Computerized Instrumentation Design

PHY 215

Prepared by Kerry Kuehn and Jeff Brown,
Wisconsin Lutheran College

October 31, 2024

Contents

1 Introduction to Computerized Instrumentation 1

1.1 Overview 1
1.2 Motivation 2
1.3 Equipment 2
1.4 Writing and submitting laboratory documentation 4
1.5 Additional course resources 5

2 Getting Started 7

2.1 System Administration 7
2.1.1 System capabilities exercise 7
2.2 Getting started with python 8
2.2.1 Installing python exercise 8
2.2.2 Running python from the command line exercise 8
2.3 Interactive development environment (IDE) 9
2.3.1 Installing Thonny exercise 9

3 Basic Python Programming 11

3.1 Variables and simple data types 11
3.1.1 Simple messages exercise 11
3.1.2 Name cases exercise 11
3.1.3 Name eight exercise 11
3.1.4 Names exercise 11
3.1.5 Guest list exercise 12
3.1.6 Seeing the world exercise 12
3.1.7 Animals exercise 12
3.1.8 Cubes exercise 12
3.1.9 Buffet exercise 12
3.2 Putting it all together 12
3.2.1 Temperature Conversion exercise 13

4 Introduction to Microcontrollers 15

4.1 ESP32 microcontroller setup 15
4.1.1 Downloading ESP32 tutorial package exercise 15
4.1.2 Flashing firmware to ESP32 exercise 16

4.2 Using the ESP32 online and offline . . . 17
4.2.1 Testing code online exercise . . . 17
4.2.2 Testing code offline exercise . . . 17
4.2.3 “Cutting the cord” exercise . . . 18
4.3 Reading assignment . . . 18

5 Digital output **19**
5.1 Number Systems . . . 19
5.1.1 Binary numbers exercise . . . 20
5.1.2 Hexadecimal numbers exercise . . . 20
5.1.3 Number systems in history exercise . . . 20
5.2 Using the ESP32 digital output ports . . . 21
5.2.1 Blink LED exercise . . . 21
5.2.2 Button and LED exercise . . . 22
5.2.3 Mini table lamp exercise . . . 22
5.2.4 Flowing light exercise . . . 22
5.2.5 Breathing LED exercise . . . 22
5.2.6 Meteor flowing light exercise . . . 22
5.2.7 RGB LED exercise . . . 22
5.2.8 Gradient LED exercise . . . 22
5.2.9 Switching speed exercise . . . 22

6 A Little More Python Programming **25**
6.1 If statements . . . 25
6.1.1 Conditional tests exercise . . . 25
6.1.2 Stages of life exercise . . . 25
6.2 User input and while loops . . . 26
6.2.1 Multiples of 10 exercise . . . 26
6.2.2 Movie tickets exercise . . . 26
6.3 Functions . . . 26
6.3.1 Message exercise . . . 26
6.3.2 T-shirt exercise . . . 26
6.3.3 City names exercise . . . 26
6.3.4 Messages exercise . . . 27
6.3.5 sandwiches exercise . . . 27
6.3.6 imports exercise . . . 27
6.4 Classes . . . 27
6.4.1 users exercise . . . 27
6.4.2 login attempts exercise . . . 27
6.4.3 admin exercise . . . 28
6.4.4 privileges exercise . . . 28
6.4.5 imported admin exercise . . . 28
6.5 Files and Exceptions . . . 28
6.5.1 learning python exercise . . . 28
6.5.2 guest book exercise . . . 29
6.5.3 common words exercise . . . 29

6.5.4 favorite number exercise 29
6.6 Putting things together 29
6.6.1 Temperature conversion exercise 29

7 A/D Conversion 31
7.1 Analog and digital signals 31
7.2 Resolution and Sampling Speed 32
7.2.1 Dynamic range exercise 32
7.2.2 Digital recording studio exercise 33
7.3 Digital to Analog Conversion 33
7.3.1 Analog output exercise 34
7.4 Putting it together 34
7.4.1 Potentiometer exercise 34
7.4.2 Digital signal processing exercise 35
7.4.3 Soft light exercise 35
7.4.4 Nightlamp exercise 35

8 Temperature Measurement and Control 37
8.1 Introduction 37
8.2 Joule heating 37
8.2.1 Joule heating exercise 38
8.3 Wire gauge and electrical resistivity 38
8.3.1 Resistivity exercise 38
8.4 Heat capacity and thermal response 39
8.4.1 Calorimetry exercise 39
8.4.2 Basic heater setup exercise 39
8.4.3 Theoretical thermal response time exercise 40
8.5 A bit of first-order response theory 41
8.6 Thermistor basics 41
8.6.1 Setting up the thermistor circuit 42
8.7 Drude theory 42
8.8 Feedback and control 43
8.8.1 Digital control of the HEXFET exercise 44
8.8.2 Temperature regulation exercise 45
8.9 Thermistor temperature calibration 45
8.9.1 Thermistor calibration runs 46
8.10 Least squares fitting to data 46
8.10.1 Least squares fit to data 48
8.10.2 Plot of residuals 48
8.10.3 A better temperature controller 48
8.11 Errors in data and parameters 49
8.11.1 Errors in thermistor data 50
8.11.2 Scientific Writing Assignment 50

9 Timers and Interrupts **53**
9.1 Introduction . . . 53
9.2 Timers . . . 53
9.2.1 Blink an LED . . . 54
9.2.2 Sequencing events . . . 55
9.3 Interrupts . . . 56
9.4 Noisy switches . . . 56
9.5 Summary . . . 59

10 Thermal Diffusion Experiments **61**
10.1 Introduction . . . 61
10.2 Heat flow equation . . . 61
10.2.1 Diffusion equation solution . . . 63
10.2.2 Integration practice . . . 64
10.2.3 Reduced temperature and time . . . 64
10.2.4 Heat capacity . . . 65
10.3 Experimental setup . . . 65
10.3.1 Circuit assembly . . . 65
10.3.2 Control program . . . 65
10.4 Conducting the experiment . . . 67
10.4.1 Data collection . . . 67
10.4.2 Data analysis . . . 67
10.4.3 Final paper . . . 67

List of Figures

1.1 The ESP32 WROVER microcontroller mounted *via* a GPIO Extension Board onto a protoboard. 3

8.1 Basic heater circuit 40
8.2 Thermistor circuit 42
8.3 HEXFET temperature controller 44

9.1 Blink LED 54
9.2 (Left) Mechanical switches can vibrate on/off before settling into a stable state. (Right) Oscilloscope trace of a real switch demonstrating this phenomenon. 57

10.1 Thermal diffusion circuit 66

List of Tables

5.1 Comparison of binary, hexadecimal, and decimal representation of numbers. 21

Chapter 1

Introduction to Computerized Instrumentation

1.1 Overview

This manual is designed to accompany an introductory course on the uses of a small computer in the laboratory. The primary aims of this course are to teach you (1) how set up a computer–controlled laboratory experiment and (2) how to produce a publication-quality scientific paper detailing your results. Along the way, you will learn a bit about computer system administration and programming, analog and digital electronics, scientific data collection and analysis, temperature measurement and regulation, and the mathematical theory of heat conduction.

During a traditional 15-week semester, the students meet with the instructor twice a week for three hour laboratory sessions. Typically, students will need to do three additional hours of work per week outside of scheduled class time in order to complete all of the assignments. Course grades will be assigned based on completion of exercises recorded in a laboratory notebook and a final scientific paper. Details regarding the laboratory notebook expectations and scientific paper format are provided in a later section.

Students enrolled in this course often have a wide range of backgrounds: some come with only a strong desire to learn; others, because of their previous study, could practically teach the course. For this reason, the course is designed to be flexible in that students are encouraged to work at their own pace. Naturally, the present laboratory manual cannot cover every problem or challenge that might arise when working with computers in a laboratory setting. So I strongly encourage those of you who are more experienced to share your understanding with students who have not had as much preparation. Be generous! I

think you will find that in explaining concepts to others, you will deepen your own understanding as well.

1.2 Motivation

Why set up a computer-controlled laboratory experiment? Primarily for automation. Although an experimenter can certainly record the output of an instrument and record it in his or her laboratory notebook, when this needs to be done ten or a hundred or a thousand times, the chance of a scribal error increases to an unacceptable level. Automation allows for reduction of errors in data collection. Automation also reduces the amount of tedious human labor. The experimenter can do other tasks–often remotely–while his or her experiment is running instead of focusing on repetitive tasks. In addition, automation allows for performance that is simply unattainable otherwise. A human being can not record a hundred temperature readings every second. A computer can. Finally, automation allows for information to be readily stored in digital form, which is much simpler to manipulate and analyze.

A word of warning about automation, however, is in order. Automation can often introduce systematic errors which, if not detected and corrected in time, can prove catastrophic. To take just one example, consider the loss of NASA's Mars Climate Orbiter in September of 1999. A programming error went undetected and a $330 million dollar science project (in expensive 1999 dollars...) was lost in an entirely automated fashion as it approached surface of Mars. Apparently, there was a programming error due to an incorrect thrust unit conversion from the imperial to metric system. The adage that a computer is only as smart as its designer is entirely appropriate. Automation is not a silver bullet and should not be thought of as such. A great deal of planning and careful experimentation is necessary before a reliably automated system can be produced. Do not be afraid to experiment; trial and error is the way to learn. Hopefully, this course will give you a deeper understanding of computer technology so that you may intelligently use it to your advantage.

1.3 Equipment

This course has evolved since its inception in early 2000. Originally, the course employed a "Wintel" desktop personal computer[1] running the QNX real-time operating system. An analog-to-digital (A/D) computer board[2] was installed in one of the ISA slots of the PC's motherboard. Such an A/D board is similar to a keyboard, mouse, or monitor, in that it allows the computer's microprocessor to communicate with the outside world; it is unlike these three peripheral devices in that it is not designed to communicate with a human, but rather with a piece

[1] In "Wintel" machines, the computer's motherboard was developed around the Intel processor architecture and it often used a Microsoft Windows-like operating system. Such PCs are descendants of the original IBM-PC.

[2] Measurement Computing, CIO-DAS-08 JR

of equipment that produces a voltage (*e.g.* the signal from a thermistor) or that needs an electronic signal to operate (*e.g.* a heater or a light emitting diode). The C-Programming language was used to communicate with the A/D board.

As of 2024, an internal A/D board is no longer used. It has been replaced by an external ESP32 micro-control unit attached via a USB cable to the PC. The ESP32 board is mounted on a proto-typing board (proto-board for short), as shown in Fig.1.3, in order to connect auxiliary electronic equipment. On the proto-board, you will be constructing circuits using jumper wires, resistors, capacitors, light emitting diodes (LEDs), field effect transistors (FETs), thermistors, and switches. To operate and test these circuits, you will also need an oscilloscope, a function generator, and a DC power supply. The Python Programming language will be used to communicate with the ESP32 micro-controller.

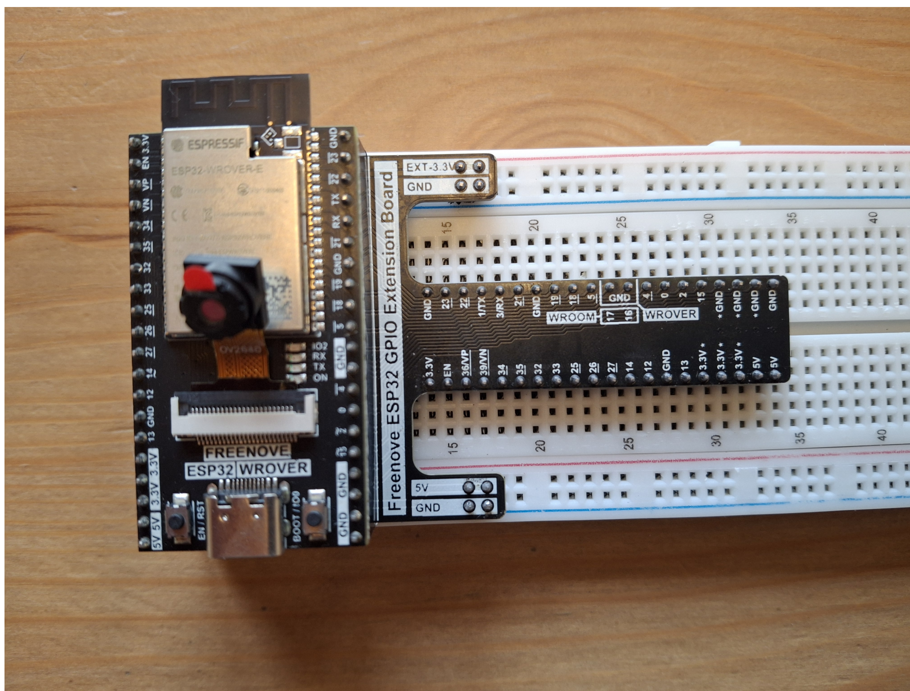


Figure 1.1: The ESP32 WROVER microcontroller mounted *via* a GPIO Extension Board onto a protoboard.

In order to carry out the temperature control and heat diffusion experiments, your laboratory instructor will provide you with (1) a small block of aluminum in order to test your temperature measurement and control techniques, and (2) a rod of copper to measure its heat capacity and thermal conductivity. The

aluminum block and the copper rod each contain an integral heater and one or more mounted thermistors.

1.4 Writing and submitting laboratory documentation

The exercises in this manual involve writing computer code, performing calculations, building electronic circuits, collecting and analyzing experimental data, and occasionally answering theoretical questions. I hope you find them interesting! In any case, whenever you complete an exercise, you must documented your work and submit it electronically to the course instructor. In this way, both you and the instructor will have a detailed record of all that you have accomplished in this course. **You will only receive credit for laboratory exercises that are electronically submitted to the instructor in a timely manner.** For consistency, let us now agree to adopt a consistent naming convention for each document that is electronically submitted; if a student with the initials A.C.K is submitting exercise 2.1.1, then his submitted file name should be:

```
phy215_ex_2_1_1_ack
```

There are many ways to produce appropriate electronic laboratory documentation. Your instructor will provide you with some recommendations and a few example documents to show you what is expected. Generally speaking, the laboratory documentation should provide its reader with enough information that he or she has a fighting chance of reproducing what you did. More specifically, your laboratory documentation should include:

- the exercise number and title (*e.g.* **Ex. 2.1.1: System capabilities**);
- your **name and the date(s)** on which the exercise was performed;
- a concise description of what you have done using complete sentences and correct grammar, spelling, and punctuation;
- neat sketches or photographic images of your laboratory setup;
- proper electronic circuit diagrams;
- clear pseudo-code or code snippets that include appropriate comments;
- mathematical calculations with clear explanation;
- experimental results, data tables, and properly labelled graphs;
- descriptions of difficulties and your solutions or work-arounds;
- any additional thoughts or insights that you think may be helpful.

1.5 Additional course resources

There is no single text for this course. We will primarily follow the exercises outlined in this manual, which was based, in part, on undergraduate courses taught at the University of California at Santa Barbara and at Cornell University. Some sections of this manual are adapted directly from the highly informative *IBM PC in the Laboratory*, by Thompson and Kuckes[6], the text formerly used at Cornell. Here are a few additional resources that you will find very helpful:

- ESP32 micro-controller documentation, especially the ESP32 data-sheet and the ESP32 getting-started tutorial available for download as pdf files from FreeNove.
- *Python Crash Course, 3rd Edition*, by Eric Matthes. This book provides an excellent and comprehensive introduction to programming in Python.
- *The Art of Electronics* by Horowitz and Hill. This is a classic covering analog and digital electronics.

Chapter 2

Getting Started

2.1 System Administration

Without an operating system (OS), your computer is merely a collection of metal and plastic and semiconductor parts. The operating system allows a user to manage a computer's resources–its disk drive, monitor, keyboard, programs, network and sound cards, and so on. Examples of popular operating systems are Linux, UNIX, Mac OSX, and Microsoft Windows.

In this course, you will use your own computer with your choice of operating system. Minimally, you will need a computer that has a keyboard, monitor, and one USB port that will allow the ESP32 microcontroller to communicate with the computer.

2.1.1 System capabilities exercise

If you have not already done so, now is a perfect time to familiarize yourself with your personal computer.

1. What operating system are you using? What version?

2. What kind of computer are you using? What is the make and model? When was this manufactured?

3. What kind of ports does your computer have? Provide any notable information about these ports.

4. How much memory does your hard disk have? What is your processor speed? How much RAM (Random Access Memory) does your computer have?

5. About how many copies of the book of Genesis would fit on your hard disk? (Hint: This is a bit tricky. To answer this question, you might take this approach: (i) estimate the memory required per ASCII character (in

bits or bytes), (ii) estimate how many characters there are per page of Genesis, and (iii) estimate how many pages there are in Genesis.)

When writing up your laboratory documentation for this first exercise, don't forget to give context for your answer, instead of just writing down an answer. For example, for the first question above, you could write

```
1. My operating system: macOS Sonoma 14.1.
```

2.2 Getting started with python

All of the applications you run, whether they are mail programs or web browsers or even graphical user interfaces, are computer programs written by a programmer. In this course, we will be developing our own applications with an eye toward controlling laboratory equipment and carrying out scientific experiments. The programming language we will use is called Python. You may already be familiar with programming. If so, this should be review for you. If not, it's time to learn! Let's begin by installing python and writing our first program.

2.2.1 Installing python exercise

Let's install Python on your computer (if you have not done so already). Much of what follows is an abbreviated version of the instructions for setting up Python on macOS that is contained in *Python Crash Course* by Eric Matthes. For detailed instructions on installing Python on other operating systems, please refer to the appropriate section (Appendix A) of that text.

1. Open a terminal. Enter **python3**. If you get a message about installing *command line utilities tools*, then you do not have python. If you do not have *python3* or later, you will need to install it. Otherwise, you can skip the next step.

2. Go to *https://python.org* and download the latest version of Python. Run the installer. After installation, a finder window will open. Double-click *install Certificates.command*. This will allow you to install additional libraries more easily when necessary.

2.2.2 Running python from the command line exercise

Now let's try to run some python code.

1. Open a terminal on your PC. Type **python3**. This should start a python terminal session. A python prompt should appear:

   ```
   >>>
   ```

2. If it does not, then you may need to double-click on the file *Update Shell Profile.command* that is located in your newly installed Python directory. As explained in the first few lines of this file (which can be examined using a text editor), double-clicking this file executes a script that will update your shell profile when the 'bin' directory of python is not in the path of your shell. Open a new terminal and type **python3** again; a python prompt should now appear:

   ```
   >>>
   ```

3. Enter the following text:

   ```
   >>> print("Hello Python interpreter!")
   ```

The python interpreter should echo back to you the text contained between your quotation marks. If so, congratulations! You have successfully installed the python interpreter!

2.3 Interactive development environment (IDE)

In this course, you will be running most of your python programs using an IDE (interactive development environment). An IDE combines a text editor with other tools that are useful for software development, such as an interpreter, a debugger, various automation tools, and a terminal for interacting with your operating system. One example of an IDE is *VS Code*; another is *PyCharm*; a third is *Thonny*. We will plan to use *Thonny* in this course.

2.3.1 Installing Thonny exercise

1. Navigate your web browser to *thonny.org* and download the latest version of *Thonny* for your operating system.

2. If you are using macOS, use the package manager to install the software.

3. Start up *Thonny* by double-clicking on the application icon.

4. In the *Thonny* editor, which is presently untitled, enter the following text:

   ```
   print("Hello Thonny User!")
   ```

5. Save this file as *hello_thonny.py* in an appropriate directory. At this point, it is important to start thinking about how you will orgainze your files that you generate in this course. You will have many. So you may wish to create a dedicated directory called *src_py*, or something to that effect, in which to put your python source code.

6. Run your code by selecting

```
Run > Run Current Script
```

The result of running your code should appear in the window titled *Shell*.

Congratulations! You have successfully installed *Thonny* and used it to create and run some python code. Now let's write some more sophisticated code.

Chapter 3

Basic Python Programming

3.1 Variables and simple data types

In the following exercises, you will learn a bit about how to use variables and other simple data types in *Python*. The exercises listed in this section can be found in the gray "try it yourself" boxes of Eric Matthes' book titled *Python Crash Course, Ed.3*, henceforth referred to as PCC. If you do not know how to do the exercises, as you probably will not if you don't already know *Python*, then you should read the pages that precede each exercise.

3.1.1 Simple messages exercise

Read PCC pages 16-19 and do exercise 2-2: assign a message to a variable, and print that message. Then change the value of the variable to a new message, and print the new message.

3.1.2 Name cases exercise

Read PCC pages 19-25 and do exercise 2-4: use a variable to represent a person's name, and then print that person's name in lowercase, uppercase, and title case.

3.1.3 Name eight exercise

Read PCC pages 26-29 and do exercise 2-9: write addition, subtraction, multiplication, and division operations that each result in the number 8. Be sure to enclose your operations in *print()* calls to see the results. Your output should be four lines, with the number 8 appearing once on each line.

3.1.4 Names exercise

Read PCC pages 34-36 and do exercise 3-1: store the names a few of your friends in a list called *names*. Print each person's name by accessing each element in

the list, one at a time.

3.1.5 Guest list exercise

Do exercise 3-4: If you could invite anyone, living or deceased, to dinner, who would you invite? Make a list that includes at least three people. Then use your list to print a message to each person, inviting them to dinner.

3.1.6 Seeing the world exercise

Do exercise 3-8.

3.1.7 Animals exercise

Do exercise 4-2: Think of at least three different animals that have a common characteristic. Store the names of these animals in a list, and then use a for loop to print out the name of each animal. Modify your program to print a statement about each animal, such as *A dog would make a great pet.* Add a line at the end of your program , stating what these animals have in common. You could print a sentence, such as *Any of these animals would make a great pet!*

3.1.8 Cubes exercise

Do exercise 4-8: A number raised to the third power is called a *cube*. For example, the cube of 2 is written as $2 * *3$ in *Python*. Make a list of the first 10 cubes, and use a *for* loop to print out the value of each cube.

3.1.9 Buffet exercise

Do exercise 4-13: A buffet-style restaurant offers only five basic foods. Think of five simple foods, and store them in a *tuple*. Now use a *for* loop to print each food the restaurant offers. Try to modify one of the items, and make sure that *Python* rejects the change. Now the restaurant changes its menu, replacing two of the items with different foods. Add a line that rewrites the *tuple*, and then use a *for* loop to print each of the items on the revised menu.

3.2 Putting it all together

We have just spent considerable time learning how to do basic *Python* programming. Now let's try to apply what we've learned. We will write a program that converts a bunch of Fahrenheit temperature values to Celsius temperature values.

3.2.1 Temperature Conversion exercise

1. Write a program that generates 100 fahrenheit-celsius pairs starting at 0 degrees fahrenheit and going up to 300 degrees fahrenheit. Hint: your program should use a loop and should compute the value of celsius that corresponds to each value of fahrenheit.

2. Modify the program so that it prints a nice header above the table. The header should be justified so that it lines up with the numbers.

3. Now modify the program so that in addition to printing the Fahrenheit and Celsius temperatures, it also prints the temperature in Kelvin. Modify the header accordingly.

4. Change your program so that it uses floating-point arithmetic and prints out the results to one place behind the decimal point. Also: see if you can make it really pretty by aligning the columns of numbers on the decimal points.

Congratulations! You have come a long way in learning how to write *Python* programs run by your personal computer. In the next chapter, we will learn how to set up and program an ESP32 WROVER microcontroller using *Python*. This will allow us to control external devices, such as LEDs, thermometers, heaters, and just about anything that operates using an electronic signal.

Chapter 4

Introduction to Microcontrollers

In the previous chapter, we worked through a number of exercises with the goal of learning how to do basic *Python* programming. These exercises were taken largely from the book *Python Crash Course*. In the present chapter, we will learn how to set up the ESP32 WROVER microcontroller.

4.1 ESP32 microcontroller setup

Your instructor should provide you with a Freenove Basic Starter Kit for ESP32. Let's get this working.

4.1.1 Downloading ESP32 tutorial package exercise

1. The documentation, tutorials, and sample code for the ESP32 are all available as a zipped file by clicking here: Freenove Basic Starter Kit for ESP32. Download this package to your PC and place it in an appropriate directory for this class.

2. The ESP32 can be programmed in either C or micro-python. Since we will be programming in *Python*, you should open the tutorial that is found in the *Python* directory of your downloaded package. You may wish to print a hard-copy of the *Python* tutorial for easy future reference.

3. Read the first seven pages of the *Python* tutorial. Open the starter kit when you get to the section describing the ESP32-WROVER and take a look at it. Be sure you can identify the ESP32-WROVER, the extension board, and the protoboard.

Since we previously installed Thonny on your PC, some of the work in Chapter 0 of the *Python* tutorial is already completed. Nonetheless, it is worthwhile look-

ing through this section to make sure that your installation is correct. In particular, you may wish to look at the section on Basic Configuration of Thonny. Most importantly, in order to run *Python* programs directly on the ESP32, we will need to flash firmware to the ESP32. The instructions for downloading and installing the firmware described on page 26 of the tutorial.

4.1.2 Flashing firmware to ESP32 exercise

1. Connect your PC to the ESP32 micro controller using a USB cable. Check to be sure that the ESP32 device can be identified. On macOS, this can be done by inspecting the USB hardware in *System Information* . If the device cannot be recognized, then you may need to install the CH34x usb to serial device driver, which is included with the ESP32 package you downloaded. All this is explained in the file CH34X_drv_instal_instructions. After installing the driver, on macOS you can check that the device drivers are working properly by typing

   ```
   >>> ls /dev/tty*
   ```

   in the command line of a terminal. In the terminal, you should see listed

   ```
   /dev/tty.wchusbserial21420
   ```

   Note: the above procedure will not work if you are using a Windows machine. If you are using a Windows machine, you will need to use the *Device Manager* to find the correct COM port (usually COM4). You may also need to change the default connection speed from 9600 bits/second to 115200 bits/second. This can be done using the *Port Settings* tab of the Device Manager.

2. When you are sure that you can use your PC's USB port, open Thonny. The Thonny IDE should display the files that are on your personal computer (this computer) and also on your ESP32 WROVER (the micropython device). Thonny should also display an "untitled" window that can be used to generate and edit files, and also a "shell" window that acts as a terminal for running *Python* commands.

3. Next, follow the firmware installation instructions provided in the manual. In short, you should begin by selecting as the interpreter: Micropython (ESP32). This implies that any code you run in the Thonny shell will be interpreted/executed by the connected ESP32 device, rather than by your local PC, as you had been doing in the past. You will also need to set the port correctly. For example:

   ```
   /dev/cu.wchusbserial21420
   ```

Once again, the above procedure will not work if you are using a Windows machine; the appropriate COM port must be selected.

Then you will need to flash the firmware on your ESP32 device. Be sure to select the correct target port (wchusbserial) and MicroPython family (ESP32) and variant (Espressif ESP32). When you finally click "install", it should take a few minutes to write the firmware to the ESP32 device.

4. Test the shell command by typing "print('hello world')" in the shell window. Press enter and verify that the code runs on your ESP32 device. Congratulations! Now you can run *Python* code using either the ESP32 or your PC, depending on what you choose as your interpreter.

4.2 Using the ESP32 online and offline

When you downloaded the Freenove Basic Starter Kit for ESP32, you downloaded a number of *Python* sample programs. These can be found in the newly downloaded *Python* Codes subdirectory on your PC. In the following two exercises, found on pages 32 - 38 of the tutorial, you will learn how to run code both online and offline.

4.2.1 Testing code online exercise

1. After starting Thonny, open the *HelloWorld.py*, script which is located in the *Python* Codes directory that you downloaded to your PC.
2. Run the script by clicking Thonny's green arrow button.

Note that if you press the reset key of ESP32, the code will not be executed again. If you want to have the code run each time you press the reset key of the ESP32, you will need to run the script offline.

After the your press the reset button, the device first automatically run the file *boot.py*, which was placed in the root directory of the ESP32 when you installed the firmware. It then runs *main.py*. Finally, it enters "shell". In order to execute a user's program after pressing the reset button, we need to upload a different *boot.py* file than the one that is installed already. Let's do that.

4.2.2 Testing code offline exercise

1. Make a copy of the *Python* Codes directory on your PC. Then if we modify files, we will still have the original unmodified files.
2. Expand the *00.1_Boot* subdirectory in the *Python* codes directory on your PC. Double click on the *boot.py* file that is located in this subdirectory. The file should open so you can look at the code. This a modified version of the original *boot.py* file; it is modified so as to allow programs to run offline.

3. Upload this modified *boot.py* file to your MicroPython device. This new boot file should appear on the MicroPython device
4. Now upload *HelloWorld.py* to the MicroPython device.
5. Press the reset key on the device; see that the code is executed each time you press the reset key on your device. This should be the case for any code you now load onto the device.
6. Delete the *HelloWorld.py* file from your microcontroller. Now select *Blink.py* in the *01.1 _Blink* directory on your PC. Upload this file to the microcontroller's root directory. Press the reset button on your device and see the blue led on the device begin to blink. Do it again.
7. Finally, delete the *Blink.py* file from your microcontroller. Press the reset button on the device and see the blinking stop.

Perhaps it is not clear to you that the ESP32 device is running code all by itself. This exercise will demonstrate that it is by removing the USB cable altogether.

4.2.3 "Cutting the cord" exercise

1. Using Thonny, open the file *Blink.py* on your PC. Using the "file" tab, save the *Blink.py* file to your microcontroller device as *main.py*.
2. Use a 9-volt battery to power your ESP32 WROVER microcontroller and disconnect the USB cable entirely.
3. Press the reset button and watch the LED blink. This should demonstrate that any code you save to the microcontroller as *main.py* will be run even if not attached to your computer so long as the microcontroller has power.

Note: *Thonny* should display the files that are located on your PC and the files that are located on your Microcontroller in different panes or file browsers of the IDE. It it does not, then you may need to add the file browser tool to Thonny. To add it go to View ¿ Files and you'll see two panes: the top pane shows your PC files and the bottom one shows the files stored on the microcontroller. Also note that in the filename tab, square brackets [filename.py] indicate that the file is located on the microcontroller. No brackets means it is stored on your PC.

4.3 Reading assignment

To prepare for what comes next, your should read through the end of Chapter 0 of the tutorial. This will give you a sense of what the various pins of the ESP32 can do. Also, you should begin to read Chapter 1. In the next chaper, we will learn more about how to program the ESP32. Keep in mind that our goal is to use this device to facilitate doing scientific experiments.

Chapter 5

Digital output

In the previous chapter, we learned how to program the ESP32 mircrocontroller to carry out simple commands. In the present chapter we will learn how to build elementary circuits on the protoboard (on which the ESP32 microcontroller is mounted) that allow the microcontroller's digital output ports to perform basic tasks, such as turning on and off an LED. In the next chapter, we will learn how to use the analog input ports of the ESP32 to read voltages from external devices. Remember that we are working up to a point where we can carry out a scientific experiment involving the measurement of the thermal conductivity of a rod of copper. As you do the following exercises, you might start pondering how such a task might be accomplished.

5.1 Number Systems

Although we write programs in high-level languages such as Python, which look more or less like English, the computer translates these programs into its own language, called machine code. Machine code consists of just ones and zeros.[1] A number system consisting of just ones and zeros is called a binary number system. The hexadecimal number system provides a shorthand notation to represent large binary numbers. It will be very useful to become acquainted with binary and hexadecimal number systems and their relationship to our decimal number system with which we are more accustomed.[2]

The decimal number system is so called because it uses base 10. A number such as 293 can be described by the equation

[1]At the level of hardware, a one or a zero corresponds to a tiny device–such as wire or a capacitor–acquiring a high or a low voltage. For purposes of this course, we can largely overlook the physical implementation of digital ones and zeros.

[2]The ancients used alternative numbering systems. For example, we inherited the sexagesimal (base-60) system of recording both time and small angles from the ancient Babylonians.[3] Thus, there are sixty minutes in an hour, sixty seconds in a minute, sixty thirds in a second, sixty fourths in a third, *etc.*.

$$293 = 2 \times 10^2 + 9 \times 10^1 + 3 \times 10^0 \tag{5.1}$$

That is: 2 one-hundreds plus 9 tens plus 3 ones. The binary number system is so called because it uses base 2 instead of base 10. This means that a number such as 23 can be described in the binary number system by the equation

$$23 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \tag{5.2}$$

That is: 1 sixteen plus 0 eights plus 1 four plus 1 two plus 1 one.

5.1.1 Binary numbers exercise

1. Write the following decimal numbers in binary: 12, 18, 39, 241.
2. Write out a binary multiplication table from binary 000 to binary 111. Hint: this should be an 8 by 8 table.
3. Challenge: what type of logical operation(s) must be used in order to multiply binary numbers? (bitwise AND? OR? XOR? or something else?)

The hexadecimal number system is so called because it uses base 16. In Tab. 5.1 are shown correspondences between some binary, hexadecimal, and decimal numbers. Whereas the 'nice round numbers' in decimal are multiples of 10: $10^0 = 1$, $10^1 = 10$, $10^2 = 100$, and $10^3 = 1000$, the 'nice round numbers' in binary are multiples of 2: $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, $2^3 = 8$, $2^4 = 16$, $2^5 = 32$, $2^6 = 64$, and $2^7 = 128$. Similarly, the 'nice round numbers' in hexadecimal are multiples of 16, such as $16^0 = 1$, $16^1 = 16$, $16^2 = 256$, and $16^3 = 4096$. Also, notice in Tab. 5.1 that every four binary digits corresponds to one hexadecimal digit. It is thus customary to group binary numbers in clusters of four digits. For example, the hexadecimal number 2B5 is written in binary as 0010 1011 0101. Often, to denote that a number is in hexadecimal notation, a 0x is place before it. In this notation, we would write the previous hex number as 0x2B5.

5.1.2 Hexadecimal numbers exercise

1. Write the following decimal numbers in binary and hexadecimal notation: 7, 42, 826.
2. Write the following hexadecimal numbers in decimal and binary notation: 0x03, 0xA7, 0x3BF1

5.1.3 Number systems in history exercise

1. Who invented the decimal number system? In what century was it introduced to Europe, and by whom? Write down the sources of your answers.
2. What base does the Roman numeral system use?
3. Why might the ancient Babylonians have used a base 60 numbering system?

binary	hexadecimal	decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
0001 0000	10	16

Table 5.1: Comparison of binary, hexadecimal, and decimal representation of numbers.

5.2 Using the ESP32 digital output ports

Why are we learning about binary and hexadecimal number sytems? As it turns out, writing a binary value to a particular digital output port of the ESP32 board allows us to set the voltage of that port HI (digital one) or LO (digital zero). Multiple digital output ports can be simultaneously set to a desired value by writing a string of digital values to a series of digital output ports. For example, writing the binary number 1110 can be used turn on the first three digital output ports and turn of the last one. We will come back to this later. In the mean time, let's do a few simpler exercises.

In these exercises, you will be asked to set up some circuits, download code to your micropython device, and run the code. Make an attempt to understand how the code works, but don't worry too much if you don't understand all the details. Some of the code uses loops and conditional statments. We'll come back and learn more about these topics in the next chapter.

5.2.1 Blink LED exercise

Read pages 54 - 61 in the freenove python tutorial. Carry out Project 1.2: Blink.

5.2.2 Button and LED exercise

Read pages 62 - 68 in the freenove python tutorial. Carry out Project 2.1 Button and LED.

5.2.3 Mini table lamp exercise

Read pages 69-73 in the freenove python tutorial. Carry out Project 2.2: Mini table lamp

5.2.4 Flowing light exercise

Read pages 74-79 in the freenove python tutorial. Carry out Project 3.1 Mini table lamp

5.2.5 Breathing LED exercise

Read pages 80-86 in the freenove python tutorial. Carry out Project 4.1: Breathing LED

5.2.6 Meteor flowing light exercise

Read pages 87- 92 in the freenove python tutorial. Carry out Project 4.2: meteor flowing light

5.2.7 RGB LED exercise

Read pages 93-98 in the freenove python tutorial. Carry out Project 5.1: random color light

5.2.8 Gradient LED exercise

Read pages 99- 100 in the freenove python tutorial. Carry out Project 5.3: gradient color light

5.2.9 Switching speed exercise

1. Write a program that flips an LED on and off once per second. Put a printout of the working program in your notebook.

2. Modify your program so that the switching speed is 500 Hz.

3. Plug in the BNC cable with the oscilloscope probes into CH 1 of the oscilloscope. Hook up the probes across the diode and observe the diode voltage as a function of time. The ground lead of the oscilloscope probe should be connected to the ground of the system, *i.e.* the lower voltage of the LED. Never connect the oscilloscope probe ground to any point of a circuit which is not ground. Set the scope trigger control to AUTO; be

sure the small switch on the probe tip is set to 1x; set the vertical scale to 1.0 V/DIV and the VARIABLE knob to the CALibrated position; set a 0 V baseline by using the ground switch and vertical position knob on the scope. Set the horizontal scale to 1 ms TIME/DIV. You may need to adjust the intensity, focus, position, and TRIG LEVEL to observe a nice square wave.

4. What is the highest frequency square wave that you can generate? Hint: You may want to just omit the delay statements in your program.

Chapter 6

A Little More Python Programming

In Chap.3, we learned the basics of how to program in python. In particular, we went through a number of exercises from *Python Crash Course*, that taught us about variables and simple data types, such as integers and lists. We even learned how to write a program that converts Fahrenheit to Celsius temperatures. We then proceeded, in Chap.5, to write some python programs that allowed us to control the digital output ports of the ESP32 microcontroller. Well, not exactly. Technically, we copied and pasted code that somebody else wrote. Some of this code was probably a bit opaque because you were not familiar with some of the functions and data types that these programs employed. In this chapter, we will step back, so to speak, and learn more about python programming so we can intelligently write and use our own code. To do so, we will go through a few more exercises from *Python Crash Course* (PCC). You should read the sections of text that precede the prescribed exercises and then carry them out. Remember to write down what you did in your lab book!

6.1 If statements

The next few exercises are from Chapter 5 of PCC.

6.1.1 Conditional tests exercise

Do exercise 5-1: write a series of conditional tests. Print a statement describing each test and your prediction for the results of each test.

6.1.2 Stages of life exercise

Do exercise 5-6: Write an if-elif-else chain that determines a person's stage of life. Set a value for the variable age and print an appropriate message.

6.2 User input and while loops

The next few exercises are from Chapter 7 of PCC.

6.2.1 Multiples of 10 exercise

Do exercise 7-3: Multiples of ten: Ask the user for a number, and then report whether the number is a multiple of 10 or not.

6.2.2 Movie tickets exercise

Do exercise 7-5: Movie tickets. A movie theater charges different ticket prices depending on a person's age. If a person is under 3, the ticket is free; it they are between 3 and 12, it is 10 USD; if they are over 12, the ticket is 15 USD. Write a loop in which you ask users their age, and then tell them the cost of their ticket.

6.3 Functions

The next few exercises are from Chapter 8 of PCC

6.3.1 Message exercise

Do exercise 8-1: Message: write a function called display_message() that prints one sentence telling everyone what you are learning about in this chapter. Call the function, and make sure the message displays.

6.3.2 T-shirt exercise

Do exercise 8-3: T-Shirt: write a function called make_shirt() that accepts a size and the text of a message that should be printed on the shirt. The function should print a sentence summarizing the size of the shirt and the message printed on it. Call the function using positional arguments to make a shirt. Call the function a second time using keyword arguments.

6.3.3 City names exercise

Do exercise 8-6: City names: write a function called city_country() that takes in the name of a city and its country. The function should return a formatted string containing the city, country pair. Call your function with at least three city-country pairs, and print values that are returned.

6.3.4 Messages exercise

Do exercise 8-9: messages: make a list containing a series of short text messages. Pass the list to a function called show_messages(), which prints each text message.

6.3.5 sandwiches exercise

Do exercise 8-12: sandwiches: Write a function that accepts a list of items a person wants on a sandwich. The function should have one parameter that collects as many items as the function call provides, and it should print a summary of the sandwich that's being ordered. Call the function three times, using a different number of arguments each time.

6.3.6 imports exercise

Do exercise 8-16: imports: Using a program you wrote that has one function in it, store that function in a separate file. Import the function in to your main program file, and call the function using several approaches (see PCC pg. 154).

6.4 Classes

The following exercises are designed to introduce you to the concepts of object-oriented-programming. You will write *classes* that describe real-world things, and you will create *objects* based on these classes. Since this topic is a bit ore abstract, you are highly encouraged to read the text of PCC preceding each exercise.

6.4.1 users exercise

Do exercise 9-3: users: Make a class called User. Create two attribues called first_name and last_name, and then create several other attributes that are physically stored in a user profile. Make a method called describe_user() that prints a summary of the user's information. Make another method called greet_user() that prints a personalized greeting to the user. Create several instances representing different users, and call both methods for each user.

6.4.2 login attempts exercise

Do exercise 9-5: login attempts: Add an attribute called login_attempts to your User class from Ex.6.4.1. Write a method called increment_login_attempts() that increments the value of login_attempts by 1. Write another method called reset_login_attempts() that resets the value of login_attempts to 0. Make an instance of the User class and call increment_login_attempts() several times. Print the value of login_attempts to make sure it was incremented properly, and

then call reset_login_attempts(). Print login_attempts again to make sure it was reset to 0.

6.4.3 admin exercise

Do exercise 9-7:Admin: An administrator is a special kind of user. Write a class called Admin that inherits from the User class you wrote in Ex.6.4.2 Add an attribute, privileges, that stores a list of strings like "can add post", "can delete post", "can ban user", and so on. Write a method called show_privileges() that lists the administrator's set of privileges. Create an instance of Admin, and call your method.

6.4.4 privileges exercise

Do exercise 9-8: privileges: Write a separate Privileges class. The class should have one attribute, privileges, that stores a list of strings. Move the show_privileges() method to this class. Make a Privileges instance as an attribute in the Admin class. Create a new instance of the Admin and use your method to show its privileges.

6.4.5 imported admin exercise

Do exercise 9-11:imported admin: Start with your work from Ex.6.4.4. Store the classes User, Priviliges, and Admin in one module. Create a separate file, make an Admin instance, and call show_privileges() to show that everything is working correctly.

6.5 Files and Exceptions

Let's learn about one more topic: files. Then we will re-write our Fahrenheit to Celsius conversion program using some of the stuff we've learned. Start by taking a look at the tutorial in the official python documentation: `https://docs.python.org/3/tutorial/inputoutput.html#tut-files`. Then complete the following exercises which use the *open()* command.

6.5.1 learning python exercise

Learning python: Open a blank file in your text editor and write a few lines summarizing what you've learned about python so far. Start each line with the phrase In Python you can... Save the file as learning_python.txt in the same directory as your exercises from this chapter. Write a program that reads the file and prints what you wrote two times: print the contents once by reading in the entire file, and once by storing the lines in a list and then looping over each line.

6.5.2 guest book exercise

Guest book: Write a while loop that prompts users for their name. Collect all the names that are entered, and then write these names to.a file called guest_book.txt. Make sure each entry appears on a new line in the file.

6.5.3 common words exercise

Common words: Copy some selected text from Project Gutenberg into a text file on your computer. You can use the count() method to find out how many times a word or phrase appears in a string. Write a program that reads your text file and determines how many times the word "the" appears in the text. This will be an approximation because it will also count words such as "them" and "then". Try counting "the ", with a space in the string, and see how much lower your count is.

6.5.4 favorite number exercise

Favorite number: write a program that prompts for the user's favorite number. Use json.dumps() to store this number in a file. Write a separate program that reads in this value and prints the message "I know your favorite number! It is ___."

6.6 Putting things together

In Ex.3.2.1, you wrote a program that converts a number of Fahrenheit temperature to Celsius temperature. In this section you will use what you have learned to write a more sophisticated version of this program.

6.6.1 Temperature conversion exercise

Write a program that prompts the user for a Fahrenheit temperature, then returns the value in Celsius. The returned value should be printed to the screen and also stored in a list. When the user is finished entering data, all of the converted values should be stored in a convenient (*i.e.*, space delimited or comma delimited) format in a text file.

Chapter 7

A/D Conversion

Natura non facit saltus.

7.1 Analog and digital signals

Most processes in nature change in a continuous fashion. For example, the color of a rainbow gradually fades from red to violet; the pitch of a human voice gradually drops from high to low; and the temperature of a cup of tea continuously falls from hot to cold. Such natural processes are said to be “analog”: they do not change in discrete steps, or jumps. In fact, until the development of quantum theory in the early 20th century, this axiom was taken by many to be an essential feature of nature.[1]

Computers, however, are not natural; they are artificial. And as described in Chap.5, most modern computers operate using digital signals: sequences of ones and zeros. These signals are said to be digital, as opposed to analog, in the sense that there is a discontinuous jump between a one and a zero. Why do computers use digital signals? Primarily, because digital data is robust; it is easier to store and to manipulate than analog data.

Now in order for a digital computer to store or manipulate information about analog processes, it must convert analog information into digital information. This is accomplished using two basic devices: an *electronic transducer* and an *analog to digital converter*. The transducer first converts some analog value–such as light color, air pressure, or temperature–into an analog voltage. For example, a photodiode detects light intensity variations and converts these into voltage variations. A microphone detects air pressure variations and converts these into voltage variations. A thermistor detects temperature variations and converts these into voltage variations.

[1] This axiom that “nature does not make jumps” informed the work of Gottfried Wilhelm Leibniz in developing the method of infinitesimal calculus and also Charles Darwin in developing his theory of common descent.

The analog-to-digital converter (ADC), then, converts the analog voltage output of the transducer into an integer that can be represented digitally: as a sequence of ones and zeros that your computer can store and manipulate. Your ESP32 microcontroller has two 12-bit ADCs (more on this in a moment). These two ADCs support measurements of analog voltage on 18 different channels (analog-enabled pins). In the following exercises, we will build circuits using transducers and learn how to use the analog to digital converter of your ESP32. First, however, let us learn a bit more about two important measurement issues: resolution and sampling speed.

7.2 Resolution and Sampling Speed

There are two main issues when performing analog to digital conversion: *resolution* and *sampling speed*. Let's talk about resolution first. Consider a measuring device such as a meter stick. On a meter stick, the larger the number of ticks drawn between zero and one meter, the higher the resolution with which the meter stick may be read. For example, if the meter stick has 100 ticks, then the meters stick has a resolution of one centimeter; if the meter stick has 1000 ticks, then the meters stick has a resolution of one millimeter. It has a higher resolution.

The number of ticks drawn on a meter stick is akin to the number of bits in the analog input register of the ADC. The larger the number of bits, the greater the resolution of the ADC. For example, a 12-bit ADC has 2^{12} or 4096 "ticks". This means any analog voltage value it reads must be assigned to a number between 0 and 4095. This implies that if the analog to digital converter accepts a range of voltages between 0 to 10 volts, then the resolution of that ADC is 10volts/4096 = 0.0024 volts. Any variation in the ADC input voltage that is smaller than 2.4 mV will thus not be detected by the 12 bit ADC.

7.2.1 Dynamic range exercise

1. The resolution in voltage of an ADC is $\Delta V/2^n$ where ΔV is the total input range and n is the number of bits of the digital output. What is the resolution of an 8-bit ADC if it has an input range +5 to -5 V?

2. Since the amplitude of an analog signal can be adjusted by an amplifier circuit to fill the input range of the ADC, the resolution can be better described by the dynamic range; this is the ratio of the maximum to the minimum voltage measurable by the ADC. What is the dynamic range of the above-mentioned 8-bit ADC? What about the one we are using in class? By the way, the dynamic range (DR) is often expressed in decibels (dB); that is DR = 20 log (ratio) in dB. Give your answers in both forms, as a ratio and in dB.

Now let's talk about sampling speed. A complication arises when the analog voltage signal is changing–perhaps rapidly. An ADC requires some "settling

time" in which to perform the analog to digital conversion. If the analog voltage signal changes too quickly, then the ADC will not be able to accurately track the changing signal.

There is a famous result known as the Sampling Theorem, formulated by Shannon (1949), building on earlier work by Nyquist (1924), which states that in order to reconstruct the original signal from a sampled signal accurately, the ADC sample rate should be at least twice the highest frequency in the input signal. This is called the *Nyquist Frequency*. If sampling is performed at a lower rate than the Nyquist frequency, then the higher frequency components of the signal can masquerade, so to speak, as lower frequency components in the digital recording. This is an effect called *aliasing*.[2] An example of aliasing with which you may be familiar is when a spinning automobile wheel is filmed at 30 frames per second. If the sampling speed (the rate at which images are captured by the camera, in this case) is too small, then the wheel may curiously appear to be rotating backwards when viewing the film recording.

7.2.2 Digital recording studio exercise

Analog to digital converters are used quite often to store musical recordings. A high-fidelity digital recording should represent the true analog signal as faithfully as possible. This way, when we use a digital to analog converter (more on this later) to reproduce an analog signal from digitally stored data, it will (in the case of a sound recording) sound just like the original signal.

1. Suppose a digital recording studio wants to faithfully record the audio spectrum from 20 to 20,000 Hz. What must be the sample rate so as to avoid aliasing?

2. What is the maximum conversion time the studio's ADC can have?

7.3 Digital to Analog Conversion

The inverse of analog to digital conversion is, you guessed it, digital to analog conversion. Digital to analog conversion is routinely used to convert digitally stored data into music or your favorite movie. Your ESP32 microcontroller has two digital to analog converters. The analog output pins, GPIO25 and GPIO26, are different than the previously used digital output pins in that the digital output pins can only take on just two discrete voltage values (hi and lo). The analog output pins, on the other hand, can take on a number of voltage values that depends on the resolution of the digital to analog computer. The ESP32's DAC have 8-bit resolution. This means that our DACs can each output 2^8 distinct output voltages.[3]. Digital to analog conversion is fairly straightforward.

[2]In order to guarantee that no higher frequency signal is being inadvertently sampled by the ADC is to place a low-pass filter between the transducer and the ADC itself.

[3]If you are a careful thinker, you may protest that the output of a DAC is not, technically, analog. You would be correct.

First, a number corresponding to the desired voltage is sent to one of the analog output channels. Next, the analog output channel converts this into a voltage. Wallah.

7.3.1 Analog output exercise

1. What output voltages (and on what channels) would result by sending the following binary integers to the following analog output channels? Hint: what are the minimum and maximum output voltages?

case	GPIO25	GPIO26
1	1101 0001	0000 1000
2	0000 0000	0000 0000

7.4 Putting it together

The following exercises involve setting up a circuit and using both the ESP32's ADC and DAC. To prepare, you should carefully read pages 115 - 123 of the tutorial. The first exercise involves using a rotary potentiometer. What is a potentiometer?

7.4.1 Potentiometer exercise

In short, a potentiometer is a type of electrical resistor. Most resistors have two terminals; when an electrical current is passed through the resistor, a voltage difference develops between the two terminals of the resistor. The relationship between the electrical current, I, the resistance, R, and the voltage difference, V, is given by Ohm's law:

$$V = IR \tag{7.1}$$

A potentiometer is a three-terminal resistor. The additional terminal is adjustable, in the sense that it has a sliding contact point between the other two terminals. This additional terminal, then, will have a voltage value somewhere between that of the other two. It acts as a so-called *voltage divider*.

In a rotary potentiometer, the particular location (and hence the voltage value of) the sliding contact point can be easily adjusted by rotating a small knob.[4]. In what follows, we will attach two first two terminals of a rotary potentiometer to 3.3 volts and ground, respectively. We will use the ESP32's ADC to read the voltage of the third terminal. This voltage will of course vary as we rotate the knob of the potentiometer. We will then use this reading to inform an LED how brightly to glow.

[4]In a so-called *rheostat*, the location of the contact point along a large coil of wire can be adjusted by moving a slider.

1. Connect a potentiometer across a known (e.g. 3.3 or 5) volt power supply and observe the voltage of the wiper (the center connection on the potentiometer) using a multimeter or an oscilloscope. Be sure you attach the ground of the scope to the same ground as that of your protoboard. (You should never connect the oscilloscope probe ground to any point of a circuit which is not grounded).
2. Now set up the complete circuit as shown on page 119 - 120 and load the code 08.1_AnalogRead from the Python_Codes directory on your computer onto your ESP32 microcontroller.
3. Run the code and describe what happens as you turn the wiper. Make a printout of the code, place it in your lab book, and make sure you understand how it works. How might you modify the code so that it uses a different input pin?

7.4.2 Digital signal processing exercise

1. Use a function generator to apply a sinusoidally varying voltage to one of the analog input channels of your ESP32. Write a program that sample the voltage over a reasonably long time interval. Save the data in an appropriately named file. What is the highest frequency sine wave that you can reliably sample?
2. Now write a program that generates an output voltage of a desired frequency. Use the oscilloscope to monitor the analog output channel. What is the highest frequency sine wave that you can reliably play?

7.4.3 Soft light exercise

Instead of using the DAC to control the brightness of an LED, we can use a digital output channel. That is what we will do in the following exercise:

1. Build the circuit shown on page 134 of the tutorial.
2. Load the code 10.1_Soft_LED onto your microcontroller and run the script.
3. Make a printout of the code and explain how it works.

7.4.4 Nightlamp exercise

Next, we will use the ESP32's ADC to read the voltage across the terminals of a photoresistor (a transducer the senses the brightness of light). Then we will use a digital output port to control the brightness of an LED.

1. Read pages 135-139 of the tutorial and build the circuit on page 138.
2. Load the code 11.1_Nightlamp onto your microcontroller and run the script.
3. Explain how the code works.

Chapter 8

Temperature Measurement and Control

8.1 Introduction

You are probably familiar with how a thermostat works in your home: you assign a temperature (say, 68 degrees). A thermometer reads the temperature of the house. If the temperature is too low, a heater is turned on until the temperature rises above the target temperature, at which point the heater is turned off. If the temperature subsequently falls below the target, the heater is turned back on.

In this chapter, we will build a thermostat that regulates the temperature of an aluminum block at a desired temperature. We will begin by learning how to build a heater. Then we will learn how to use a thermistor, which is a kind of electronic thermometer. Finally, we will put this together to build a thermostat. Let's begin by talking a bit about Joule heating.

8.2 Joule heating

When an electrical current is passed through a wire, the wire tends to heat up. This amount of heat energy generated per second by the electrical current is governed by the so-called *Joule heating* formula

$$P = I^2 R \tag{8.1}$$

where P is the power, typically measured in Joules per second (or Watts), I is the electrical current, in Amperes, and R is the electrical resistance of the wire, in Ohms. So if we know the current and the resistance, we can calculate the heating power. Generally speaking, for a given current, the larger the resistance, the greater the heating.

Suppose, on the other hand, that we don't know the electrical current. Instead, we use a battery with a known voltage to drive a current through the heater wire. We can still calculate the heating power by substituting Ohm's law, $I = V/R$ to obtain

$$P = \frac{V^2}{R}. \tag{8.2}$$

8.2.1 Joule heating exercise

Suppose a manganin wire has a resistance of 8 ohms. The ends of the wire are attached to a 12 volt power supply.

1. How much electrical current flows down the wire?
2. How many watts of heat are generated by the wire?
3. What do you think would happen if the power supply can only supply a maximum current of 1 ampere?

8.3 Wire gauge and electrical resistivity

The resistance of a wire, such as the one considered in Ex.8.2.1, depends on the type of material, the length of the wire, and the cross-sectional area of the wire. This relationship is given by the formula

$$R = \frac{\rho L}{A} \tag{8.3}$$

where R is the resistance (in Ohms), ρ is the volume resistivity of the material (in Ohm-meters), L is the length of the wire (in meters), and A is the cross-sectional area of the wire (in square meters). The cross-sectional area of a wire depends on its diameter of the wire, and the diameter is specified by the gauge of the wire.

8.3.1 Resistivity exercise

Consider the manganin wire in Ex.8.2.1 that has a resistance of 8 ohms. Suppose that the wire is 30-gauge wire ("American Wire Gauge" or AWG 30).

1. Look up the diameter of AWG 30 wire. What is the cross sectional area of this wire?
2. What is manganin wire made of? Look up the electrical resistivity, ρ, of manganin wire.
3. How long must his wire be so as to have a resistance of $R = 8$ Ohms?

8.4 Heat capacity and thermal response

When a heater wire is wrapped around an object, such as an aluminum block, the temperature of the object rises. How quickly does it rise? It depends on the heating power, the mass of the object, and the type material out of which the object is made. Consider the fundamental equation of calorimetry:

$$\Delta Q = mc\Delta T \tag{8.4}$$

This formula relates the amount of temperature rise of an object, ΔT, to the amount of heat deposited, ΔQ, the mass of the object, m, and the specific heat of the object, c. Recall that the heating power, P, is the amount of heat energy generated, ΔQ, in a given time interval, Δt. Namely,

$$P = \frac{\Delta Q}{\Delta t} \tag{8.5}$$

8.4.1 Calorimetry exercise

1. Combine Eq.8.4 and Eq.8.5, to show that the rate of temperature rise can be given by

$$\frac{\Delta T}{\Delta t} = \frac{P}{mc} \tag{8.6}$$

8.4.2 Basic heater setup exercise

Now let's build a push-button heater circuit. The diagram is shown in Fig.8.1. The heater is a segment of manganin wire wrapped around a block of aluminum. Manganin, like tungsten in an incandescent light bulb, has rather high electrical resistivity compared to copper. When the pushbutton is depressed, the circuit is completed and current flows through the heater.

CAUTION: Whenever you are building or modifying any electrical circuit, be sure to turn off the power supply. This will reduce the chance of you getting shocked, and it will also reduce the chance of you damaging equipment. Also: even though the wiring is rather simple in this circuit, it is a good idea to get in the habit of using conventional wire color codes. As a rule of thumb, use black or red wires for high voltagewires and white or green wires for ground wires. Taking the time to do this will make it easier to trace any circuit errors when something doesn't function correctly.

1. Set up the circuit as shown in Fig.8.1. Be sure to include a sketch of the circuit in your documentation!

2. The mercury thermometer should be inserted into the aluminum block. Press the pushbutton and observe the temperature rise. While the button is pressed, record how fast the temperature rises (roughly, in degrees per second).

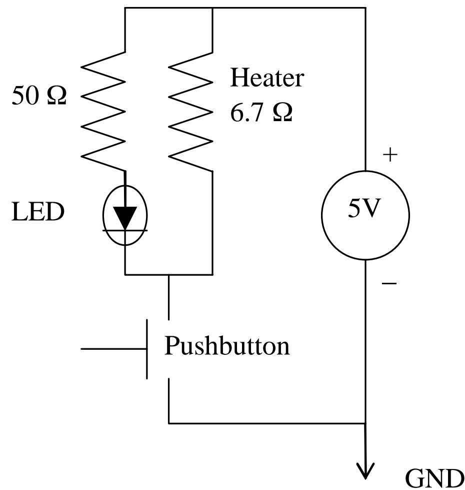


Figure 8.1: Basic heater circuit; the LED indicates when the heater is on.

3. Does the temperature immediately begin to fall when you release the pushbutton? Why do you think it behaves in this manner?

4. Does it take longer to rise by 5 degrees above room temperature, or to fall by 5 degrees back to room temperature? Why might this be?

8.4.3 Theoretical thermal response time exercise

1. Using Eq.8.6 and the thermal properties of aluminum, calculate the theoretical rate of temperature rise expected for your aluminum block. In computing the mass, m, of the block, you will need to know its volume and density. According to the *CRC Handbook of Chemistry and Physics*, the specific gravity (the ratio of the weight to that of water at 4 degrees Celsius) of aluminum is about 2.7. Also: the specific heat of aluminum is about 0.22 calories per gram-degree celsius. You'll need to convert this to appropriate units.

2. Do your calculations match your experimental results from the previous exercise? Does the block heat up more quickly or more slowly than anticipated from your calculations? What are your calculations not taking into account?

8.5 A bit of first-order response theory

You probably found that your calculated thermal response time is shorter than the observed thermal response time. This is at least in part because the thermometer itself takes time to respond to changes in the block temperature. This is a general characteristic of measurement instruments: they take time to respond to a signal. The thermometer is called a first-order instrument; its response is determined by the differential equation

$$\tau \frac{dT_{therm}}{dt} + T_{therm} = KT_{block} \tag{8.7}$$

K is called the static sensitivity (or the calibration factor). Usually $K = 1$, so that when the thermometer temperature, T_{therm}, stops changing (such that the first term equals zero), T_{therm} equals the block temperature, T_{block}. τ is a characteristic thermal response time of the thermometer. It depends on the heat capacity of the thermometer and the quality of the thermal contact between the block and the thermometer. For an abrupt change in the block temperature, the solution to this differential equation[1] is given by

$$T_{therm} = KT_{block}\left(1 - e^{-t/\tau}\right) \tag{8.8}$$

Notice that when $t = \tau$, the temperature will have risen to $1 - e^{-1} \approx 2/3$ of its final value. So introducing a sudden step in the block temperature and observing the response of the thermometer provides a way of determining the time constant of the thermometer. If, instead of a step change in the block temperature, the block is continually heated, one may expect a perpetual lag in the thermometer temperature reading behind the actual temperature of the block. This all may sound a bit complicated, but it is important when trying to regulate temperature accurately in a changing environment.

8.6 Thermistor basics

Mercury thermometers are simple, accurate, and reliable devices. Unfortunately, a mercury thermometers does not readily lend itself to computer automated temperature measurement and control. Instead, we will use a thermistor. A thermistor is a device whose electrical resistance depends strongly on its temperature. Let's first set up a circuit, shown in Fig.8.2, to measure temperature using a thermistor. Basically, the voltage that is read at V_T depends on the ratio of the fixed resistance, R_1, and the thermistor resistance, R_T. If the two happen to be equal, then V_T will be exactly half of the power supply voltage, V_0. If the thermistor resistance goes down or up, then the voltage V_T will also go down or up.

[1]The thermal problem is formally analogous to the electrical problem in which a capacitor is being charged through a resistor by a power supply. The heat capacity is analogous to capacitance; the thermal resistance between the heater and the aluminum block is analogous to resistance.

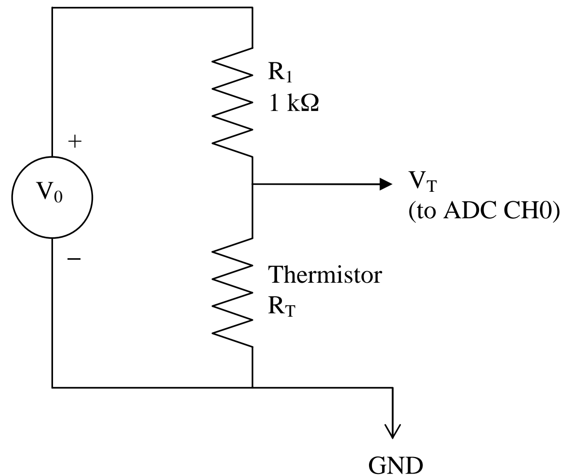


Figure 8.2: Thermistor circuit

8.6.1 Setting up the thermistor circuit

1. The thermistor should already be mounted in the aluminum block that you used in the previous exercise. Using a 5 volt power supply as V_0, assemble the circuit as shown in Fig. 8.2 on your protoboard.

2. The relationship between V_T and V_0 is given by the voltage divider formula

$$\frac{V_T}{V_0} = \frac{R_T}{R_T + R_1} \tag{8.9}$$

   Where does this formula come from? Explain using Kirchoff's laws of circuit analysis.

3. Invert the above equation to obtain a formula for R_T in terms of the other variables. Now, by measuring V_T, you can determine the thermistor resistance.

4. Write a program which periodically (once per second) reads the voltage, V_T, over the course of, say, 180 seconds, converts these values into thermistor resistance values, and then prints them to a file.

8.7 Drude theory

How does the resistance of a thermistor depend on temperature? According to the classical *Drude theory*, the resistivity of a metal depends on the density of

charge carriers, n, the charge of each charge carrier, e, and the mass of each charge carrier, m, as well as on the time, τ, between collisions of the mobile charge carriers and the stationary protons.

$$\rho = \frac{m}{e^2 n \tau} \tag{8.10}$$

Typically, each atomic nuclei donates one or more electrons to the pool of mobile electrons. The number density of electrons is therefore huge, perhaps $10^{22}/\mathrm{cm}^3$, and is nearly independent of temperature. The collision time, however, decreases as the temperature rises, since the protons vibrate more vigorously as the temperature rises. Therefore the resistivity of metals rise with temperature.

On the other hand, the resistivity of a homogeneous semiconductor, such as Germanium, drops when the temperature rises. Although the Drude theory is not entirely applicable to semiconductors (actually, neither is it to metals), it gives qualitative insight into their behavior. For semiconductors, the number density of charge carriers is very small at low temperatures. This is because the electrons are typically tightly bound to the atomic nuclei. As the temperature rises, atomic vibrations become increasingly energetic. The energy required to strip an electron from a nuclei is given by

$$E_g = k_B T_0, \tag{8.11}$$

where k_B is Boltzmann's constant and T_0 is some characteristic temperature. The probability of an electron being liberated from any given atom by thermal agitation is given by

$$\begin{aligned} P &= e^{-E_g/k_B T} \\ &= e^{-T_0/T}. \end{aligned} \tag{8.12}$$

Thus, the number density (number per unit volume) of free electrons in a semiconductor varies as

$$n = n_0 e^{-T_0/T}. \tag{8.13}$$

Using the Drude theory, we see that the resistance of a semiconductor may be written as

$$R = R_0 e^{T_0/T}. \tag{8.14}$$

In this expression, R_0 is a minimum resistance value at very high temperature, T_0 is an activation temperature (in Kelvin), and T is the absolute temperature (in Kelvin) (0 degrees Celsius = 273.16 Kelvin). The exponential variation of the resistance with temperature arises due to the rapid variation of the number of charge carriers with temperature. R_0, on the other hand, depends on the dimensions of the semiconductor and some factors which depend only weakly upon temperature.

8.8 Feedback and control

Although we have not yet calibrated our thermistor against a known temperature scale (we will do that a bit later, using the mercury thermometer), we may

still use it as a rudimentary thermometer since we know that its resistance varies with temperature. In this section, we will continuously monitor the temperature of the aluminum block by measuring the resistance of the thermistor, and turn the heater on or off so as to maintain a predetermined thermistor resistance (and hence temperature).

Although the output ports of your ESP32 board have a range of several volts, they cannot supply much electrical current and so, in general, cannot drive external circuitry loads directly. HEXFETs are one variety of enhanced mode power FETs (Field Effect Transistors) which are particularly suited for controlling large amounts of power by using the digital signals coming out of a computer. In short, we will set up the circuit shown in Fig.8.3 that uses a HEXFET to turn on or off a heater to regulate the temperature of our block.

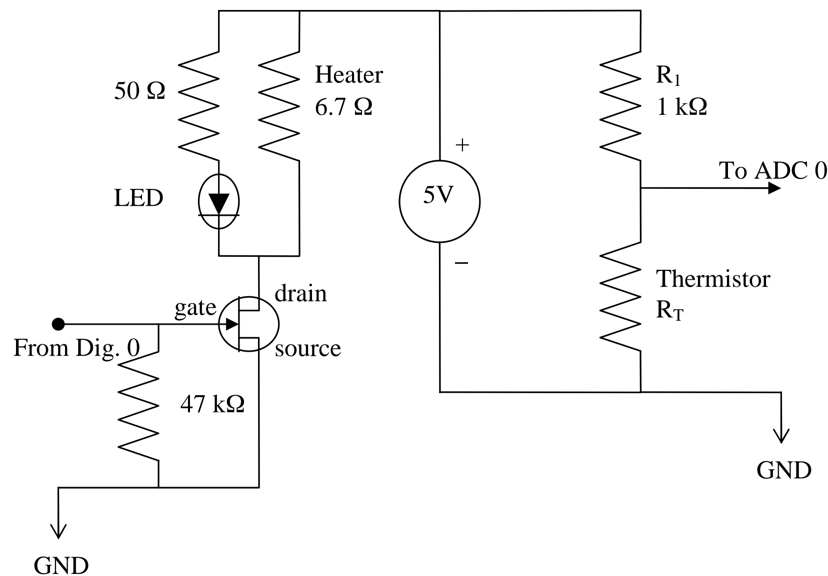


Figure 8.3: HEXFET temperature controller

8.8.1 Digital control of the HEXFET exercise

1. Set up the circuit shown in Fig. 8.3. The HEXFET will act like the push button switch you used earlier, but it will be controlled by a digital output port of your ESP32. When a HI signal is applied to the gate of a HEXFET, the device conducts current like a closed switch; when a LO

signal is applied, the device acts like an open switch, *i.e.*, it has infinite resistance.

NOTE: you may need to use two separate power supplies, instead of a single one, as shown in Fig. 8.3. This is because: if the HEXFET is drawing a lot of current, then you may inadvertently load down the power supply, causing its voltage to drop. This, in turn, will change your thermistor reading.

2. Connect the gate of the HEXFET to one of your digital output ports. Wait to turn on the power to the power supply until after you write your program (next step).

3. Write a short program that will turn the heater on for 10 seconds then turn the heater off.

4. Test your code. Does it work?

8.8.2 Temperature regulation exercise

1. Now that you can turn the heater on and off, write a program to regulate the temperature. The program should first ask the user to enter a target thermistor resistance. Be careful to pick a reasonable value for the target resistance. Next, it should execute a loop in which it 1) reads the voltage across the thermistor, converts this to a resistance, and prints its value to the screen, and 2) turns the heater on or off so as to approach the target thermistor resistance. Run the program and demonstrate to your laboratory instructor that the thermistor stabilizes at the target resistance. When testing, be sure to turn off the heater manually (or with a statement at the end of your program) to ensure that the heater does not overheat.

8.9 Thermistor temperature calibration

We now have a working temperature controller. Unfortunately, we do not know the temperature at which we are controlling. We will now calibrate the thermistor against a mercury thermometer inserted into the aluminum block. Remember that the block may take a short while to equilibrate at the target temperature. What we want to do, then, is to select a target resistance, initiate our temperature controller, wait a short while until the resistance stabilizes, then record the resistance and the corresponding temperature from the mercury thermometer. We will repeat this procedure for several values or resistance between room temperature and about 100 degrees Celsius. This will give us calibration data: thermistor resistance versus temperature.

8.9.1 Thermistor calibration runs

1. Modify your previous program so that, in addition to printing the resistance values to the screen, it saves them in a numerical list. It should collect data once per second for perhaps 240 seconds, or until the thermometer reaches equilibrium. After data collection, it should prompt the user to enter the temperature shown on the mercury thermometer and should record this. Finally, it should create an appropriately named data file with two columns. In the first column, it should record the time; in the second column, it should record the thermistor resistance value. Test to be confident that the program you have written generates a data file.

2. Inspect your data file (preferably by using a plotting program) to be sure that the resistance had equilibrated by the end of the data collection run. If it had, then you know the resistance corresponding to one particular temperature. If it had not, then you should modify your program so that it collects data for a bit longer. Once you are satisfied, perform 10 or 15 calibration runs over a wide range of temperatures. Make a table of resistance versus temperature in your lab notebook.

8.10 Least squares fitting to data

Thus far, we have a table of our thermistor's resistance and a number of temperature values. We would like to find the resistance at *any* given temperature within the range in which we calibrated our thermometer. To do so, we will need to find a mathematical equation which fits out data: Eq. 8.14. This equation can be cast in the form of a straight line; then we can use a linear least-squares fit to fit this formula to our data. Let us be a little more precise. Taking the natural logarithm of Eq. 8.14 gives

$$\ln R = \ln R_0 + \frac{T_0}{T}. \tag{8.15}$$

By setting

$$y = \ln R \qquad A = T_0 \qquad x = \frac{1}{T} \text{ and } \qquad B = \ln R_0, \tag{8.16}$$

Eq. 8.15 becomes

$$y = Ax + B, \tag{8.17}$$

which is the equation for a straight line. By finding values for A and B from the linear plot, values for R_0 and T_0 can be easily calculated. In doing the experiment, you have acquired data at a sequence of values of temperature T_i or alternatively $X_i = 1/T_i$. Each of these temperatures yielded an experimental resistance value R_i or alternatively Y_i. The model equation yields a theoretical resistance value R_i^{th} for each temperature, *i.e.*, for each X_i a theoretical value $Y_i^{th} = \ln R_i^{th}$ is given. The task is to find values for A and B to minimize

the error between the experimental and theoretical values, $E_i = Y_i^{th} - Y_i$. A common type of analysis minimizes the sum of the squares of the individual errors. Calling the total square of the error E_T, we get

$$\begin{aligned} E_t &= \sum_i E_i^2 \\ &= \sum_i \left(Y_i^{th} - Y_i\right)^2 \\ &= \sum_i \left(AX_i + B - Y_i\right)^2 \end{aligned} \tag{8.18}$$

To minimize this error with respect to the parameters A and B we take derivatives with respect to A and B and set them equal to zero:

$$\begin{aligned} \partial E_t/\partial A &= 0 \\ &= \sum_i 2X_i \left(AX_i + B - Y_i\right) \\ \partial E_t/\partial B &= 0 \\ &= \sum_i \left(AX_i + B - Y_i\right) \end{aligned} \tag{8.19}$$

Taking A and B out of the summations and collecting terms gives

$$\begin{aligned} AS_{XX} + BS_X &= S_{XY} \\ AS_X + BS &= S_Y \end{aligned} \tag{8.20}$$

where

$$\begin{aligned} S_{XX} &= \sum_i X_i^2 \\ S_Y &= \sum_i Y_i \\ S_X &= \sum_i X_i \\ S &= \sum_i 1 \\ S_{XY} &= \sum_i X_i Y_i \end{aligned} \tag{8.21}$$

Then solving for A and B

$$\begin{aligned} D &= SS_{XX} - S_x^2 \\ A &= \frac{SS_{XY} - S_X S_Y}{D} \\ B &= \frac{S_{XX} S_Y - S_{XY} S_X}{D} \end{aligned} \tag{8.22}$$

8.10.1 Least squares fit to data

1. Write a program to find values for A and B using a linear least squares fit to arrays of data $x[i]$ and $y[i]$. Use the program to obtain the model fit to your resistance and temperature data and obtain values for T_0 and R_0. When you are satisfied with your program, make a printout and put it in your lab book.

2. Using your favorite plotting program, plot the theoretical fit as a line together with your experimental values. It might be a good time to become familiar with a plotting program such as *MatPlotLib* or *Igor Pro*. Importantly: your plot should be clear and pleasant to behold. Your axes should be labelled neatly; the axis limits should be nice round numbers; the data should fill (but not fall off the edge of) the graph; the font should be clean; the data points should be large enough to see, *etc.*. In short: it should be a work of art. Make a printout of your plot and put it in your lab book.

The least squares fit assumes the measured data will be randomly scattered about the theoretical fit. The plot of the previous exercise does not show this clearly. A quick visual test of this assumption is to make a plot of the difference between the data and the fit, *i.e.*, plot the errors E_i. These are called the *residuals*.

8.10.2 Plot of residuals

1. Make a plot of the difference between the measured data and the theoretical fit to the data of the previous exercise. By inspection determine if the assumption of random errors was justified. Print out your plot and put it in your lab book.

8.10.3 A better temperature controller

1. Now that we have determined R_0 and T_0, we may easily convert any temperature into a thermistor resistance and vice versa. Modify your temperature controller program so that instead of asking for a target resistance, it asks for a target temperature. It should then proceed, as before, to regulate the temperature at the target value for some time. Note: In order for your c-code to invoke mathematical functions, such as sin() or ln(), you will need to include the "math.h" header file. You may also need to use the flag "-lm" to link to the math library when you compile your code.

2. Finally, change your program so that instead of using the digital output port to turn the heater on or off, it uses the analog output port. Instead of simply turning the heater on or off, which causes significant overshoot, the DAC should apply a voltage to the gate of the HEXFET which is proportional to the difference between the target temperature and the

measured temperature. This should allow for more precise temperature control. To display how well your temperature controller works, collect a data set and make a plot of temperature versus time. How well can you control the temperature? One way to characterize the level of control is the root mean square temperature fluctuation.

8.11 Errors in data and parameters

In fitting a theoretical model to data in the least squares method, the implicit assumption has been made that each data point has been measured with the same reliability. This is often not the case and it is then important to include a measure of the data reliability when fitting a model to these data. Another result of frequent interest which is not obtainable by the simple least squares fit is to determine how much the fitted parameters can vary without straining the fit to the data (how good is the fit?).

To make a statement of how good a measurement is we usually quote the value measured together with an expected error; for example a voltage is $V \pm \Delta V$ volts. An accepted definition of ΔV is that it is the root mean square (rms) value of the random error inherent in the measurement.

Consider a plot of a proposed theoretical fit and the data points $Y_i \pm e_i$ at a series of parameter values X_i. Assume the X_i are well determined (no uncertainty). The true variation of $Y(X)$ is given as some function of X. For sake of discussion assume that Y is of the form $Y(X) = AX + B$ where the parameters A and B are to be determined.

The total error can now be written as

$$E_T = \sum_i \left[\frac{(AX_i - B) - Y_i^{ex}}{e_i} \right]^2 \tag{8.23}$$

where e_i is the error in the data point Y_i. A small error e_i at data point Y_i will cause the difference between the model and the data point to be weighted heavily in the sum. Thus the points with small errors have a stronger effect on the fit. Proceeding as before yields the same formula for A and B except that now

$$\begin{aligned}
S_{XX} &= \sum_i X_i^2/e_i^2 \\
S_Y &= \sum_i Y_i^2/e_i^2 \\
S_X &= \sum_i X_i/e_i^2 \\
S &= \sum_i 1/e_i^2 \\
S_{XY} &= \sum_i X_i Y_i^{ex}/e_i^2
\end{aligned} \tag{8.24}$$

By standard rules of error propagation analysis, the errors in the estimates of A and B are determined to be

$$\begin{aligned} e_A^2 &= S/D \\ e_B^2 &= S_{XX}/D \end{aligned} \tag{8.25}$$

where $D = SS_{XX} - S_X^2$ as before.

Keep in mind that the estimation of the parameters $A \pm e_A$ and $B \pm e_B$ by the least squares method is a statistical one, i.e., given the data and the model junction, the calculated parameters A and B are the most likely ones for the system. The method assumes that the errors made in the measurements are random. It does not consider any systematic errors which may be lurking in your data. These last need to be ferreted out by careful thought and experimentation.

8.11.1 Errors in thermistor data

1. Make an evaluation of the error in your resistance determinations with the ADC and reanalyze the thermistor data with error considerations. To simplify error analysis, assume some reasonable constant error ($\Delta R_i = \Delta R$ for all i) and simplify the error equations by factoring the errors out of the sums.

8.11.2 Scientific Writing Assignment

An important part of this course is learning how to write a scientific paper. To this end, you should write a scientific paper, no more than four pages in length, which describes your work in calibrating your thermistor. This paper will serve as a warm-up, so to speak, for your final paper in this course. It will provide you with an opportunity to obtain feedback from your instructor regarding your scientific writing. To help you, your instructor will provide you with a few well-crafted scientific papers that can serve as examples. Your paper should include the following elements:

1. A *heading* which includes the title, the name of the authors, and the affiliation of the authors (institution and address).

2. An *Abstract* which states succinctly the most important results of your experiment.

3. An *Introduction*, which provides background on the problem which is being addressed in the paper. The introduction typically describes previous theoretical or experimental work that has been done on the problem.

4. An *Experimental apparatus and procedure* section description of your apparatus design and operation. This is the place to report the dimensions of your apparatus, the make and model of any equipment used, the experimental conditions, and the sequence of events that were carried out to perform a typical experiment.

5. A *Results and discussion* section in which you present your data, data plots, and your method of analysis including any formulae. Are your results reasonable? What are the most significant sources of error in your experimental results? This section may also involve some discussion for difficulties or avenues for further work.

Chapter 9

Timers and Interrupts

9.1 Introduction

In this chapter, you will learn how to use hardware timers and interrupts to orchestrate events in a synchronous manner. These features are very useful for scheduling future microcontroller tasks, such as reading a pin value at regular intervals, or triggering a function call when a button is pressed.

9.2 Timers

Micropython's `machine` module provides many useful functions and classes to access the esp32's available hardware. One of these classes is the `Timer` class, which provides an easy-to-use interface for requesting that a function be called in the future. `Timer` objects give you precise control over when an event takes place and therefore provide a more efficient and precise alternative to "busy-waiting" or polling:

```
while condition:
    task()
    sleep(duration)
```

where a function is called within a `while` loop and a `sleep()` is used to limit how often it's called. An additional perk to using `Timer` objects is that you now free up the esp32 to perform other tasks, instead of sleeping most of the time.

The esp32 has 4 hardware timers, enumerated 0-3, that can be controlled by the software `Timer` class. `Timer`s can be initialized to run in two modes:

1. `ONE_SHOT` mode where a function is called once after a given delay, and

2. `PERIODIC` mode where a function is repeatedly called at a given frequency or period.

The function that you give to a Timer object to call on your behalf has been traditionally called a `callback` function. Pause now and read the documentation on Timers.

9.2.1 Blink an LED

Your first task is to set up a basic timer that periodically blinks an LED. Please follow the steps below, and refer back to the Freenove tutorial's Project 2.1 if needed.

1. Connect an LED and resistor in series between an output pin and ground.
2. Create a `Pin` object, making sure to specify it in output mode (`Pin.OUT`).
3. Create a `callback` function which takes in a timer object as its only argument. Inside the function toggle the LED state (on/off) using `led.value(not led.value())`.
4. Create a `Timer` object, selecting which of the 4 hardware timers to use (0-3).
5. Finally, initialize the timer to periodically toggle the led on/off every 500ms.

Running the program, you should see the LED blink on/off with a 1s period (Figure 9.1). In order to stop the `Timer` you must call the object method `deinit()`.

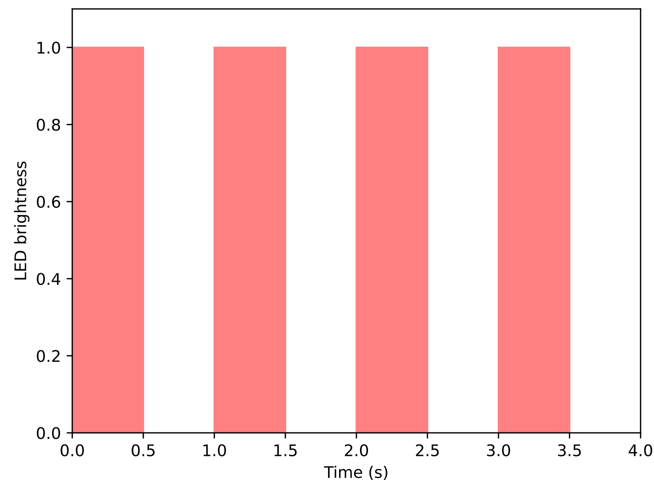
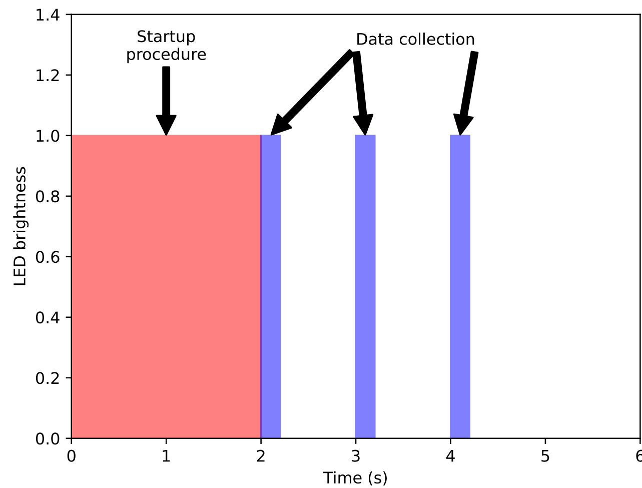


Figure 9.1: Blinking an LED using a timer with a period of 500ms

Note that while the LED is flashing, the Python interpreter is available for interacting with. Add to the bottom of your Python script a `while` loop (or another Timer!) that prints "Hello, multitasking!" to demonstrate that you can flash an LED while simultaneously printing.

9.2.2 Sequencing events

Consider an experimental apparatus where some hardware must be initialized before periodic data collection can take place. Let's model this situation using two LED's: The red LED is on during initialization, then shuts off indicating the apparatus is ready. A blue LED is then pulsed periodically until data collection is complete (3 samples collected). See 9.2.2) for an example.



The general strategy for orchestrating the red/blue sequences is to use timers to wake up and change the states of the LED. There are many ways to accomplish this, but one approach could be:

1. Turn on the red LED, then start a timer that wakes up in 2 seconds.

2. The timer's function would turn off the red LED, and then start a new timer that periodically flashes the blue LED.

3. To stop data collection, the callback function that is turning on/off the blue LED can increment a counter, and then `deinit()` it's own timer object. Remember that the `Timer` object is the argument passed into the callback function.

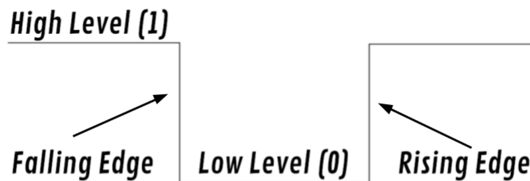
9.3 Interrupts

How do hardware timers work? By using hardware interrupts. Interrupts are requests given to the esp32's runtime that instruct the hardware to "react" to a change in an event.

The easiest way to demonstrate interrupts is with a switch. Refer back to Project 2.1 if you need help connecting a switch. Let's set up an interrupt that increments a counter whenever we press down on the switch.

```
1 counter = 0
2 def increment_counter(pin):
3     global counter
4     counter += 1
5     print(counter)
```

Create a `Pin` object as we normally do. Then register a handler (callback) function to be called when the Pin changes value. The voltage at the pin is 3.3V before pressing and drops to 0V when pressed. Since the value drops when pressed we want to trigger on the Falling Edge of the signal (Figure 9.3).




```
1 switch = Pin(21, Pin.IN)
2 switch.irq(handler=increment_counter, trigger=Pin.IRQ_FALLING)
```

Press the switch and ensure that it increments the counter.

9.4 Noisy switches

Press the encoder 10 times. What is the final count? It's probably different than 10 (if not do it again, and press quickly!). The imprecise counting happens because hardware switches are noisy (they don't cleanly turn on or off but mechanically vibrate). These vibrations call the handler functions many times instead of just once (Figure 9.2).

You may remember in Project 2.2 that mechanical vibrations can be ignored via a combination of `if`'s and and a `while` loop (non-ideal).

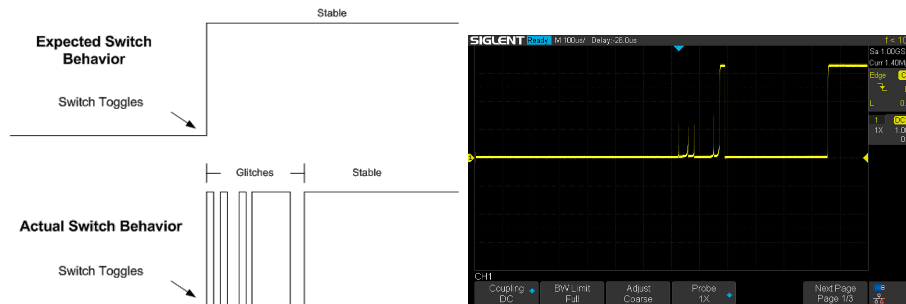


Figure 9.2: (Left) Mechanical switches can vibrate on/off before settling into a stable state. (Right) Oscilloscope trace of a real switch demonstrating this phenomenon.

```
button = Pin(21, Pin.IN, Pin.PULL_UP)

while True:
    if not button.value():
        time.sleep_ms(20)
        if not button.value():
            handler()
            while not button.value():
                time.sleep_ms(20)
```

The above code waits for 20ms after a press in hopes that the oscillations calm down. If it's still oscillating, then it keeps waiting in 20ms intervals until the signal stabilizes. This strategy works, but we essentially lock up the CPU from performing any other tasks.

The same waiting process can be done much more efficiently using interrupts and Timers. The general strategy is the same as the looping code above:

1. Recognize the first button press

2. Set up a period of waiting

3. Wake up and check if the switch is done vibrating

4. Call the handler function once

5. Go back to step 1

We will use a combination of interrupts and `Timer` objects to wait for the mechanical vibrations to stablize before calling the callback function. The code is below and heavily commented, so please take some time to understand how it works.

```
from machine import Pin, Timer
from micropython import schedule
import micropython
# space for exception messages when inside irq
micropython.alloc_emergency_exception_buf(100)



class DebouncedSwitch:
    """Tolerates mechanical vibrations of a switch and calls
    the given callback after the switch has settled down.
    Callback is called on falling edge, i.e. pressing the switch
    descreases the voltage on the pin.

    """
    def __init__(self, pin_num, callback, delay=50, timer_id=0):
        self.pin = Pin(pin_num, Pin.IN)
        self.callback = callback
        self.delay = delay

        self.timer = Timer(timer_id)
        self.timer.deinit()

        # 1. recognize first button press
        self.pin.irq(self._start_timer, trigger=Pin.IRQ_FALLING)

    def _start_timer(self, pin):
        self.pin.irq(None) # disable irq to ignore any bounces
        # 2. Set up waiting period
        self.timer.init(period=self.delay,
                        callback=self._timer_wakeup)

    def _timer_wakeup(self, timer):
        # 3. Wake up and check if
        timer.deinit() # stop timer
        if not self.pin.value(): # true if stabilized
            # 4. Call handler once (we must use schedule
            # to call our function if within an irq)
            schedule(self.callback, self.pin)

        # 5. Go back to step 1
        self.pin.irq(self._start_timer, trigger=Pin.IRQ_FALLING)
```

You can place this code into a separate Python file and then `import` it into your project. Create a switch object using

```
switch = DebouncedSwitch(21, increment_counter)
```

and then try pressing the debounced switch 10 times (quickly). Is it better behaved than before? Note that the Python shell is available for typing and performing other tasks. The `DebouncedSwitch` class is asynchronously handling the vibrations of the switch in an efficient manner.

9.5 Summary

In this Chapter, you learned how to precisely orchestrate a sequence of events using Micropython's `Timer` object, and how to use interrupts to trigger software changes when a hardware event takes place (e.g. a pin's voltage changing). You learned that these features are supported by the esp32 hardware which provides 4 hardware timers that can trigger interrupts. We then demonstrated a realistic application of these features by creating a software object to handle a mechanical switch that does not cleanly turn on/off. Timers and interrupts are a great alternative to busy-waiting or polling.

Chapter 10

Thermal Diffusion Experiments

10.1 Introduction

In the previous chapter, we built a temperature controller that employed a heater and a thermistor. A temperature scale for the thermistor was obtained by calibrating it against a mercury thermometer. We will now put our knowledge to use to study the properties of matter. In particular, we will apply a heat pulse to the end of a copper rod and measure the temperature at points along its length. From these measurements, we will determine the thermal conductivity and the heat capacity of our sample of copper.

10.2 Heat flow equation

When we calibrated our thermistor, we modelled the temperature dependence of its resistance using a particular formula, $R = R_0 e^{T/T_0}$. We did not choose this formula arbitrarily; rather it was physically motivated by our understanding of the conduction of electrons in a semi-conducting material. Furthermore, the parameters that appear in the formula, T_0 and R_0, have physical meanings: the activation temperature, and the limiting resistance at very high temperatures, respectively.

We will use a similar procedure in modelling heat flow in our copper rod. We will use what is called the diffusion equation. The specific heat, c, and the thermal conductivity, k, are parameters that appear in the diffusion equation. In this section, we will derive the diffusion equation.

Nearly all energy eventually becomes heat. Heat is a form of energy associated with the random motion of particles. Furthermore, heat tends to flow from warmer to colder objects. As an example, if a warm object, T_a is brought into thermal contact with a cold object, T_b, an amount of heat ΔQ will flow from

a to b. The temperature change of object a is given by $\Delta T_a = -\Delta Q/m_a c_a$, that of object b is given by $\Delta T_b = -\Delta Q/m_b c_b$. Here, m is the mass of the object, measured in kilograms, and c is its specific heat, measured in Joules per kilogram-Kelvin.

The rate of heat transport is called the power, and is given by

$$P = \Delta Q/\Delta t. \tag{10.1}$$

Instead of two objects, we might consider heat transport through a rod of length L whose left end is held at a warmer temperature than its right end. The amount of heat deposited in a particular segment of length Δz, in time Δt, is simply the difference between the amount of heat coming in from the left and going out to the right:

$$\Delta Q = [P(z) - P(z + \Delta z)]\Delta t. \tag{10.2}$$

Linearizing this equation, we obtain

$$\Delta Q = \left[P(z) - P(z) - \left(\frac{\mathrm{d}P}{\mathrm{d}z}\Delta z\right)\right]\Delta t. \tag{10.3}$$

Thus, the rate of heat flow is given by

$$\frac{\Delta Q}{\Delta t} = -\left(\frac{\mathrm{d}P}{\mathrm{d}z}\right)\Delta z. \tag{10.4}$$

Now, recall that when a bit of heat ΔQ is added to a mass m, its temperature increases by the amount $\Delta T = \Delta Q/mc$. The rate of temperature rise for a segment of length Δz and density ρ can then be written as

$$\frac{\Delta T}{\Delta t} = \frac{1}{\rho A c \Delta z}\frac{\Delta Q}{\Delta t}. \tag{10.5}$$

Combining Eqs.10.4 and 10.5, we see that

$$\frac{\Delta T}{\Delta t} = -\frac{1}{\rho A c}\left(\frac{\Delta P}{\Delta z}\right). \tag{10.6}$$

In seems reasonable that the power, or rate of heat transport, is proportional to the gradient of the temperature, dT/dz, as follows:

$$P = -kA\frac{\mathrm{d}T}{\mathrm{d}z}. \tag{10.7}$$

Here, A is the rod's cross sectional area (square meters) and k is the thermal conductivity (watts/cm-kelvin). Plugging this into the previous equation, we obtain

$$\frac{\Delta T}{\Delta t} = -\frac{1}{\rho A c}\left(\frac{\mathrm{d}}{\mathrm{d}z}\left[-kA\frac{\mathrm{d}T}{\mathrm{d}z}\right]\right). \tag{10.8}$$

If the thermal conductivity is independent of temperature, and hence position, which we will assume, it can be pulled outside of the spatial derivative along with the cross sectional area. In the limit of $\Delta t \to 0$, we can then write Eq. 10.8 in the form:

$$\frac{dT}{dt} = \frac{k}{\rho c}\frac{\mathrm{d}^2 T}{\mathrm{d}z^2}. \tag{10.9}$$

This is the *diffusion equation*. We may define $D \equiv k/\rho c$, the *thermal diffusivity*, whose units are cm^2/second. For our copper rod, $k \approx 4 \times 10^7$ erg/cm-s-K, $\rho \approx 9$ g/cm^3, and $c \approx 4 \times 10^6$ erg/g-K. (We here use the CGS, rather than the MKS system of units.) You should look up the precise values for these physical quantities in the *CRC Handbook of Chemistry and Physics*.

The diffusion equation may be solved analytically by considering the ideal case of an infinitely long one-dimensional rod and assuming that all of the heat is deposited at one location at $t = 0$. We will consider this case. The solution to the diffusion equation is then

$$T' = B_1 + B_2 \frac{1}{\sqrt{t}} e^{-z^2/4Dt}. \tag{10.10}$$

The reason for the T-prime notation will become apparent in a moment. (It does *not* indicate a derivative.)

10.2.1 Diffusion equation solution

1. Determine the thermal diffusivity for a piece of copper
2. Verify that the above solution in fact satisfies the diffusion equation.

We have here two unidentified parameters, B_1 and B_2. After a long time has passed, the second term becomes zero, and the rod should be at ambient temperature, T_0'. Thus, $B_1 = T_0'$. Defining $T = T' - T_0'$, we may write the solution to the diffusion equation as

$$T = B_2 \frac{1}{\sqrt{t}} e^{-z^2/4Dt}. \tag{10.11}$$

T is therefore the difference between the ambient temperature and the temperature of the rod at a particular point at a particular time. We will call it the "excess temperature." To identify B_2, we recognize that the total heat added to the rod, Q, may be found by integrating the quantity $dq = mcT$ over the length of the rod:

$$\begin{aligned} Q &= \int_0^\infty \mathrm{d}q \\ &= \int_0^\infty \mathrm{d}z\, \rho A c T \\ &= \frac{\rho A c B_2}{\sqrt{t}} \int_0^\infty \mathrm{d}z\, e^{-z^2/4Dt}. \end{aligned} \tag{10.12}$$

Performing this Gaussian integral and solving for yields

$$B_2 = \frac{Q}{A\sqrt{\pi \rho c k}}. \tag{10.13}$$

10.2.2 Integration practice

1. Perform the above integral and verify the equation for B_2

Now we can write the solution to the diffusion equation in terms of meaningful variables:

$$T = \frac{Q}{A\sqrt{\pi \rho c k t}} e^{-z^2/4Dt}. \tag{10.14}$$

This equation for T is still a bit complex; to elucidate its meaning, let us identify a characteristic time scale, τ, and a characteristic temperature scale, Θ, associated with a particular position along the rod. We define

$$\begin{aligned} \tau &= \frac{z^2}{4D} \text{ and} \\ \Theta &= \frac{2Q}{A\rho c z\sqrt{\pi}}. \end{aligned} \tag{10.15}$$

Now, we can write the solution to the diffusion equation in dimensionless form:

$$\frac{T}{\Theta} = \sqrt{\frac{\tau}{t}} e^{-\tau/t} \tag{10.16}$$

10.2.3 Reduced temperature and time

1. If one joule (1×10^7 erg) of heat is added to a 1/8 inch diameter copper rod, what is the value of Θ at a the location of each of your thermistors?
2. What is the value of τ at the location of each of your thermistors?
3. Verify that, using the above definitions, you can write the solution to the diffusion equation in this dimensionless form.
4. Verify that the maximum of T/Θ occurs at a time t such that $t/\tau = 2$. This will be used in your experiments to determine the diffusivity of copper.
5. Verify that at $t/\tau = 2$, $T/\Theta = 0.43$. This will be used in your experiments to determine the value of ρc.

What is the point of this kind of analysis? If we have a thermometer placed along the rod a few centimeters from the heat source, we can measure its temperature as a function of time. If we know the amount of heat added to the rod, Q and the maximum temperature, T_{max}, at the location of the thermometer, z, we can then determine the value of ρc. Furthermore, from the time at which T_{max} occurs, we can determine τ, and hence D. Finally, from D and ρc, we can determine k, the thermal conductivity of copper

10.2.4 Heat capacity

1. Verify that

$$\rho c = \frac{0.86Q}{\sqrt{\pi} A z T_{max}} \tag{10.17}$$

10.3 Experimental setup

The apparatus which we will use to measure the thermal conductivity of copper consists of a 1/8 inch outer diameter (o.d.) copper rod soldered to an aluminum base. Affixed to the rod with Stycast 2850 (thermally conductive) epoxy are (i) a 2.2 Ohm carbon resistor at one end which serves as the heater, and (ii) two thermistors spaced along the length of the rod. Insulated 10 mil[1] o.d. manganin wire is used to make electrical connections between the heater and thermistors and an 8 pin hermetic electrical feed-through epoxied into the aluminum base. Insulated 22 gage copper wires connect the feed-through to the proto-board.

The proto-board contains two circuits, very similar to the ones you have constructed previously for your temperature controller. On the left side of Fig. 10.1 is the heater part of the circuit. Notice that now we use an 18 volt power supply. We want to dissipate a lot of energy in the heater in a short time period, so you should use a power supply that can deliver up to four or five amps.

On the right side of Fig. 10.1 is the thermometer part of the circuit. The only difference between this and your previous circuit is that now we will be reading the voltage across two thermistors at different locations on the copper rod.

10.3.1 Circuit assembly

1. Assemble the circuit on your proto-board, as shown in Fig. 10.1. Make a sketch of the circuit in your lab notebook, clearly labelling the circuit elements.

Note: Since the heat input to the rod will be rather brief (and hence, not much heating power), the temperature rise that is measured by each thermistor may be quite small. To resolve the temperature rise, we may need to amplify the input to the ACD channels. More on this to come...

10.3.2 Control program

1. Write a program that will, first, turn the heater on for an interval of one or two seconds; second, record the temperature of the two thermistors several times per second and store them in an array; third, ask the user for an output file name; fourth, open an output file and store the temperature versus time data in the data file. Print out a copy of the program once you get it working and place a copy into your lab book.

[1] One "mil" is one thousandth of an inch. This is sometimes also called a "thou".

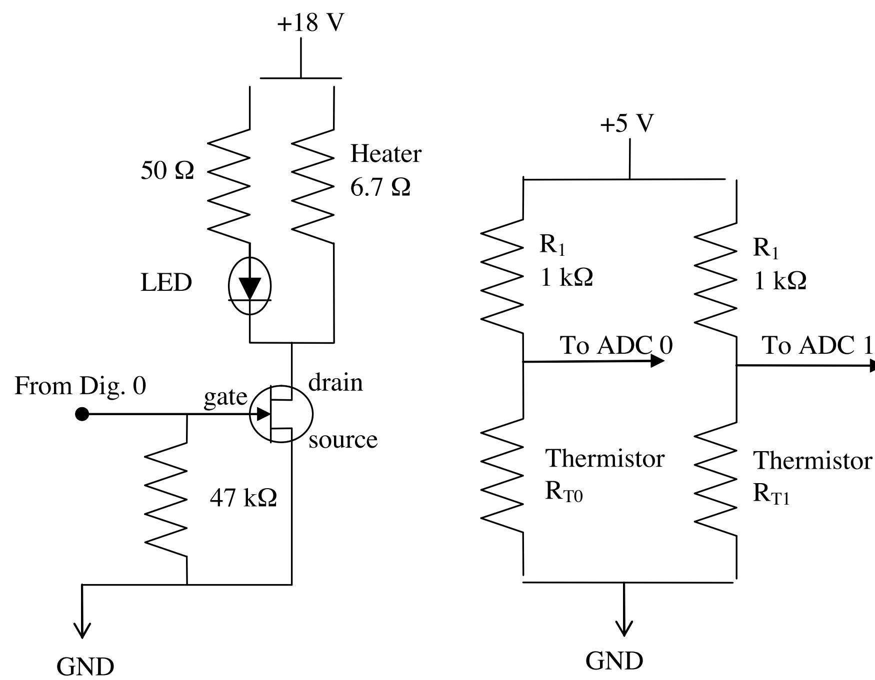


Figure 10.1: The heater and temperature control circuits for the thermal diffusion experiments.

10.4 Conducting the experiment

10.4.1 Data collection

1. Use your control program to apply a short heat pulse to the end of the copper rod and measure the evolution of the temperature at the thermistor locations as a function of time. You will need to collect data for perhaps four minutes in order for the temperature to return to room temperature after applying the heat pulse.

10.4.2 Data analysis

1. View your data using a plotting program. By identifying the maximum temperature for a particular thermometer, you can determine ρc, the product of the density and the specific heat of copper. By identifying the time at which this maximum occurs, you can determine the value of the diffusion constant, D. From these two measurements, you can determine the thermal conductivity, k.

2. You may need to make the following correction. Recall that the analysis assumed that the heat was imparted to the rod instantaneously. In our case, we dissipated energy over perhaps one second. Therefore, you may need to shift the right (on the time axis) by a half a second to (roughly) account for this.

3. Also, you may obtain better results for the thermal conductivity if you perform a curve fit of the solution to the diffusion equation,

$$T = \frac{Q}{A\sqrt{\pi \rho c k t}} e^{-z^2/4Dt}. \tag{10.18}$$

   to your data, allowing ρc and D to be fitting parameters.

4. Another source of error arises from the fact that not all of the heat Q dissipated in the heater travels through the copper rod; some of it travels through the surrounding air. Would this lead you to overestimate or to underestimate the thermal conductivity of the copper rod?

5. Evacuate the air from the vicinity of the copper rod and repeat the experiment to obtain a more accurate value for k.

10.4.3 Final paper

Now that you have set up your experiment, collected your data, and performed some analysis, you will need to share your results with the world! To this end, you will need to write a short scientific paper. The paper should be no more than four pages long, and should be composed in the style of the Physical Review. You may wish to look at a few papers to get a feel for the formatting. Your paper should include the following elements:

1. An *abstract* which states succinctly what you did and what you found.

2. An *introduction or overview* which describes the relevance of your experiments and previous work on this subject. You will need to do a literature search to find previous work on the topic. I'd start with the SAO/NASA Astrophysics Data System (http://adsabs.harvard.edu) or a Google scholar search.

3. An *experimental apparatus and procedure* section description of your apparatus design and operation. This is the place to report the dimensions of your apparatus, the make and model of any equipment used, the experimental conditions, and the sequence of events that were carried out to perform a typical experiment.

4. A *results and discussion* section in which you present your data, your method of analysis and your results. Be sure to compare your results with any previous or accepted values. If they differ, you need to provide a reasonable explanation as to why this is so. Which results are most reliable? What are the most significant sources of error in your experimental results?

Bibliography

[1] P Horowitz and W Hill. The Art of Electronics, 1989.

[2] B Kernighan and D M Ritchie. The C programming Language, 2017.

[3] Daniel F Mansfield and N J Wildberger. Plimpton 322 is Babylonian exact sexagesimal trigonometry. *Historia Mathematica*, 44(4):395–419, 2017.

[4] S Mueller. Upgrading and Repairing PCs, 2003.

[5] D E Simon. An Embedded Software Primer, 1999.

[6] B G Thompson and A F Kuckes. *IBM-PC in the Laboratory*. Cambridge University Press, Cambridge, 2009.