1 firmware send the bits of `U`? How long does Task 1 take to send the three characters of `"USC"` once, and what is its useful bit rate?

5.2 ◦ Using Table 5.3, how many bytes does UTF-8 need for (a) *naïve* (*ï* is `U+00EF`), (b) `€5` (€ is `U+20AC`), (c) the two-character Chinese word for Japan, code points `U+65E5` and `U+672C`, and (d) `EE105`? (e) Encode € by hand and give its bytes in hexadecimal.

5.3 ◦ The lab sends its 392-bit frame (360 data bits) with OOK and with PAM-4, at 2 ms and at 1 ms per symbol. For each of the four cases find the baud, the bit rate, the time to send the frame, and the useful bit rate.

5.4 ◦ A link uses eight brightness levels (PAM-8) and a symbol period of 2 ms. (a) What are its baud and bit rate? (b) How long does it take to send `"Hello, world!"` (13 ASCII characters), with no framing, if the last symbol is padded with zeros?

5.5 ◦ The letter `K` is `0x4B`. Sketch its OOK and Manchester waveforms (MSB first; Manchester 1 = low then high). Count the level transitions in each waveform within the byte. Why does the Manchester version make the receiver's job easier, and what does it cost?

5.6 ◦ A hospital wants wireless data in its patient rooms. (a) Give two reasons LiFi might suit it better than WiFi, and two drawbacks. (b) Why must a LiFi lamp switch much faster than about 200 Hz? (c) Why does the demo board's photodiode include a daylight-blocking filter?

5.7 ◦ (a) A 12-bit ADC spans 0 to 3.3 V. Find its LSB, its RMS quantization noise, and the ideal SNR for a full-scale sine wave. (b) How many bits does a 3.3 V converter need for an LSB of at most 1 mV? (c) Express in decibels a power ratio of 2, a power ratio of 1000, and a voltage ratio of 10. Express an SNR of 25 dB as a power ratio and as an RMS voltage ratio. (d) The received optical power drops by a factor of 4 and the noise stays the same. By how many decibels does the electrical SNR fall?

5.8 • A ceiling LED lamp 2 m above a desk emits 1 W of optical power with a Lambertian pattern ($m = 1$). A receiver with a 1 cm^2 photodiode of responsivity 0.4 A/W faces straight up. Find the received power and photocurrent (a) directly below the lamp and (b) on the desk 1.5 m to the side. By how many decibels does the optical power drop between the two spots?

5.9 • An LED with a half-power angle of 20° emits 1 mW toward a photodiode ($A = 0.23\,\text{mm}^2$, $\mathcal{R} = 0.55\,\text{A/W}$) 4 cm away on its axis. The photodiode feeds a TIA with $R_f = 3\,\text{M}\Omega$ and the ADC. (a) Find the Lambertian order m, the received power, the photocurrent, the TIA output, and the ADC reading. (b) Up to what distance does the reading stay above 200 counts? (c) At what distance does the TIA output reach 3.3 V and saturate? (d) At 4 cm, both boards are tilted so that $\phi = \psi = 10°$. What is the reading now, and why does a narrow beam make the link sensitive to tilt?

5.10 • The TIA of Figure 5.29 has $R_f = 200\,\text{k}\Omega$ and $C_f = 5\,\text{pF}$. The photocurrent switches between 0 and 4 µA. (a) What output voltage swing (magnitude) corresponds to 4 µA? (b) Find $f_{3\text{dB}}$, the rise time, and the approximate maximum OOK bit rate. (c) What is the largest C_f that gives a rise time below 0.5 µs?

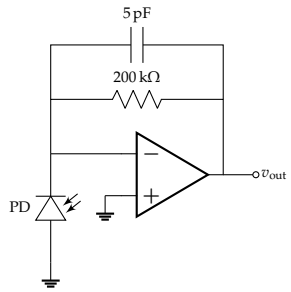


Figure 5.29: Transimpedance amplifier for Problem 5.10.

5.11 •• In step E2 of Lab 5.1, the falling edge of the received light passes 90% of the step at 8.21 ms and 10% at 8.97 ms. The re-

ceiver's TIA has $R_f = 3\,\text{M}\Omega$. (a) Find the fall time, τ, $f_{3\text{dB}}$, the feedback capacitance, and $R_{b,\max} \approx 1/(2t_f)$. (b) For OOK with levels 0 and A through this first-order receiver, a 1 that follows a long run of 0s has reached $A(1 - e^{-t/\tau})$ at a time t after the start of its bit, and a 0 that follows a long run of 1s has fallen to $Ae^{-t/\tau}$. Show that the noise-free eye height at time t is $A(1 - 2e^{-t/\tau})$, and evaluate it as a fraction of A at mid-bit and at the end of the bit for bit periods of 4, 2, 1, and 0.5 ms. (c) Below what bit period does the eye close at mid-bit, and below what period does it close even at the end of the bit?

5.12 • A photodiode drives a 100 kΩ load resistor (Figure 5.30) with 5 µA of photocurrent over a bandwidth of 50 kHz. (a) Find the signal voltage, the thermal and shot noise voltages, the total noise, and the SNR in dB. (b) At what photocurrent would the shot noise equal the thermal noise?

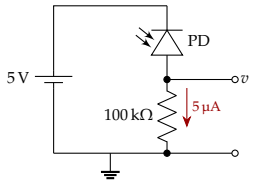


Figure 5.30: Circuit for Problem 5.12.

5.13 •• A receiver uses a TIA with $R_f = 1\,\text{M}\Omega$ and $C_f = 47\,\text{pF}$ and a 12-bit ADC over 3.3 V. The LED adds a signal photocurrent of 0.5 µA on top of 2 µA of ambient photocurrent (the output, 2.5 V, stays below 3.3 V). (a) Find τ, the noise bandwidth, the thermal noise of R_f, the shot noise of the total photocurrent, and the quantization noise, all as voltages at the ADC. Which dominates? Find the total and the SNR of the 0.5 V signal in dB. (b) An optical filter removes the ambient light. What are the total noise and the SNR now? (c) Repeat (a) and (b) with a 16-bit ADC. When does shading the receiver pay off?

5.14 • The light receiver of the fusion demo shown in class is the divider of Figure 5.31: a light-dependent resistor (LDR) from 5 V to pin A0 and 10 kΩ from A0 to ground, read by a 10-bit ADC with a 5 V reference (4.88 mV per count). The LDR measures 40 kΩ in room light and 4 kΩ when the transmitter LED shines on it. (a) Find the A0 voltage and ADC reading in each case. (b) The firmware sets its threshold to the ambient reading plus 50 counts. What threshold does it choose? (c) After calibration the room gets brighter. Below what LDR resistance does the receiver see a false 1 with the LED off?

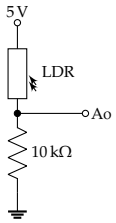


Figure 5.31: LDR divider for Problem 5.14.

5.15 • An OOK receiver sees levels of 0 V and 0.4 V with Gaussian noise of 40 mV RMS. (a) With the threshold at 0.2 V, what is the BER, and how many errors do you expect in a 1 MB file (8×10^6 bits)? (b) Ambient light was miscalibrated and the threshold sits at 0.1 V. What is the BER now? (c) The same swing is used for PAM-4 with equally spaced levels and midpoint thresholds. Es-

timate the BER (assume Gray coding, so one symbol error costs one bit). Use $Q(5) = 2.9 \times 10^{-7}$, $Q(2.5) = 6.2 \times 10^{-3}$, $Q(7.5) \approx 0$, and $Q(1.667) = 0.048$.

5.16 • Two boards have moved apart, and the receiver now sees 6 counts for a 0 and 46 counts for a 1, with $\sigma = 2.5$ counts on both. (a) With the threshold at the midpoint, what is the BER? (b) The threshold is still fixed at 40 counts from an earlier, closer setup. What is the BER, and how many errors do you expect in the 360-bit lab message? (c) PAM-4 with the same 6 to 46 count swing, equally spaced levels, midpoint thresholds, and Gray coding: estimate the symbol error probability and the BER. Use $Q(8) = 6.2 \times 10^{-16}$, $Q(2.4) = 8.2 \times 10^{-3}$, $Q(2.67) = 3.8 \times 10^{-3}$, and $Q(13.6) \approx 0$.

5.17 ◦ Name the most likely cause of each eye diagram, and one change that would open it: (a) thick top and bottom bands but sharp, narrow crossings; (b) thin bands but crossings smeared from side to side; (c) an eye that opens only near the end of each symbol, with slow curved edges.

5.18 • (a) A channel has 2 MHz of bandwidth and an SNR of 25 dB. What is its Shannon capacity? (b) What SNR, in dB, would the same channel need to carry 30 Mbit/s?

5.19 •• *Demo board design.* The 12 ms recording of Figure 5.20 shows a lowest level of 41.5 counts, a highest level of 174.2 counts, and noise $\sigma = 1.9$ counts on every level. Suppose you keep the same lowest and highest levels but use M equally spaced levels with midpoint thresholds. Use Eq. (5.19) for the symbol error probability and, with Gray coding, BER $\approx P_s / \log_2 M$. (a) Compute the BER for $M = 2, 4, 8, 16$. (b) What is the largest M that keeps the BER below 10^{-6}, and what bit rate does it give at 12 ms per symbol? (c) Why might the real link fall short of this prediction?

5.20 •• Part C of the lab measures this staircase (dark level 3 counts, noise $\sigma = 2$ counts):

drive (V)	0	2.75	2.80	2.85	2.90	2.95	3.00	3.05	3.10	3.15	3.20	3.25	3.30
level (counts)	3	4	12	27	47	70	96	121	143	160	172	180	185

(a) What is the turn-on voltage, defined as the first drive voltage whose level is more than 5σ above the dark level? (b) Choose the OOK drive voltages and threshold as the notebook does (0 V and the brightest step). (c) Choose PAM-4 drive voltages whose received levels are evenly spaced from 3 to 185 counts, by linear

interpolation in the table, and give the three thresholds. (d) Find the smallest half-gap in units of σ. Compare with drive voltages 0, 2.90, 3.10, and 3.30 V and midpoint thresholds. (e) Explain from the shape of the staircase why the two middle drive voltages are not evenly spaced.

5.21 • An earlier version of the lab receiver printed one ADC sample every T_{sample} and assumed `N = int(T_send / T_sample)` samples per symbol, deciding on the samples with indices $N/2, N/2+N, \ldots$ (integer division, counting from the first sample of the frame, which arrives at time 0). The sender's symbol time was 12 ms, but the receiver sampled every 5 ms. (a) What is N, and what symbol duration does the receiver effectively assume? (b) Which symbol does each of the first ten decision samples land in? (c) Roughly how many symbols does the receiver decode from the 196-symbol PAM-4 message? (d) Which whole-millisecond receiver intervals would have worked?

5.22 • A receiver has one sample per bit, converted to ± 1, and looks for the 4-bit preamble `1100` $(+1, +1, -1, -1)$. It records -1, -1, $+1$, $+1$, -1, -1, $+1$, -1. (a) Compute $c[k] = \sum_{j=0}^{3} x[k+j]\,p[j]$ for $k = 0$ to 4 and find the start. (b) The fourth sample is received wrong (-1 instead of $+1$). Recompute $c[k]$. What goes wrong? (c) Give two reasons the lab uses a 16-bit preamble and about 16 samples per bit.

5.23 • *Python.* Write a function `decode(samples, thresholds, sps)` that follows the lab receiver in a simple form: map each sample to a level from 0 to 3 with `np.searchsorted(thresholds, samples)`; find the first and the last sample above the lowest threshold (the frame starts and ends with level 2, because `0xAA` begins and ends with the bits `10`); starting at the first, take the middle sample of each symbol (index `first + sps//2 + k*sps`); convert each level to two bits; strip the 16 marker bits at each end; and decode UTF-8. Test it on a synthetic recording: frame the text `"EE105"` with `0xAA 0x55` and `0x55 0xAA`, map the bit pairs `00` to `11` to levels of 4, 62, 120, and 178 counts, hold each symbol for 6 samples, add Gaussian noise with $\sigma = 2$ counts, and add 30 idle samples of 4 counts at each end. With thresholds halfway between the levels, how many symbols and samples does the recording hold, and what does your function return?

5.24 • A link sends 7-bit ASCII with an even-parity bit in the most significant position. The receiver gets the bytes `0xC5`, `0xC4`, and `0xD7`. Which pass the parity check, and which characters do they

decode to? The sender actually sent `0xC5` three times. What happened to the third byte, and what does this show about parity?

5.25 ° Add an even-parity bit in the most significant position to the 7-bit codes of `U`, `S`, and `C`, and give the three bytes in hexadecimal. The receiver then gets `0xD3` in the second position. Does it pass the check?

5.26 • A sender appends to `"EE105"` the sum of its bytes modulo 256. (a) Find the checksum in decimal and hexadecimal. (b) The receiver gets `"EF105"` with the original checksum. Is the error caught? (c) It gets `"EE015"`. Is that caught? (d) Show that any change to a single byte always changes this checksum. (e) Roughly what fraction of random corruptions of several bytes slip through?

5.27 • Bits flip independently with probability $p = 0.02$. (a) Find the bit error probability after majority decoding for a triple-repetition code and for a five-fold repetition code. (b) At 1000 bit/s on the line, how long does it take to send the 360 data bits uncoded, with each repetition code, and with Hamming (7,4)? (c) For Hamming (7,4), what is the probability that a 7-bit block has two or more errors, so that the decoder cannot fix it?

5.28 • The Hamming (7,4) codeword for the data $d_1d_2d_3d_4 = 0101$ is `0100101` (positions 1 to 7). Decode each received word, giving the syndrome, the corrected codeword, and the data: (a) `0100101`, (b) `0100111`, (c) `1100101`, (d) `0000100`. In (d), positions 2 and 7 both flipped. What did the decoder do?

5.29 •• Encode the 7-bit ASCII code of `a` (`0x61`) with the Hamming (11,7) code of Table 5.4 (d_1 = least significant bit), and add the overall parity bit p_0 of the extended (12,7) code. (a) List the codeword at positions 1 to 11 and give p_0. (b) Position 6 flips. Compute the syndrome and the overall parity check, and show how the receiver corrects the error. (c) Positions 3 and 6 both flip. What does the receiver conclude, and what would a plain (11,7) decoder have done?

5.30 • Three thermometers read a patient's temperature as 36.9 °C ($\sigma = 0.3$ °C), 37.4 °C ($\sigma = 0.6$ °C), and 36.5 °C ($\sigma = 1.0$ °C). (a) Find the inverse-variance weights, the fused estimate, and its standard deviation. (b) Compare with a plain average of the three readings and its standard deviation.

5.31 • A complementary filter runs every $\Delta t = 5\,\text{ms}$. (a) Choose α for a crossover time constant of 1 s. (b) Starting from $\hat{\theta}_0 = 0^\circ$, the gyroscope reads $10\,^\circ/\text{s}$ in the next two steps and the accelerometer reads 0.3° and then 0.2°. Find $\hat{\theta}_1$ and $\hat{\theta}_2$. (c) The gyroscope has a bias of $0.2\,^\circ/\text{s}$. What steady error does the filter settle at with the board still?

5.32 •• A robot moves along a line. Each step it commands a move of 1 m, with process noise $Q = 0.5\,\text{m}^2$. A range sensor measures position with variance $R = 2\,\text{m}^2$. Start from $\hat{x}_0 = 0\,\text{m}$, $P_0 = 1\,\text{m}^2$. Run two Kalman steps with measurements $z_1 = 1.4\,\text{m}$ and $z_2 = 1.7\,\text{m}$, giving $\hat{x}^-$, P^-, K, $\hat{x}$, and P at each step. Why does K change from step 1 to step 2?

5.33 • A receiver fuses a light channel that garbles 3% of characters and a sound channel that garbles 8%, independently. (a) With a parity check that flags every garbled character and OR-fusion, what fraction of characters is lost? (b) Three channels each garble 5% of characters; the receiver takes a majority vote and loses a character whenever two or more channels are wrong. What fraction is lost? (c) Show that majority voting over three channels helps only when each channel's error probability is below 0.5.

6
The Final Project: Building Your Own Intelligent Hardware

In this chapter you will learn to

- Recognize a good project: a real need, a measurable result, a feasible core, and a size that fits the time.
- Turn an idea into a one-sentence project statement and a short list of requirements with numbers.
- Estimate the signal, data, power, and accuracy budgets of a design before building it.
- Plan the work so that the riskiest question is answered first and a thin end-to-end system runs early.
- Test a prototype against its goal and report honest numbers, including where it stops working.

Each of the first six chapters has a lab that exercises one part of the demo board: a touch panel, a diode curve, a heartbeat, a digit classifier, a message carried on a beam of light. The final project asks you to do what the engineers behind a device like the brain implant of Chapter 0 had to do: start from a problem, choose the pieces, make them work together, and show with measurements that the result does what you said it would.

Nothing in this chapter is new physics. What is new is the work around the technology: choosing a problem worth solving, cutting it to a size that fits a few weeks, finding out early what will not work, and judging the result fairly. These habits matter more than any single circuit, and they are the same whether the project is a class demo or a product. The chapter follows the life of a project: finding a good one (Section 6.1), turning it into a specification (Section 6.2), carrying it out (Section 6.3), and testing it (Section 6.4).

6.1 *Finding a Good Project*

Start from a need, not from a part

The most common weak project begins with a component: "we have a camera, let us use it." The result is a demonstration of the component, not a solution to anything. A strong project begins with a person and a situation: a runner who wants to know her heart rate without a chest strap, a lab partner who keeps forgetting whether the hot plate is still warm, a classroom where the lights stay on in empty rooms. Ask who has the problem, when it happens, how often, and what they do about it today. If you cannot name the person and the moment, the need is probably not real yet.

The sense, process, communicate test

Chapter 0 described every electronic system as a chain: sense the physical world, process what was sensed, and communicate or act on the result. Write your idea as that chain, one line per link. Each link should name a concrete part and a chapter of this book: *sense* the pulse with the optical sensor (Chapter 3), *process* it into a heart rate with a filter and a peak detector (Chapter 3), *communicate* it over the light link (Chapter 5). If a link is blank, you have found the part of the idea that needs more thought. If every link is "the laptop does it," the project probably does not need hardware.

Five properties of a good project

Table 6.1 lists the properties that separate strong projects from weak ones. They pull against each other. A project that is easy to finish is often not interesting, and a project that answers a real need is often too big. The art is finding the version of the idea that has all five.

Property	Question to ask
Need	Who has this problem, and in what situation? Would they use the result?
Measurable	What single number will show that it works, with a target and units?
Feasible	Do the parts and the skills exist? What is the one thing most likely to fail?
Sized	Can the thinnest complete version run within the first third of the time?
Integrative	Does it join at least two blocks so that the joint itself takes engineering?

Table 6.1: Five properties of a good project. Score each candidate idea against them before choosing.

A sixth property is not optional: the project must be *safe and honest*. Work only with the low voltages of the board and USB power, test

on yourself and on teammates who agree to it, and do not present a measurement as a medical result. A heart-rate reading from your prototype is an engineering measurement to compare with a reference, not a diagnosis.

Where ideas come from

Ideas rarely arrive by staring at a blank page. A few methods work reliably. *Observe a routine*: list everything you measure, check, or remember during one day, and ask which of those a small device could do better. *Extend a lab*: each lab measured something in a controlled setting; ask what it would take to measure the same thing in daily life, and what would break. *Combine two blocks*: pick two modules and ask what one could do for the other (the classifier deciding what the light link should send, the thermistor deciding when the camera should look). The module selector powers one block at a time, so a project that needs two blocks at once uses two boards, one for each block, or the sensors built into the Nano 33 BLE Sense itself (motion, sound, color and light, temperature, humidity, and pressure), which work at any selector position. *Take a product apart on paper*, as Chapter 0 did with the brain implant: name its blocks, then build the smallest version of one interesting block. *Start from a constraint*: "no wires," "runs a day on a coin cell," "works in bright sunlight"; constraints often lead to the most interesting engineering.

WORKED EXAMPLE 6.1 *Choosing between three ideas with a decision matrix*

A team has three candidates: (A) a pulse monitor that sends heart rate over the light link, (B) a board that switches modes with hand gestures, and (C) a breathing monitor from the slow drift of the PPG. They weight the five properties of Table 6.1 as need 0.20, measurable 0.25, feasible 0.25, sized 0.15, integrative 0.15 (the weights sum to 1) and score each idea from 1 to 5: A (4, 5, 4, 4, 4), B (2, 3, 5, 5, 3), C (4, 4, 3, 3, 3). Which idea should they pick, and what does the matrix not tell them?

SOLUTION. The weighted score is the sum of weight times score:

$$S_A = 0.20(4) + 0.25(5) + 0.25(4) + 0.15(4) + 0.15(4) = 4.25,$$
$$S_B = 0.20(2) + 0.25(3) + 0.25(5) + 0.15(5) + 0.15(3) = 3.60,$$
$$S_C = 0.20(4) + 0.25(4) + 0.25(3) + 0.15(3) + 0.15(3) = 3.45.$$

Idea A wins clearly. Idea B is the easiest to build but scores low on need: few people want to switch a demo board with gestures. Idea C has a real need but its core (separating breathing from heartbeat

and motion in the PPG) is the riskiest. The matrix makes the team's judgment explicit, but the scores are still opinions. Its real value is the discussion it forces: if two teammates gave C a feasibility of 2 and 5, they should find out why before choosing. ■

The project statement

Compress the chosen idea into one sentence with four parts:

> *For* [who] *who* [need], we will build [system] *that* [function], and we will know it works *when* [measurable result] *under* [conditions].

Compare two versions of the same idea. Weak: "A smart heart-rate monitor using the board and machine learning." Strong: "For a student exercising in a dorm room who wants a heart rate without a chest strap, we will build a finger sensor that sends the rate to a laptop over light, and we will know it works when the rate is within ± 5 beats per minute of a commercial pulse oximeter for 8 of 10 one-minute recordings taken at rest and after 30 squats." The strong version tells the team what to build, what to measure, and when to stop.

6.2 *From Idea to Specification*

Requirements with numbers

A requirement is a statement you can test. Split them into three kinds. *Functional* requirements say what the system does ("reports heart rate once per second"). *Performance* requirements say how well, with numbers ("within ± 5 bpm," "bit error rate below 10^{-3} at 5 cm"). *Constraints* limit the design ("runs from USB power," "uses only parts in the kit," "finished in five weeks"). Five to ten requirements are enough. If a requirement has no number, ask what number would make it testable.

Budgets: check the design on paper first

Before building, add up the four quantities that most often sink a design. A *signal budget* follows the size of the signal from the sensor through amplifiers to the ADC: is it larger than the noise and the ADC step (Chapter 3)? A *data budget* compares the bits per second a block produces with what the next link can carry (Chapter 5). A *power budget* compares the average current with the battery or supply. An *accuracy budget* adds up the error of each part, from the sensor tolerance to the ADC step, and compares the total with the target. Delays can be budgeted the same way, from event to output. Each budget takes ten minutes with a calculator and can save weeks.

Worked Example 6.2 *Data and power budgets for a fall detector*

A team plans a fall detector: the Nano's accelerometer samples three axes at 100 Hz with 16 bits per axis, and the board should report to a laptop over the light link of Chapter 5, which in the lab runs at up to about 2 kbit/s. The board will run from a 500 mAh battery and draws about 20 mA. Can it stream the raw data? How long will the battery last?

Solution. The raw data rate is

$$R = 3 \text{ axes} \times 16 \,\tfrac{\text{bits}}{\text{sample}} \times 100 \,\tfrac{\text{samples}}{\text{s}} = 4800 \text{ bit/s} = 4.8 \text{ kbit/s},$$

more than twice what the link carries, before adding any framing. So the detection must run on the Arduino, and only events (for example a 20-byte "fall at time t" message) cross the link. The battery life is $t = 500\,\text{mAh}/20\,\text{mA} = 25\,\text{h}$, about one day, which is enough for a demonstration but not for a product. Both conclusions change the design before any code is written: on-board processing becomes a requirement, and the power budget says where to look if longer life is ever needed. ■

Worked Example 6.3 *Accuracy budget for a thermometer*

A thermometer uses the divider of Lab 3.1: a 10 kΩ NTC thermistor ($\beta = 3950$ K) on top and a 10 kΩ resistor on the bottom, from 3.3 V, read by the 10-bit ADC. The requirement is $\pm 0.5\,^{\circ}$C near 25 °C. The thermistor and the resistor each have a ±1% tolerance. Will the uncalibrated design meet the requirement?

Solution. Near 25 °C ($T = 298.15$ K) the thermistor changes by $dR_T/dT = -\beta R_T/T^2 = -3950 \times 10^4/298.15^2 \approx -444\,\Omega/\text{K}$. With $V = 3.3\,R/(R + R_T)$ the output changes by

$$\frac{dV}{dT} = \frac{3.3\,R}{(R+R_T)^2}\left(-\frac{dR_T}{dT}\right) = \frac{3.3 \times 10^4 \times 444}{(2\times 10^4)^2} \approx 36.7\,\text{mV/K}.$$

One ADC step is $3.3/1024 = 3.22$ mV, or 0.088 K. Reading each code as the middle of its step (Chapter 3), rounding adds at most 0.044 K. A 1% (100 Ω) error in either part moves the output by $3.3 \times 10^4 \times 100/(2\times 10^4)^2 = 8.25$ mV, or 0.225 K. In the worst case the errors add: $0.044 + 0.225 + 0.225 = 0.494$ K, just inside the requirement (with the plain code-times-step conversion the ADC alone can add 0.088 K, and the worst case, 0.538 K, would fail); if they are independent they combine as a root sum of squares to about 0.32 K. The design meets the requirement, but with no margin in the worst case, so the plan should include a one-point calibration against a reference thermometer. ■

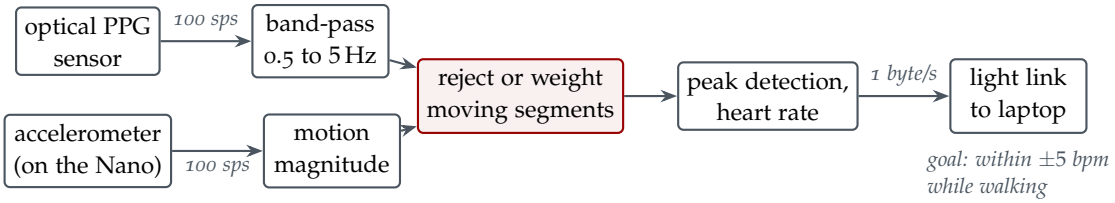


Figure 6.1: A block diagram for a motion-tolerant heart-rate monitor, with the interface rates written on the arrows and the measurable goal next to the output.

The block diagram and its interfaces

Draw the system as blocks before building anything (Figure 6.1). Each block becomes one person's job, and each arrow is an *interface* the team agrees on in writing: a voltage range and impedance for analog signals, a sample rate and format for data, a message format and rate for communication. Most integration failures are interface failures: one person sends volts, the other expects ADC counts; one sends 100 samples per second, the other assumes 50.

6.3 Carrying Out the Project

Answer the riskiest question first

Every project rests on a few assumptions. List them and mark the one that, if false, kills the project: "the PPG signal survives walking," "the classifier works under room lighting," "the link reaches 5 cm." Test that assumption in the first week with the crudest possible experiment, a few lines of code and a recorded data file. If it fails, you have lost a week, not the project, and you can change the idea while there is still time.

Build the thinnest complete path, then improve it

The second rule is to connect everything early. Make every block exist in its simplest form (a fixed threshold instead of a classifier, a print statement instead of a radio) and pass data from the first block to the last. Then improve the block that limits the result most. A team that perfects its sensor circuit for four weeks and connects it in the fifth discovers in the fifth week that the processing needs a different sample rate. Figure 6.2 contrasts the two approaches.

Milestones

A project of five or six weeks works well with four milestones, each ending in something you can show: (1) the riskiest assumption tested with recorded data; (2) the thin path running end to end, however crudely; (3) the first full measurement against the goal; (4) the design

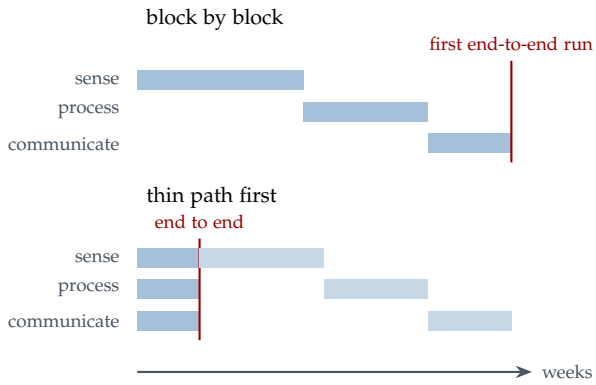


Figure 6.2: Building block by block delays the first complete run to the end. Building a thin complete path first exposes integration problems in the first weeks, while there is still time to fix them.

frozen, with the remaining time spent on testing, finding limits, and preparing the demonstration. Put the dates in the plan and review them every week. When a milestone slips, cut scope, not testing.

Debugging a system

When a system does not work, the fastest path is to divide it. Measure the signal at each interface on your block diagram, starting from the sensor, and find the first place where it is wrong. Measure rather than guess: print the raw ADC values, plot the waveform, check the supply voltage. Change one thing at a time and write down what you changed. Two tools from the labs help separate hardware faults from software faults: the simulated board (`SIMULATE = True`) runs your processing code without the hardware, and a recorded data file lets you rerun the processing on the same input until it is right. Before suspecting anything subtle, check the module selector, the jumpers, the USB cable, and the ground connection.

Keeping a record

Keep a shared log: every measurement with its date, settings, board, and result, and every design decision with the reason. Put the code under version control (a shared Git repository is enough) and save recorded data with a short note of how it was taken. The log turns into the results of the project, and it rescues the team when a change breaks something that used to work.

Working as a team

Divide the work by blocks of the diagram, so each person owns a block and its interfaces, and give one person the job of keeping the whole system running end to end. Meet briefly and often: each person says what works, what is next, and what is blocking them, and the team

writes down decisions. Agree on what "done" means for each block (it meets its interface and passes its test), so that nobody spends a week polishing a block that was already good enough.

6.4 Testing and Judging the Result

A test plan says what you will measure, against what reference, how many times, and under what conditions. For a heart-rate monitor the reference is a commercial pulse oximeter, the conditions are rest and after exercise, and ten recordings of a minute each give a usable picture. For a classifier the test set must be recorded separately from the training set (Chapter 4). For a link, the bit error rate needs enough bits to see errors at the rate you claim (Chapter 5).

> **Worked Example 6.4** *Did the heart-rate monitor meet its goal?*
>
> Eight one-minute recordings give these differences between the prototype and the reference, in beats per minute: $+2$, -3, $+1$, $+4$, -2, $+6$, 0, -1. The goal was "within ± 5 bpm." Summarize the result.
>
> Solution. The mean difference (the bias) is $(2 - 3 + 1 + 4 - 2 + 6 + 0 - 1)/8 = 7/8 = 0.875$ bpm, small compared with the goal. The sample standard deviation is 3.04 bpm, and the largest error is 6 bpm. Seven of the eight recordings are within ± 5 bpm, so the prototype meets the goal in 87.5% of trials. An honest report says exactly that, shows the one failure, and explains it if possible (for example, the recording with $+6$ was taken right after exercise, while the hand was still moving). ■

Find the limits

Every design fails somewhere: too much ambient light, too much motion, too far apart, too cold. Strong projects find where and show it, by sweeping one condition until the result breaks. "The link works up to 6 cm under room lighting and 3 cm near a window" is a stronger result than "the link works."

Communicating the result

Whatever form the final presentation takes, lead with the problem in one sentence, show the system working early, and give the measured result next to the goal you set. Show the block diagram rather than every schematic. End with the limits you found and what you would change. A modest system that works, with honest numbers, is worth more than an ambitious one that never ran.

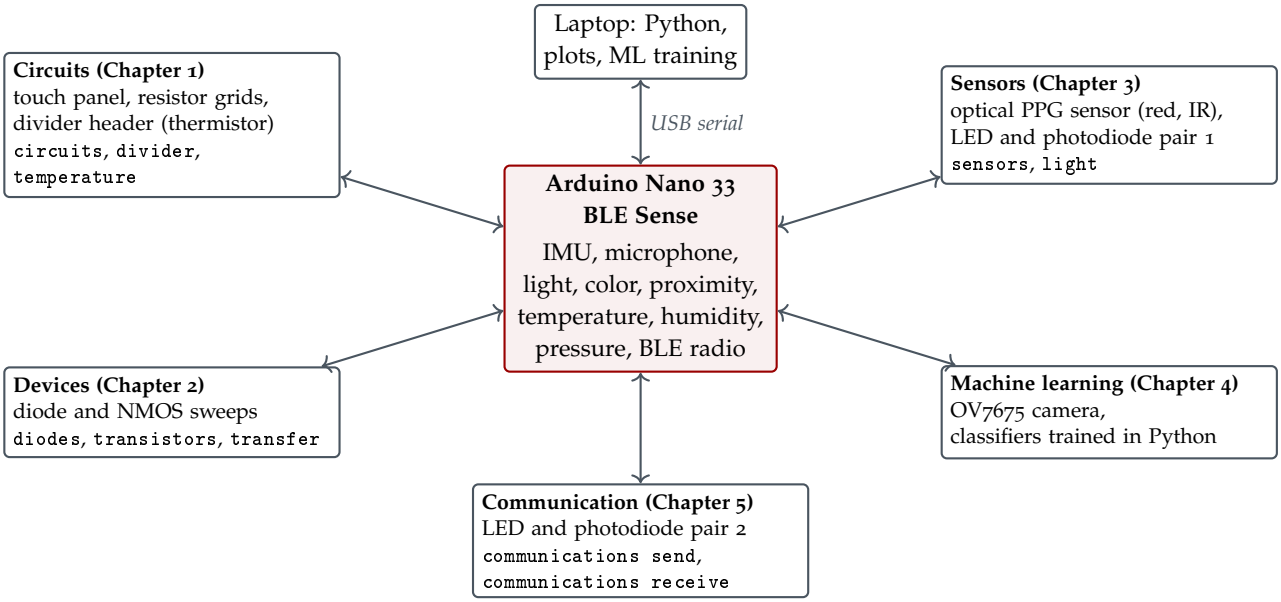


Figure 6.3: Building blocks for the final project: the demo board's blocks with the firmware commands used in the labs, the sensors and radio of the Nano 33 BLE Sense, and the laptop.

6.5 *Building Blocks and Ideas*

Figure 6.3 collects the hardware you can draw on: the blocks of the demo board with the firmware commands you used in the labs, the sensors and radio built into the Arduino Nano 33 BLE Sense, and the laptop running Python.

The ideas below are seeds, not assignments. Each names the chain it would build and a number that would show it works; the best projects usually come from changing one of them to fit a need the team knows well.

- **Pulse monitor over light.** PPG, filtering and peak detection on the Arduino, heart rate sent over the light link. *Measure:* error against a commercial pulse oximeter.
- **Motion-tolerant heart rate.** Use the accelerometer to reject or correct PPG segments during motion. *Measure:* error while walking in place, with and without the correction.
- **Smart thermometer with alarm.** Thermistor divider, calibration, and an alarm, on the Nano's own LED or sent over Bluetooth, when the temperature leaves a range. *Measure:* calibration error over a range and the alarm delay.
- **Hand signs over light.** Classify a few hand signs with the camera on one board and send each as a character over the light link from a second board to a third. *Measure:* accuracy and end-to-end character

error rate.

- **Reliable light link.** Add framing, a checksum, and an automatic threshold to the LiFi code; raise the bit rate until errors appear. *Measure:* bit error rate against distance and bit rate.
- **Automatic curve tracer.** Sweep the diode and transistor from Python and fit their model parameters. *Measure:* fitted parameters with uncertainties and fit residuals.
- **Colorimeter.** White light through dilutions of food dye, measured with the red, green, and blue channels of the Nano's color sensor (the board's photodiodes see only near-infrared light, which food dyes barely absorb), and the Beer-Lambert law to estimate an unknown concentration. *Measure:* calibration curve and the error on an unknown.
- **Breathing monitor.** Recover the breathing rhythm from the slow part of the PPG or from the accelerometer on the chest. *Measure:* error in breaths per minute against counted breaths.

The Arduino documentation keeps a list of community projects built on the Nano 33 BLE Sense, many of which combine its sensors with machine learning or wireless links: `docs.arduino.cc`. Reuse the lab code freely, and say which parts you reused.

Summary

- A good project starts from a need, has one measurable result, has a feasible core whose biggest risk is known, fits the time, and joins at least two blocks.
- Write the idea as one sentence: for whom, what need, what system, how you will know it works, under what conditions.
- Check signal, data, power, and accuracy budgets on paper before building.
- Test the riskiest assumption first, run a thin end-to-end path early, and improve the block that limits the result.
- Test against a reference, report the numbers including failures, and find the conditions where the design stops working.

6.6 Problems

6.1 ° Rewrite this project statement so it has all four parts of the template in Section 6.1: "An AI-powered smart glove for health."

6.2 ° List the sense, process, and communicate links for a device that warns a student when the air in a room has been still and warm for too long. Name the chapter of this book that covers each link.

6.3 • A decision matrix uses the weights of the worked example in Section 6.1. Idea D scores (5, 2, 4, 3, 4) on need, measurable, feasible, sized, and integrative. Compute its weighted score and compare it with idea A. What single change to idea D would raise its score most?

6.4 • A microphone is sampled at 8 kHz with 10 bits per sample. Can the raw samples be sent over a 2 kbit/s light link? If the board instead sends one loudness value (one byte) every 100 ms, what fraction of the link does it use?

6.5 • A prototype draws 18 mA on average from a 220 mAh battery. How long does it run? By what factor must the average current fall to reach 48 h?

6.6 • In the thermometer budget of Section 6.2, the parts are replaced by 5% tolerance parts. What are the worst-case and root-sum-square errors near 25 °C? Does the design still meet ± 0.5 °C without calibration?

6.7 • Ten recordings of a breathing monitor give errors of -1, 0, $+2$, $+1$, -3, 0, $+1$, $+4$, -1, 0 breaths per minute against counted breaths. Find the mean error, the sample standard deviation, and the fraction within ± 2 breaths per minute.

6.8 •• Your team's riskiest assumption is that the light link works under room lighting at 5 cm. Design a one-day experiment that tests it: what you will record, how many bits, what result would make you change the plan.

A
Answers to Chapter Problems

Final answers to every end-of-chapter problem are collected here. Check your result only after you have a complete solution of your own; when your answer differs, the worked examples in the chapter are the best place to find the step that went wrong. Numerical answers are rounded to three significant figures unless the problem asks otherwise.

Chapter 0: Electrical Engineering and Electronic Systems

0.1 (a) $1024/64 = 16$ electrodes per thread. (b) $5\,\mu\text{m} = 5000\,\text{nm} = 5 \times 10^{-6}\,\text{m}$; $8\,\text{mm} = 8000\,\mu\text{m}$. (c) $1024 \times 19\,300 = 1.98 \times 10^{7}$ samples per second.

0.2 (a) Electrodes: sense, Chapter 3. (b) Amplifiers: sense/process (analog front end), Chapter 1 (op-amps in Chapters 2 and 3). (c) ADC: process (the bridge to numbers), Chapters 0 and 3, built from the transistors of Chapter 2. (d) Spike detector: process, Chapter 4. (e) Radio: communicate, Chapter 5. (f) Decoder: process, Chapter 4. (g) Battery and coil: power, Chapters 0 and 1. In the kit: (a) the photodiodes, pulse sensor, and thermistor; (e) the LED-to-photodiode light link (pair 2); (f) the neural network that classifies camera images.

0.3 (a) $1.976 \times 10^{8}\ \text{bit/s} \times 86\,400\,\text{s}/8 = 2.13 \times 10^{12}$ bytes, about 2.13 TB per day. (b) $197.6 \times 10^{6}/1 \times 10^{6} = 197.6\,\text{s}$, about 3.3 minutes for one second of data. (c) $256 \times 10 \times 19\,300 = 49.4\,\text{Mbit/s}$; no, about 49 times more than the link carries.

0.4 (a) $1024 \times 15 \times 24 = 368\,640\ \text{bit/s} \approx 0.369\,\text{Mbit/s}$, 36.9% of the link. (b) $10^{6}/(1024 \times 24) = 40.7$ spikes per second per channel. (c) 1024×4 bits $\times$ 40 per second $= 163\,840\ \text{bit/s} \approx 0.164\,\text{Mbit/s}$. (d) It loses the exact time of each spike within its 25 ms interval; a 4-bit count reaches at most 15 per interval, or $15/0.025\,\text{s} = 600$ spikes per second.

0.5 (a) $4 \times 6 + 10 = 34\,\text{mW}$; $I = 34\,\text{mW}/3.7\,\text{V} = 9.19\,\text{mA}$.

(b) $0.2\,\text{Ah} \times 3.7\,\text{V} = 0.74\,\text{Wh} = 2664\,\text{J}$; $0.74/0.034 = 21.8\,\text{h}$. (c) $197.6 \times 10^6 \times 10 \times 10^{-9}\,\text{J} = 1.98\,\text{W}$, which together with the rest of the implant (2.01 W in all) would empty the battery in $0.74/2.01 \approx 0.37\,\text{h}$, about 22 minutes (and heat the tissue); spike messages need only $0.33 \times 10^6 \times 10 \times 10^{-9}\,\text{J} = 3.3\,\text{mW}$.

0.6 (a) $4096 \times 10 \times 19\,300 = 790.5\,\text{Mbit/s}$; $4096/256 = 16$ chips using $16 \times 6 = 96\,\text{mW}$. (b) 12 bits ($2^{12} = 4096$). (c) $4096 \times 20 \times 32 = 2.62\,\text{Mbit/s}$, which does not fit; the limit is $10^6/(4096 \times 32) = 7.63$ spikes per second per channel. (d) For example: send spike counts per time interval instead of time-stamped events (4-bit counts every 25 ms need $4096 \times 4 \times 40 = 0.655\,\text{Mbit/s}$; loses exact timing); send only the channels the decoder needs (loses data); or use a faster radio (costs power). Shorter messages with coarser time stamps cannot fit by themselves: the 12-bit channel number alone already needs $4096 \times 20 \times 12 = 0.98\,\text{Mbit/s}$.

0.7 (a) Sensing (turns reflected light into a number); (b) processing; (c) communication; (d) both power (5 V supply) and communication (serial data); (e) sensing (turns light into image data); (f) communication (it modulates the light-link transmitter LED); (g) processing (turns raw samples into information).

0.8 (a) $2300 \times 2^{26} = 1.54 \times 10^{11}$, about 154 billion, within 3% of 150 billion. (b) $\log_2(10^{12}/1.5 \times 10^{11}) = 2.74$ doublings, or 5.5 years: about 2028 or 2029. (c) $1000 = 2^{9.97}$, so the doubling time is $20/9.97 = 2.0$ years.

0.9 (a) $(66/0.75)^{1/22} = 88^{1/22} = 1.226$, about 22.6% per year; doubling time $\ln 2/\ln 1.226 = 3.4$ years. (b) $66\,\text{MHz} \times 1.226^{19} \approx 3.15\,\text{GHz}$, just above the 1 to 3 GHz range. (c) $66\,\text{MHz} \times 1.226^{31} \approx 36.3\,\text{GHz}$; real chips stopped near a few GHz because power dissipation (heat) grows with switching speed, so designers added cores instead.

0.10 (a) 4.7 kΩ; (b) 0.0022 µF; (c) 15 000 µA; (d) 0.005 µm; (e) 230 transistors per μm^2 (since $1\,\text{mm}^2 = 10^6\,\mu\text{m}^2$).

0.11 (a) 15.6 ns; (b) 62.5 ns; (c) $64\,\text{MHz}/16\,\text{kHz} = 4000$ clock cycles per audio sample.

0.12 (a) $2\,\text{A} \times 3600\,\text{s} = 7200\,\text{C}$. (b) $2\,\text{Ah} \times 3.7\,\text{V} = 7.4\,\text{Wh} = 26.6\,\text{kJ}$. (c) $2000\,\text{mAh}/10\,\text{h} = 200\,\text{mA}$ and $7.4\,\text{Wh}/10\,\text{h} = 0.74\,\text{W}$.

0.13 (a) $R = 2.0\,\text{V}/4.0\,\text{mA} = 500\,\Omega = 0.5\,\text{k}\Omega$. (b) $3.3\,\text{V}/500\,\Omega = 6.6\,\text{mA}$. (c) $P = VI = 3.3 \times 6.6\,\text{mA} = 21.8\,\text{mW}$. (d) $\text{V}^2/\Omega = \text{V}^2/(\text{V/A}) = \text{V A} = \text{W}$.

0.14 (a) Voltage is the difference in energy per charge between two points; "at 3.3 V" means 3.3 V above the agreed reference node, ground. (b) Opposite to the conventional current: electrons drift from the − terminal toward the + terminal. (c) L doubles and A grows by

$2^2 = 4$, so R changes by $2/4$: it halves. (d) The pin can deliver only about 15 mA; the voltage may be similar, but the motor needs more than 30 times that current (and power), so it needs a separate supply and a transistor or driver to switch it.

0.15 (a) $262144/2 = 131\,072$ samples. (b) $131\,072/16\,000 = 8.19$ s. (c) $262\,144/24.7 \times 10^6 = 10.6$ ms. (d) $4096/2 = 2048$ samples, lasting 0.128 s.

0.16 (a) $\Delta = 3.3/1024 = 3.22$ mV; (b) $1.20/3.3 \times 1023 = 372$; (c) $700 \times 3.3/1023 = 2.26$ V; (d) $3.3/4096 = 0.806$ mV.

0.17 Need $3.3/2^N \leq 1$ mV, so $2^N \geq 3300$ and $N \geq \log_2 3300 = 11.7$: 12 bits, giving $\Delta = 0.806$ mV. (11 bits would give 1.61 mV, too coarse.)

0.18 (a) $1.2\,\text{V}/4096/200 = 1.46\,\mu$V; $1024 \times 12 \times 19\,300 = 237.2$ Mbit/s (20% more than with 10 bits). (b) $1.2\,\text{V}/256/200 = 23.4\,\mu$V, 3.97 times the 5.9 µV noise, so small spikes would span only a few steps. (c) 10 bits: its step (5.86 µV) matches the noise, so 12 bits would mostly record noise at a higher data rate, and 8 bits would throw away real signal.

0.19 (a) $128 + 32 + 16 + 4 + 2 = 182 =$ `0xB6`. (b) `0x3FF` $= 1023$; `0x57` $= 87$. (c) $300 = 1\,0010\,1100_2 =$ `0x12C`. (d) 6 bits ($2^6 = 64$), 10 bits, 12 bits.

0.20 (a) More than 8 kHz. (b) More than 20 Hz; $25 \times 10 = 250$ samples. (c) $1/19\,300$ Hz $= 51.8\,\mu$s; frequencies below 9.65 kHz. (d) $16\,000 \times 2 = 32\,000$ bytes per second.

0.21 (a) $1.0/0.00165 = 606.06 \rightarrow 606$, giving 0.9999 V; $2.5/0.00165 = 1515.15 \rightarrow 1515$, giving 2.4998 V. (b) $1.65/3300 = 0.05\%$ of 3.3 V.

0.22 (a) $255 \times 1.0/3.3 = 77.3$, so 77, giving an average of 0.996 V. (b) $D = 191/255 = 74.9\%$, average 2.47 V. (c) The pulses repeat far faster than the eye can follow, so the eye averages them and sees a steady brightness set by the duty cycle.

0.23 (a) 690: 2.226 V, Module 1; 350: 1.129 V, Module 3; 240: 0.774 V, Camera; 60: 0.194 V, no position selected. (b) Module 3 and Camera: $1.1 - 0.83 = 0.27$ V, about 84 codes. (c) Real readings differ from the nominal levels by noise and resistor tolerance by a few codes; a threshold halfway between levels accepts any reading closer to one level than to its neighbor.

0.24 (a) Period 1000 ms, so 1 Hz, with duty cycle $250/1000 = 25\%$. (b) $t_{\text{half}} = 1/(2 \times 2.5) = 200$ ms each.

0.25 (a) $512 \times 255 = 130\,560$, and $130\,560/1023 = 127.6$, truncated to 127. (b) $127/255 \times 3.3\,\text{V} = 1.64$ V. (c) At most 20 lines per second (each pass waits 50 ms). (d) `3/1023` divides two integers, which in C gives 0, so the product is 0; write `3.3 / 1023.0` (or `3.3 / 1023`) so the division is done in floating point.

0.26 (a) $115200/10 = 11520$ bytes/s; (b) $11520/6 = 1920$ lines/s; (c) $3 \times 2 \times 100 = 600$ bytes/s, which fits in 9600 baud (960 bytes/s); (d) 1200 bytes/s does not fit in 960 bytes/s; 19200 baud (1920 bytes/s) is the lowest that works.

0.27 (a) `v = codes*3.3/1023` gives 0.0387, 1.652, 3.300, 2.474, 0.803, and 1.945 V; (b) `np.argmax(v)` $= 2$; (c) `v.mean()` $= 1.702$ V; (d) `(v > 2.0).sum()` $= 2$.

0.28 (a) Store `win = {'S1': (1000, 1020), ...}` and return the first key with `lo < code < hi`, else `None`. (b) `['S2', 'S6', 'S1', 'S8', 'S7', 'S9', None]`. (c) 1.613 V to 1.677 V: 37.1 mV below and 27.4 mV above 1.65 V. (d) S1 reads code 1023, above its window, so it is never reported. S9 reads code 0, and the open interval needs a code above 0, so it is reported only when noise lifts the reading to 1 or more. Fix: widen the end windows to include 1023 and 0 (for example S1 ≥ 1000, S9 ≤ 200), or match each reading to the nearest expected code instead of fixed windows.

0.29 (a) `[c*9/5 + 32 for c in temps_c]` gives `[65.3, 69.8, 77.9, 86.0, 60.8]`. (b) `[f for f in temps_f if f > 70]` gives `[77.9, 86.0]`. (c) `d = {day: c for day, c in zip(days, temps_c)}`; `sum(c > 20 for c in d.values())` gives 3.

0.30 (a) `levels.reshape(4, 3).mean(axis=1)` gives 43.7, 66.7, 47.7, 42.7 dB. (b) Index 1, the interval from 15 s to 30 s. (c) `levels[levels > 55]` gives `[60, 72, 68]`. (d) `[42, 60, 50, 43]` (every third reading) and `[46, 43, 41, 44]` (the last four). (e) `np.sort(levels)[::-1][:3]` gives `[72, 68, 60]`.

0.31 (a) $\begin{bmatrix} 20 & 1 \\ 45 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 412 \\ 537 \end{bmatrix}$, so $a = 5$ codes per °C and $b = 312$. (b) $T = (450 - 312)/5 = 27.6\,^{\circ}\text{C}$. (c) `(codes - 312) / 5` gives 23.6, 28.6, and 31.6 °C. (d) $1/5 = 0.2\,^{\circ}\text{C}$ per code.

0.32 (a) $x + y = 600$ and $0.10x + 0.40y = 0.20 \times 600 = 120$; `np.linalg.solve([[1, 1], [0.1, 0.4]], [600, 120])` gives $x = 400$ mL of 10% and $y = 200$ mL of 40%. (b) `A @ x` returns `[600, 120]`. (c) $x = -100$ mL, $y = 700$ mL: a negative volume, so no mixture of 10% and 40% can reach 45%. The solver returns numbers whether or not they make physical sense; you must check.

0.33 (a) 10×3 ms $= 30$ ms; (b) 30 characters take 900 ms, plus the 1000 ms pause, 1.9 s; (c) $8/0.030$ s $= 267$ bit/s; (d) $200 \times 3.3/1023 = 0.645$ V; (e) $8/0.010$ s $= 800$ bit/s.

0.34 (a) Sense: LEDs, photodiode, and read circuit; process: microcontroller with ADC; communicate: Bluetooth radio; power: battery. (b) $72/60 = 1.2$ Hz; $25/1.2 = 20.8$ samples per beat, 40 ms apart. (c) $25 \times 16 \times 2 = 800$ bit/s $= 100$ B/s, or 8.64 MB per day. (d) Streaming raw data lets the phone run heavier analysis and keep the full

waveform; computing on the patch sends far less data, which saves radio energy and battery life.

Chapter 1: Circuits

1.1 (a) $Q = 4\ \text{Ah} \times 3600 = 14\,400\,\text{C}$. (b) $E = QV = 53.3\,\text{kJ}$, which is 14.8 Wh. (c) $4000/200 = 20\,\text{h}$.

1.2 (a) $2.5 \times 10^{-6}/1.602 \times 10^{-19} = 1.56 \times 10^{13}$ electrons per second. (b) 1.5 mC. (c) 0.1 mC per pulse; average $(0.1\,\text{mC})(100)/1\,\text{s} = 10\,\text{mA}$.

1.3 $R = \rho L/(wt) = 0.194\,\Omega$; it drops 0.194 V at 1 A and dissipates 0.194 W. Twice as wide: $0.097\,\Omega$.

1.4 (a) KVL around the loop the two sources form would need $V_1 = V_2$, so different values contradict it. (b) KCL at the junction between them would need $I_1 = I_2$. (c) The short forces $V = 0$ across it, so $I = V/R = 0$. (d) The new branch adds conductance, and $G_{\text{par}} = G_{\text{old}} + G_{\text{new}}$ always grows, so R falls.

1.5 (a) Four nodes: the source's top, the node joining R_1, R_2, R_3, R_4, R_5, the node joining R_4, R_5, R_6, and ground. $R_2 \parallel R_3$ (that node to ground) and $R_4 \parallel R_5$; R_7 is shorted (both ends on the ground wire). (b) $R_{\text{eq}} = 1 + 0.5 \parallel 1.5 = 1.375\,\text{k}\Omega$, so $I = 4.36\,\text{mA}$.

1.6 With $V_a = 10\,\text{V}$: $V_b = 6\,\text{V}$ and $V_c = 3\,\text{V}$. (a) E_3: 6 V; E_4: 3 V; E_5: 3 V. (b) E_1: 5 mA out of its + terminal; E_4: 2 mA; E_5: 4 mA. (c) Absorbed, in mW: E_1 −50, E_2 12, E_3 6, E_4 6, E_5 12, E_6 14; the total is 0. Only E_1 supplies power.

1.7 (a) $I = (12 - 3)/6 = 1.5\,\text{A}$. Absorbed: 12 V source −18 W, 4 Ω 9 W, 3 V source 4.5 W, 2 Ω 4.5 W. (b) $I = 15/6 = 2.5\,\text{A}$; −30 W, 25 W, −7.5 W, 12.5 W: both sources deliver. (c) In (a), where current enters the cell's + terminal.

1.8 $R_{AB} = 30\,\Omega \parallel \big(10 + 20 \parallel (15 + 5)\big) = 30\,\Omega \parallel 20\,\Omega = 12\,\Omega$.

1.9 $V_{R1} = 12 - 6 = 6\,\text{V}$, so $R_1 = 6/3 = 2\,\text{k}\Omega$; $R_3 = 6/2 = 3\,\text{k}\Omega$; R_2 carries $3 - 2 = 1\,\text{mA}$, so $R_2 = 6\,\text{k}\Omega$. Check: KVL, $6 + 6 = 12\,\text{V}$; KCL at the top node, $3 = 1 + 2\,\text{mA}$.

1.10 (a) $R_2 = 27.5\,\text{k}\Omega$ (bottom), $R_1 = 72.5\,\text{k}\Omega$ (top). (b) $12 \times 27/102 = 3.18\,\text{V}$, count 985. (c) $12^2/102\,\text{k}\Omega = 1.41\,\text{mW}$.

1.11 $R_{\text{par}} = 54.5\,\Omega$, $V = 1.64\,\text{V}$; currents 16.4 mA, 8.18 mA, 5.45 mA.

1.12 The conductances add to $0.5 + 0.333 + 0.167 = 1\,\text{mS}$, so $V = 6\,\text{V}$; branch currents 3 mA, 2 mA, and 1 mA; 5 V across the 5 kΩ resistor; the source delivers 36 mW.

1.13 (a) $R_1 = 6\,\text{k}\Omega$, $R_2 = 4\,\text{k}\Omega$ (0.5 mA). (b) $R_2 \parallel 10\,\text{k}\Omega = 2.86\,\text{k}\Omega$, so $V_o = 1.61\,\text{V}$, 19% low. (c) Need $R_{\text{th}} \le 10/19 = 0.526\,\text{k}\Omega$; with $R_2/(R_1 + R_2) = 0.4$, $R_{\text{th}} = 0.4R_1$, so $R_1 \le 1.32\,\text{k}\Omega$ and $R_2 = \frac{2}{3}R_1 = 0.877\,\text{k}\Omega$. The divider then draws 2.28 mA, about 4.6 times as much.

1.14 (a) $R_4 = 1234 \times 100/1000 = 123.4\,\Omega$. (b) $R_2/R_1 \geq 47000/11110 = 4.23$; choose 10. (c) Neither: at balance no current flows through the meter, and both dividers scale with V_s in the same way.

1.15 (a) 0 V unstrained; -2.50 mV under load. (b) With $R_4 = R + \Delta R$, $V_x - V_y = V_s\left[\frac{1}{2} - \frac{R+\Delta R}{2R+\Delta R}\right] \approx -\frac{V_s}{4}\frac{\Delta R}{R}$. (c) The single divider sits near 2.5 V and the 2.5 mV change is less than one 3.2 mV ADC step; the bridge removes the large common voltage, so its output can be amplified before the ADC without saturating.

1.16 (a) Voltmeter probes on the two ends of R_2, nothing disconnected. (b) Open the circuit at one end of R_2 and insert the ammeter in series with R_2 only. (c) Power off and lift one end of R_2, then use the ohmmeter. (d) The ammeter's near-zero resistance shorts R_2 (and R_3), so the full 5 V appears across R_1 and $5\,\text{V}/R_1$ flows through the meter; if R_1 is small, that current blows the meter's fuse.

1.17 (a) 5.00 V. (b) 4.76 V, -4.8%. (c) $R_{\text{in}} \geq 49.5\,\text{M}\Omega$. (d) 3.33 V.

1.18 (a) True 100 mA, measured $5/51 = 98.0$ mA, -2.0%. (b) $5/1 = 5$ A and 25 W in the shunt; the meter's internal fuse opens.

1.19 (a) $0.1\,\text{V}/2\,\text{A} = 50\,\text{m}\Omega$. (b) $I^2R = 0.2$ W. (c) $5/2.55 = 1.961$ A instead of 2 A, an error of -1.96%.

1.20 Each strip draws 0.25 A. Six strips draw 1.5 A (safe); a seventh raises it to 1.75 A, above the rating, and the fuse opens.

1.21 In mS, mA, and V: $\mathbf{G} = \begin{bmatrix} \frac{1}{2}+\frac{1}{6}+\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4}+\frac{1}{12} \end{bmatrix}$, $\mathbf{i} = \begin{bmatrix} 6 \\ 1 \end{bmatrix}$; $V_1 = 9.26$ V, $V_2 = 9.94$ V; 0.171 mA flows in the 4 kΩ resistor from V_2 toward V_1.

1.22 (a) $\mathbf{G} = \begin{bmatrix} 2 & -0.5 & -0.25 \\ -0.5 & 2 & -0.5 \\ -0.25 & -0.5 & 1 \end{bmatrix}$ mS, $\mathbf{i} = [9,\ 0,\ 1]$ mA; $G_{13} = -\frac{1}{4}$ because the top 4 kΩ resistor joins nodes 1 and 3. (b) $V_1 = 5.5$ V, $V_2 = 2.25$ V, $V_3 = 3.5$ V. (c) In: $1.625 + 0.625 = 2.25$ mA; out through 1 kΩ: 2.25 mA.

1.23 (a) $V_x - V_y = 5(0.5 - 0.5238) = -119$ mV. (b) $R_{\text{th}} = 1 \parallel 1 + 1 \parallel 1.1 = 0.5 + 0.524 = 1.024\,\text{k}\Omega$. (c) $-0.119\,\text{V}/1.524\,\text{k}\Omega = -78.1\,\mu$A: 78.1 µA flows through the meter from V_y to V_x. (d) Nodal analysis gives $V_x = 2.539$ V and $V_y = 2.578$ V, the same current.

1.24 (a) 0.825 V. (b) 48.5 mm from the grounded edge. (c) $80/1023 \approx 0.078$ mm per count.

1.25 (a) 1.974 V: S2, 28 mV above its level. (b) 1.468 V: S8, but 114 mV above its level, far more than the few tens of millivolts a held button shows; suspicious (a floating input or a button being pressed or released). (c) 2.516 V: S4, 20 mV above. (d) 0.387 V: S9, but 387 mV above 0 V; suspicious, most likely a floating input.

1.26 (a) $3.3 \times 10/14.7 = 2.245$ V, count 696. (b) From 2.172 V (top high, bottom low) to 2.315 V. (c) 0.224 mA and 0.741 mW. (d) No: the

divider output and the ADC reference both scale with the rail, so the ratio, and the count, stay the same.

1.27 (a) 2.64 V, 1.98 V, 1.32 V, 0.66 V; counts 818, 614, 409, 205. (b) Worst case about ± 0.08 V (25 counts) at the middle taps; levels are 205 counts apart, so ± 100 counts is safe. (c) 0.66 mA and 2.18 mW. 10 kΩ cuts the standby power tenfold; very large values make the node sensitive to ADC input current, leakage, and noise.

1.28 (a) S1 to S9: 3.3, 2.3, 1.3, 2.4, 1.4, 0.8, 2.0, 1.2, 0 V. (b) 0.100 V (31 counts), shared by S8 and S3, S3 and S5, and S2 and S4. (c) S2 reads as S7, S3 as S8, S5 as S8, S7 as S2, and S8 as S3; S1, S4, S6, and S9 are still right. (d) It builds its thresholds from the levels measured on this board, so all nine buttons are still separated by at least 0.1 V.

1.29 (a) The smallest gap peaks at $R_a \approx 45\,\Omega$, about 0.265 V (82 counts). (b) At 50 Ω the gap is 0.169 V (52 counts), still workable. (c) Near $R_a = 59.5\,\Omega$, S2 and S7 have the same voltage (about 2.00 V), and by the 180° symmetry so do S3 and S8 (about 1.30 V), so those pairs cannot be told apart.

1.30 (a) $V_{\text{th}} = 6$ V, $R_{\text{th}} = 2 + 3 \parallel 6 = 4\,\text{k}\Omega$, $I_N = 1.5$ mA. (b) 3 V. (c) Setting $dP_L/dR_L = 0$ gives $R_L = R_{\text{th}}$; $P_{\max} = V_{\text{th}}^2/(4R_{\text{th}}) = 2.25$ mW.

1.31 (a) $4.00 = 4.80 \times 1000/(R_{\text{th}} + 1000)$ gives $R_{\text{th}} = 200\,\Omega$; $I_N = 24$ mA. (b) $4.80 \times 500/700 = 3.43$ V. (c) $R_L = 200\,\Omega$, $P = 4.80^2/800 = 28.8$ mW.

1.32 (a) 10 V in series with 2 kΩ. (b) 3 mA in parallel with 3 kΩ. (c) $10/3 = 3.33$ mA.

Chapter 2: Semiconductor Devices

2.1 p-type. $p \approx N_A = 5 \times 10^{16}\,\text{cm}^{-3}$ (majority holes); $n = n_i^2/N_A = 2.0 \times 10^3\,\text{cm}^{-3}$ (minority electrons).

2.2 n-type (compensated). $n \approx N_D - N_A = 3 \times 10^{16}\,\text{cm}^{-3}$; $p = n_i^2/n = 3.3 \times 10^3\,\text{cm}^{-3}$.

2.3 Heating silicon breaks more bonds, so n_i grows exponentially and far outweighs the small drop in mobility: resistance falls. Copper's electron density is fixed, and stronger lattice vibration lowers its mobility: resistance rises.

2.4 (a) $\rho = 0.520\,\Omega\,\text{cm}$. (b) $L = RA/\rho = 3.84\,\mu\text{m}$ (about 1.9 squares of 5.2 kΩ sheet resistance).

2.5 (a) $\lambda = 365$ nm, ultraviolet. (b) $E = 0.800$ eV. Silicon cannot detect it ($0.80 < 1.12$ eV); germanium can ($0.80 > 0.67$ eV, cutoff near 1850 nm).

2.6 (a) 1.88 eV (red) and 1.32 eV (infrared). (b) Yes: both exceed silicon's 1.12 eV gap (silicon absorbs out to about 1107 nm). (c) $1 \times$

$10^{-6}\,\mathrm{W}/(1.32\times1.602\times10^{-19}\,\mathrm{J}) = 4.73\times10^{12}$ photons per second.

2.7 (a) $V_{bi} = 0.835\,\mathrm{V}$. (b) $W = 0.331\,\mu\mathrm{m}$, 99 % of it on the lightly doped n side. (c) $C_j = \varepsilon_s/W = 3.14\times10^{-8}\,\mathrm{F/cm^2}$ ($31.4\,\mathrm{nF/cm^2}$).

2.8 (a) $V_{bi} = 0.775\,\mathrm{V}$. (b) Zero bias: $W = 1.00\,\mu\mathrm{m}$, $C = 104\,\mathrm{pF}$; at 3.3 V reverse: $W = 2.30\,\mu\mathrm{m}$, $C = 45.2\,\mathrm{pF}$. (c) C falls by a factor of 2.29, so the response is 2.29 times faster.

2.9 $V_D = 0.656\,\mathrm{V}$ at 1 mA and 0.716 V at 10 mA; difference 59.6 mV; $r_d = V_T/I_D = 2.59\,\Omega$ at 10 mA.

2.10 $I = (3.3-1.9)/220 = 6.36\,\mathrm{mA}$. For 10 mA: $R = 140\,\Omega$.

2.11 (a) $V_{out} = 0$ for $A = B = 0$ and 2.6 V for the other three cases: an OR gate. (b) A high, B low: diode A carries 0.26 mA, diode B is reverse biased and carries 0. Both high: 0.13 mA each. (c) Each stage loses 0.7 V and has no gain, so logic levels degrade until high and low can no longer be told apart.

2.12 (a) $3.3 = I(110\,\Omega) + V_T\ln(I/I_S+1)$ gives $I = 23.3\,\mathrm{mA}$. (b) Junction 0.738 V; terminals $0.738 + (0.0233)(10) = 0.970\,\mathrm{V}$. (c) Constant drop gives $(3.3-0.7)/100 = 26.0\,\mathrm{mA}$, 12 % high; it misses the 0.23 V drop across the series resistance.

2.13 (a) 11.3 V and 11.3 mA. (b) 12 V. (c) The diode conducts while $v_{in} > 0.7\,\mathrm{V}$: 48.1 % of each cycle.

2.14 (a) Cutoff, $I_D = 0$. (b) Triode ($0.4 < 1.2$), $I_D = 0.400\,\mathrm{mA}$. (c) Saturation ($3 \geq 1.2$), $I_D = 0.720\,\mathrm{mA}$.

2.15 (a) $I_D = 1.00\,\mathrm{mA}$, $V_{DS} = 3.0\,\mathrm{V} \geq 1\,\mathrm{V}$: saturated. (b) Saturation would give 4 mA and $V_{DS} = -3\,\mathrm{V}$, impossible. Triode: $V_{DS}^2 - 4.5V_{DS} + 2.5 = 0$ gives $V_{DS} = 0.649\,\mathrm{V}$, $I_D = 2.18\,\mathrm{mA}$. (c) $g_m = 2\,\mathrm{mA/V}$, gain $-g_mR_D = -4$.

2.16 $V_{SG} = 3.3\,\mathrm{V}$, overdrive 2.5 V. (a) Drain at 0.5 V: $V_{SD} = 2.8\,\mathrm{V} \geq 2.5\,\mathrm{V}$, saturation, $I_{SD} = \frac{1}{2}(0.5)(2.5)^2 = 1.56\,\mathrm{mA}$. (b) Drain at 2.8 V: $V_{SD} = 0.5\,\mathrm{V}$, triode, $I_{SD} = 0.5[(2.5)(0.5) - \frac{1}{2}(0.5)^2] = 0.563\,\mathrm{mA}$. (c) Into the source and out of the drain.

2.17 (a) Two decades in 0.18 V: 90 mV/decade; $n = 0.090/(2.303\times0.0259) = 1.51$. (b) $I_{off} = 10\,\mathrm{nA}\times10^{-0.30/0.090} = 4.6\,\mathrm{pA}$.

2.18 (a) 14.4 µW at 1.2 V, 6.4 µW at 0.8 V, a factor of 2.25 lower. (b) $V_{\mathrm{th}} = 0.4\,\mathrm{V}$: $I_{off} = 10\,\mathrm{pA}$, total 10 mW; $V_{\mathrm{th}} = 0.3\,\mathrm{V}$: $I_{off} = 178\,\mathrm{pA}$, total 178 mW. (c) Lowering V_{th} to keep speed at low V_{DD} raises leakage exponentially (about 18× per 100 mV here), so static power soon dominates.

2.19 (a) Exact 133 Ω; use 130 Ω: $I = 15.4\,\mathrm{mA}$, $P_R = 30.8\,\mathrm{mW}$, $P_{LED} = 46.2\,\mathrm{mW}$. (b) 10 mA to 20 mA. With only 0.3 V of headroom, a ±0.1 V spread in V_F changes the current by a factor of two.

2.20 $\mathcal{R} = 0.531\,\mathrm{A/W}$, $I_{ph} = 10.6\,\mu\mathrm{A}$, $R_F = 188\,\mathrm{k}\Omega$.

2.21 Open-circuit voltage 22.3 V; maximum power $0.78 \times 22.3 \times 3.0 = 52.2$ W; efficiency $52.2/350 = 14.9\,\%$.

2.22 (a) Gain $1 + 47/10 = 5.7$; $V_{out} = 1.43$ V. (b) The formula asks for 4.56 V, beyond the 3.3 V rail, so the output stops near 3.3 V and feedback is lost. (c) $3.3/5.7 = 0.579$ V. (d) None (rule 1).

2.23 (a) $I = 0.55$ mA and $V_D = 1.20 - 0.55 = 0.65$ V. (b) $(1.20 - 0.70)/1\,\text{k}\Omega = 0.50$ mA. (c) Both inputs sit at $V_{DAC}/2 = 0.60$ V.

2.24 (a) 2.4 V, leaving 0.9 V. (b) 3.75 MΩ. (c) 1.07 nA per count with 3 MΩ, 0.86 nA with 3.75 MΩ. (d) The cathode: the photocurrent is then drawn out of the virtual-ground node and supplied through R_F, so the output is positive, which a single-supply op-amp requires.

2.25 (a) $V_+ = V_A/2.01$, so $V_{out} = 2V_+ - V_B = 0.9950\,V_A - V_B$. (b) With $V_B = 0.80$ V, A2 reads 2.484 V, which is 16.4 mV low: a current error of −16.4 µA, about 5 ADC counts (0.7 %). (c) The formula gives $0.9950 \times 3.3 - 3.290 = -6.4$ mV; the output cannot go negative, so A3 reads 0 and the 10 µA is invisible. The error is set by the common-mode voltage V_A, not by the current, so it swamps small currents.

2.26 (a) $V_- = V_A/2$ by rule 2, so the current in R_1 from B is $(V_B - V_A/2)/R$. (b) $(0.78 - 1.65)/1\,\text{M}\Omega = -0.87$ µA: 0.87 µA flows from the amplifier into node B, so the diode carries 0.87 µA more than the sense resistor and the reading is that much low, about 0.27 of an ADC step (0.035 % of the 2.52 mA). (c) 87 µA, about 27 ADC steps (3.5 %).

2.27 (a) The source sits at the virtual node, $V_{ref} = 2.0$ V; $V_{out} = V_{ref} - I_D R_F = 1.0$ V. (b) 0 to 2.0 mA (the output falls from 2.0 V to 0 V). (c) $V_{GS} \le 3.3 - 2.0 = 1.3$ V and $V_{DS} \le 1.3$ V; the gate cannot reach the typical threshold, so the transistor barely turns on. (d) $I_{max} R_F = V_{ref}$ and $V_{GS,max} = 3.3 - V_{ref}$, which sum to 3.3 V: gate drive and current range trade off, which the grounded-source sense-resistor circuit avoids.

2.28 (a) $V_{GS} = 2.2$ V, $I_D = 1.46$ mA, $V_{DS} = 0.04$ V. (b) Triode: the overdrive is at least 0.2 V, more than V_{DS}. (c) $V_{DAC2}/1\,\text{k}\Omega = 1.5$ mA.

2.29 (a) $I_D = 20(0.2)^2 = 0.80$ mA, $V_{DS,sat} = 0.20$ V. (b) $V_{DAC2} = 0.20 + 0.80 = 1.00$ V. (c) The plateau is 3.20 mA from $V_{DS} = 0.40$ V on, which needs $V_{DAC2} = 0.40 + 3.20 = 3.60\,\text{V} > 3.3$ V.

2.30 (a) $n = 0.06/(V_T \ln 4) = 1.67$, $I_S = 4.77 \times 10^{-10}$ A. (b) $V_D = 0.700$ V. (c) Higher: at larger currents the drop across the diode's series resistance adds to the junction voltage. The board cannot reach 5 mA anyway: with the DAC at most 3.3 V and 1 kΩ in series, the current stays below 3.3 mA.

2.31 (a) 0.080 mA at $V_{DS} = 2.90$ V; 0.320 mA at $V_{DS} = 1.70$ V; at most $3.3\,\text{V}/5.0\,\text{k}\Omega = 0.66$ mA. (b) $\sqrt{I_D}$ rises from 0.283 to 0.566 in 0.40 V: slope 0.707, so $K = 0.500\,\text{mA/V}^2$ and $V_{\text{th}} = 1.40$ V. Overdrives

0.40 and 0.80 V are below V_{DS} = 2.90 and 1.70 V: both saturated. (c) Saturation would need $0.5(1.4)^2 = 0.98$ mA, more than the 0.66 mA the circuit allows (V_{DS} would be -1.6 V). Triode with $0.5(2.8V_{DS} - V_{DS}^2) = (3.3 - V_{DS})/5.0$ gives $V_{DS} = 0.486$ V, $I_D = 0.563$ mA. (d) Keep points with $0.05\,\text{mA} \le I_D \le 0.6\,I_{max}$ (as in the lab), then `m, b = np.polyfit(vgs, np.sqrt(i), 1)`, $K = m^2$, $V_{\text{th}} = -b/m$, and check $V_{DS} = V_{DAC2} - I_D R_{path} \ge V_{GS} - V_{\text{th}}$ for every point used.

Chapter 3: Sensors

3.1 (a) Temperature changes resistance; read with a voltage divider into the Arduino ADC (pin A1). (b) Light produces a photocurrent; read with a transimpedance amplifier followed by an ADC. (c) Light returning from the finger (modulated by blood volume) produces a photocurrent that the chip amplifies and digitizes internally; the Arduino reads finished numbers over I²C. (d) Strain changes resistance by a tiny amount; read with a Wheatstone bridge, a difference amplifier, and an ADC.

3.2 (a) 20 mV/kPa. (b) 1.40 V.

3.3 (a) Mean 2.0115 V; RMS noise 2.29 mV. (b) SNR = 52.4, or 34.4 dB.

3.4 (a) 5.99 kΩ. (b) 16.1 °C.

3.5 (a) 25 °C: $R_T = 10.0$ kΩ, $V_{\text{out}} = 1.650$ V, code 512. 34 °C: $R_T = 6.78$ kΩ, $V_{\text{out}} = 1.966$ V, code 610. (b) 33.3 mV per degree; one LSB (3.22 mV) is 0.097 °C.

3.6 $|dV/dR_T| = V_S R/(R_T + R)^2$, whose derivative with respect to R is proportional to $R_T - R$, so the maximum is at $R = R_T(T^*)$ (dR_T/dT does not depend on R). Best $R = 6.78$ kΩ; a 6.8 kΩ resistor raises the sensitivity from 33.3 to 34.5 mV per degree, about 3.8 %.

3.7 (a) $I = 0.25$ mA, $P_T = 0.625$ mW. (b) 0.31 K. (c) $P_T = 0.73$ mW, self-heating 0.37 K, sensitivity 21.7 mV/K instead of 36.7 mV/K: more heating and less signal, a bad trade.

3.8 (a) Write $T_5 - T_f = (T_0 - T_f)x$ and $T_{10} - T_f = (T_0 - T_f)x^2$ with $x = e^{-5/\tau}$; eliminate x: $(T_5 - T_f)^2 = (T_0 - T_f)(T_{10} - T_f)$, and the T_f^2 terms cancel. (b) $T_{\text{final}} = 32.0$ °C, $\tau = 5.0$ s. (c) $t = \tau \ln(9.0/0.1) \approx 22.5$ s.

3.9 (a) -0.7 °C. (b) $V_{\text{out}} = 1.642$ V, computed $R_T = 10.10$ kΩ, reported 24.77 °C (0.23 °C low). (c) The code computes $R_{\text{nom}}(V_S/V - 1) = R_T R_{\text{nom}}/R$, always $R_T \times 1.0101$. At 37 °C the error is -0.24 °C, so an offset found at 25 °C removes it to within about 0.02 °C.

3.10 (a) About 60 Hz (one bump every 16.7 ms). (b) A1 was floating and picked up power-line hum, cut off at 0 V by the ADC: the selector was not on Module 1, J8 was not on the Module 1 end, or the parts

were not seated. (c) $R_T = 430\,\text{k}\Omega$ and $-40.9\,^\circ\text{C}$; the divider was not connected, so the formula does not apply.

3.11 (a) $\delta R = 0.735\,\Omega$; $V_o = 2.62\,\text{mV}$ (approximation 2.625 mV). (b) A gain of about 381.

3.12 (a) 0.500 V, 1.266 V, 1.607 V. (b) About 106 °C ($R_1 = 140.9\,\Omega$). (c) Slope 27.8 mV/K, so 0.116 °C per step.

3.13 (a) 0.900 V, code 279. (b) 0.32 °C per LSB, about 3.7 times coarser than the thermistor divider's 0.088 °C. (c) 0.107 °C per LSB; the output reaches 3.3 V at 60 °C.

3.14 (a) LSB = 0.806 mV; code 1241. (b) Half an LSB, 0.403 mV. (c) The fifth harmonic of 3.33 Hz is 16.7 Hz, so $f_s > 33.3\,\text{Hz}$.

3.15 (a) 10 Hz at 50 Hz sampling, 40 Hz at 100 Hz, 1 Hz at 59 Hz, and the true 60 Hz at 1 kHz. (b) Sampling at 59 Hz turns the hum into a 1 Hz oscillation that looks like a 60 bpm heartbeat.

3.16 (a) 10 Hz and 5 Hz. (b) From 116.5 to 119.3 and from 120.7 to 123.5 samples per second. (c) A factor of 0.042, or $-27.6\,\text{dB}$.

3.17 (a) $1 + 47/2.2 = 22.4$. (b) 1.12 V.

3.18 (a) $(1 + 22/10)(-47/10) = -15.0$. (b) $-1.50\,\text{V}$. (c) 0.332 V amplitude (the second stage reaches $\pm 5\,\text{V}$ first; the first stage would not clip until 1.56 V).

3.19 (a) $R_f = 1.5\,\text{M}\Omega$. (b) $C_f = 10.6\,\text{nF}$ (use 10 nF, $f_c \approx 10.6\,\text{Hz}$). (c) $-0.75\,\text{V}$. (d) Reverse the photodiode so the photocurrent flows out of the inverting node; then $V_{\text{out}} = +I_S R_f$.

3.20 (a) 23.6 nA and 688 nA; the LED adds 664 nA. (b) $2 \times 663.9 + 23.6 = 1351\,\text{nA}$, more than the 1.1 µA full scale: the output clips at code 1023. (c) $3.0\,\text{V}/1.35\,\mu\text{A} = 2.2\,\text{M}\Omega$.

3.21 (a) $R_f = 1\,\text{M}\Omega$; the AC swing of V_1 is 10 mV peak to peak (0.995 V to 1.005 V). (b) $V_{\text{DAC}} = 1.0\,\text{V}$ and $R_{2f}/R_2 = 100$, for example $R_2 = 10\,\text{k}\Omega$, $R_{2f} = 1\,\text{M}\Omega$. (c) From 0.5 V to 1.5 V (inverted relative to V_1), well inside 0 V to 3.3 V.

3.22 (a) 3.18 Hz (with $R_T \parallel R = 5\,\text{k}\Omega$). (b) A factor of 0.053, or $-25.5\,\text{dB}$. (c) $\tau_{RC} = 50\,\text{ms}$, more than 100 times shorter than 6 s: no noticeable slowing.

3.23 (a) $A = 0.301$; $A = 2.00$. (b) 31.6 %. (c) $A = 1.50$, so 3.16 %.

3.24 (a) $A = 0.602$; $c = 3.01 \times 10^{-5}\,\text{M}$ (30.1 µM). (b) $T = 0.25^3 = 0.0156$, about 1.6 %.

3.25 $C_{\text{HbO}_2} = 45.0\,\mu\text{M}$, $C_{\text{Hb}} = 5.0\,\mu\text{M}$, saturation 90 %.

3.26 (a) For a small ΔA, $10^{-\Delta A} = e^{-\Delta A \ln 10} \approx 1 - \Delta A \ln 10$, so the fractional drop of the light is about $\Delta A \ln 10$. (b) Exactly, $-\log_{10} 0.99 = 4.365 \times 10^{-3}$. The approximation gives $0.01/\ln 10 = 4.343 \times 10^{-3}$,

which is 0.5 % low. (c) Both AC/DC values carry the same factor $\ln 10$, so it cancels in their ratio.

3.27 (a) $R = 0.50$, $\mathrm{SpO_2} = 97.5\,\%$. (b) $R = 1.00$, $\mathrm{SpO_2} = 85\,\%$: hypoxemic, should seek medical attention. (c) Hard pressure squeezes the arteries and pushes blood out of the tissue, which shrinks and distorts the pulsatile AC signal, so the measured R no longer reflects arterial blood.

3.28 (a) $S = (\varepsilon_{\mathrm{Hb},r} - R\,\varepsilon_{\mathrm{Hb},ir}) / [(\varepsilon_{\mathrm{Hb},r} - \varepsilon_{\mathrm{HbO_2},r}) + R(\varepsilon_{\mathrm{HbO_2},ir} - \varepsilon_{\mathrm{Hb},ir})]$, which with the given numbers is $S = (3227 - 726R)/(2907 + 428R)$. (b) $R = 320/1154 = 0.277$ for 100 %; $R = 3227/726 = 4.44$ for 0 %. (c) Theory 88.2 %, empirical 95.0 %; red and infrared light scatter differently in tissue and travel different path lengths, which the simple Beer-Lambert model ignores.

3.29 (a) Red 0.431 %, infrared 0.978 %, $R = 0.441$, $\mathrm{SpO_2} = 99.0\,\%$. (b) Mean $R = 0.575$, $\mathrm{SpO_2} = 95.6\,\%$; median $R = 0.435$, $\mathrm{SpO_2} = 99.1\,\%$. Report the median: the one bad beat (1.30) drags the mean.

3.30 (a) $34/25\,\mathrm{s} = 1.36\,\mathrm{Hz}$, 81.6 bpm. (b) 79.2 and 84.0 bpm (bins are 0.04 Hz, or 2.4 bpm, apart). (c) 60 s.

3.31 (a) `ibi = np.diff(idx)/100` and `hr = 60/ibi`. IBIs 0.91, 0.88, 0.91, 0.93, 0.89 s; rates 65.9, 68.2, 65.9, 64.5, 67.4 bpm; mean IBI 0.904 s, average 66.4 bpm. (b) 0.125 Hz, or 7.5 bpm.

3.32 (a) $R = R_T(37\,^\circ\mathrm{C}) = 5.99\,\mathrm{k\Omega}$; the nearest standard value is $6.2\,\mathrm{k\Omega}$. (b) 35 °C: 1.610 V, code 499; 41 °C: 1.812 V, code 562; 33.9 mV/K at 37 °C; 0.095 °C per LSB. (c) $G = 2.70/0.2014 = 13.4$; 0.0071 °C per LSB. The noise of the divider and amplifier, not the ADC step, now sets the resolution.

3.33 Convert code to volts (`V = code*3.3/1024`), volts to resistance (`RT = 10e3*(3.3/V - 1)`), and resistance to temperature with Eq. (3.6) (`T = 1/(1/298.15 + np.log(RT/1e4)/3950) - 273.15`). Codes 512, 560, and 610 give 25.0 °C, 29.3 °C, and 34.0 °C.

Chapter 4: Machine Learning

4.1 (a) $\mathbf{z} = (4, 4.5)$. (b) ReLU: (4, 4.5); sigmoid: (0.982, 0.989). (c) $2 \times 3 = 6$ MACs.

4.2 (a) $\mathbf{w} \cdot \mathbf{x} = -0.5$, in 5 MACs. (b) 131 072 weights, 32 biases (131 104 parameters), 131 072 MACs. (c) 256 inputs give 8192 MACs, 16 times fewer.

4.3 $\sigma' = e^{-z}/(1+e^{-z})^2 = \sigma(1-\sigma)$. $\sigma(0) = 0.5$, $\sigma'(0) = 0.25$, $\sigma(4) = 0.982$, $\sigma'(4) = 0.0177$. $\sigma(-z) = 1/(1+e^{z}) = e^{-z}/(1+e^{-z}) = 1 - \sigma(z)$.

4.4 $z = -0.4$. Step: 0; ReLU: 0; sigmoid: 0.401; tanh: -0.380.

4.5 (a) $z = -1.5,\ 0.5,\ -0.5,\ 2.5,\ 0.5$, outputs $0, 1, 0, 1, 1$. (b) The line $2x_1 - x_2 - 1.5 = 0$ crosses the x_1 axis at 0.75 and the x_2 axis at -1.5; the output is 1 below and to the right of it (larger x_1, smaller x_2). (c) The boundary stays where it is, because z only changes by a positive factor. With a sigmoid, the output now goes from near 0 to near 1 across a band 10 times narrower: the decision is sharper.

4.6 (a) $-2 < b \leq -1$ (for example $b = -1.5$). (b) $w_1 = 1$, $w_2 = -1$, $b = -0.5$ (outputs $0, 0, 1, 0$ for $(0,0), (0,1), (1,0), (1,1)$).

4.7 (a) $\mathbf{w} = (1, 1)$, $c = 3$. (b) Without ReLU $y = 0$; with ReLU the first layer gives $(0, 1)$ and $y = 1$. Only the nonlinearity makes the two-layer network differ from one linear layer.

4.8 (a) $\mathbf{z}^{(1)} = (1.5, -3, -0.5)$, $\mathbf{h} = (1.5, 0, 0)$. (b) Logits $(1.7, -0.35)$, $\mathbf{p} = (0.886, 0.114)$, class 1. (c) $C = -\ln 0.114 = 2.17$. (d) 17 parameters $(9 + 8)$ and 12 MACs $(6 + 6)$.

4.9 (a) 29 530 parameters; 29 472 MACs. (b) 115.4 kB in float32, 28.8 kB in int8. (c) 0.921 ms.

4.10 (a) Convolution 1: 80 parameters, 56 448 MACs; convolution 2: 1168 and 225 792; 784-64 layer: 50 240 and 50 176; 64-10 layer: 650 and 640. Totals: 52 138 parameters, 333 056 MACs. (b) 203.7 kB in float32, 50.9 kB in int8; 10.4 ms. (c) $8 \times 28 \times 28 = 6272$ values, 24.5 kB; $203.7 + 24.5 = 228.2$ kB, which fits in 256 kB with little room to spare (in int8 it fits easily). (d) 3.97 times fewer parameters and 3.66 times fewer MACs.

4.11 (a) $\mathbf{p} = (0.085, 0.631, 0.052, 0.232)$; predicted class idle. (b) $C = 1.46$ if up-down, $C = 0.461$ if idle.

4.12 (a) $a = 0.0180$. (b) $C = 0.482$, $\partial C/\partial z = (a - y)\, a(1 - a) = -0.0173$. (c) $C = 4.02$, $\partial C/\partial z = a - y = -0.982$, 56.6 times larger. (d) Since $\partial z/\partial b = 1$, z rises by 0.0173 (to -3.983) with the quadratic cost but by 0.982 (to -3.018) with the cross-entropy, which learns far faster from a confident mistake.

4.13 $\gamma = 0.25$: 1.5, 2.25, 2.625 (steady approach). $\gamma = 0.5$: 3, 3, 3 (exact in one step). $\gamma = 0.75$: 4.5, 2.25, 3.375 (oscillating, converging). $\gamma = 1.25$: 7.5, -3.75, 13.125 (diverging). $w_{k+1} - 3 = (1 - 2\gamma)(w_k - 3)$, so it converges for $0 < \gamma < 1$.

4.14 (a) 8400, 1800, and 1800 windows. (b) $8400/32 = 262.5$, so 263 updates; the last batch holds 16 windows. (c) 5260 updates; full-batch gradient descent makes only 20.

4.15 (a) $\partial C/\partial a = -y/a + (1 - y)/(1 - a)$ and $\partial a/\partial z = a(1 - a)$, whose product is $a - y$. (b) $z = 0.1$, $a = 0.525$, $C = 0.644$, $\partial C/\partial \mathbf{w} = (-0.475, -0.950)$, $\partial C/\partial b = -0.475$. (c) $\mathbf{w} = (0.7375, 0.225)$, $b = 0.3375$, $z = 1.525$, $a = 0.821$, $C = 0.197$.

4.16 (a) $z_1 = 1.5$, $h = 1.5$, $\hat{y} = 2.25$, $C = 0.281$. (b) $\delta_2 = \hat{y} - y =$

-0.75: $\partial C/\partial w_2 = -1.125$, $\partial C/\partial b_2 = -0.75$; $\delta_1 = \delta_2 w_2 = -1.125$: $\partial C/\partial w_1 = -2.25$, $\partial C/\partial b_1 = -1.125$. (c) $w_1 = 0.725$, $b_1 = 0.6125$, $w_2 = 1.6125$, $b_2 = 0.075$; $\hat{y} = 3.40$, $C = 0.0803$ (it overshoots the target, but the cost is lower). (d) $z_1 = -1$, so $h = 0$ and the ReLU slope is 0: $\partial C/\partial w_1 = \partial C/\partial b_1 = \partial C/\partial w_2 = 0$, and only b_2 learns ($\partial C/\partial b_2 = -3$). A ReLU neuron whose input is negative for every training example gets no gradient and never recovers (a "dead" neuron).

4.17 (a) Both hidden neurons are on, with $\mathbf{h} = (0.25, 0.5)$; the logits are $(-0.25, 0.875)$, so $\mathbf{p} = (0.245, 0.755)$ and $C = 0.281$. (b) $\boldsymbol{\delta}^{(2)} = (0.245, -0.245) = \partial C/\partial \mathbf{b}^{(2)}$. $\partial C/\partial \mathbf{W}^{(2)}$ has rows $(0.0613, 0.1225)$ and $(-0.0613, -0.1225)$. $\boldsymbol{\delta}^{(1)} = (0.368, -0.735) = \partial C/\partial \mathbf{b}^{(1)}$. $\partial C/\partial \mathbf{W}^{(1)}$ has rows $(0.368, 0.735)$ and $(-0.735, -1.471)$. (c) After the step, $W_{11}^{(1)} = 0.426$, $W_{12}^{(1)} = -0.647$, $W_{21}^{(1)} = 1.147$, $W_{22}^{(1)} = 0.544$, $b_1^{(1)} = 0.676$, $b_2^{(1)} = -0.853$, $W_{11}^{(2)} = 0.988$, $W_{12}^{(2)} = -1.025$, $W_{21}^{(2)} = -0.488$, $W_{22}^{(2)} = 2.025$, $b_1^{(2)} = -0.049$, and $b_2^{(2)} = 0.049$. The first hidden neuron switches off ($z = -0.191$), the logits become $(-1.465, 2.848)$, $p_2 = 0.987$, and $C = 0.0133$. (d) Both methods give -1.471.

4.18 Overfitting after about epoch 20. Helpful: early stopping near epoch 20, more or augmented training data, dropout or other regularization, a smaller model. Training for more epochs or enlarging the model would make it worse.

4.19 (a) Choosing the best of 40 settings by the test score fits the choice to the test set, which has become a validation set. Tune on a separate validation set (or by cross-validation) and measure the final choice once on the untouched test set. (b) 1000 images per fold, 4000 for training in each round, 5 networks per setting (200 for all 40 settings).

4.20 (a) $240/300 = 0.80$. (b) $TP = 75$, $FN = 25$, $FP = 26$, $TN = 174$; sensitivity 0.75, specificity 0.87, precision 0.743, $F_1 = 0.746$. (c) 1 and 7 (20 sevens called 1, 15 ones called 7): both are mostly a single vertical stroke.

4.21 (a) Threshold 0.5: $TP = 80$, $FP = 40$, $FN = 20$, $TN = 1860$; accuracy 0.970, sensitivity 0.80, specificity 0.979, precision 0.667, $F_1 = 0.727$. Threshold 0.8: $TP = 65$, $FP = 10$, $FN = 35$, $TN = 1890$; accuracy 0.978, sensitivity 0.65, specificity 0.995, precision 0.867, $F_1 = 0.743$. (b) For the night alarm, 0.8: 87% of its alarms are real, so staff keep trusting it, at the price of more missed events. For screening, 0.5: it catches 80% of the arrhythmias instead of 65%, and the cardiologist removes the false positives.

4.22 $s = 0.005039$; integers -60, 16, 127, -79; dequantized -0.3024, 0.0806, 0.6400, -0.3981; largest error 0.00236, within $s/2 = 0.00252$.

4.23 (a) 976.6 kB, 244.1 kB, and 122.1 kB. (b) Only the 4-bit version

$(122.1 + 40 = 162.1$ kB); int8 needs 284.1 kB, and float32 is far too large. (c) 49.7 kB.

4.24 The scales are $s_w = 0.006299$ and $s_x = 0.01181$, and the integers are $\mathbf{q}_w = (71, -127, 32)$ and $\mathbf{q}_x = (127, 42, 85)$. Integer sum $9017 - 5334 + 2720 = 6403$; $s_w s_x = 7.440 \times 10^{-5}$; result 0.4764 against the exact 0.475 (0.29% high). The largest product here is 9017. Any product of two 8-bit numbers is at most $128 \times 128 = 16\,384$ and fits in 16 bits, but a sum of 784 of them can reach about 12.8 million, which needs more than 16 bits.

4.25 (a) 0.372 ms and 3.72 µJ. (b) 14.9 µW computing plus 5.0 µW asleep, 19.9 µW. (c) $1332\,\text{J}/19.9\,\mu\text{W} = 6.70 \times 10^7$ s, about 775 days (2.1 years). (d) 312 pJ per MAC, about 1040 times more: a general-purpose processor spends most of its energy fetching and decoding instructions, moving data between memory and registers, and running its clock, not on the arithmetic. Dedicated accelerators remove most of this overhead.

4.26 (a) $I_1 = 3.5\,\mu\text{A}$, $I_2 = 2.75\,\mu\text{A}$. (b) $V_{o1} = -70\,\text{mV}$, $V_{o2} = -55\,\text{mV}$. (c) 0.688 µW.

4.27 (a) $G^+ = \begin{pmatrix} 20 & 0 \\ 0 & 30 \end{pmatrix}$, $G^- = \begin{pmatrix} 0 & 40 \\ 10 & 0 \end{pmatrix}$ µS. (b) $(-6, 5)$ µA, equal to $G_0\mathbf{WV} = 40\,\mu\text{S} \times (-0.15, 0.125)$ V. (c) A ±10% cell error is a ±10% weight error (a few bits of precision); compensate with program-and-verify tuning of each cell, calibration of the column outputs, or training the network with weight noise.

4.28 (a) $64 \times 20\,\mu\text{S} \times 0.1\,\text{V} = 128\,\mu\text{A}$. (b) $R_f \le 1\,\text{V}/128\,\mu\text{A} = 7.81\,\text{k}\Omega$. (c) 2048 MACs per read; 1.02×10^{10} MACs per second. (d) One step is $128\,\mu\text{A}/256 = 0.5\,\mu\text{A}$, the current of a 5 µS cell at 0.1 V, a quarter of the full cell range. The 8-bit ADC, not the cells, limits how finely the column result, and with it a single weight, can be resolved.

4.29 (a) Every second pixel in each direction (columns $0, 2, 4, \ldots$ and rows $0, 2, 4, \ldots$), one pixel in four: 19 200 of the 76 800. (b) 25 600 characters, 2.22 s; 14 400 characters, 1.25 s. (c) 96×96, 3.43 picture pixels per network pixel; 72×72, 2.57. (d) The Nano's serial port is a native USB connection, which moves data at USB speed regardless of the baud setting. The camera's frame rate (5 frames per second, up to 200 ms of waiting) and the processing on both ends set the picture time.

4.30 (a) 33-20-10-4: 934 parameters, 900 MACs; 375-20-10-4: 7774 parameters, 7740 MACs; features cut the MACs by a factor of 8.6. (b) 28.1 µs and 242 µs. (c) About 35 µW.

4.31 Errors per epoch 1, 3, 3, 2, 1, 0: the sixth epoch is the first error-free pass. Final $\mathbf{w} = (2, 1)$, $b = -2$; the boundary $2x_1 + x_2 - 2 = 0$ passes through $(1, 0)$, which gives $z = 0$ and output 0.

4.32 (a) $795H + 10 \le 16\,384$ gives $H = 20$, using 15 910 bytes (15.5 kB). (b) 15 880 MACs, 0.496 ms. (c) Measured values vary; expect the 784-20-10 network to score a few points higher than 784-8-10 (roughly 92% versus 90%) for 2.5 times the memory.

Chapter 5: Communications

5.1 `U` = `0x55` = `01010101`, `S` = `0x53` = `01010011`, `C` = `0x43` = `01000011`. Task 1 sends the least significant bit first, so `U` goes out as 1,0,1,0,1,0,1,0. Three 30-ms frames take 90 ms; the useful rate is $24/90\,\text{ms} = 267\,\text{bit/s}$.

5.2 (a) 6 bytes (*ï* takes two, `0xC3 0xAF`). (b) 4 bytes (€ takes three). (c) 6 bytes (three per character). (d) 5 bytes. (e) The 16 bits of `U+20AC` are `0010 000010 101100`, so € is `0xE2 0x82 0xAC`.

5.3 OOK at 2 ms: 500 baud, 500 bit/s, 0.784 s, and 459 bit/s useful. OOK at 1 ms: 1000 baud, 1000 bit/s, 0.392 s, and 918 bit/s. PAM-4 at 2 ms: 500 baud, 1000 bit/s, 0.392 s, and 918 bit/s. PAM-4 at 1 ms: 1000 baud, 2000 bit/s, 0.196 s, and 1837 bit/s.

5.4 (a) The symbol rate is 500 baud and the bit rate is $500 \times 3 = 1500\,\text{bit/s}$. (b) 104 bits need $\lceil 104/3 \rceil = 35$ symbols, or 70 ms.

5.5 `K` = `01001011`. OOK has 5 transitions inside the byte; Manchester (16 half-bit levels `HL LH HL HL LH HL LH LH`) has 10, including one in the middle of every bit. The guaranteed mid-bit edge keeps the receiver's clock aligned and holds the average brightness at 50%; the cost is twice the symbol rate, so about twice the bandwidth.

5.6 (a) Advantages: no radio interference with medical equipment, light stays in the room (privacy, and every room can reuse the channel), large unregulated spectrum. Drawbacks: needs line of sight (blocked by people, curtains, bedding), needs a separate uplink (often infrared), and coverage is limited to the lit area. (b) Below a few hundred hertz the eye sees flicker, and the average brightness must look steady. (c) It passes only about 750 to 1050 nm, so it blocks much of the visible light of room lamps, which would otherwise shift the baseline, risk saturation, and add shot noise. Sunlight and filament lamps still get through, because they contain near infrared.

5.7 (a) LSB = 0.806 mV, RMS quantization noise 0.233 mV, ideal SNR $6.02(12) + 1.76 = 74.0$ dB. (b) 12 bits (11 bits give 1.61 mV). (c) 3.01 dB, 30 dB, 20 dB; 25 dB is a power ratio of 316 and a voltage ratio of 17.8. (d) The signal voltage falls by 4, so the electrical SNR falls by 16, or 12.0 dB.

5.8 (a) $P_r = 7.96\,\mu\text{W}$, $I_{ph} = 3.18\,\mu\text{A}$. (b) $d = 2.5\,\text{m}$ and $\cos\phi = \cos\psi = 0.8$: $P_r = 3.26\,\mu\text{W}$, $I_{ph} = 1.30\,\mu\text{A}$. The power drops by 3.88 dB.

5.9 (a) $m = 11.1$, $P_r = 0.278\,\mu\text{W}$, $I_{ph} = 0.153\,\mu\text{A}$, $v_{\text{out}} = 0.458\,\text{V}$, about

142 counts. (b) Up to $4\,\text{cm} \times \sqrt{142/200} = 3.37\,\text{cm}$. (c) Closer than about 1.5 cm (an estimate: at that range the LED is no longer a point source). (d) The factor $\cos^{11.1}(10^\circ)\cos(10^\circ) = 0.83$ gives about 118 counts. A large m concentrates the light near the axis, so the intensity falls quickly off axis.

5.10 (a) $|v_{\text{out}}| = (4\,\mu\text{A})(200\,\text{k}\Omega) = 0.80\,\text{V}$. (b) $\tau = 1\,\mu\text{s}$, so $f_{3\text{dB}} = 159\,\text{kHz}$ and $t_r = 2.2\,\mu\text{s}$, giving $R_{b,\max} \approx 1/(2t_r) = 227\,\text{kbit/s}$. (c) $C_f \le 0.5\,\mu\text{s}/(2.2 \times 200\,\text{k}\Omega) = 1.14\,\text{pF}$.

5.11 (a) $t_f = 0.76\,\text{ms}$, $\tau = 0.345\,\text{ms}$, $f_{3\text{dB}} = 461\,\text{Hz}$, $C_f = 115\,\text{pF}$, $R_{b,\max} \approx 658\,\text{bit/s}$. (b) Eye height $= A(1 - e^{-t/\tau}) - Ae^{-t/\tau} = A(1 - 2e^{-t/\tau})$. At mid-bit: 0.994, 0.889, 0.530, and 0.030 of A for 4, 2, 1, and 0.5 ms; at the end of the bit: 1.000, 0.994, 0.889, and 0.530. (c) The mid-bit eye closes for $T_b < 2\tau \ln 2 = 0.48\,\text{ms}$; the end-of-bit eye closes for $T_b < \tau \ln 2 = 0.24\,\text{ms}$.

5.12 (a) $v_s = 0.50\,\text{V}$; $v_{\text{th}} = 9.10\,\mu\text{V}$, $v_{\text{sh}} = 28.3\,\mu\text{V}$, total 29.7 µV; SNR $= 2.83 \times 10^8 = 84.5\,\text{dB}$. (b) $I = 2kT/(qR) = 0.517\,\mu\text{A}$.

5.13 (a) $\tau = 47\,\mu\text{s}$, $B_n = 5.32\,\text{kHz}$; $v_{\text{th}} = 9.4\,\mu\text{V}$, $v_{\text{sh}} = 65.3\,\mu\text{V}$, $v_q = 233\,\mu\text{V}$ (dominant); total 242 µV, SNR 66.3 dB. (b) $v_{\text{sh}} = 29.2\,\mu\text{V}$, total 235 µV, SNR 66.6 dB: almost no change, because quantization dominates. (c) With 16 bits $v_q = 14.5\,\mu\text{V}$: total 67.5 µV and SNR 77.4 dB with ambient light (its shot noise now dominates), 33.9 µV and 83.4 dB without. Shading pays off once the other noise sources are smaller than the shot noise of the ambient light.

5.14 (a) Room light: 1.00 V, about 205 counts. LED on: 3.57 V, about 731 counts. (b) About 255 counts (1.25 V). (c) A0 exceeds 255 counts when the LDR falls below about 30 kΩ.

5.15 (a) BER $= Q(0.2/0.04) = Q(5) = 2.9 \times 10^{-7}$, about 2.3 expected errors in 8×10^6 bits. (b) 0s now fail at $Q(2.5) = 6.2 \times 10^{-3}$ and 1s at $Q(7.5) \approx 0$, so BER $\approx 3.1 \times 10^{-3}$. (c) Spacing 0.133 V, half-gap 1.667σ: $P_s = 1.5\,Q(1.667) = 0.072$, BER ≈ 0.036.

5.16 (a) Half-gap 20 counts $= 8\sigma$: BER $= Q(8) = 6.2 \times 10^{-16}$. (b) 1s fail at $Q(2.4) = 8.2 \times 10^{-3}$ and 0s practically never, so BER $= 4.1 \times 10^{-3}$, about 1.5 errors per message. (c) Spacing 13.3 counts, half-gap 2.67σ: $P_s = 1.5\,Q(2.67) = 5.7 \times 10^{-3}$ and BER $\approx 2.9 \times 10^{-3}$. A badly placed threshold costs more than going from OOK to PAM-4.

5.17 (a) Amplitude noise (ambient light, a weak signal, a wide bandwidth): shade the receiver, move closer, narrow the bandwidth, or average samples. (b) Timing jitter or a clock mismatch: resynchronize on edges or a preamble, or measure the symbol period. (c) A slow receiver (large R_fC_f, intersymbol interference): use a smaller R_fC_f or longer symbols, and sample late in the symbol.

5.18 (a) SNR $= 316$, $C = (2\,\text{MHz})\log_2(317) = 16.6\,\text{Mbit/s}$. (b) $\log_2(1 + \text{SNR}) = 15$, so SNR $= 32{,}767 = 45.2\,\text{dB}$.

5.19 (a) Swing 132.7 counts. $M = 2$: $d/2\sigma = 34.9$, BER effectively 0 ($\sim 10^{-267}$). $M = 4$: 11.6, BER $\approx 10^{-31}$. $M = 8$: 4.99, BER $\approx 1.77 \times 10^{-7}$. $M = 16$: 2.33, BER $\approx 4.67 \times 10^{-3}$. (b) $M = 8$, giving $3/12\,\text{ms} = 250\,\text{bit/s}$. (c) The LED's light output is nonlinear in drive voltage, so equally spaced codes do not give equally spaced levels; ambient light drifts; clock mismatch moves samples toward transitions; and the noise may not be Gaussian.

5.20 (a) 2.85 V (27 counts; 2.80 V gives only 12, below $3 + 5\sigma = 13$). (b) 0 V (3 counts) and 3.30 V (185 counts), threshold 94 counts. (c) Levels 3, 63.7, 124.3, and 185 counts need 0, 2.936, 3.058, and 3.30 V; thresholds 33.3, 94.0, and 154.7 counts. (d) Half-gap 30.3 counts $= 15.2\sigma$. With 0, 2.90, 3.10, and 3.30 V the levels are 3, 47, 143, and 185; the smallest gap is 42 counts, a half-gap of $21 = 10.5\sigma$. (e) The curve is steep between about 2.9 and 3.1 V and flat above, so equal steps in received level need small voltage steps in the middle and a large one at the top.

5.21 (a) $N = \lfloor 12/5 \rfloor = 2$, so the receiver assumes 10 ms symbols. (b) Samples at 5, 15, 25, ..., 95 ms land in symbols 0, 1, 2, 2, 3, 4, 5, 6, 7, 7: every sixth sample repeats a symbol. (c) About $2352/10 \approx 235$ symbols instead of 196, so the bits slip and the text is garbled. (d) Any interval that divides 12 ms evenly and leaves at least two samples per symbol: 1, 2, 3, 4, or 6 ms.

5.22 (a) $c = -4, 0, 4, 0, -2$: the preamble starts at $k = 2$. (b) $c = -2, 2, 2, -2, -2$: offsets 1 and 2 tie, so one wrong sample makes the start ambiguous. (c) A 16-bit preamble gives a peak of 16, which a few wrong bits cannot move to another offset, and about 16 samples per bit average the noise and let the correlation place the start to a fraction of a bit.

5.23 The frame has $5 \times 8 + 32 = 72$ bits, or 36 symbols; the recording holds $36 \times 6 + 60 = 276$ samples. With thresholds 33, 91, and 149 counts (half-gaps of 14.5σ) the function returns `"EE105"` with no errors.

5.24 `0xC5` (four 1s) passes and decodes to `E`; `0xC4` (three 1s) fails; `0xD7` (six 1s) passes and decodes to `W`. `0xD7` differs from `0xC5` in two bits, so the parity stayed even and the error went undetected: parity misses any even number of flipped bits.

5.25 `U` (`1010101`, four 1s) $\rightarrow$ `0x55`; `S` (`1010011`, four 1s) $\rightarrow$ `0x53`; `C` (`1000011`, three 1s) $\rightarrow$ `0xC3`. `0xD3` = `11010011` has five 1s and fails.

5.26 (a) $69 + 69 + 49 + 48 + 53 = 288$, and $288 - 256 = 32 =$ `0x20`. (b) The sum becomes 289, checksum 33: caught. (c) The sum is still 288: not caught, because a sum ignores order. (d) Changing one byte by δ with $0 < |\delta| < 256$ changes the sum by δ, which is not a multiple of 256. (e) About 1 in 256, or 0.4%.

5.27 (a) Triple: $3p^2(1-p) + p^3 = 1.18 \times 10^{-3}$. Five-fold: $10p^3(1-p)^2 + 5p^4(1-p) + p^5 = 7.8 \times 10^{-5}$. (b) 0.36 s uncoded, 1.08 s and 1.80 s

with repetition, and 0.63 s with Hamming (7,4) (90 blocks of 7 bits). (c) $1-(1-p)^7-7p(1-p)^6 = 7.86\times10^{-3}$.

5.28 (a) Syndrome 0: no error, data 0101. (b) Syndrome 6: flip position 6 to get `0100101`, data 0101. (c) Syndrome 1: the parity bit p_1 was hit; corrected word `0100101`, data 0101. (d) Syndrome $2\oplus7=5$: the decoder flips position 5 and outputs `0000000`, data 0000. A double error is "corrected" into a third error; Hamming (7,4) handles only single errors, and an extended (SECDED) code would at least detect this one.

5.29 (a) Positions 1 to 11: `0 1 1 0 0 0 0 0 0 1 1` ($p_1=0$, $p_2=1$, $p_4=0$, $p_8=0$), and $p_0=0$. (b) Checks c_2 and c_4 fail, syndrome $=6$, and overall parity fails: a single error at position 6, so flip it back. (c) Syndrome $=3\oplus6=5$ with overall parity passing: a double error is detected and the block is rejected. A plain (11,7) decoder would flip position 5, leaving three data bits wrong and decoding `f` (`0x66`) instead of `a`.

5.30 (a) Weights 0.746, 0.187, 0.067; fused estimate 36.97 °C with $\sigma=0.26$ °C. (b) Plain average 36.93 °C with $\sigma=\sqrt{0.3^2+0.6^2+1.0^2}/3=0.40$ °C, worse than the best single thermometer.

5.31 (a) $\alpha=\tau/(\tau+\Delta t)=1/1.005=0.995$. (b) $\hat\theta_1=0.995(0.05)+0.005(0.3)=0.0512°$; $\hat\theta_2=0.995(0.0512+0.05)+0.005(0.2)=0.1017°$. (c) $b\tau=(0.2\,°/\mathrm{s})(1\,\mathrm{s})=0.2°$.

5.32 Step 1: $\hat x^-=1$, $P^-=1.5$, $K=0.429$, $\hat x=1.17$ m, $P=0.857$. Step 2: $\hat x^-=2.17$, $P^-=1.357$, $K=0.404$, $\hat x=1.98$ m, $P=0.809$. K falls because the estimate's variance has shrunk relative to the sensor's, so the filter trusts its prediction more; it settles toward a steady-state gain.

5.33 (a) $0.03\times0.08=0.0024$ (0.24%). (b) $3(0.05)^2(0.95)+(0.05)^3=0.00725$ (0.73%). (c) The vote fails with probability $3p^2-2p^3$, which is less than p when $2p^2-3p+1=(2p-1)(p-1)>0$, that is, for $p<0.5$.

Chapter 6: The Final Project

6.1 One possible answer: "For a lab technician who handles cold samples and wants to know when a glove has been touching ice too long, we will build a glove with a thermistor on the fingertip that flashes a warning light after a set time below 5 °C, and we will know it works when it alerts within 10 s of the limit in 9 of 10 trials with the fingertip held on ice in a room-temperature lab." Any answer must name the user, the need, the system, and a measurable result with conditions.

6.2 Sense: temperature (the thermistor divider of Chapter 3, or the Nano's own temperature sensor) and humidity (the Nano's humidity

sensor, listed in Chapter 0); the Nano cannot sense air motion directly, so still air is inferred from readings that change little over time; process: averaging, thresholds, and a timer on the Arduino, Chapters 0, 3, and 4; communicate: an LED, a message over the light link of Chapter 5, or the Nano's Bluetooth radio (Chapter 0).

6.3 $S_D = 0.20(5) + 0.25(2) + 0.25(4) + 0.15(3) + 0.15(4) = 3.55$, below idea A's 4.25. Raising "measurable" (weight 0.25, score 2) helps most: defining a clear numerical goal that scores 5 adds 0.75, giving 4.30.

6.4 The raw rate is $8000 \times 10 = 80\,\text{kbit/s}$, forty times the link capacity, so no. One byte every 100 ms is 80 bit/s, or 4% of the 2 kbit/s link.

6.5 $220/18 \approx 12.2\,\text{h}$. For 48 h the average current must be $220/48 \approx 4.58\,\text{mA}$, a factor of about 3.93 lower.

6.6 Each 5% part causes about 1.125 K of error. Worst case $\approx 0.044 + 1.125 + 1.125 \approx 2.29\,\text{K}$; root sum of squares $\approx 1.59\,\text{K}$. It does not meet $\pm 0.5\,^{\circ}\text{C}$ without calibration.

6.7 Mean error 0.3 breaths per minute; sample standard deviation about 1.89 breaths per minute; 8 of 10 (80%) within ± 2.

6.8 Open-ended. A good plan records a known message of at least several thousand bits at 5 cm with the room lights on (and off as a control), counts bit errors, and decides in advance a threshold (for example, more than one error in 2000 bits) that would make the team change the link design or the distance.

About the Author



Yasser Khan is the Andrew and Erna Viterbi Early Career Chair and an Assistant Professor in the Ming Hsieh Department of Electrical and Computer Engineering and Biomedical Engineering at the University of Southern California, which he joined in 2022. There he teaches EE 105, Introduction to Electrical Engineering, the course this book was written for.

He earned his B.S. in Electrical Engineering from the University of Texas at Dallas and his M.S. in Electrical Engineering from KAUST. He completed his Ph.D. in Electrical Engineering and Computer Sciences at the University of California, Berkeley, in 2018, and then conducted postdoctoral research in the Department of Chemical Engineering at Stanford University.

His research uses additive manufacturing and hardware-enabled AI to develop next-generation wearables, implantables, and ingestibles for precision health and psychiatry. Among his honors are the 2026 DARPA Young Faculty Award, the 2025 Air Force Office of Scientific Research Young Investigator Award, the 2025 IEEE Sensors Council Early Career Technical Achievement Award, the 2025 USC Viterbi Junior Faculty Research Award, the 2024 Packard Fellowship, and the 2023 Google Research Scholar Award. Earlier in his career he received the EECS Departmental Fellowship at UC Berkeley, the Discovery Scholarship and Graduate Fellowship at KAUST, and the Academic Excellence Scholarship at UT Dallas. His work has been published in *Nature*, *Science*, *Cell*, *PNAS*, the *Advanced Materials* family of journals, and other leading venues, and has been featured by BBC News, *The Wall Street Journal*, *Newsweek*, and NSF News.