An Introduction to Building Electronics Projects with Arduino

Reto Trappitsch

Fall semester 2021

Contents

Preface

The notes are structured into 6 chapters. The first chapter mainly describes basics that students should already know from their introduction to Physics classes. It will be briefly reviewed in the first session. Subsequently, we will discuss one chapter per workshop session. The notes are prepared as we go, and you can always find the latest version, but also solutions to the examples in the form of code examples, on GitHub. If you find typos, errors, or other issues please let me know. The most recent copy of the LaTeX files and figures can also be found on

The lecture notes contain clickable links in dark blue. Furthermore, boxes throughout the text discuss are used for the following contents:

Background information on topics that do not necessarily fit into the text but are important to keep in mind will be given in a box like this.

Think about it more! These boxes will challenge you to think a problem through for yourself and go into more detail.

**Exercise 0** Exercises are given in these boxes. Flex your coding muscles and practize what you've learned.

Question 0 Questions will be given in these boxes. They should solidify your background knowledge.

This workshop was first designed for interested students at the Materials Research Science and Engineering Center (MRSEC) at Brandeis University in fall 2021. Support is provided by the Brandeis NSF MRSEC, DMR-2011486.
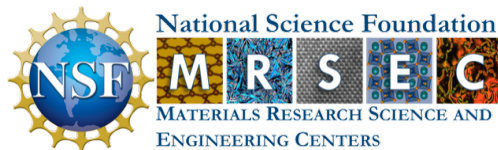
Acronyms

AC alternating current

ADC analog-to-digital converter

DAC digital-to-analog converter

DC direct current

I/O input / output

IDE integrated development environment

LED light-emitting diode

MRSEC Materials Research Science and Engineering Center

PWM pulse width modulation

SQUID Simplifying Quantitive Imaging Development and Deployment

0. Introduction

Scientific research that focuses on experiments and measurements has rapidly grown in the recent past, mainly thanks to significant improvements in engineering, instrument availability, and computing power. While many companies provide state-of-the-art research instrumentation and setups, cutting-edge scientific discovery often still thrives from home-built setups.

In addition to scientific instrument development and availability, the consumer / hobby marked has seen a huge increase in home-made electronics.[1] This development has especially been facilitated by products such as Arduinos and Raspberry Pis, as well as the huge maker community. Automation of research experiments can often benefit from such existing, low-cost products in order to significantly enhance an experiment or measurement. Furthermore, full low-cost instruments enabling frugal research have also been developed based on such platforms, see, e.g., the Simplifying Quantitive Imaging Development and Deployment (SQUID) project.

0.1. Basic Physics to Remember

Building electronics is not just fun because you can hold your final product in your hand and play with it, but also since it is a direct application of basic physics. Remember your introductory classes in physics!

Ohm's law   Throughout this workshop, you will encounter Ohm's law very frequently. This law states the current I through a conductor with resistivity R is directly proportional to the voltage U across the conductor. We can write this as:

$$I = \frac{U}{R} \tag{0.1}$$

$$U = RI \tag{0.2}$$

This basic relationship will become really important when designing circuits.

---

[1]For example, have a look at this article in the New York Times. Looking at the images clearly shows a 3D printed case as well as a standard Arduino cloud interface.

> 🚀 **Maximum current** Arduino pins, as we will discover later, can supply 5 V to, e.g., an light-emitting diode (LED). Since an LED is a diode, it's resistance (if connected properly) is close to zero (see also Wikipedia). Therefore, applying a 5 V voltage would result in an infinite current across this component. How would you add a resistor to limit the potential current to a maximum of 10 mA?

Electric power If a current flows through a resistor, electrical energy is transferred. The energy per time that is used in this resistor is the electric power P, which can be calculated as

$$P = UI. \tag{0.3}$$

Often, this electrical power is dissipated as heat. For example, an incandescent light bulb creates light (and heat!) by applying a voltage to a filament that is generally made of tungsten. The filament heats up and emits light. Electronic components generally have a maximum power rating, also often expressed as a maximum current rating.

> 🚀 **Maximum power versus maximum current** Assume you have a component that shows tells you a power consumption of at most 5 W at a current of 1 A. What is the maximum voltage that you can apply? What is the resistance of this element at this voltage?

> 🐾 **Electronic components and symbols** From your introductory physics class you should be familiar with basic electronic components such as resistors, capacitors, etc., and their symbols. We will discuss various components during this workshop. A good overview of components to refresh your memories can be found here on Wikipedia. Standard electronic symbols, which are really useful for drawing circuit diagrams, can be found here.

0.2. Analog and Digital

If you turn on a radio, and it is too loud, you can use the volume know, which is nothing else than an adjustable potentiometer, in order to regulate the volume of the sound. This volume can be adjusted over a whole range of settings. The potentiometer adopts linearly depending on its position, giving you an analog control over the volume. Mapping the volume from 0 (quiet) to 1 (loud), you can reach any value in between. Digital signals on the other hand are either on or off. Arduinos generally have many digital input / output (I/O) pins which can be either high (5 V) or low (grounded). If you connect a 3.3 V battery to an input pin, of course via a resistor in order to not exceed the current maximum, the switch would either tell you that it is high or low,
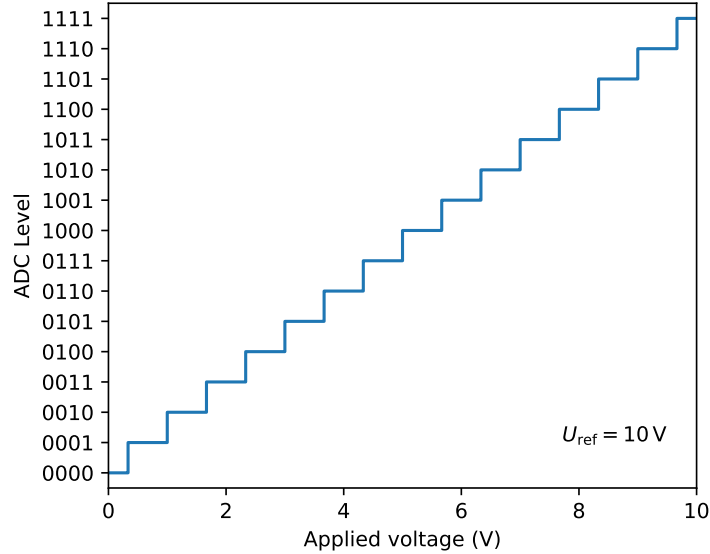
Figure 0.1.: *A 4 bit ADC with given levels. The reference voltage is 10 V.*

depending on the threshold that are actually set in order to determine this. Any kind of microprocessor *only* understands digital signals.

0.2.1. Analog-to-Digital Conversion

In order to measure as signal from a sensor, e.g., a photodetector or a temperature sensor as we will use later, a device called analog-to-digital converter (ADC) can be used. Above we mentioned that any kind of microprocessor only understands digital signals. The same is also true for an ADC. While a digital I/O pin has two levels (high / on or low / off), an ADC generally has many more levels in between. The resolution of an ADC is generally expressed in bits.

> **Bit** For any microprocessor, the two possible states (high and low) are generally expressed as 1 and 0. Binary numbers (base 2) are therefore the ideal representation to express different states. A digital I/O pin has 1 bit resolution, which means it can either be 1 or 0. Higher resolution means that more bits are available to set states. For two bits, i.e., a binary number with with two digits, the possible states are 00, 01, 10, 11. This means that 2 bit resolution has a total of four steps. For n bits, the number of available steps are 2^n.

Figure 0.1 shows the levels of a 4 bit ADC in binary as a function of the voltage that
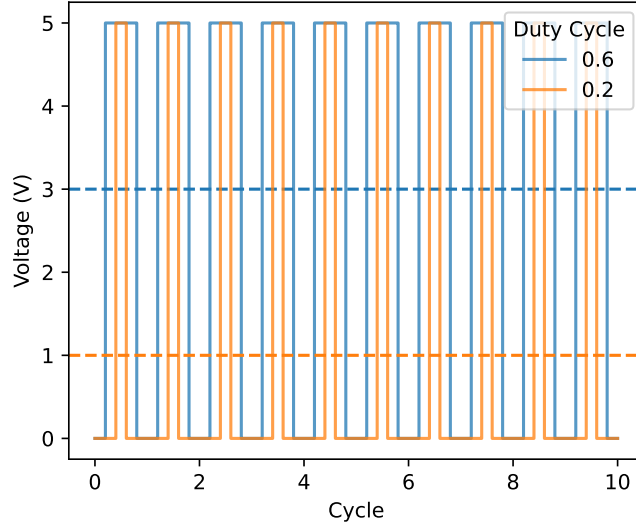
Figure 0.2.: *PWM cycles and average voltage for two different duty cycles.*

would be measured. The reference voltage here is $U_{\text{ref}} = 10\,\text{V}$. This reference voltage is the voltage that the ADC can measure at most, i.e., the voltage that it will return when the ADC level is 1111.

Knowing the resolution n of an ADC, we can easily calculate the minimum voltage difference that can be determined as

$$\Delta U = \frac{U_{\text{ref}}}{n}. \tag{0.4}$$

For the given example above in Figure 0.1, the minimum voltage would thus be $\Delta U = 0.625\,\text{V}$. Anything smaller voltage difference requires a higher resolution ADC.

0.2.2. Digital-to-Analog Conversion

Of course, we sometimes require the opposite of an ADC and need to convert digital signal into the analog world. The device that allows for this transformation is a digital-to-analog converter (DAC). A true DAC takes a digital signal and returns an analog voltage by dividing a reference voltage as many times as necessary. The same resolution limitations as for an ADC also apply to a DAC. For example, a 4 bit DAC with a reference voltage of 10 V can only increase the analog output in steps of 0.625 V.

Pulse width modulation (PWM) An interesting way to have a pseudo DAC is to use a process called pulse width modulation (PWM). Figure 0.2 shows a schematic on

how this process works with a 5 V pin. The digital pin is rapidly turned on and off. If it is at 5 V for 50% of the time, a 50% duty cycle, the effective, smoothed-out voltage that can be seen by a "slow" component would be 2.5 V. Depending on the duty cycle, a pseudo-analog output can therefore be created. A great example to use PWM is to have a dimmable LED. LEDs generally have only two states, on and off, i.e., is the voltage is high enough to light them, they are bright and otherwise dark. Using PWM however, we can turn an LED on and off in rapid succession such that it looks to the human eye as if the light source itself was dimmed.

Analog output with PWM Can you come up with a way to create a smooth analog output from a PWM pin? Think about how your cellphone charger turns alternating current (AC) into direct current (DC).

0.3. Arduino

For this class, we will be using an Arduino Micro. You can find more information and various alternative Arduino boards for all kinds of projects on the Arduino website.

In Appendix A, a schematic of the Arduino Micro board is given. This so-called pinout diagram specifies what all the various pins on the board mean and what they are used for. It is a very handy reference for when you develop your project.

0.3.1. Programming an Arduino

The easiest way to program an Arduino is by using the integrated development environment (IDE) for Arduino that can be found here. The website also has installation guides on how to install the IDE and Arduino driver on your computer, depending on your operating system. Please see this documentation or look at it at least briefly, since it will help you with troubleshooting in case your computer does not find the Arduino board.

The Arduino IDE is very useful when you are starting to learn how to program your Arduino. Under "File", "Examples" you can find eleven categories that give you many well-explained example snippets of code for various applications. We will make a lot of use of these examples, especially during the first few chapters of this workshop. The IDE also allows you to verify your code, i.e., check if it contains any errors (check button in the toolbar) and to upload your code to the Arduino itself (right arrow button in the toolbar). In order to upload your code to the Arduino board, make sure that you have the correct board selected. Go to "Tools", "Board" to select "Arduino Micro". Furthermore, you need to select the port on which your board is connected to the computer. To do

so, go to "Tools", "Port" to select the correct one. Now you are ready to begin uploading example code or your own code.

Programming language overview   To program your Arduino, the code must be written in C++. There are many great introductions online that can help you to get started, see also the background information box below. Therefore, we will only discuss very briefly the most important basics here. These will help you to avoid the most common mistakes.

- Variables must be declared. If you need an integer for example and assign it the value three, you can do this by declaring the variable as `int myVar = 3;`

- All command lines must be terminated by a semi colon `;`

- Functions, loops, etc. get surrounded by curly brackets

- It is up to the user to make the code look readable. If you want, you can write everything into one line since line endings and function endings are defined by the above stated rules.

- Line comments are preceeded with `//` while block comments use the following structure:

```
/*
    My comments in a block...
*/
```

In general, the minimum file structure for your Arduino code should look similar to the following.

```
// variable declarations, load libraries

void setup() {
  // setup code
}

void loop() {
  // main code that repeats
}
```

On the very top of your code, put variable declaration and initialization if required and load the necessary libraries. The `setup` function is the part that runs once when you reset or boot up your Arduino. The `loop` function will then run repeatedly and, ideally, until you unplug or reset the Arduino. We will see later how to fill these standard

functions. In addition, you can of course write your own functions with any names of your choosing, just make sure they do not collide in naming with these default functions.

Help with programming your Arduino To find further information on how to code an Arduino and get yourself started with C++, see the following linksx:

- Starters guide for programming for Arduino

- Programming reference specifically for Arduino

- Libraries in Arduino

- Glossary of commonly used terms

- User forum

Note that many components that we use come with detailed instructions and guides. For example, if you buy components from Adafruit, these parts generally come with a guide for Arduino, etc.

More Help As with so many things in life these days, more help is generally just one search on the internet away. There are many forums, articles, etc., on the web that discuss building electronics with Arduino. The hope is that this workshop helps you to discover the vast possibilities and gives you the right keywords to search your way through.

1. Blink

Like programming tutorials start with a "Hello World!" program, electronics tutorials generally start with a blink example. In this example you will learn to use digital I/O pins to drive an LED.

1.1. LEDs

1.1.1. Internal LED

Arduinos that have a built-in LED allow the user to program and use this LED. Therefore, all the hardware you need is the Arduino and a USB cable in order to connect it to your computer. Feel free to plug the Arduino into a breadboard as shown in Figure 1.1. The internal LED in Figure 1.1 can be seen just on the right side of the label that says
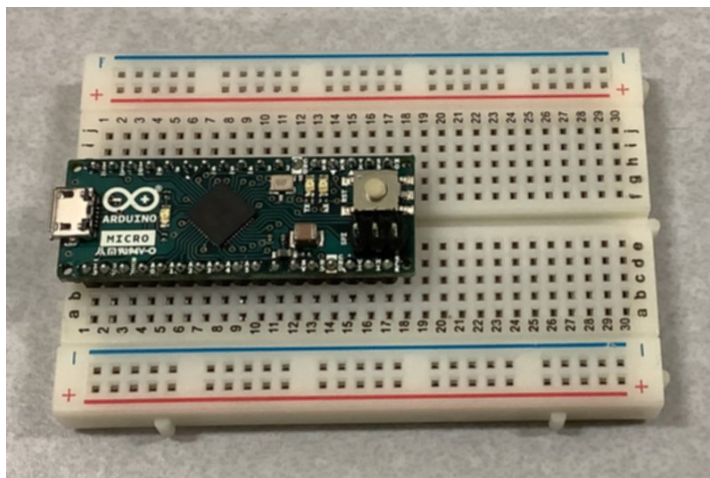
Figure 1.1.: Arduino micro plugged into a breadboard. On the left the USB connection is visible. On the right of the label where it says "Arduino", the internal LED can be seen.

"Arduino". In addition, you can see a button on the right hand side of the board (the reset button) as well as two more LEDs on the left side of this button. These additional LEDs cannot be accessed by the user and are reserved for the system.

If you start the IDE and load the basic example "Blink", you will get some code that will control the LED. The following exercises will use this simple example and slowly extend it.

> **</>** | **Exercise 0** Open the example blink file and read the comments. What is done in the setup? What does the variable `LED_BUILTIN` stand for? Study the loop, what will happen when you upload the code to your Arduino? Do so and see if your assumptions were correct. Modify the timings of the program such that the LED blinks at a different rate.

1.1.2. External LED

We can also connect an external LED to a digitial I/O pin. To use a pin as an output pin, i.e., to set its level by software, we have to define the `pinMode` to be in `OUTPUT` mode. Furthermore, connecting an LED to a pin and simply driving it can be bad for certain electronics, since an LED by itself has no resistance. Looking at equation (0.2) we can see that in such a case the current should become infinity, which might destroy the I/O pin. Fortunately, Arduinos have an internal resistance that prevent this from happening. Furthermore, we are using LEDs that have an internal resistor in addition in order to protect the LED from blowing up.
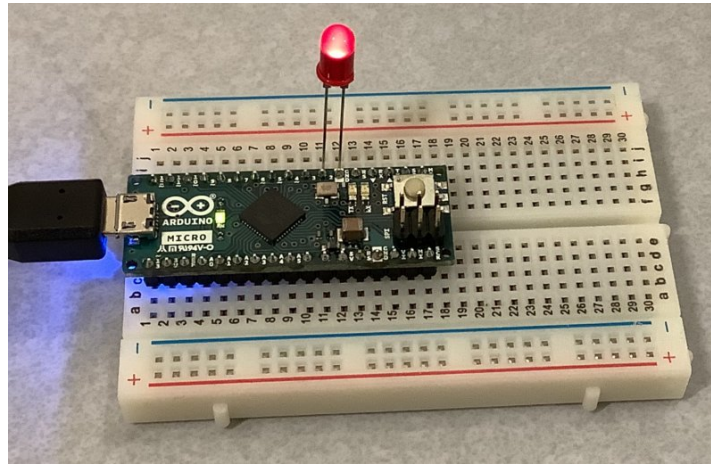
Figure 1.2.: *Arduino with one LED connected.*

Exercise 1 Draw a wiring diagram to connect your own LED to a Arduino output pin. Where do the anode and cathode of the LED connect to? Attach your LED to the arduino and modify the simple blink experiment to use your LED instead of the built-in one. If you have trouble figuring out how to connect the LED, study Figure 1.2 and remember that every electric circiut must be completed.

1.1.3. Dimming an LED

As we have discussed above, LEDs cannot be dimmed in the traditional way, i.e., by using a potentiometer to lower the voltage. However, we can use a PWM output in order to only have the LED on for a certain amount of time. This will result in our brain perceiving the LED as dimmed. We have already described the PWM outputs above in Section 0.2.2, see also Figure 0.2. In order to identify a PWM output, look at the pinout (Appendix A). Digital pins indicated with ∼ are the ones that can be used in this fashion, e.g., pin 3.

Exercise 2 Connect your LED to a PWM output pin. From the Arduino IDE, load the basic example named "Fade".

1. Read the setup and loop functions. How is the pin output set and what is the difference to the way it was set in the previous exercise?

2. Modify the fade amount to 7 and run it again. At the brigthest pint, the LED briefly blinks. Why?

3. Can you rewrite the routine in order to prevent it from blinking at the highest point? An `if` statement might be useful.

The `void loop()` function acts as a endless loop. The most commonly used looping structures are `while` and `for` loops. In this sence, the loop function is no different from a `while(True)` loop.

Exercise 3 Inside the `void loop()` function, write a `for` loop to replace the ramp. Ensure that your loop does not blink, even if you select various fade amounts.
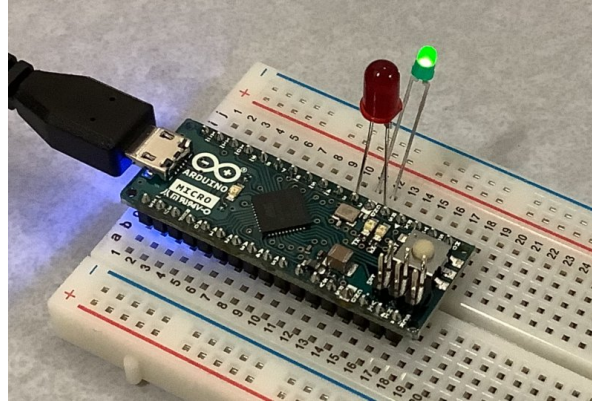
Figure 1.3.: *An Arduino with two LEDs connected.*

> **Question 0**
>
> - How does integer division in C++ work?
>
> - Ho wcan you do a division in C++ and round the result to the nearest integer?
>
> - What is boolean variable and how do you compare them, e.g., in an `if` statement?
>
> - How would you switch the state of a boolean variable?

1.1.4. Multiple LEDs

Finally, since you have more than one pin available, multiple LEDs can be set up. An example of a test setup is shown in Figure 1.3 This can be especially useful if you want to display the status of your setup using different colored LEDs.

Subfunctions in C++ Sometimes it is useful to put somem of your code into external functions, i.e., not to write them into the main loop. For example, you can write a function as following:

```cpp
void myFunction(bool toggle) {
    // your code here...
}
```

You can then call this function from the main loop by calling `myFunction(true);` if you want to assign the value `true` to the variable `toggle`.

Exercise 4 Setup two LEDs, a red and a green one. Write a program that switches between blinking the red and green LEDs. Write a subroutine that takes a boolean variable as an input and, depending on the value of this input variable, turns either the red or the green LED on (and the other one off).

1.2. Buttons

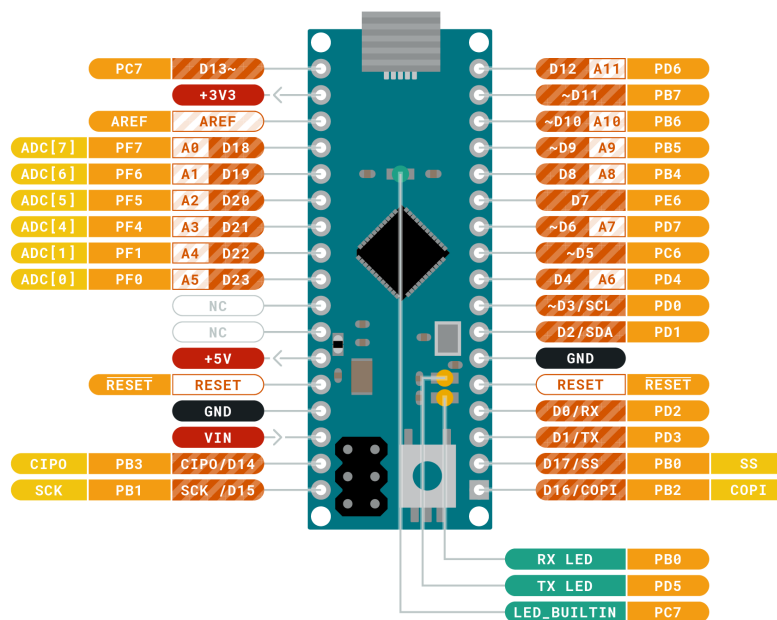# Appendices

A. Arduino Micro Pinout

ARDUINO
MICRO

		PC7	D13~		
			+3V3		
		AREF	AREF		
ADC[7]	PF7	A0	D18		
ADC[6]	PF6	A1	D19		
ADC[5]	PF5	A2	D20		
ADC[4]	PF4	A3	D21		
ADC[1]	PF1	A4	D22		
ADC[0]	PF0	A5	D23		
			NC		
			NC		
			+5V		
		RESET	RESET		
			GND		
			VIN		
CIPO	PB3		CIPO/D14		
SCK	PB1		SCK /D15		

D12	A11	PD6	
~D11		PB7	
~D10	A10	PB6	
~D9	A9	PB5	
D8	A8	PB4	
D7		PE6	
~D6	A7	PD7	
~D5		PC6	
D4	A6	PD4	
~D3/SCL		PD0	
D2/SDA		PD1	
GND			
RESET		RESET	
D0/RX		PD2	
D1/TX		PD3	
D17/SS		PB0	SS
D16/COPI		PB2	COPI

RX LED	PB0
TX LED	PD5
LED_BUILTIN	PC7

■ Ground	▮ Internal Pin	▰ Digital Pin	▮ Microcontroller's Port
■ Power	▮ SWD Pin	▱ Analog Pin	
■ LED	▢ Other Pin	▮ Default	

16