In this document, we will go through the steps to get you started on using your Arduino UNO as an oscilloscope and square wave generator. By the end this document, you will have

- acquired source code
  - for your Arduino: `oscilloscope.ino`, and
  - for your graphical oscilloscope interface: `GUI.pde`;
- uploaded `oscilloscope.ino` onto your Arduino using the Arduino IDE;
- interacted with the oscilloscope interface by running `GUI.pde` using the `processing` IDE.

Don't worry if you are not familiar with Arduino boards or the `processing` language. All of the programming has been done; you simply need to learn how to use them.

You will need a USB type A port on your computer.

1 Getting started

Acquiring source code: Begin by downloading the source code you need from the following GitHub repository. If you are experienced with Github, free feel to `clone` the repository. Otherwise, download the code as a zip file, and extract its contents to a location you will remember.

https://github.com/sillyPhotons/oscilloscope.

Github is an online service that allow programmers to share and track changes to their code, collaborate with other programmers, and even host personal websites. You will have plenty of opportunity to explore it in your undergraduate career.

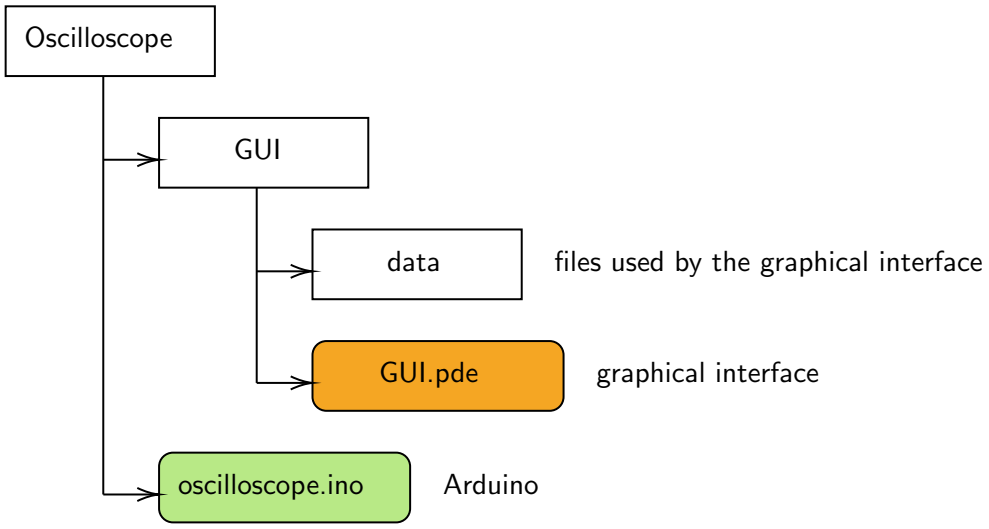

Figure 1: Avoid changing the directory structure and renaming files within the source code folder.

Arduino IDE: Next, install the Arduino IDE (short for integrated development environment) onto your computer. This is the program that will allow you to upload the `oscilloscope.ino` code onto your Arduino through a USB connection. Follow the instructions specific to your operating system:

Windows: https://www.arduino.cc/en/Guide/Windows
MacOS: https://www.arduino.cc/en/Guide/MacOSX
Linux: https://www.arduino.cc/en/Guide/Linux

While other development environments for the Arduino exist (for example, PlatformIO on Visual Studio Code), we are using the Arduino IDE for its simplicity.

Processing IDE: The last piece of software we need to install is a stable release of the `processing` IDE (version 3.5.4). The download page can be found here:

https://processing.org/download/

1.1 Uploading code onto your Arduino

Using the provided USB A/B cable, connect your Arduino to an USB A port on your computer. Next, open up the Arduino IDE. Confirm that the IDE knows which USB port you used, and that you are using an Arduino UNO board as opposed to other variants.

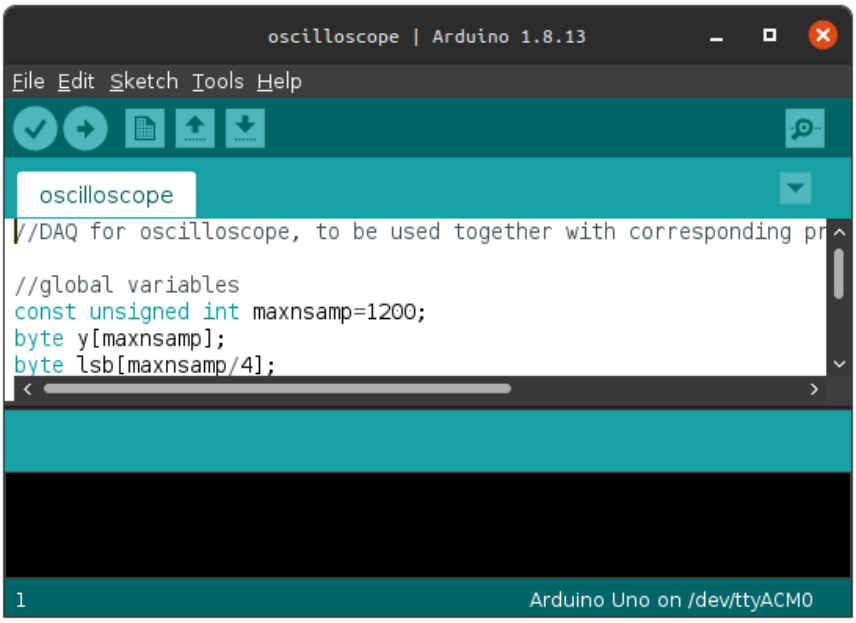

Figure 2: The development board (Arduino UNO), and the serial port that the IDE is configured to is shown in the bottom-right corner. You will need use select `Tools` in the upper tool bar, and pick the correct `Board` and `Port` option if is incorrectly configured.

Open `oscilloscope.ino` on the IDE by selecting `File` → `Open...`. Finally, select right arrow beside the check mark icon to compile and upload the code onto your Arduino. Blinking LEDs are a sign that an upload is in progress.

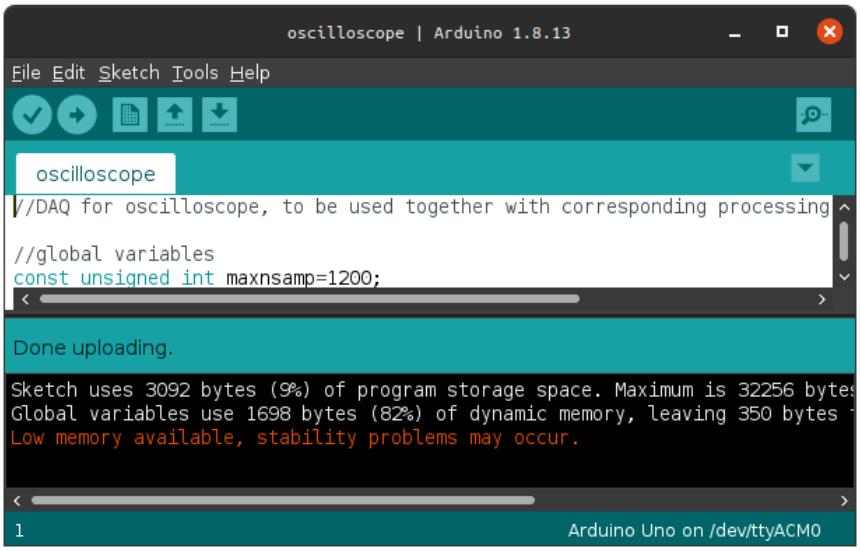

Figure 3: The IDE should display "Done uploading" if the code compiled and uploaded without error. Check out section 2 if you ran to trouble.

1.2 Accessing the oscilloscope interface

Open `GUI.pde` using the `processing` IDE.

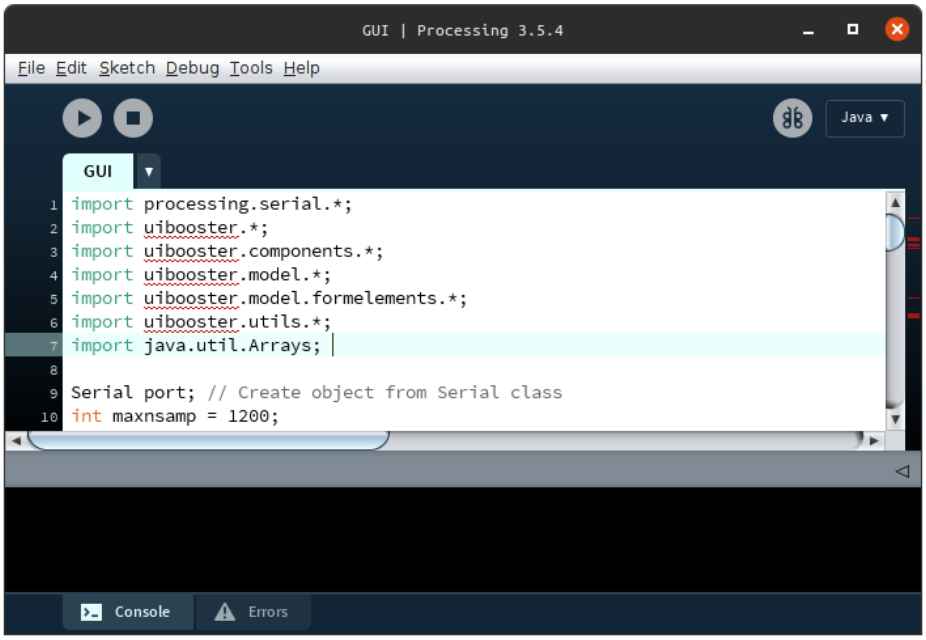

Figure 4: You should see a similar window with `GUI.pde` opened. The underlines on lines 2 to 6 tells us that the `uibooster` library needs to be installed.

Next, we will install a library called `uibooster` required to run our program. Open the contribution manager menu by selecting `Sketch` → `Import Library` → `Add Library`. Search for `uibooster` within contribution manager and select the option to install the library (figure 5).

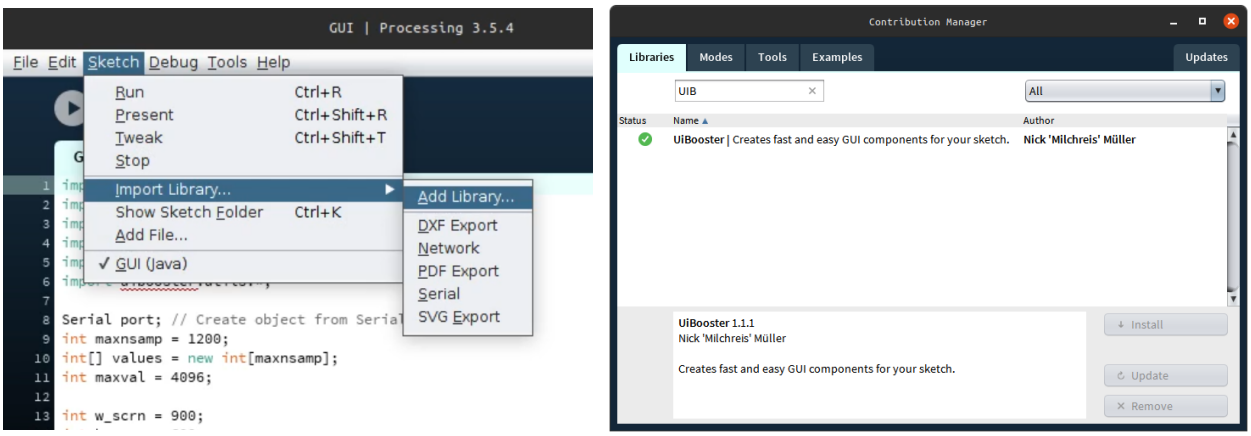

Figure 5: The contribution manager window.

Press the run button to execute the `GUI.pde` program. The interface (figure 6) for your oscilloscope should pop up on a new window.

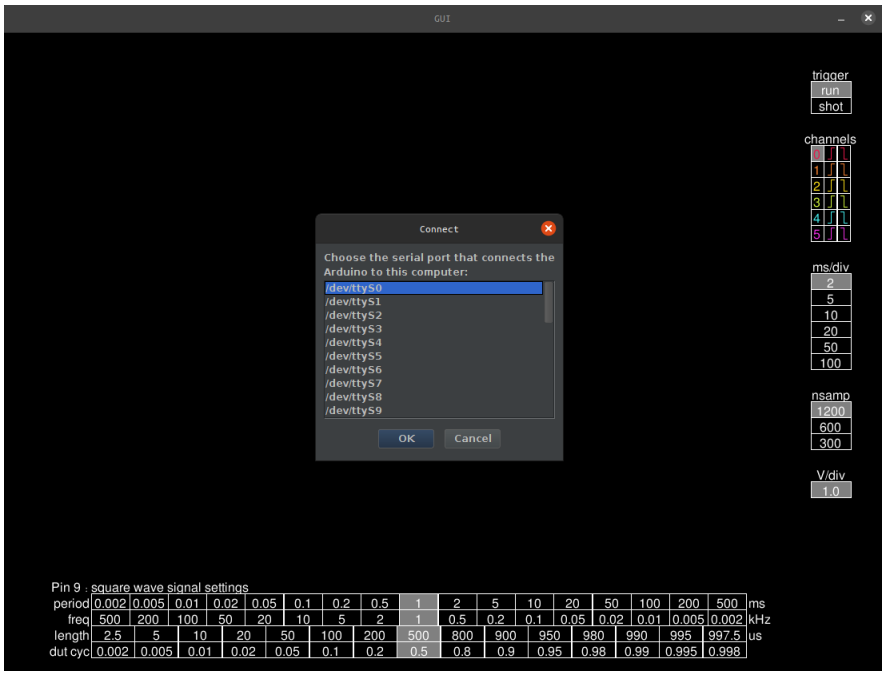

Figure 6: The oscilloscope interface will prompt you to identify the serial port that you used to connect your computer to your Arduino. This will be the same USB port you used to upload code onto your Arduino.

2 Troubleshooting

- Check the connection of your Arduino with your computer
- Make sure `uibooster` was installed
- Read the error message printed on the console of the `processing` IDE
- Ensure all previous windows were closed before opening a new one
- Ensure the correct serial port was identified
  - Tips to help you identify the correct serial port can be found here.

Read over the error message printed on the console or by your computer. Try to run your program again and see if the error is reproducible.

Avoid making changes to source code unless you were directed.

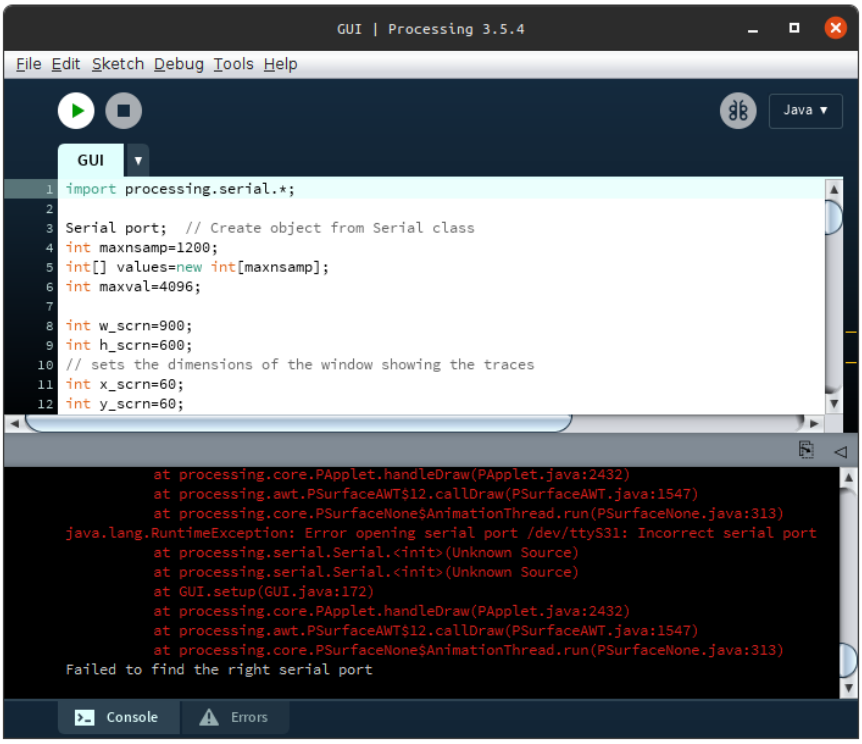

Figure 7: Error messages are printed to the console of the IDE.

3 Overview

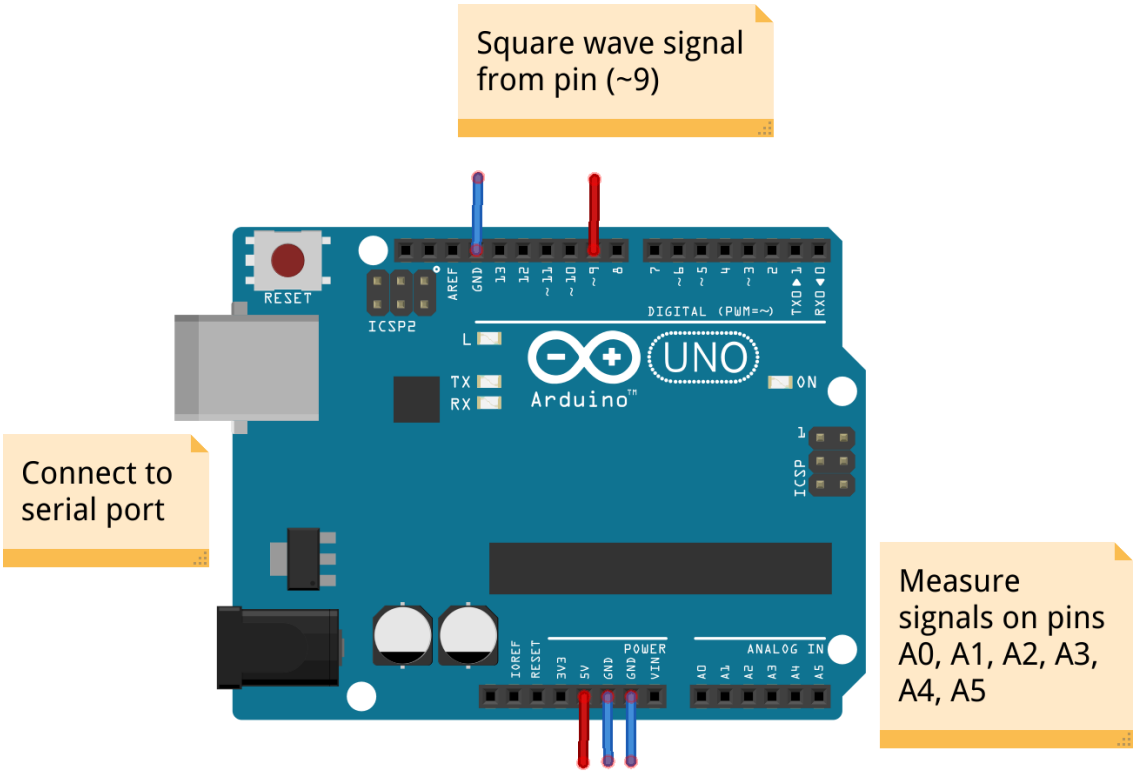

Figure 8: Three ground pins can be seen connected to blue jumper wires. The 5V and the square wave signal pins are seen connected to red jumper wires. `ANALOG IN` pins are used for measurement, seen in the bottom right corner.

The main purpose your Arduino oscilloscope is to help you visualize the voltage of any point in your circuit - as a function of time - measured with respect to the Arduino ground.

The Arduino board repeatedly measures, or samples, the voltage in each active `ANALOG IN` pin and converts each reading to a binary number through a process called analog to digital conversion. Each sample is communicated to your computer through the USB A/B cable. The running `GUI.pde` program receives each reading and plots them on the interface, forming a curve of voltage versus time called a trace.

3.1 Limitations

1. `ANALOG IN` measurement pins can only measure signals with amplitude up to 5 V. Going any higher will damage your UNO board.

2. Only trust oscilloscope readings when the input signals are between 0 and 5 V.

3.2 First Measurement: 1 kHz square wave

By default, pin 9 on your Arduino is configured to produce a 1 kHz, 5V square wave at a 50% duty cycle. We expect to see a curve similar to figure 9 when we plot the voltage of pin 9 as a function of time. Follow figure 10 to visualize the signal on the interface.

We can tell that the square wave amplitude is nearly 5 V by knowing the volts per division (shown as `V/div` on the interface) of the oscilloscope, which is 1.0 in our case. Then, each white horizontal line above the base line represents an 1.00 V increment.

We encourage you to experiment with the different options on the interface to find out what they do. What is another way to tell the amplitude of our square wave? How can we measure the period of the square wave? Did you notice anything unexpected?

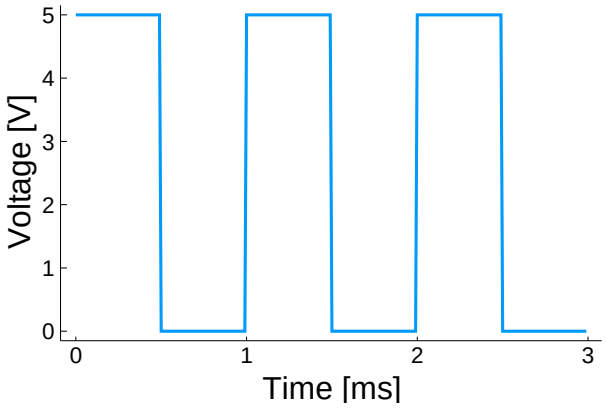

Figure 9: A 1 kHz square wave signal. 50% duty cycle means the voltage alternates between 5 V and 0 V evenly (every 0.5 milliseconds).

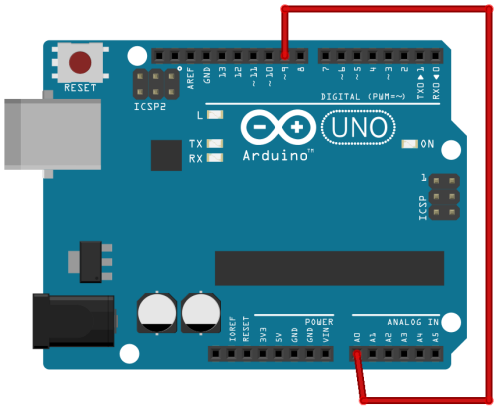

Figure 10: Voltage of pin 9 and `A0` are both referenced to the Arduino ground. Connect pin 9 to pin `A0` to see the square wave.

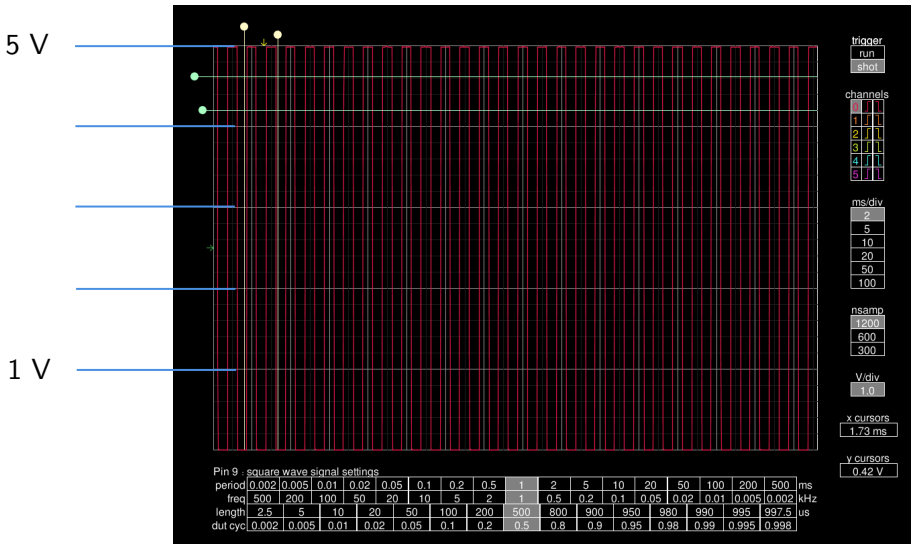

Figure 11: The 1 kHz signal produced by pin 9 seen on channel 0 of the oscilloscope.

3.3 Sharing ground

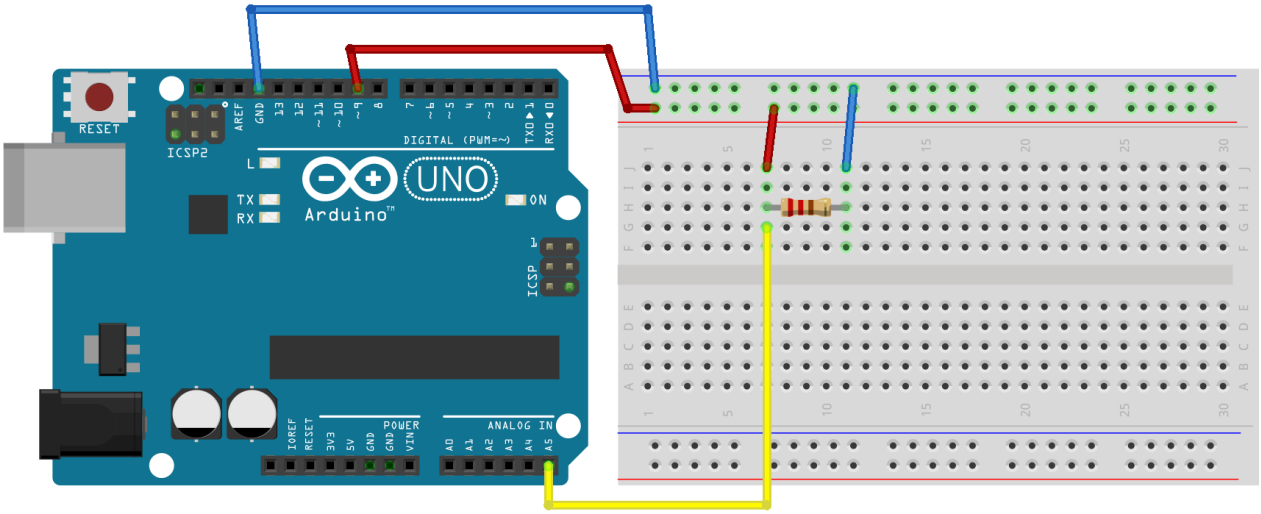

Figure 12: Circuits you are measuring should share ground with your Arduino board. Here, a blue wire connects the ground rail of the bread board to a ground pin. The supply rail is connected to pin 9. The yellow wire directs the signal from the supply rail to pin A5 to be measured by your oscilloscope.

4 Features

4.1 Square wave configuration

Pin 9 : square wave signal settings																		
period	0.002	0.005	0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10	20	50	100	200	500	ms
freq	500	200	100	50	20	10	5	2	1	0.5	0.2	0.1	0.05	0.02	0.01	0.005	0.002	kHz
length	2.5	5	10	20	50	100	200	500	800	900	950	980	990	995	997.5	us		
dut cyc	0.002	0.005	0.01	0.02	0.05	0.1	0.2	0.5	0.8	0.9	0.95	0.98	0.99	0.995	0.998			

Figure 13: The top two rows adjusts the period/frequency of the square wave output (recall that frequency $= 1/\text{period}$) from 2 Hz up to 500 kHz. The bottom two rows adjusts the pulse length and duty cycle of your signal.

To describe a square wave oscillating between 0 and x volts ($x > 0$), a common convention is to say that the square wave is "on" when it is at x volts, and "off" at 0 V.

The pulse length, L, and duty cycle, D, are both measures of the amount of time that the square wave is "on" vs "off" in every complete on-and-off cycle. They are related to the period of square wave, T, by the following relationship

$$D = \frac{L}{T} \tag{1}$$

$D > 0.5$ means that wave is "on" for more time in a full period T than it is "off". $D = 0.5$ means the wave spends 50% of the full period "on" and 50% of the full period "off" (as in figure 9).

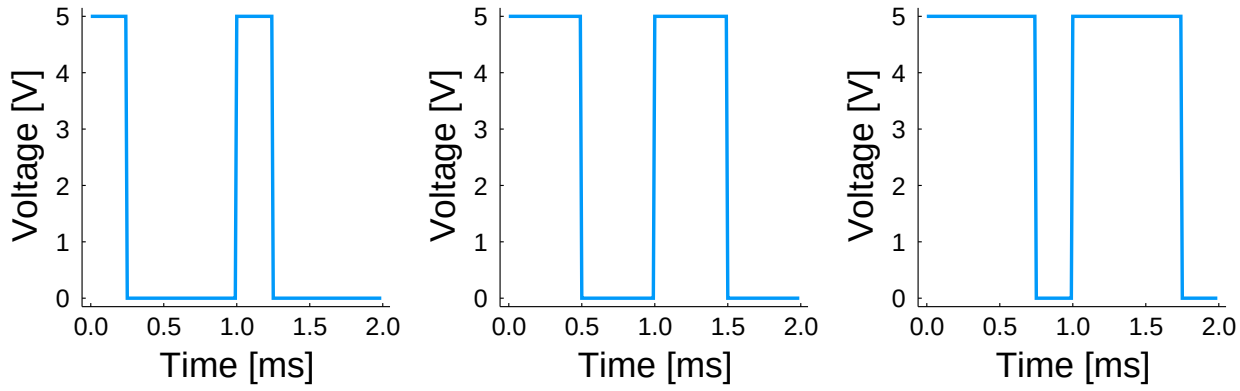

Figure 14: Duty cycle of three square waves starting from the left: 0.25, 0.5, 0.75. The pulse length increases for increasing duty cycle. The leftmost wave has pulse length $L = 0.25$ ms, the middle wave has $L = 0.5$ ms, and the rightmost wave has $L = 0.75$ ms.

4.2 Run/freeze

Figure 15: "run" and "shot" mode icons

Click on the icon to switch the oscilloscope between "run" and "shot" mode. "shot" mode freezes all signals measured on the screen.

In "shot" mode, all icons are disabled except: the icon to return to "run" mode; icons to select the number of samples be displayed on the screen; vertical and horizontal cursors.

4.3 Switching channels + triggering

Each channel number corresponds to an `ANALOG IN` pin. Clicking on the channel number actives that channel and plots its trace on the interface if it was previously inactive. One channel must be active at all times, up to a maximum of 6.

The small icons next to each channel number sets the oscilloscope to trigger on the rising edge or falling edge.

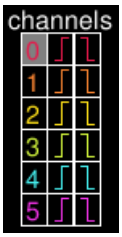

Figure 16: Channel selection icons

The same triggering configuration is shared by all channels of the oscilloscope.

4.4 Trigger level and time base

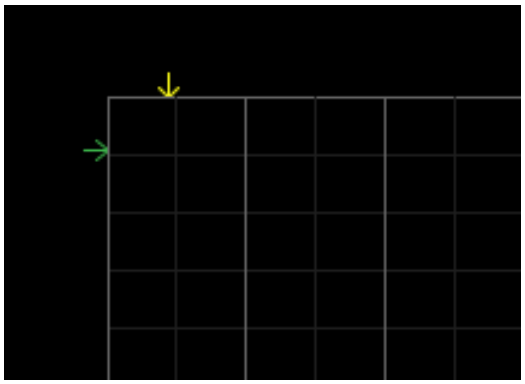

Figure 17: The yellow arrow along the upper horizontal edge specifies the trigger time position. The green arrow along left vertical edge specifies the trigger level (or trigger voltage). The arrows can be moved along their edges by double clicking on the new position you wish to move them to.

You may have noticed from measuring the 1 kHz square wave that the signal appears to be moving from one side of the grid to the other. Triggering allows us to visualize periodic signals in a steady manner without freezing them. We can expect a properly triggered signal to appear stationary, yet we are still able to small fluctuations due noise.

The only trigger option on your oscilloscope is called edge trigger. To begin, choose a trigger voltage using the green arrow. Then, decided on the trigger time position using the yellow arrow. Finally, select the option to trigger on a falling edge or a on raising edge.

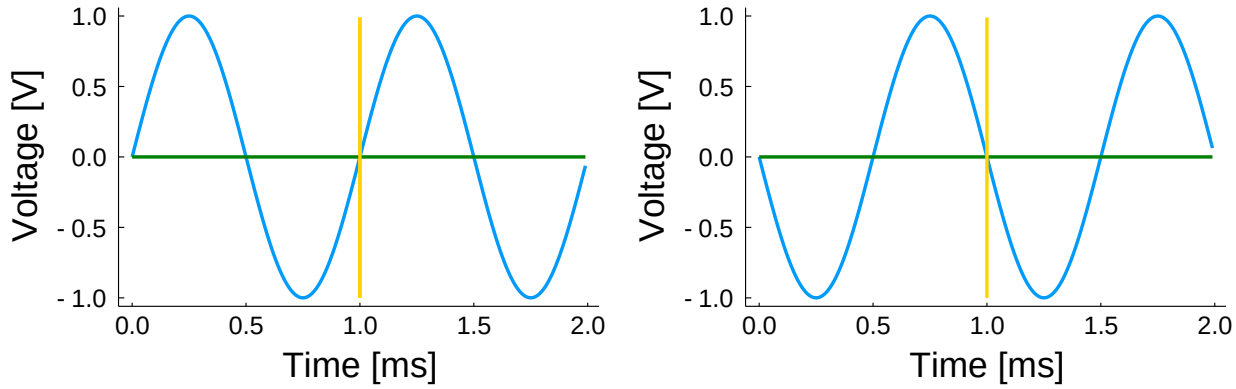

Figure 18: Trigger on a rising edge (on the left) compared to triggering on a falling edge for a sine wave signal shown in blue. The trigger level in green is at 0 V, and the trigger time position in yellow is set to the center of plots. When triggering on a rising edge, the voltage of the signal is equal to the trigger level at the trigger time position, on a region where the signal is positively sloped. Triggering on a falling edge shifts the signal so that the intersection between the trigger voltage level and time position is on a region where the signal is negatively sloped.

4.5 Time per division

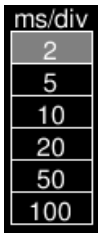

Figure 19: Icons to select time per division.

By default, the time difference between two white vertical lines on the scope is 2 milliseconds: 2 ms/div.

Selecting another icon changes the time scale of the traces. For a signal of the same frequency, a higher time per division fits more cycles of the signal into the screen.

4.6 Volts per division

Each adjacent white horizontal line represents a voltage difference of 1 V (figure 11).

4.7 Samples displayed

By default, 1200 samples are displayed on the interface at any time.

Choosing a smaller sample allows you to horizontally zoom in on the signal measured. In "shot" mode, clicking on any of the three icons repeatedly refreshes the signal.

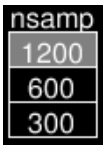

Figure 20: Icons to select number of samples displayed per trace.

4.8 Cursor measurements

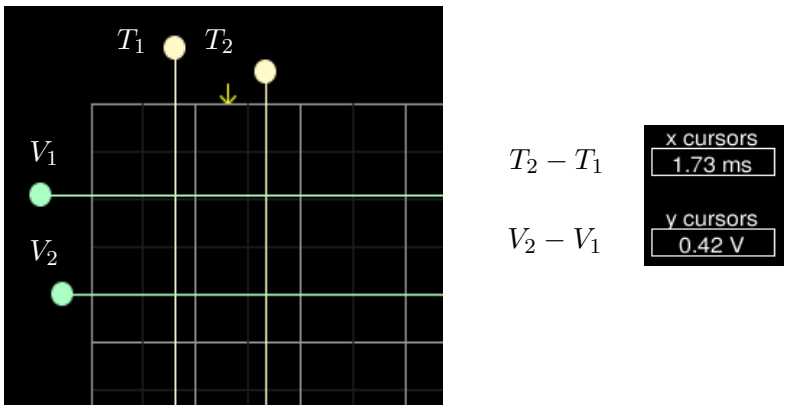

Figure 21: The mint coloured lines (called cursors) measure difference in voltage. And the beige coloured cursors measure period of time. Adjust the cursors by clicking and dragging on the circles. You can also use the `WASD` or arrow keys to fine tune the location of the last selected cursor.

4.9 Save screen and write data to file

Pressing the space bar when the interface is focused saves a screenshot of the interface and writes the voltage versus time data of each active channel into individual `.csv` files.