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 you likely already know from your introduction to Physics classes. 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 GitHub.

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

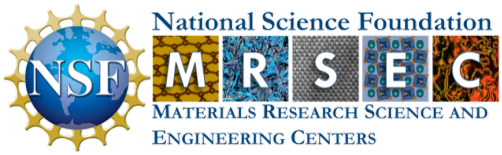
Background information on topics that do not necessarily fit into the text but are important to keep in mind will be given in a box like this.

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, complete 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.

In this workshop, we will develop a temperature regulation system in order to cool a sample. We will read the temperature using a thermistor, use buttons to control the set point, use a display to show the displayed and set temperature, and finally drive a thermoelectric cooler in order to achieve sample cooling.

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!

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:

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

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

[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.

Remember this relationship for when you design your circuits.

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, electric 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, electric 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. All electronic components have a maximum power rating, also often expressed as a maximum current rating.

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

Electronic components and symbols You are probably already 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 knob, 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. An Arduino generally has many digital input / output (I/O) pins which can be either high (5 V) or low (grounded). If

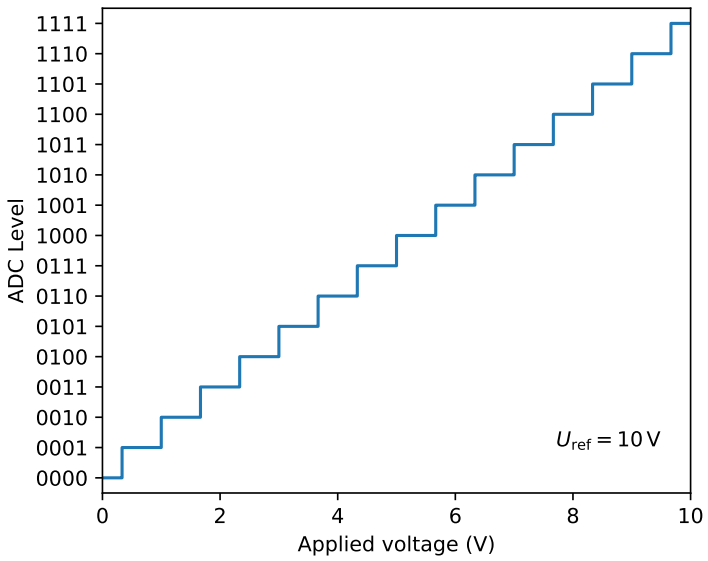


Figure 0.1.: *A 4 bit ADC with given levels. The reference voltage is 10 V.*

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, 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 an 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) can be 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 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 of this process for 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.

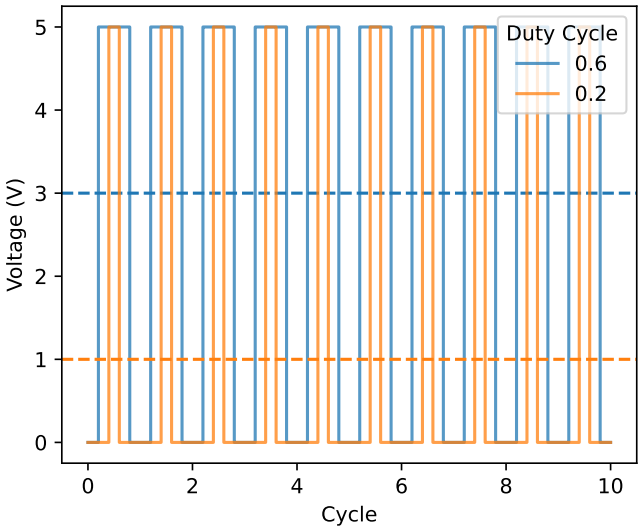


Figure 0.2.: *PWM cycles and average voltage for two different duty cycles.*

In Appendix A, a schematic of the Arduino Micro board is given. This 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 rules here. These will help you to avoid the most common mistakes.

- Commands and instructions are case sensitive
- 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 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.

Debugging When code does not do the thing we expect it to, it is often difficult to exactly see why it does not work. While the Arduino IDE catches many bugs, it cannot catch logic errors. If you simply code a script on your computer, you might debug it by printing out values to the screen. The same can be done with the Arduino. Unless you have a screen connected to the Arduino, you have to send the values to print via the serial connection to your computer for displaying. The statement `Serial.begin(9600);` should go into your setup routine. This begins the serial communication to your computer with a baud rate of 9600. Inside your main loop, you can then use the following print statements to write to your computer:

- `Serial.print("Hello␣");` prints "Hello " without a line break.
- `Serial.println("World!");` prints "World!" on the same line as the print before and then starts a new line.

To see these printed values, click in the Arduino IDE on "Tools" and then select "Serial Monitor". Note that you can also send commands from the computer to the Arduino via Serial. More information can be found here.

Help with programming your Arduino To find further information on how to program for Arduino and get yourself started with C++, see the following links:

- 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.

0.3.2. TinkerCAD

If you want to play with a virtual Arduino, give TinkerCAD a try. On its website you can simulate an Arduino, add components, write code, and then run the setup like in real life. It is a great tool to plan a project and start experimenting with setups, e.g., while you are waiting for components to ship.

1. Blink

Like programming tutorials start with a "Hello World!" program, electronics tutorials generally start with a blink example. In this chapter you will learn how to write to and read from digital I/O pins.

1.1. LEDs

1.1.1. Internal LED

Arduinos with 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

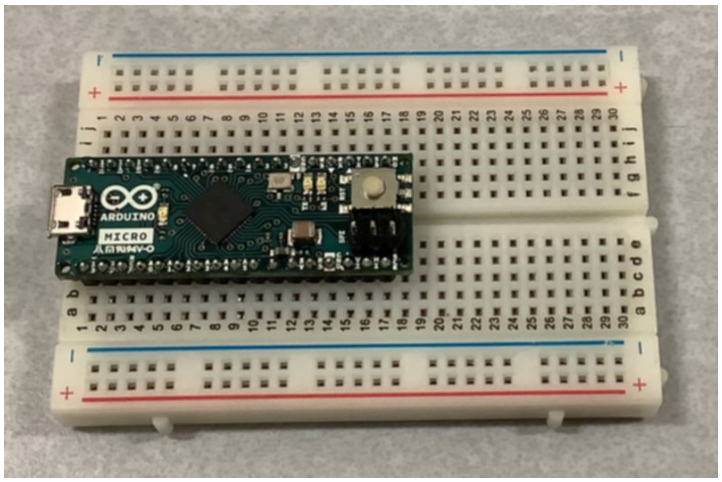


Figure 1.1.: *Arduino micro plugged into a breadboard. On the left the USB connection is visible. On the right of the label where it says "Arduino", the internal LED can be seen.*

"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 digital 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 the LED, since it has, by itself, 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 resistor that prevent this from happening. However, the LED might still get too much current, which might significantly reduce its lifetime. You can either add a resistor or use an LED with an internal one, as we are doing here (Figure 1.2).

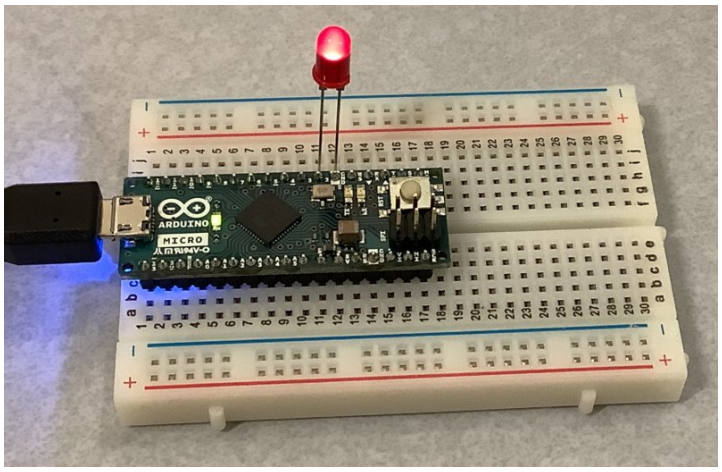


Figure 1.2.: *Arduino with one LED connected.*

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 different from how the pin was set in the previous exercise?
2. Modify the fade amount to 7 and run it again. At the brigthest point, 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 an endless loop. The most commonly used looping structures are `while` and `for` loops. In this sense, 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.

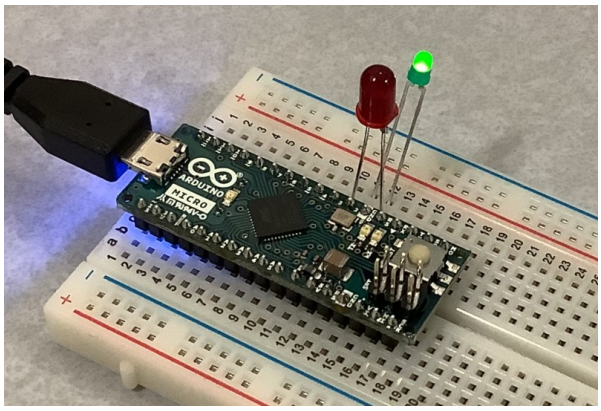


Figure 1.3.: *An Arduino with two LEDs connected.*

Question 0

- How does integer division in C++ work?
- How can you do a division in C++ and round the result to the nearest integer?
- What is a boolean variable and how do you compare two booleans, e.g., in an `if` statement?
- How would you flip the state of a boolean variable?

1.1.4. Multiple LEDs

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 some 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:

```
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`.

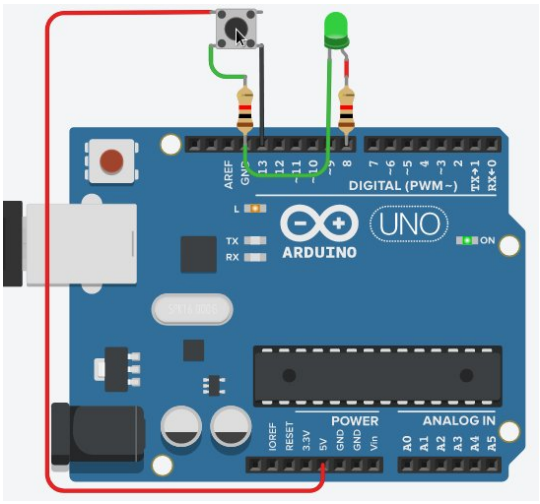


Figure 1.4.: *TinkerCAD simulation of Arduino with a button and LED connected.*

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

Most devices that allow for user interactions contain some buttons. We can use the digital I/O pins in order to connect a button. Figure 1.4 shows a TinkerCAD simulation of an Arduino with a button and an LED connected. Here, we connect the button on one side to the 5 V output of the Arduino and the other side via a 1 kΩ resistor to ground and simultaneously to a digital I/O pin. The resistor limits the current to 5 mA, see equation (0.2). In the setup routine, we need to set the `pinMode` to `INPUT`, since we want to read the value that is connected to the pin. If the button is unpressed, the pin is on the same electrical potential as ground and therefore in a low state. If the button is pressed however, the pin is at 5 V and thus in a high state. In the loop you can, e.g., read the state of the button connected to `buttonPin` as:

```
int buttonState = digitalRead(buttonPin);
if (buttonState == HIGH) {
    // the button is pressed
} else {
    // the button is not pressed
}
```

Exercise 5 Connect a button and an LED to your Arduino and write a program that turns the LED on while the button is pressed. Then adjust your routine such that, whenever you press the button, the LED is turned on for 3 s and then turns off after the time has elapsed.

Adding time In the second part of the above exercise, you have (most likely) written your program such that if the button is pressed while the LED is on, nothing happens. The light will still turn off once the set time elapses after the initial button press. Can you come up with code that would allow you to reset the countdown when pressing the button again, i.e., add time to the timer?

2. Display

In order to display the temperature of our sample cooling setup, we will connect a display to our Arduino. Displays come in many different shapes and forms, form the monitor you might be reading this on to simple LED seven-segmant displays. An overview of various displays that are ready to be deployed on Arduino can, e.g., be found here. For our specific case, a simple seven-segment display will do the trick to display the temperature.

2.0.1. Seven-segment displays

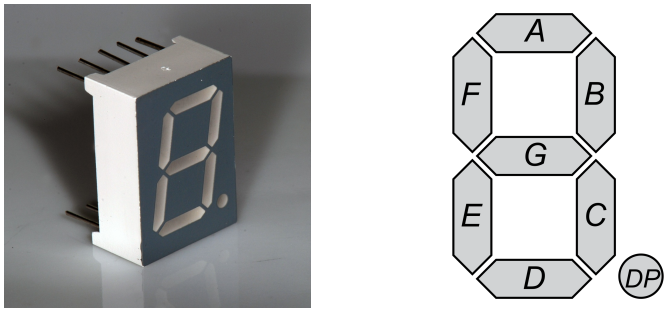


Figure 2.1.: *Seven-segment display: photo (left) and schematic (right). Credit: Peter Halasz (left) and Uln2003 (right) via Wikipedia. License: CC by SA-3.0 (left), Public domain – CC0 (right)*

As displayed in Figure 2.1, seven-segment displays contain seven individual segments that allow us to display any nummer and even some letters. In addition, they can contain a decimal point. The right hand figure shows the schematic and typical labeling of a seven-segment display. Each segment is its own LED and we could drive these LEDs by using multiple I/O pins. However, in order display multiple numbers, the number of I/O pins that we would use up would very fast become too extensive. Figure 2.1 shows that the seven-segment display has five pins on top and, not shown here, also five connectors on the bottom. For each seven-segment display you could look up the wiring diagram and notice that each segment has its own connection and that they have a total of two common grounds.

Chips such as 74HC595 that allow us to convert serial into parallel data allow us to effectively control multiple eight outputs, as required for such a display, with one only

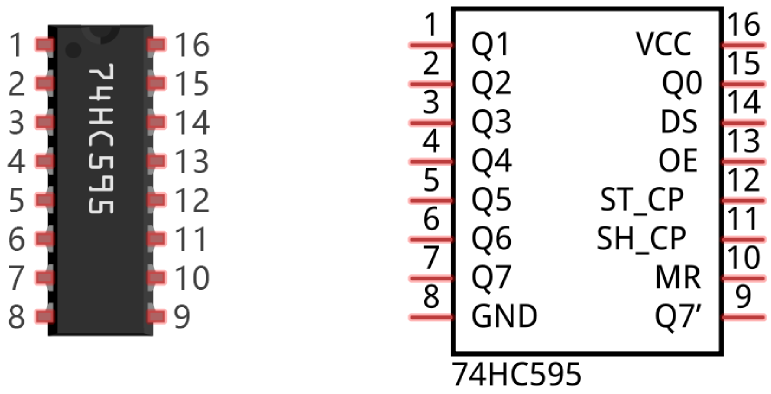


Figure 2.2.: *Schematic and pin names of a 74HC595 chip. Credit: Freenove, License: CC BY-NC-SA 3.0*

a few pins. Figure 2.2 shows a the schematic and pin names of such a chip. Note the notch on the top of the schematic that each chip contains in order to provide you with a reference on the pin assignments. You can in fact find various versions of this chip from different manufacturers, e.g., the datasheet for the Texas Instruments version can be found here. Generally, the pins have the following purposes:

Pin name	**Pin number**	**Description**
Q0 - Q7	1-7, 15	Parallel data output
GND	8	Ground
Q7'	9	Serial data output
MR	10	Remove shift register
SH_CP	11	Serial shift clock
ST_CP	12	Parallel update output
OE	13	Enable output
DS	14	Serial data input

As indicated by the names, the parallel data output pins are the eight pins that can be connected to the individual LEDs of a seven-segment system. The GND pin provides the ground connection of the chip. The serial data output (Q7') can be used to connect another 74HC595 chip in series. The MR and SH_CP take care of clearing the shift register and timing when a shift happens. The ST_CT triggers an update in the parallel output and the OE pin enables this output. Finally, the DS pin is where the serial input is given.

Want do build your own seven segment display driver? Detailed instructions on how to use a 74HC595 chip to drive a seven segment display can be found online, e.g., on in chapter 17 of this tutorial. Building such a driver is outside the scope of our workshop, however, it might be useful for you to read and understand the shifting itself and how it works. Please have a look at the mentioned tutorial.

2.1. Adafruit four digit, seven-segment display with I²C backpack

Appendices

A. Arduino Micro Pinout