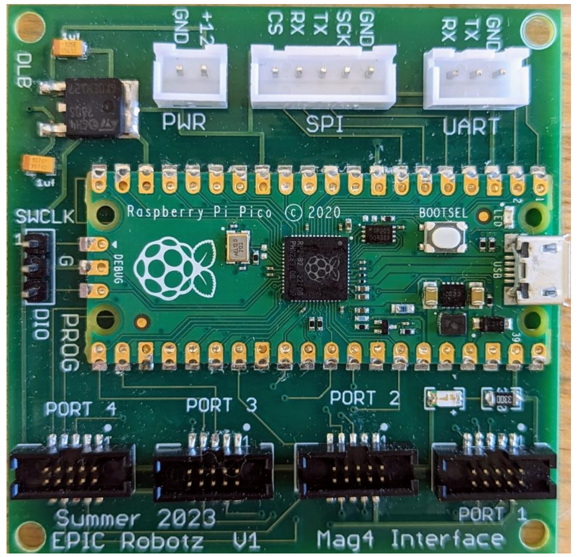
Mag4 Interface Device

EPIC Robotz

July 2023



The Mag4 Interface Device

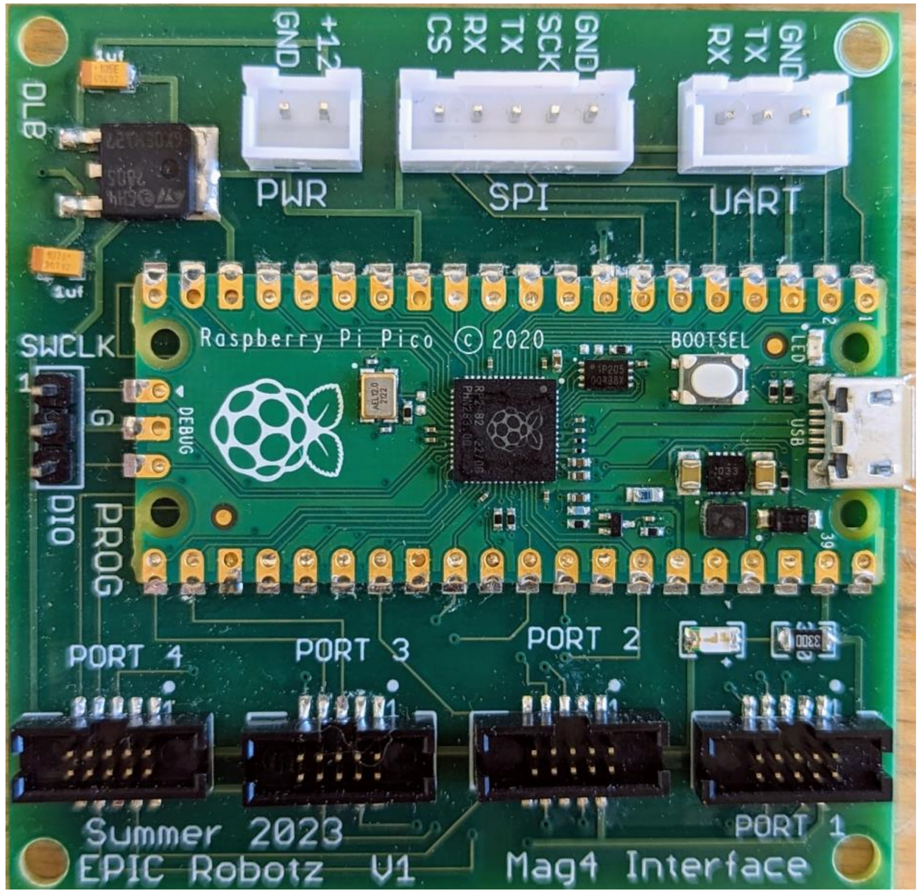
Input Power

SPI Interface to RoboRio

UART (unused)

3 Wire Debugging Port

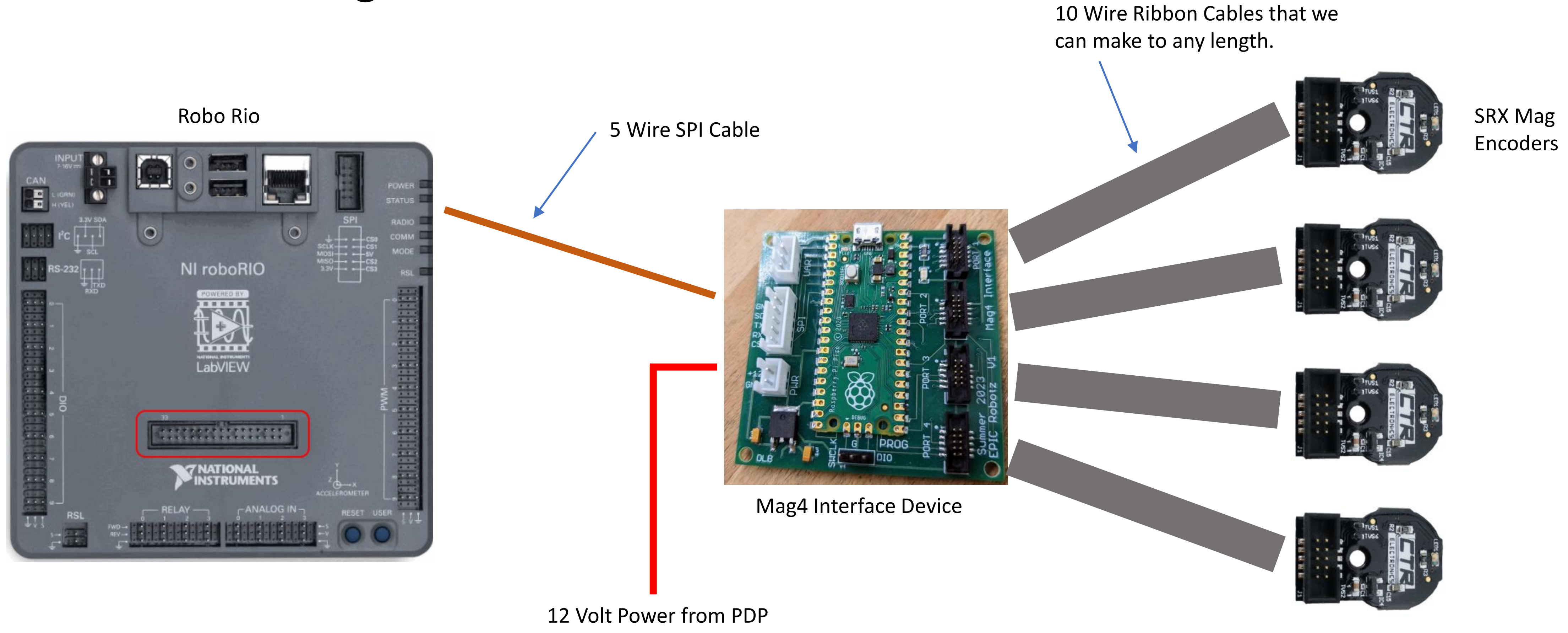
USB Port for Programming



4 SRX Mag Encoder Ports

Purpose of the Mag4 Interface Device

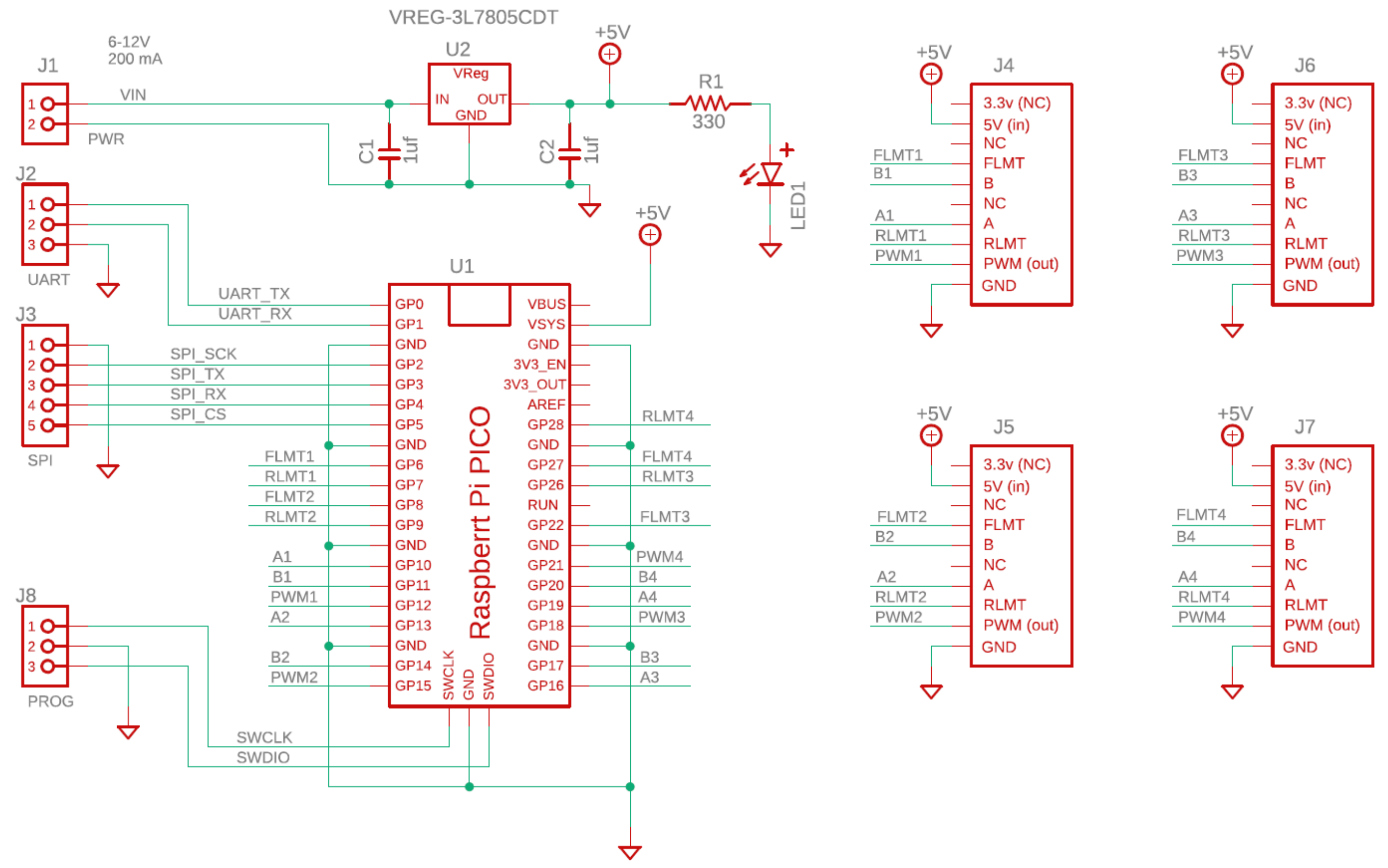
Connect 4 SRX Mag Encoders to the RoboRio:



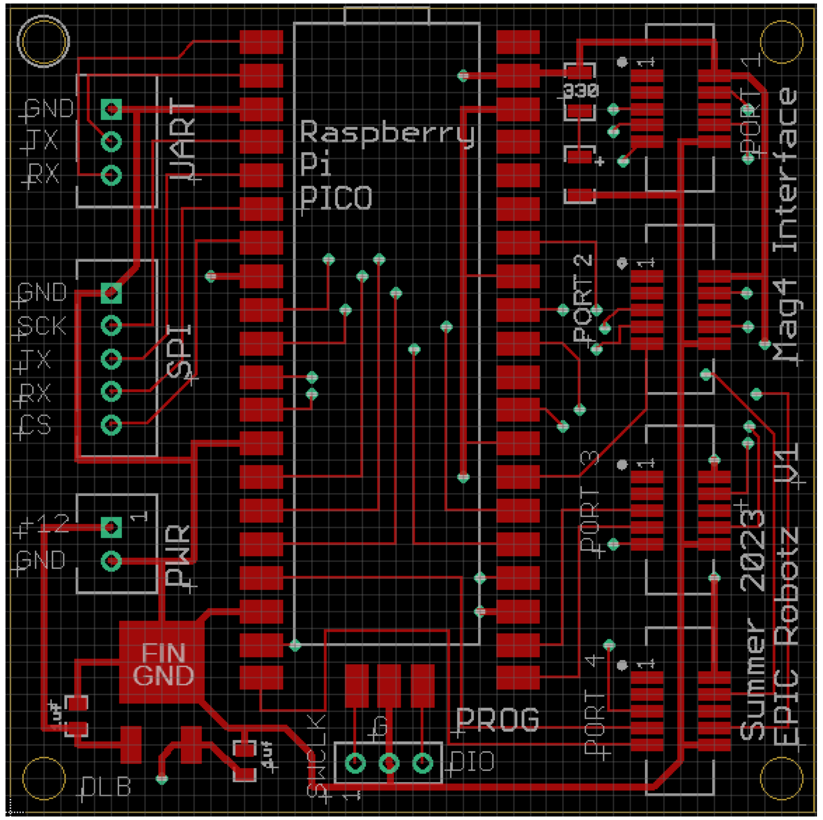
Justification of Project

- Current CANCoder cost $80 Each.
    - SRX Mag Encoders cost $40.
- CANCoders crowd the CAN Bus
    - SRX Mag Encoders avoid CAN Bus by using SPI
- CANCoders require individual Power Wires, as well as CAN wires
    - SRX Mag Encoders require single ribbon cable
- Latency Is better with SRX Mag Encoders
    - 0.3 milliseconds deterministic Latency
    - (However, Latency is NOT known to be a problem with CANCoders)
- There is possibility of fusing PWM and Quadrature Readings At Source
    - More accuracy for reading absolute angles quickly

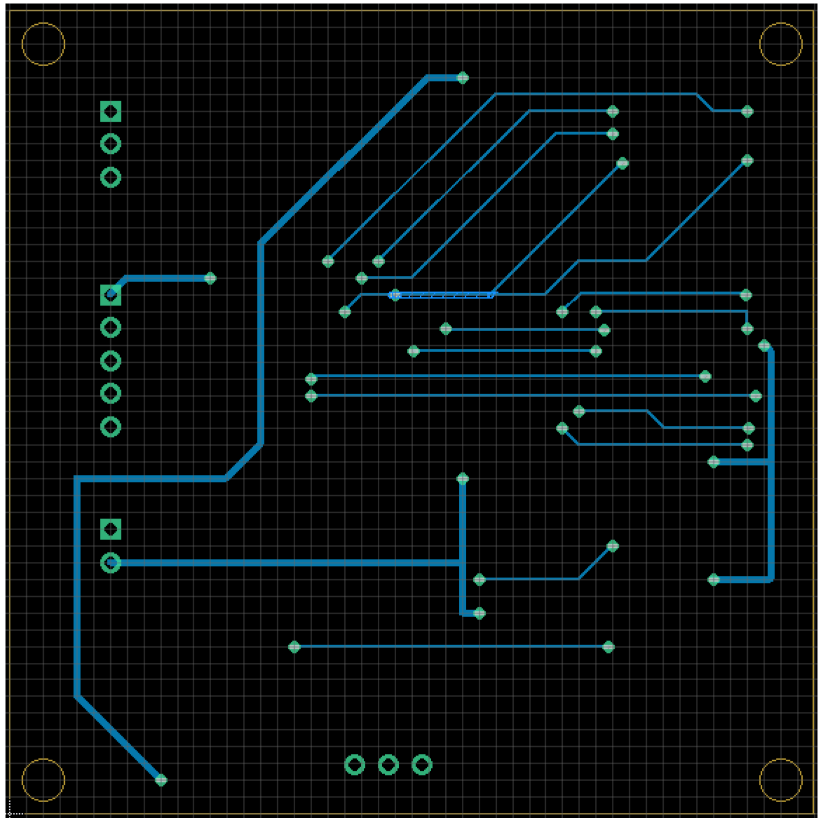
Mag4 Schematic



Mag4 PCB Layout



Top Copper and Silkscreen



Bottom Copper and Silkscreen

Bill of Materials

```
1x PCB                          -- From AllPCB
1x Raspberry Pi PICO Board      -- From Amazon
4x 10-Pin Socket                -- DigiKey, 1175-2649-1
1x LED, 1206                    -- DigiKey, 732-4989-1
1x VReg 5V 1.5A, DPAK           -- DigiKey, 497-7255
1x 330 ohm Res, 1206            -- DigiKey, RMCF 1206FT330RCT
2x Cap Tant, 1 UF, 1206         -- DigiKey, 478-2363
1x JST male Socket, 2 Pin       -- From Amazon
1x JST male Socket, 3 Pin       -- From Amazon
1x JST male Socket, 5 Pin       -- From Amazon
1x 3-Pin Header, 0.1 Pitch      -- From Amazon
```

Why use a Raspberry PICO?

(It’s a perfect fit for the Job!)

- PICO is a new microprocessor that has a unique “programmable IO” block, which allows very fast sub-programs to monitor and react to incoming IO signals.
- To correctly interface with a SRX Mag Encoder, three inputs must be monitored: Two Quadrature digital signals, and one PWM signal.
- It turns out that the PICO has 8 state machines on it’s programmable IO block. Four of these are dedicated to monitoring the four PWM signals. The other four state machines are used the count the quadrature pulses.
- The PICO also has two independent cores. One core can be dedicated to collecting the data, while the other core can be dedicated to communication with the host.

Software

Software to Simply Use the Mag4

Copy the java class “Mag4” to your Project. It is found under ../src/main/java/frc/robot/Mag4.java.

In your Robot Code:

```
        import frc.robot.Mag4.Mag4Data;
```

In your code, create a Mag4 Object in your robot code:

```
        Mag4 mag4 = new Mag4();
```

When you want the data from the Mag4 interface device, do:

```
        Mag4Data data = mag4.getData();
        float angle1 = data.angles[0];
        // similar for other parameters)
```

getData() will require about 0.3 milliseconds to run.

The returned data will have a latency of just over 0.3 milliseconds.

The data object returned contains the pulse count and angle measurements for all four encoders.

Note that Mag4 does not run in the background or use multithreaded objects.

Software for Development

- Two Categories: RoboRio and Raspberry Pi PICO
- Setup for RoboRio according to FRC instructions.
  - Must use VSCode configured for RoBoRio:
- Setup for Raspberry Pi PICO very Involved.
  - Must use VSCode configured for PICO:
  - I installed “Pico-v1.50” SDK for Windows
  - Pay special attention to environment variables:
    - PICO_INSTALL_PATH
    - PICO_SDK_PATH "C:/Program Files/Raspberry Pi/Pico SDK v1.5.0/pico-sdk“
    - I set PICO_SDK_PATH in pico_sdk_import.cmake file.
      (Not recommended, but no good alternative).

Software Roadmap

- For RoboRio
  - Main files are: (Found under: ../src/main/java/frc/robot)
    - Mag4.java -- Contains the class that interfaces with the Mag4 device
    - Mag4Tester.java – Contains code for test board
    - Robot.java – Contains code to include Mag4Tester
- For PICO
  - Main production code is under Mag4Pico/mag4
    - USE THIS TO BURN NEW DEVICES
  - The code on the PICO used to control the stepper motor for testing is under Mag4Pico/motordrive
    - This code can be used to control a generic stepper, both with a UART and USB at the same time.

Development Software for PICO

Name	Purpose	Status
adder	Experimenting with PIO blocks	Old, not important
blink	Misc Experiments	Old, not important
encoders	First versions of PWM reading	Obsolete
loopirq	Experiments with Interrupts	Useful as examples
mag4	THE MAIN CODE	Used in production
magreader	SPI Experiments	Obsolete
motordrive	Stepper Motor Driver	Used for Test Board
pio_blink	Test ability of PIO to issue interrupt	Keep as an example
pwm_rd4	For development of pwm reader. Written in c++	Obsolete
pwm_reader	Earlier version of pwm reader	Obsolete
stepper	Earlier version of motordrive. Used with testman.	Old, but used with testman
testman	Python Program to control Stepper	Keep for Example

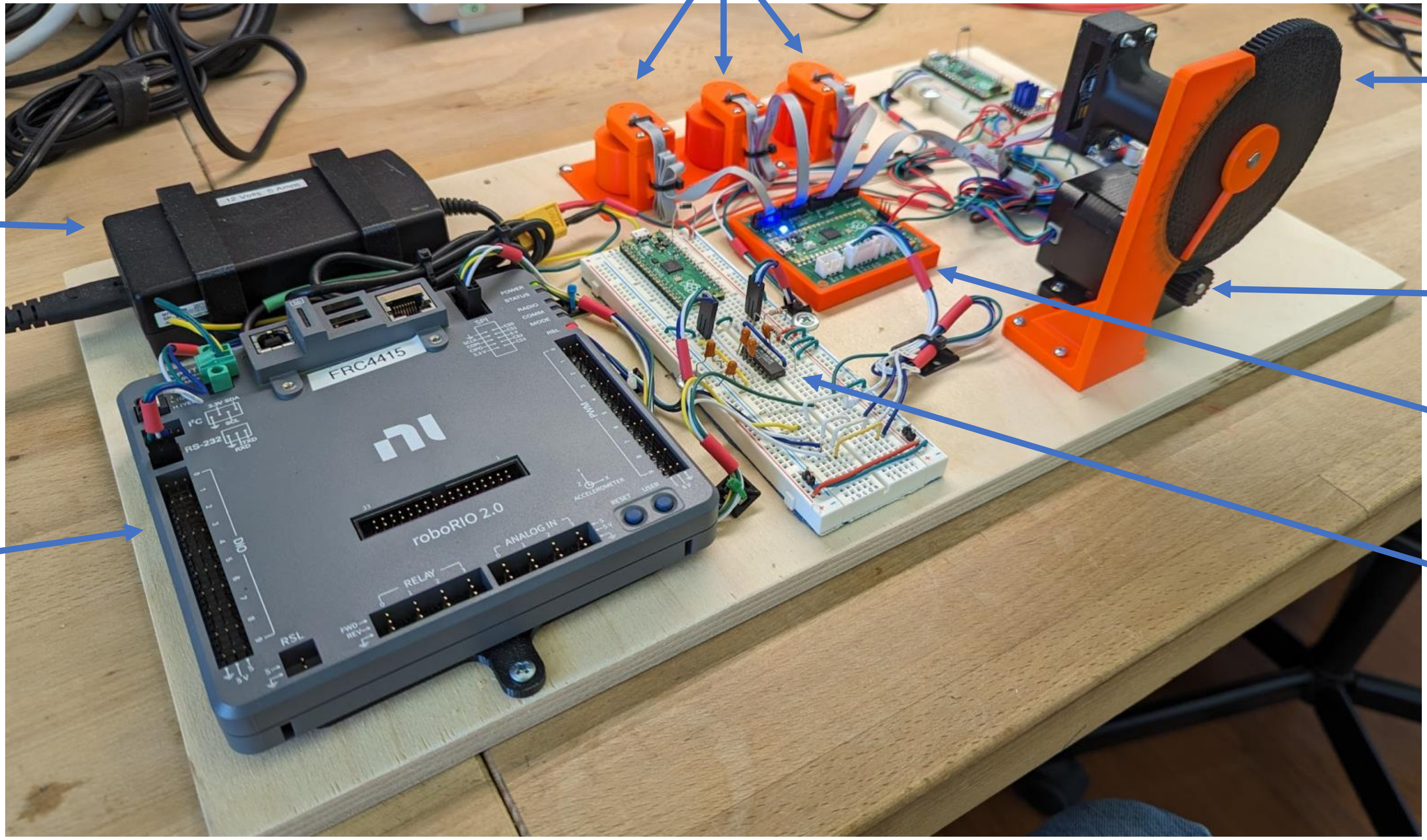
Thoughts on PICO Development

I LOVE THE PICO – Definitely my Go-To microprocessor!

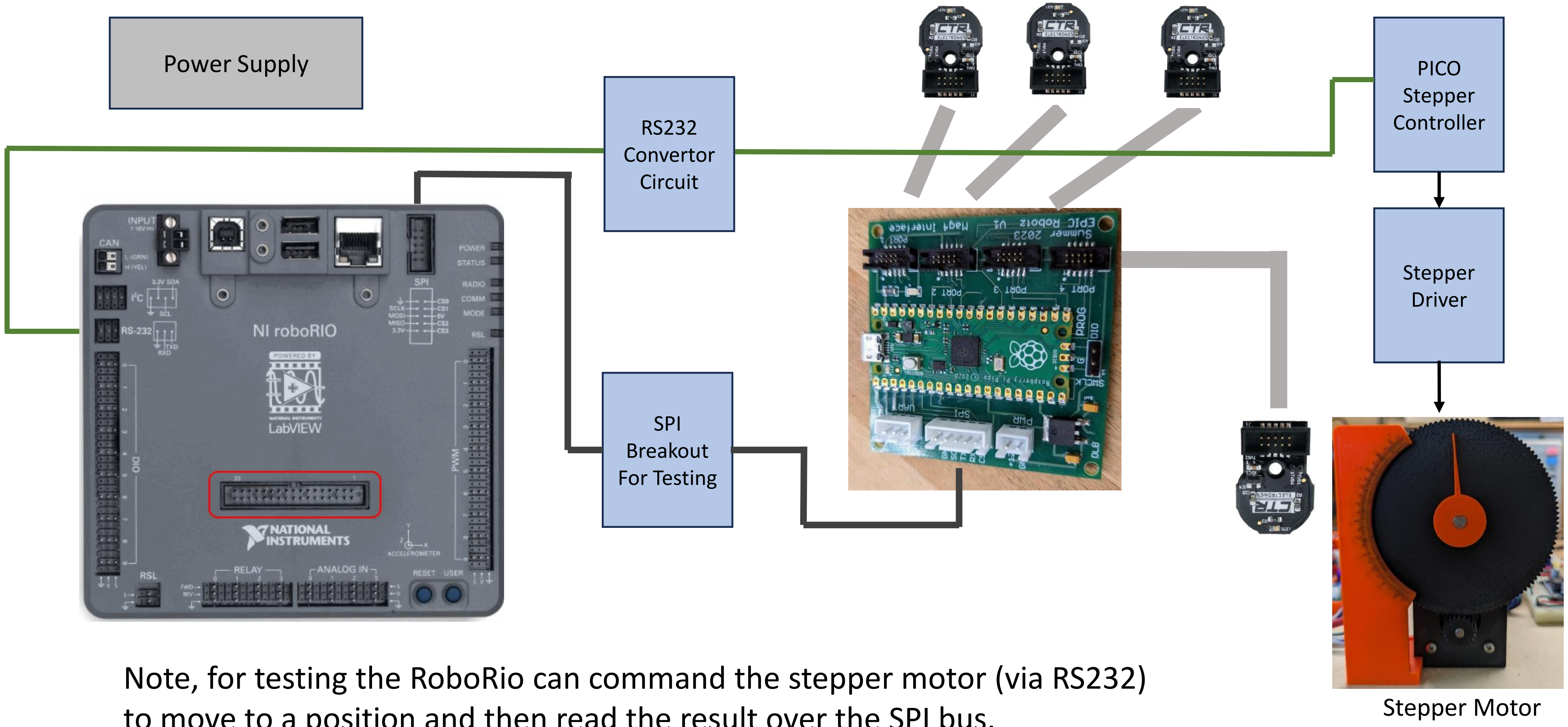
- Install SDK for Windows.
- Must get environment variables set right!
- Must use a preconfigured VSCode. Its tricky on how that is done.
- The C SDK for PICO from Raspberry is excellent!
- Understand what CMake is doing… CMake must be used.
- IDE Debugging works for one of the two cores if using a picoprobe
  - DOES not work for both cores!
- I tried the Arduino environment. Works for basic stuff, but many PICO specific features are not supported. Best to stick with C-SDK from Raspberry

Testing the Mag4 Device

Mag4 Testing Board



Block Diagram of Testing Board



Note, for testing the RoboRio can command the stepper motor (via RS232) to move to a position and then read the result over the SPI bus.

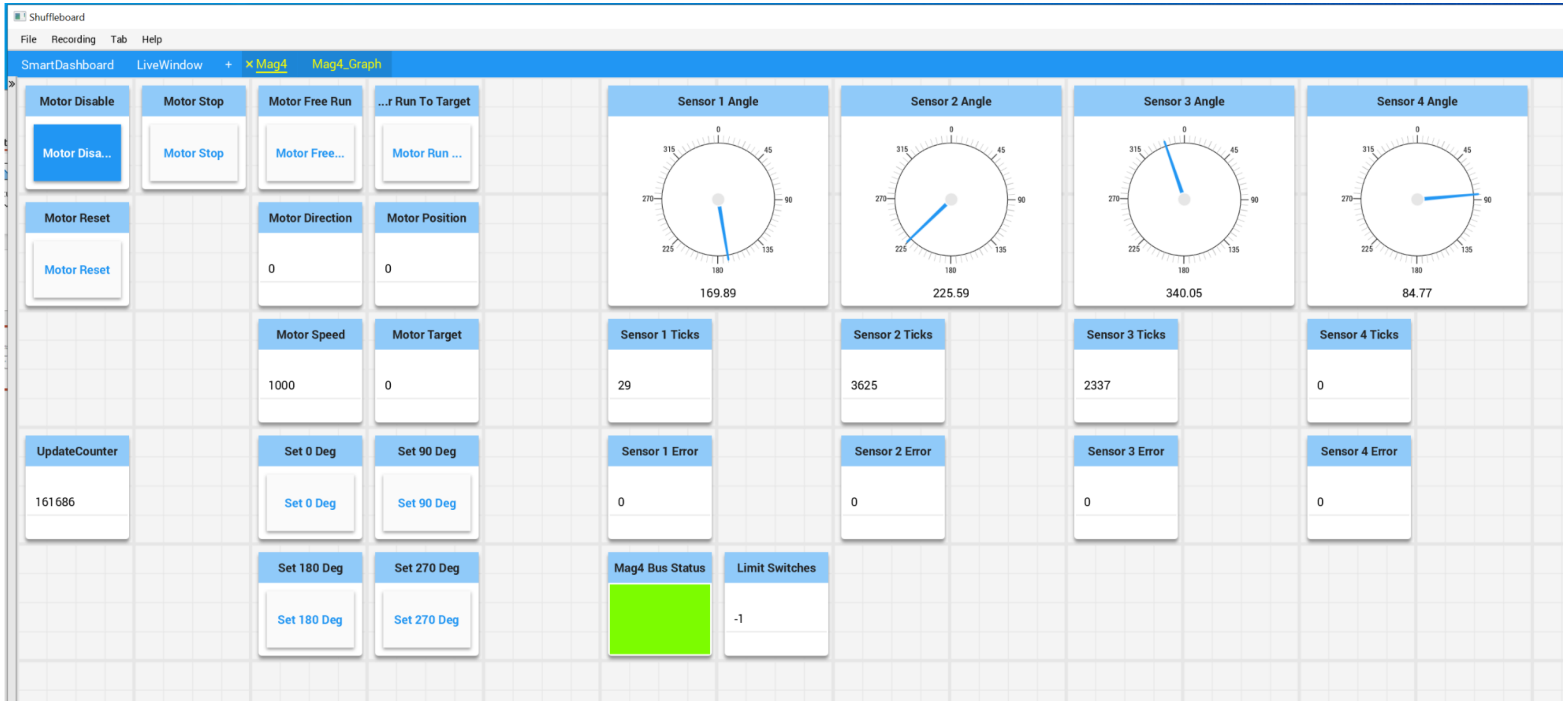
Running the Test Board Software

- Use the FRC DriverStation and ShuffleBoard
- A tab called Mag4 will appear on the ShuffleBoard
  - Under this tab, you can move the motor and watch the results.

    Note that you do NOT need to be in “Enable Mode” for this to work,

    As the RS232 interface bypasses FRC safety guidelines.
- A tap called Mag4_Graph will also appear given graphs of the movement and sensor readings for Port 4.
- Note that the buttons on the ShuffleBoard do not respond well… it might take some extra clicking to get the behavior you want.

Example Test Results



Motor Disable

Motor Stop

Motor Free Run

...r Run To Target

Motor Reset

Motor Direction: 0

Motor Position: 0

Motor Speed: 1000

Motor Target: 0

UpdateCounter: 161686

Set 0 Deg

Set 90 Deg

Set 180 Deg

Set 270 Deg

Sensor 1 Angle: 169.89

Sensor 2 Angle: 225.59

Sensor 3 Angle: 340.05

Sensor 4 Angle: 84.77

Sensor 1 Ticks: 29

Sensor 2 Ticks: 3625

Sensor 3 Ticks: 2337

Sensor 4 Ticks: 0

Sensor 1 Error: 0

Sensor 2 Error: 0

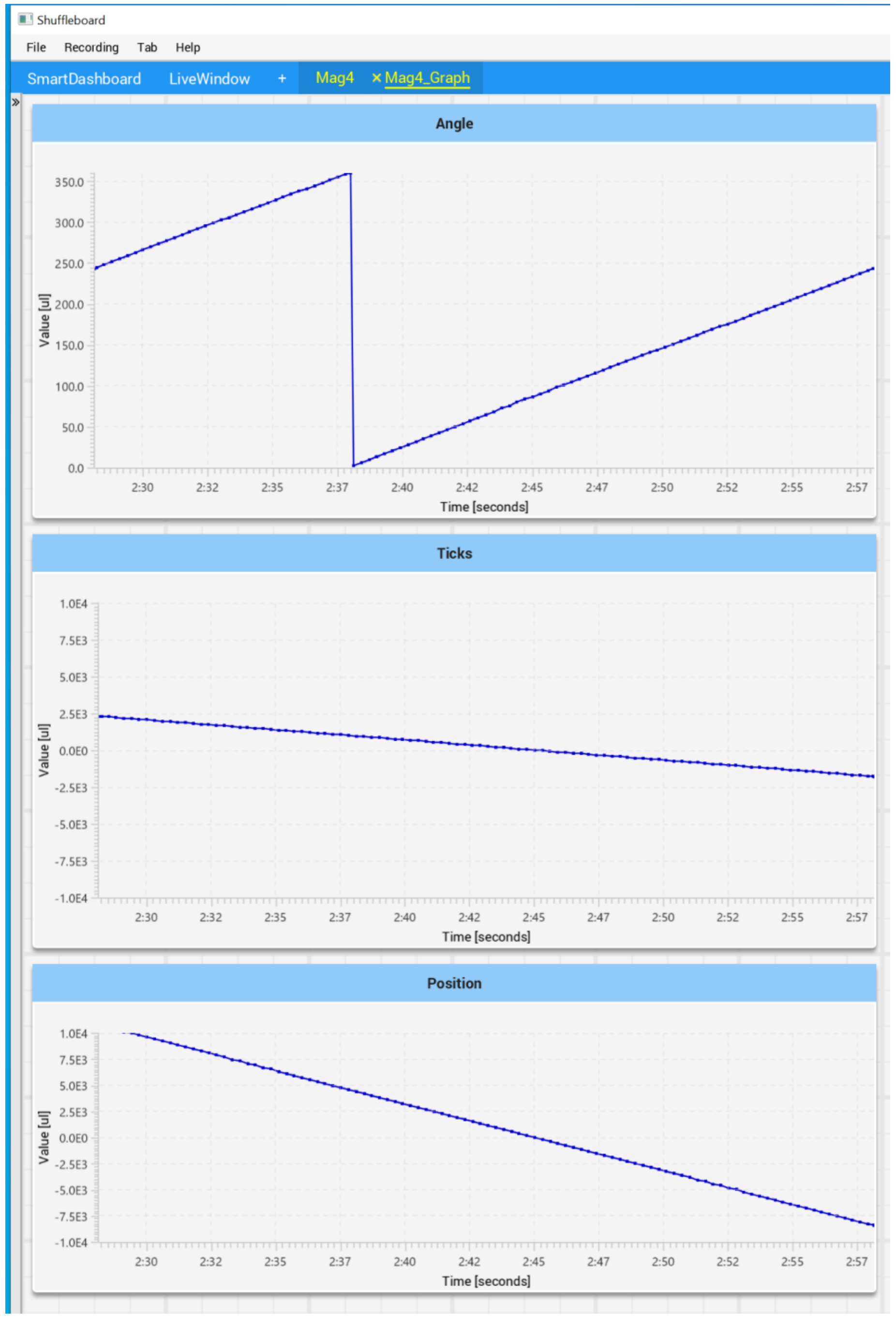
Sensor 3 Error: 0

Sensor 4 Error: 0

Mag4 Bus Status: 🟩

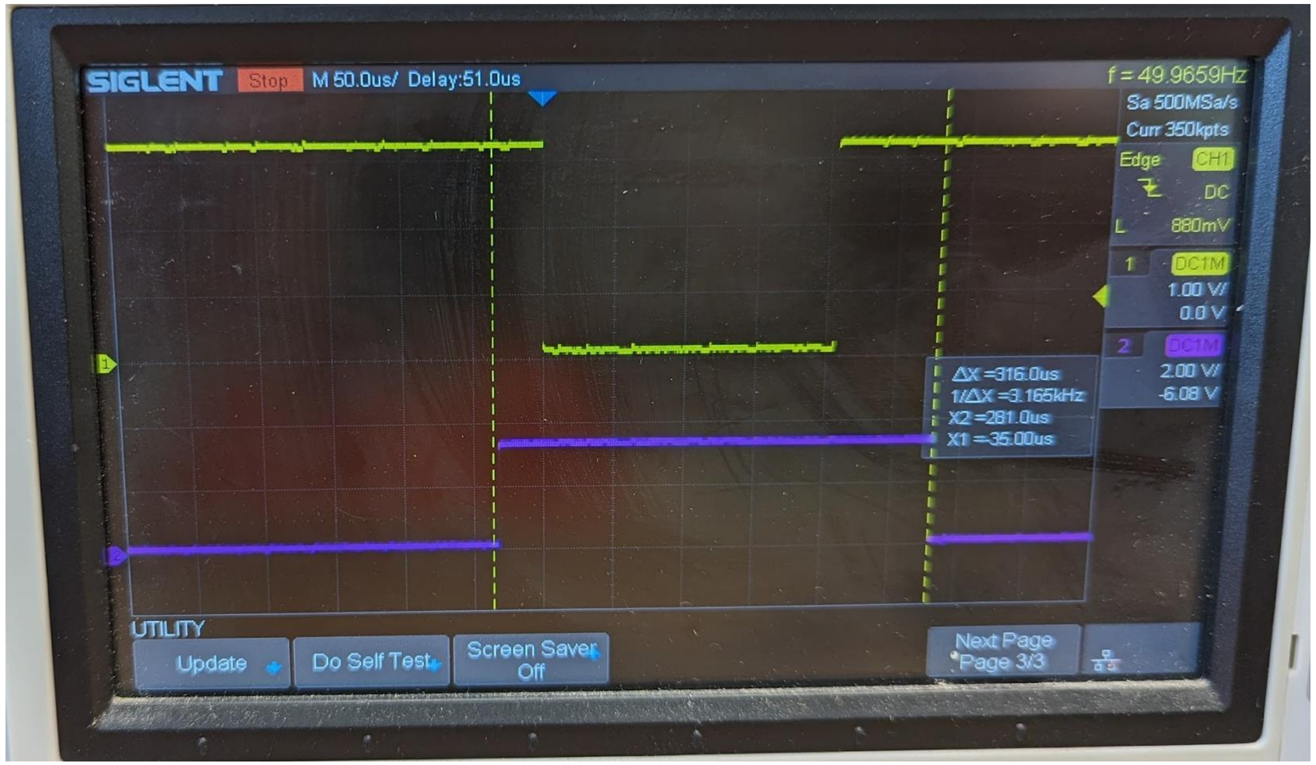
Limit Switches: -1

Example Test Results



Timing Results

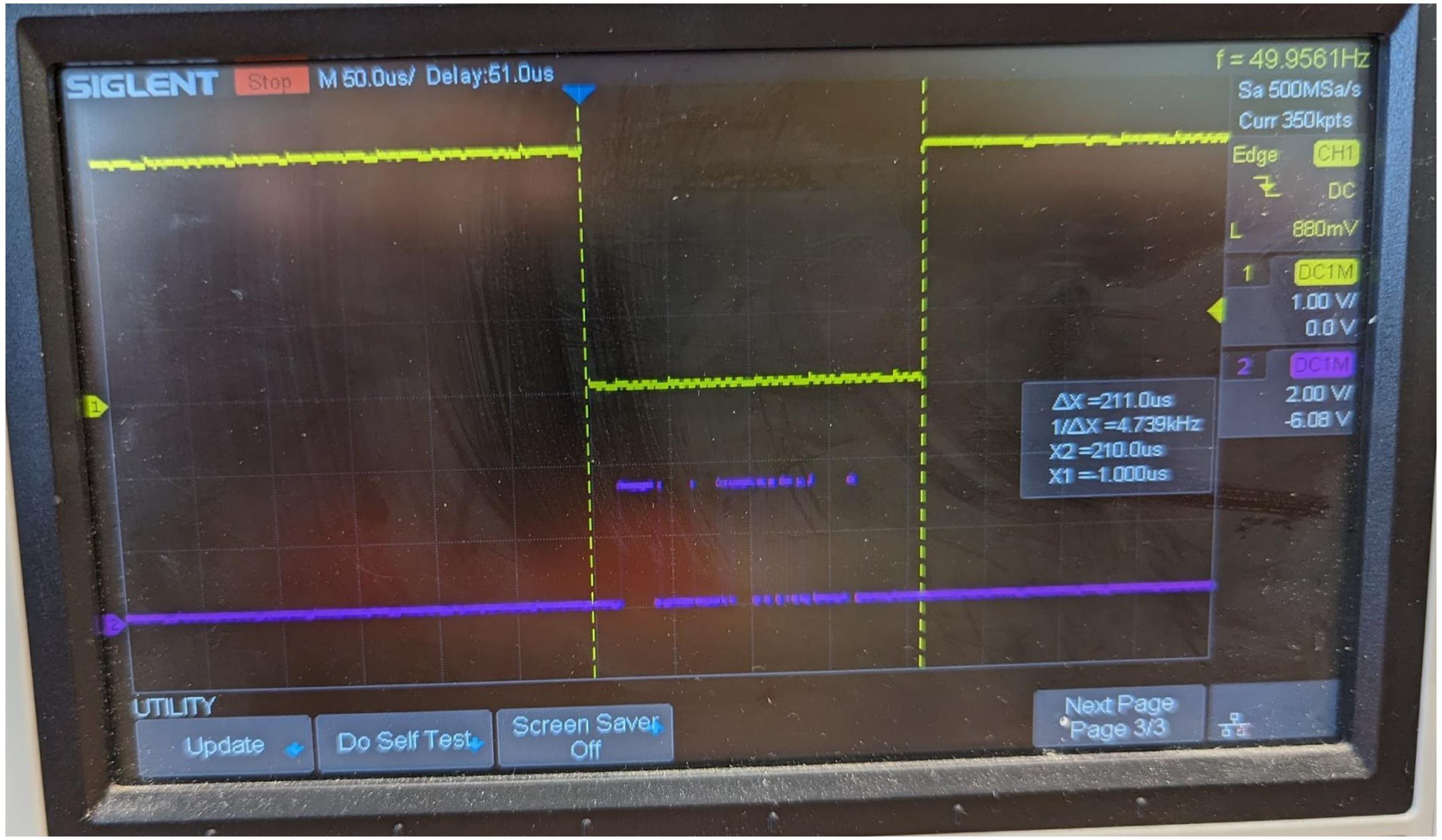
- An oscilloscope was used to verify SPI and RoboRio Times



Here, the yellow trace is the CS assert sent by the RoboRio for one transfer. The blue trace is also a signal from the RoboRio (from a DIO line) to mark the time it takes to do the transfer. The cursors indicate 316 us for a transfer.

(We set the SPI Clock rate to be 1.8MHz)

Timing Results, Part 2



This picture is similar to the previous, except here the blue trace is the data being transmitted from the Mag4 device.

Discussion of Data Fusion

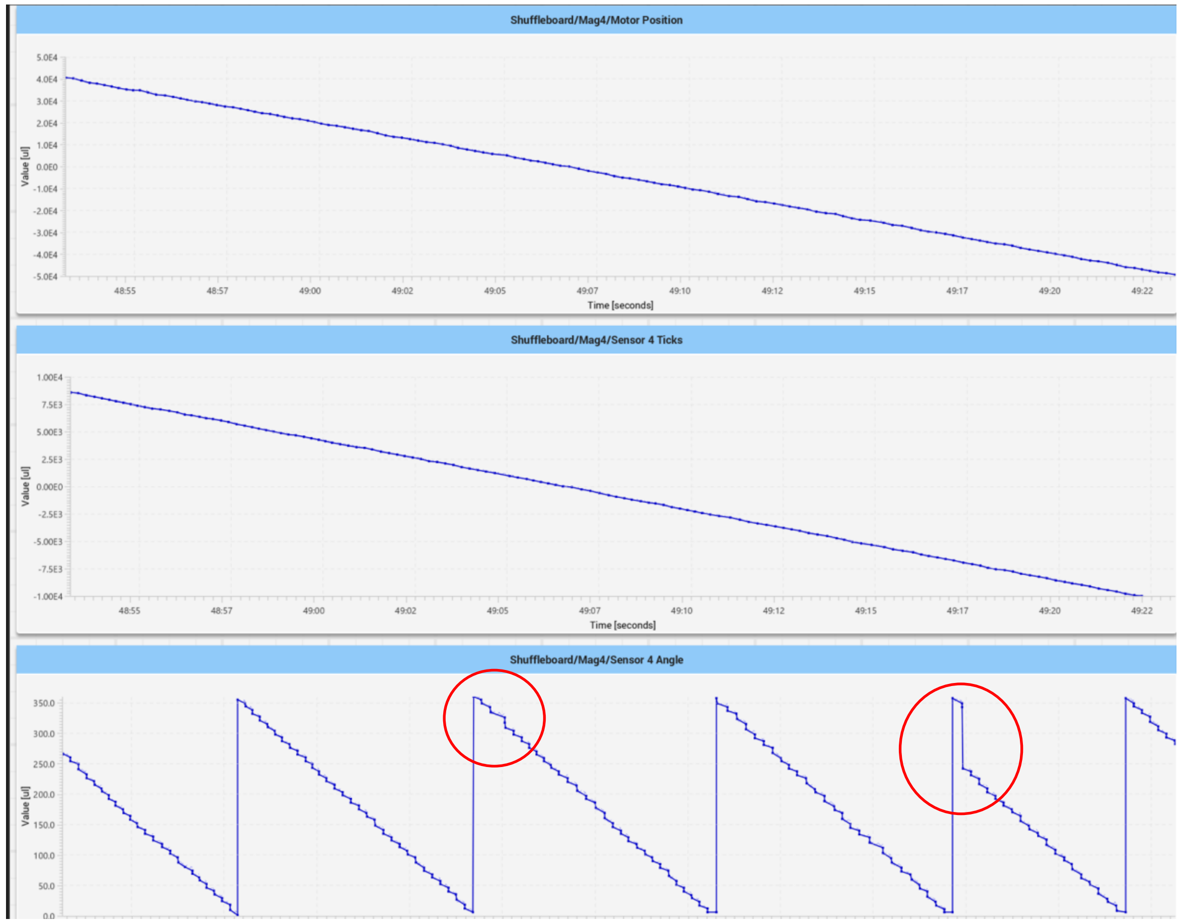
A lot of work was expended trying to fuse the readings from the quadrature with the PWM signal that give absolute angle. The PWM signal arrives once every 4 milliseconds, whereas the quadrature arrives almost instantaneously – but gives only relative information.

The idea was to marry these. Once an absolute position is known, then one can observe the quadrature and calculate absolute position without reference to the PWM.

This was attempted. It worked by recording the quadrature count at the time a PWM pulse was decoded – these two values establish a datum. Thereafter, only the quadrature was used to calculate angle.

The algorithm was improved by noticing that one could calculate angular velocity with the quadrature – and that leads to an estimate of maximum possible error on the PWM measurements. Therefore, after every PWM pulse, the algorithm would update the datum if the possible maximum error was less than or equal from before. If the velocity was zero during a PWM pulse, then no error should be in the PWM signal, and the datum would be updated on very pulse. These last steps was an attempt to correct for a possible noisy PWM measurement.

Results of Fusion Algorithm



To the left are test results of the fusion algorithm for the motor spinning at about 10 RPM. The first chart is the motor position over time, and the second is the raw quadrature count from the Mag4 device. The third chart is the reported angle.

The first two charts tell us that the motor is moving linearly, and that the Mag4 reports believable quadrature counts. However, the angles have problems, most notably at the red circles.

I believe this indicates that the algorithm has accepted an incorrect datum, and then is unable to update the datum because the continuous motion prevents another datum with a less maximum error estimate.

Unfortunately, the algorithm fails. Currently the Mag4 is configured to only return raw angle measurements from the PWM. No fusion is attempted.