XY-SCOPE DRAWER

UCTRL – Final Project

Niels Keller

Table of Contents

Introduction/Background

For the final project in CMPE 3815 our group created an oscilloscope “drawer”. This drawer is able to draw multiple types of figures on an oscilloscopes XY mode display. Originally, and primarily, this drawer was designed to draw various Lissajous curve figures. These curves are created by varying frequency and phase of sinusoids. A table of some common Lissajous curves is shown in figure 1.

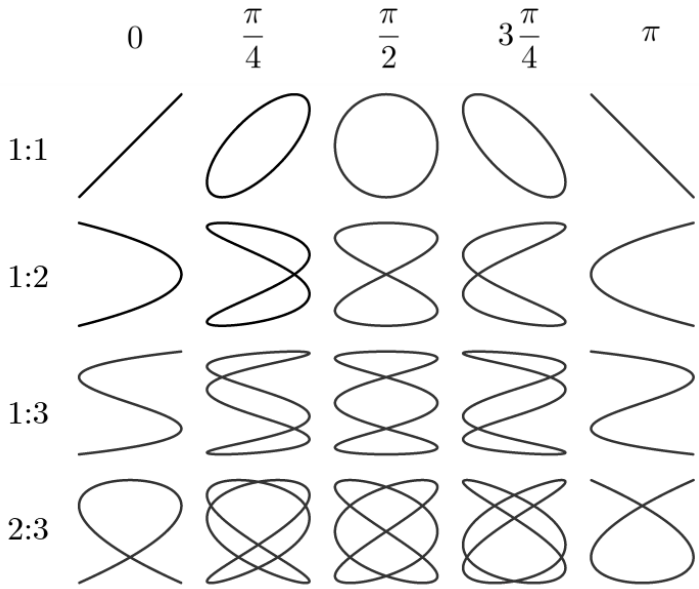

Fig. 1. Common Lissajous curves (see attributions section).

This is possible because in XY mode, oscilloscopes plot one channel against another. More specifically, at discrete time intervals the oscilloscope samples the voltage on both channels. These points are plotted. The drawer utilizes this and sends corresponding voltages to the scope. Conceptually this can be understood similarly to an Etch A Sketch.

This report includes the following topics:

1. Feature description
2. Hardware explanation
3. Software explanation
4. Development overview
5. Possible future improvements

Features

The main function of this device is to display Lissajous curves. The drawer is shown doing this in figure 2.

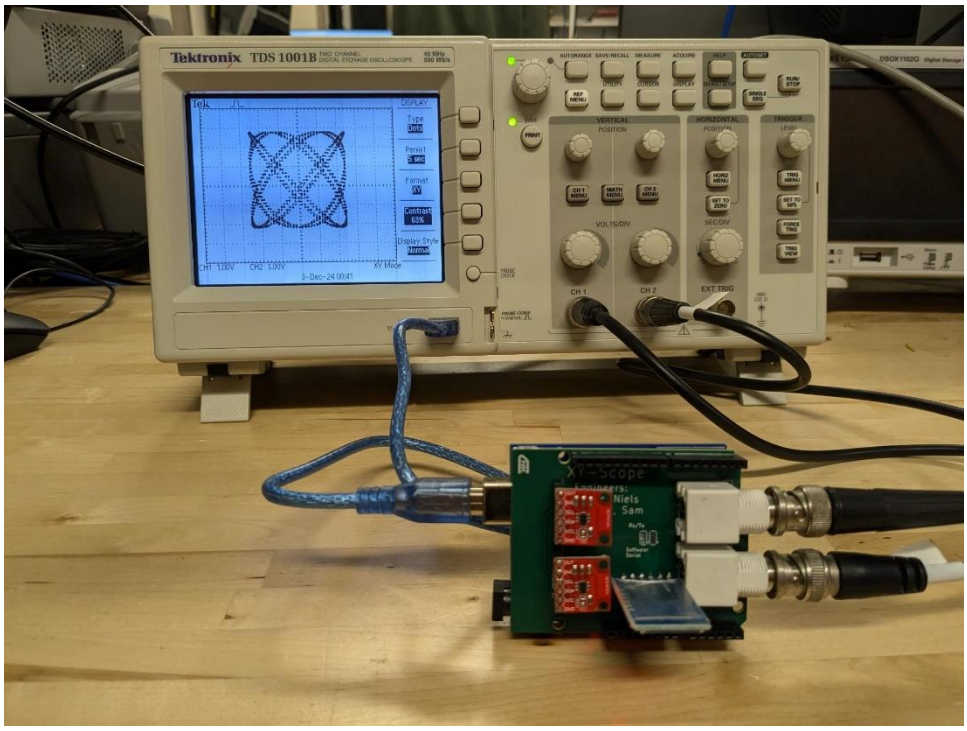

Fig. 2. Lissajous curve drawn by the drawer on an oscilloscope.

To make drawing these figures accessible an Arduino library was developed with easy to access commands, allowing these figures to be drawn in as few lines of code as possible. This library is detailed in the software section. Furthermore, displaying multiple LJ figures in quick succession gives the effect of moving. This feature is also implemented in the software library.

In addition to Lissajous curves, the drawer can also be used to draw lines. These lines are accessed with an even simpler command that the Lissajous curves, and essentially require only two points to plot between. This is covered in more depth in the software section. A demonstration with a small doodle is shown below in figure 3.

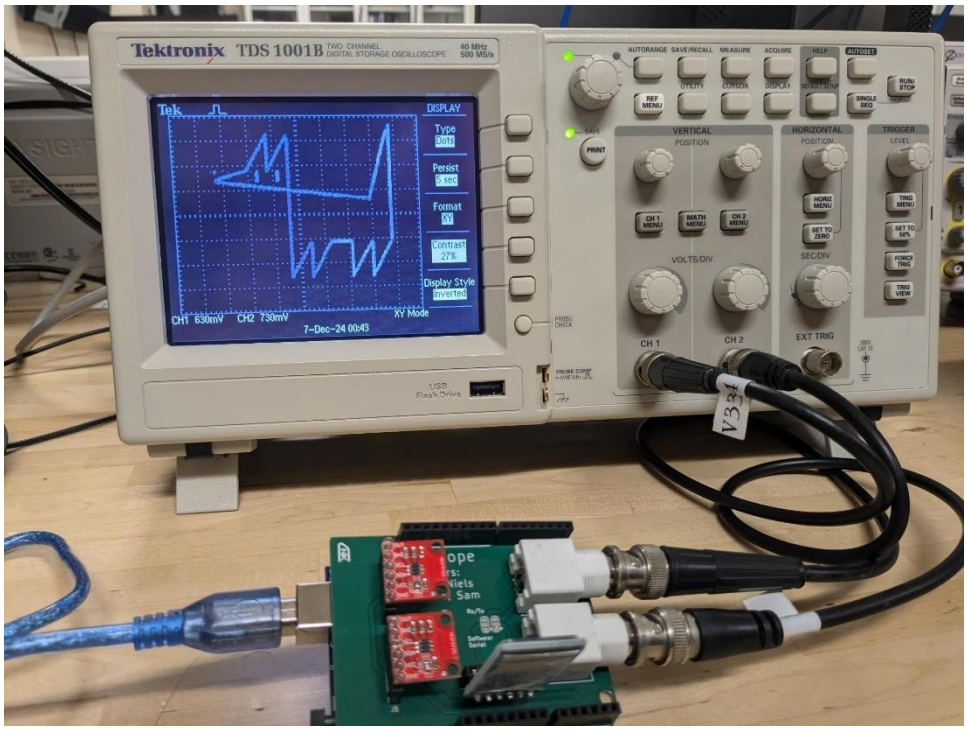

Fig. 3. Geo-Dog doodle.

Controlling the drawer is quite easy, there are two main ways to send inputs: hardcoding, and Bluetooth. The hardcoded version simply displays a single image, or multiple in a timed sequence. The Bluetooth option takes advantage of the installed Bluetooth module and allows other devices to be connected to the drawer. The current implementation uses RemoteXY, which requires the accompanying iOS/Android app to be used. This can be easily changed by cutting a trace and soldering a new bridge jumper, allowing for more versatile Bluetooth programming. This is discussed in more detail in the hardware section. The software library contains examples of both hardcoded and RemoteXY Bluetooth implementations.

Hardware

The scope drawer utilizes the following hardware:

- Arduino Uno R3
- Custom PCB Arduino Shield
- 2x MCP4725 Breakouts
- 1x JDY-16 Bluetooth module

Figure 4 shows the assembled hardware.

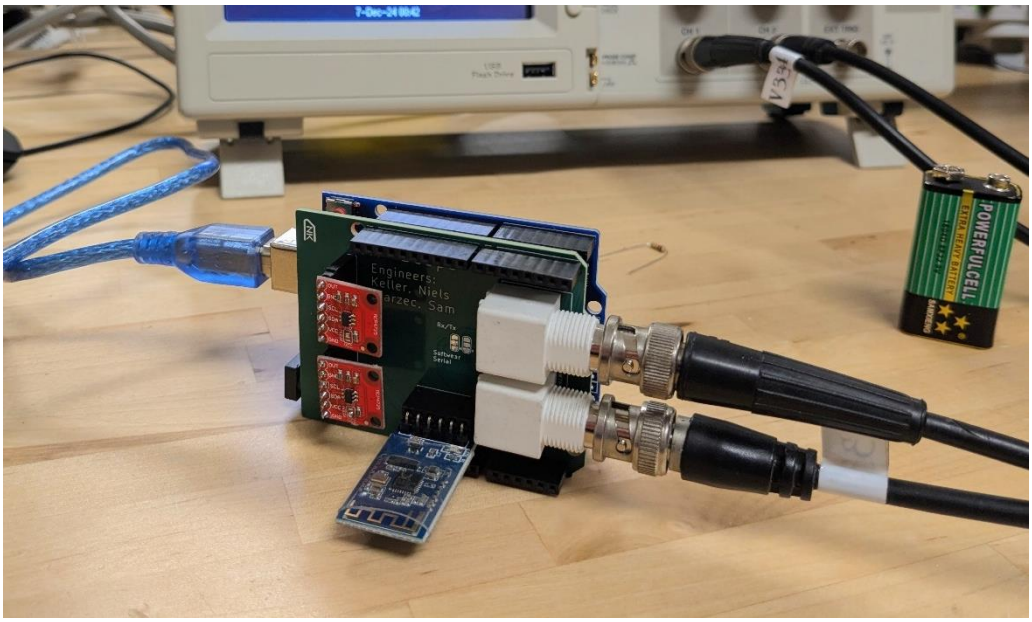

Fig. 4. Handwear closeup.

The hardware is all built on a custom PCB which holds all the other modules. Additionally, it has space to directly connect to the oscilloscope via BNC connectors. The Arduino Uno R3 on which the shield is mounted supplies power and controls both DACs via I2C. Each DAC's output is directly wired into its corresponding BNC connector. A circuit schematic is shown in figure 5.

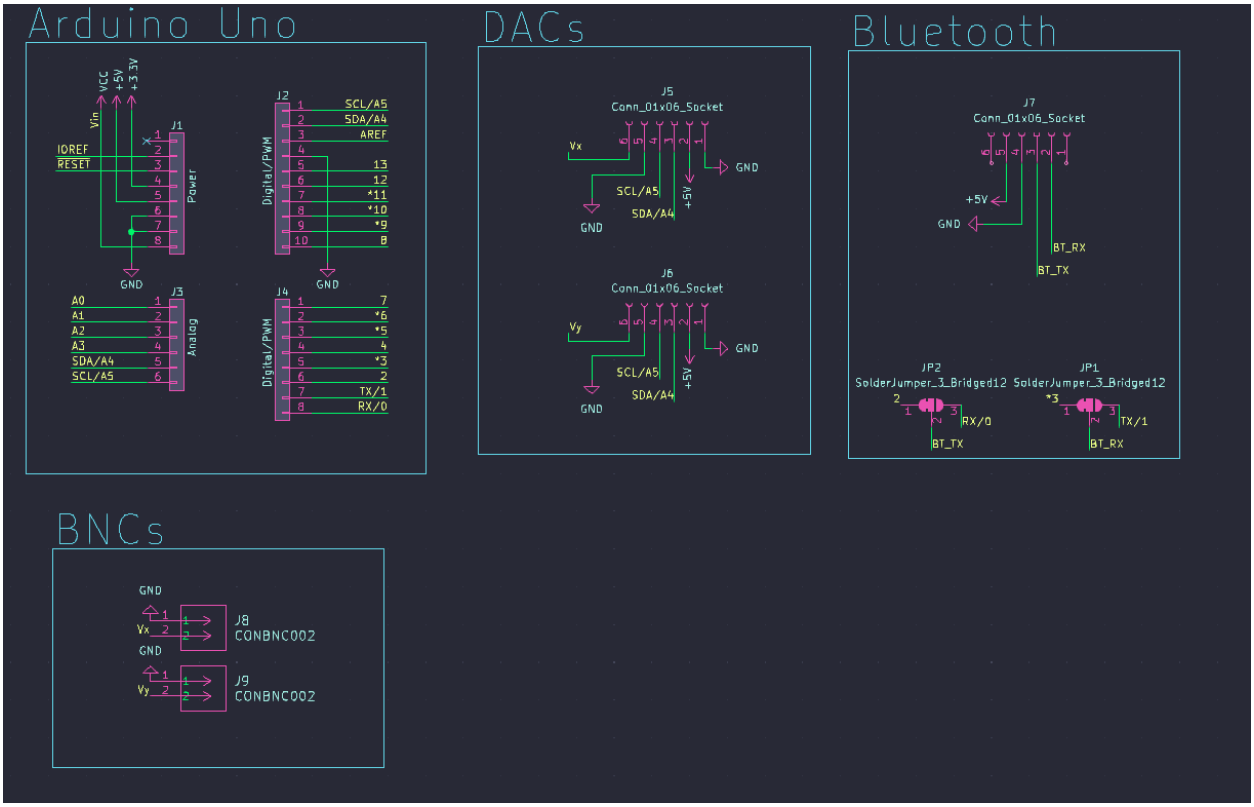

Fig. 5. Circuit schematic.

Power is delivered to the Arduino via USB cable. This also allows for the drawer to be powered from most Oscilloscopes which have USB ports for saving waveform data. In this setup, it requires no computer and runs essentially as an extension of the oscilloscope. A hardware flowchart is shown below.

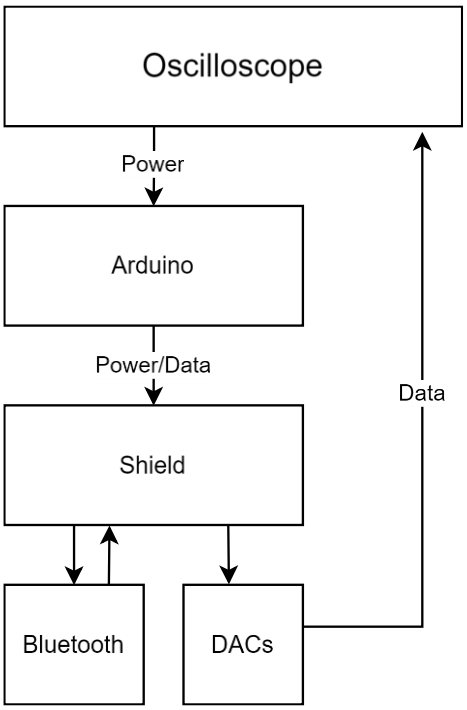

Fig. 5. Circuit schematic.

This PCB has spaces to fit each of the modules into header sockets. This design was built around ease of use and versatility of hardware. This was a requirement because of time constraints.

Software Serial vs UART

The RemoteXY library takes advantage of software-based serial. This has a number of benefits, which include being able to upload code to the Arduino while the Bluetooth module is connected. While this is useful, it can also get in the way of some sometimes. As such, the connection to the Bluetooth module can be severed and rewired to use the normal Rx/Tx pins. This is done by cutting two traces and jumping two others with solder. These pads are shown in figure 6.

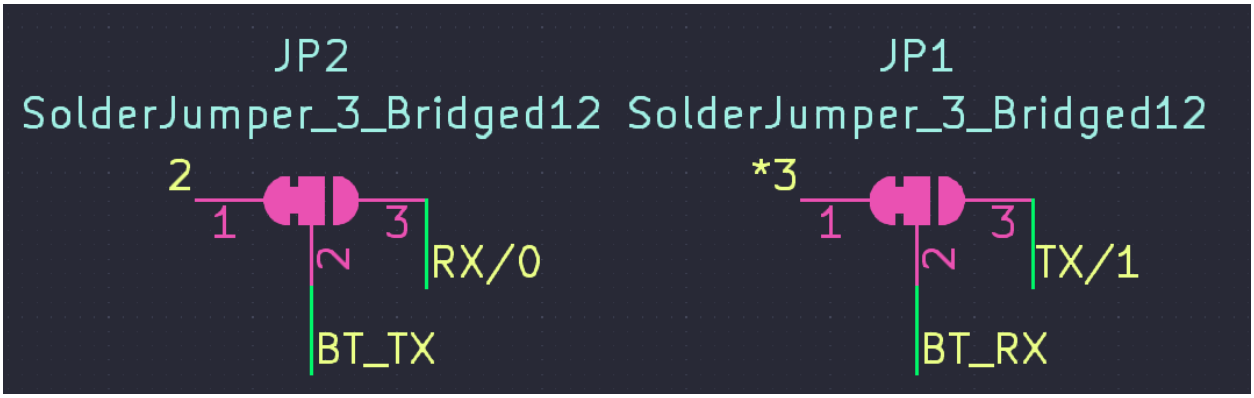

Fig. 6. Solder jumper bridge schematic.

Software

To aid in programming the drawer, an Arduino library was created. This library includes the following functions:

```
XYscope(int AddressX, int AddressY);
void dacSend(int address, int value);
void LJFigure(float ratio, float phase, int index, float fdiv);
void LJFigureMoving(float ratio, float period, int index, float fdiv);
void lineSc(int x1, int x2, int y1, int y2, long pts);
```

Detailed below are how each of the functions work, and how to use them. Additionally, examples are stored in the "examples" folder and are also accessible via the Arduino IDE after having installed the library.

XYscope(int AddressX, int AddressY)

This is the constructor. It is used to initialize the XYscope class. It accepts two integers which correspond to the addresses of the X and Y DAC's respectably. These are stored in the XYscope object.

void dacSend(int address, int value)

This method sends commands to a given DAC to set its output to a specific value. The method takes the address of the DAC that is being sent to, as well as the value to output as input parameters. Both parameters are integers. At first this seems somewhat counterintuitive for the DAC output value. This is because when sending commands to the DAC the actual voltage is never sent. Because the MCP4725 has 12 bits of resolution, the range of values allowed to be sent is 0-4095. From the data sheet, the output is given by equation 1, where Value is the input sent to the DAC:

$$V_{out} = \frac{V_{DD} * Value}{4096} \tag{1}$$

For example, if V_{DD} is set to 5 volts (as it is on the drawer) and one wanted to output 3 volts the equation could be rewritten as equation 2 and solved to be approximately 2457. This is the integer that would be passed into the value parameter.

$$Value = \left\lfloor \frac{4096 * V_{out}}{V_{DD}} \right\rfloor \quad (2)$$

Sending this data is handled by Arduino's Wire library, which handles I2C communication. I2C message structure is stipulated in the MCP4725's data sheet and example code to send the command is provided by SparkFun, the manufacturer of the breakout board.

LJFigure(float ratio, float phase, int index, float fdiv)

This method generates a Lissajous curve. The first parameter, ratio, is a float that specifies the ratio between the frequencies of the sinusoids sent to the display. Likewise, the phase parameter specifies the change in phase between the sinusoids in ($\pi *$ radians). These two parameters are all that are required to define a Lissajous curve mathematically. However, sending curves to the oscilloscope would not be feasible without the last two. First the fdiv parameter slows down the frequency. This value depends from oscilloscope to oscilloscope and requires some tuning for the curve to display correctly. This new frequency is then multiplied by index and the outcome is the input to the sinusoid function. Subsequently, the equations for the voltages at each terminal can be described in equations 3 and 4 respectively.

$$V_{out\,x} = \sin(\text{fdiv} * \text{index} * \text{ratio} + \text{phase} * \pi) \quad (3)$$

$$V_{out\,y} = \sin(\text{fdiv} * \text{index}) \quad (4)$$

This method is meant to be placed in the loop function with an incrementing index variable. The method then sends both sinusoids to the DACs each time it is called.

For example, if one wanted to display the curve in figure 2, the following code would be called in loop. This would create a figure with a frequency ratio of $\frac{5}{6}$ and a phase change of $\frac{\pi}{4}$.

```
static int i = 0;
XYscope.LJFigure(6.0/5, 1.0/4, i, 0.0075);
i++;
```

LJFigureMoving(float ratio, float period, int index, float fdiv)

This method plays a moving Lissajous curve by incrementing the phase with time. It simply calls LJFigure() with a new phase parameter based on the input parameter of period (given in seconds) and the time in milliseconds since the Arduino was switched on.

lineSc(int x1, int x2, int y1, int y2, long pts)

This method draws lines on the oscilloscope as seen in figure 3. It takes two points on a 1000x1000 plane as parameters and maps them to the oscilloscope display. It then calculates the step sizes based on the number of points. Finally, it loops and sets the DACs voltages to values along the line, incrementing by the X and Y step sizes.

Bluetooth Features

Much of the code for the Bluetooth examples was handled by the RemoteXY library. This library works with the RemoteXY website and enables the creation of GUI's which are opened on a phone The phone is then connected to the Arduino to control it. The website generates code to handle the inputs from the phone. All that needs to be done to read the data is access the data structure created by the code from the website.

For example, in the Bluetooth slider demo, one slider in the GUI corresponds to the phase of the displayed LJ figure. All that is needed to read the output is the following single line of code.

```
phase = RemoteXY.Phase/100.0;
```

Development Overview

The development of this project took place over the course of roughly three weeks. This limited the scope of the project. Additionally, it made time management an important consideration. As such the project was split up into three phases.

1. Prototyping and Testing
2. PCB Design/Assembly
3. Finalization.

Prototyping and Testing

The main goal of the first phase was to create an initial prototype. This initial prototype is shown in figure 7.

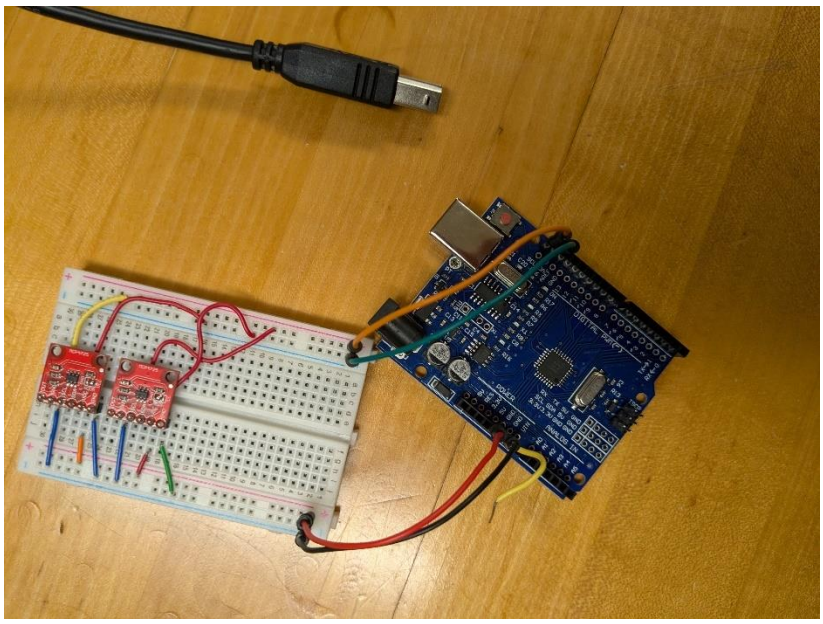

Fig. 7. First prototype

While quite rudimentary, this prototype proved to be very useful in creating initial scripts to show Lissajous curves and testing hardware configurations. This prototype only contains the Arduino and both DACs. One of the challenges during this phase was learning how to communicate with the DACs. While communicating with one DAC is rather straightforward, but without

modification, the second DAC has the same I2C address as the other. To change the I2C address, the A0 pin on the MCP4725 needs to be pulled either high or low. This sets the last bit of the address to either a “1” or “0” respectively. Luckily, the breakout board used in this project allowed for this to be down by cutting an exposed trace and bridging it to another trace with solder. This allowed the use of 2 DAC’s on the same I2C bus.

In this stage, the initial firmware was written. This firmware was mainly only used to test hardware viability. This would be finalized in later phases. This phase was concluded when basic Bluetooth was implemented.

PCB Design/Assembly

Due to the quick turnaround time of the project timeline, it was important to get the PCB design done as soon as possible to get the board in time. Because of this, the PCB was designed before the software was finalized. Another consideration was debugging time. An important part of all projects, this was one of the main considerations while designing the PCB. This was the leading reason that headers with breakout boards were chosen for the PCB. From this step, the circuit in figure 5 was designed in KiCad. Figure 5 is shown again below for convenience

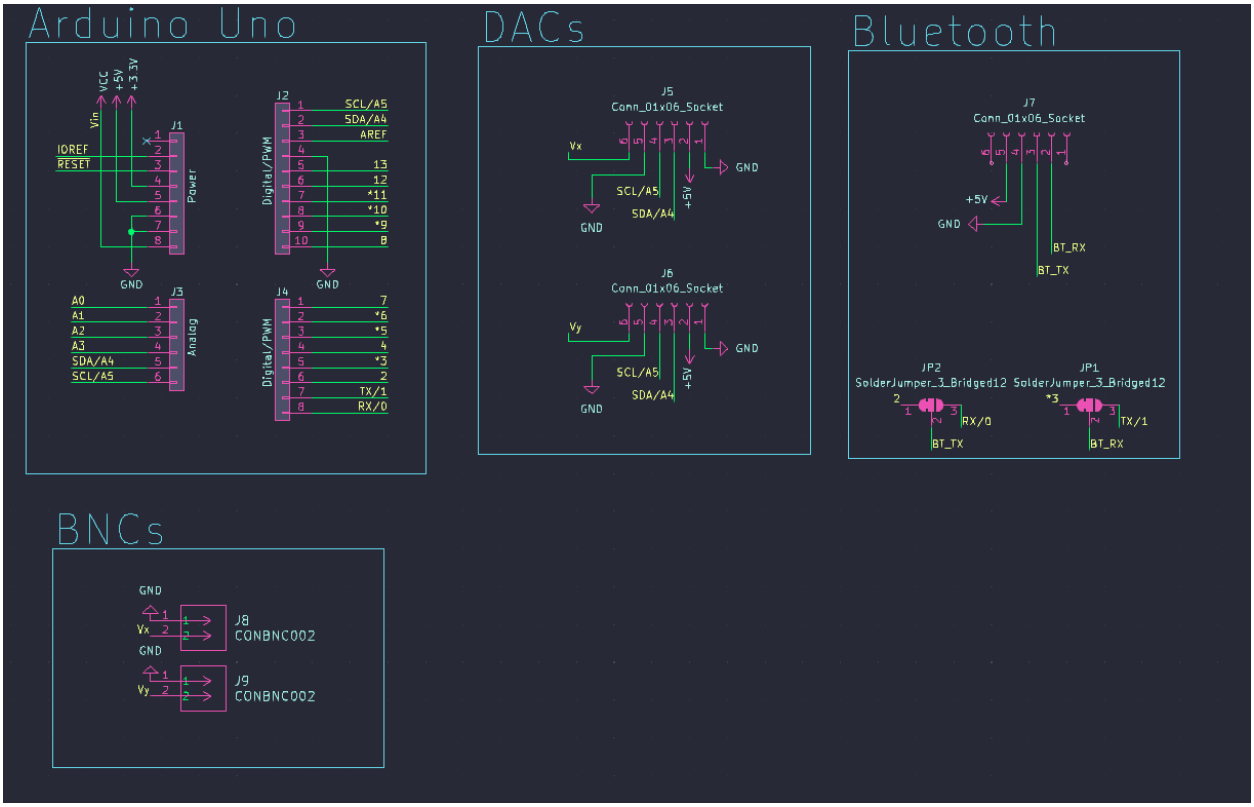

Fig. 5. Circuit schematic.

With the schematic finalized, the PCB was laid out. The finalized board layout is shown in figure 8.

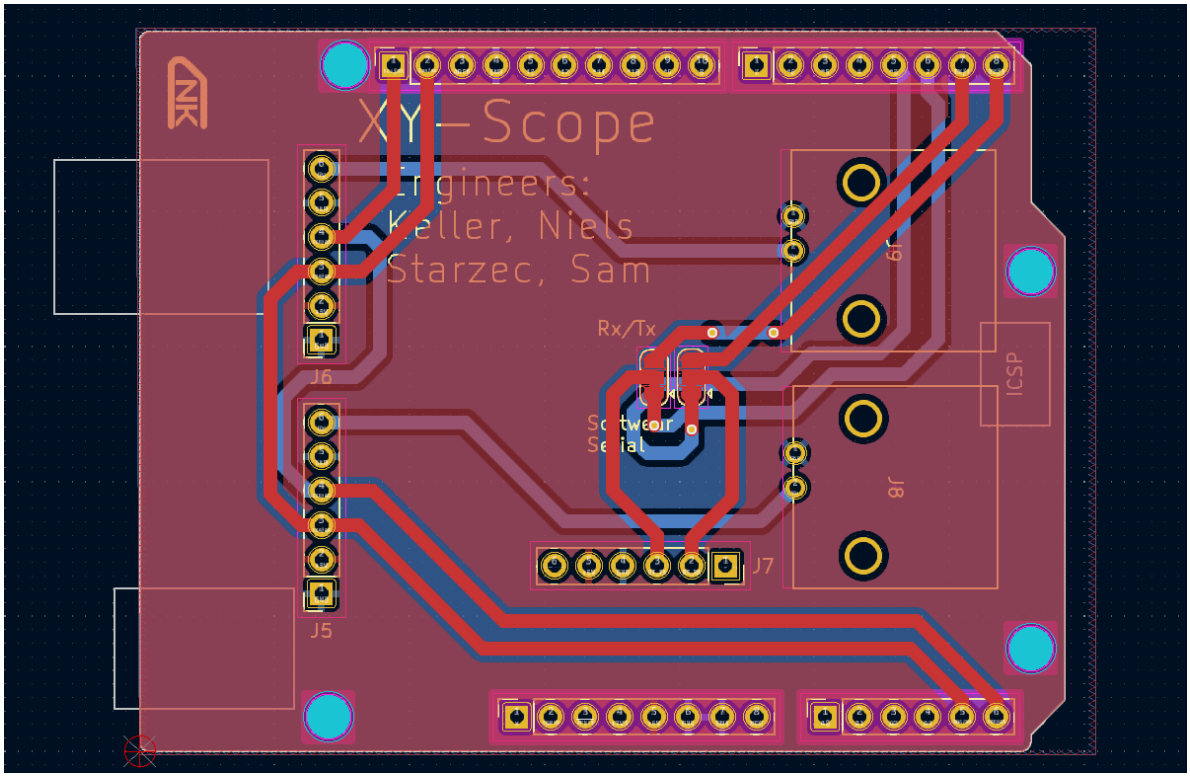

Fig. 8. PCB layout.

Finalization

The main objective of this phase was to finalize the code. Until this phase, most of the effort was focused on sending out the PCB. In this phase it was possible to use the breadboard prototype to develop and test all desired features. Additionally, this was the phase in which the Arduino library, alongside several demo and example programs were written.

Possible future improvements

This project was very successful and exceeded the requirements stipulated in the project proposal. However, due to time limitations some aspects of the project have room for improvement.

PCB

The PCB designed for this project is somewhat basic and does not hold all the hardware for this project. Instead, it uses breakout boards connected to pin headers. Had there been more time for this project, it would have been useful to design a board which doesn't require breakout boards, and instead incorporated all the required integrated circuits and support hardware on the same board.

Issues with the MCP4725

Throughout the development of this project, there were some issues with the MCP4725's update rate. Through basic testing the update rate was measured to be around 500-1000 Hz. Initially this was believed to be a limitation of the MCP4725 itself, however, the datasheet specifies a typical settling time of 6µs. This corresponds to roughly 160 kHz. Additionally, both the DAC and Arduino's I2C clock rates are around 100KHz which is much faster than what was measured. Currently our group's theory is that the update rate is limited by the Arduino's capability to process new points to send to the DAC.

Further testing is required into this issue, as the main limitation to drawing larger pictures is the rate at which points are plotted. This is because the drawings appear clearest when the point persistence is limited to only a few seconds. Otherwise, if the points are set to persist indefinitely, random noise can cause distortions in the image. These distortions are infrequent, and as such disappear quickly on lower persistence times. Additionally, moving or changing images require the presence to be set very low.

Microcontroller selection

Another aspect of this project that could have been changed would be the microcontroller used. For instance, other microcontrollers have built in DACs which might provide a faster output. Alternatively, the Arduino Uno R3 could have been replaced with an ESP32 which has a much higher clock frequency, and as such would perform the calculations for each of the points much faster.

Conclusion

In summary, this report outlines the features and limitations of the scope drawer. It provides insight into how the software and hardware work. Alongside the report, PCB design files are provided which simplify reproduction of this circuit. Finally, an Arduino library with examples was created to aid in the programming the drawer.

Attributions and Resources

- Figure 1 attribution:

  By Vhastorga - Own work, CC BY-SA 4.0,
  https://commons.wikimedia.org/w/index.php?curid=98659598

  Accessed 12/10/2024

  This work has been modified by cropping.

- SparkFun breakout board documentation
  https://learn.sparkfun.com/tutorials/mcp4725-digital-to-analog-converter-hookup-guide

- GitHub Repo
  https://github.com/NielsKeller/ScopeXY