

# Quadrature Encoder Test Kit Documentation v1.0

Ryan Lederhose

31st August 2023

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	2
1.2	Aim . . . . .	2
1.3	Scope . . . . .	2
<b>2</b>	<b>Design</b>	<b>3</b>
2.1	Hardware . . . . .	3
2.1.1	Bill of Materials . . . . .	5
2.2	Software . . . . .	5
2.2.1	Encoder Processing . . . . .	5
2.2.2	LCD Display . . . . .	7
<b>3</b>	<b>Instructions</b>	<b>8</b>
3.1	Use Instructions . . . . .	8
3.2	Troubleshooting . . . . .	10
3.3	Build Instructions . . . . .	11

List of Figures

1	Quadrature Encoder Channels <a href="#">[1]</a> . . . . .	1
2	FSM for Quadrature Encoder . . . . .	1
3	Allen Bradley 847H Encoder Pinout . . . . .	3
4	Hardware Schematic . . . . .	4
5	PCB Layout . . . . .	4
6	FreeRTOS Implementation . . . . .	5
7	LCD Display . . . . .	7
8	Power Supply . . . . .	8
9	Connection to 8-pin Plug . . . . .	8
10	Plug and Socket Connection . . . . .	9
11	Display Measurements . . . . .	9
12	PCB Test Points . . . . .	10
13	Debug Pinout . . . . .	11

List of Tables

1	Bill of Materials . . . . .	5
---	-----------------------------	---

# 1 Introduction

Encoders are used to measure the direction and velocity of mechanical motion providing information for the precise control for a wide range of applications. Encoders follow two basic geometries: rotary and linear [2], with the fundamental difference surrounding the measurement of resolution. Rotary encoders resolution is measured in pulses per revolution (PPR) whereas linear encoders resolution is measured in pulses per millimetre (PPM) [2]. The style of the encoder can also differ, with incremental and absolute encoders the most common. An incremental encoder will only read electrical signals to "provide information about the relative motion of the shaft" [2], and "it can only show how far the shaft has moved since the encoder was powered up" [2]. On the contrary, absolute encoders provides information about the absolute position of mechanical motion.

A quadrature encoder is a common implementation of rotary, incremental encoders. These encoders utilise three sets of electrical signals, or channels, to provide information about the direction, velocity and relative position of mechanical motion.

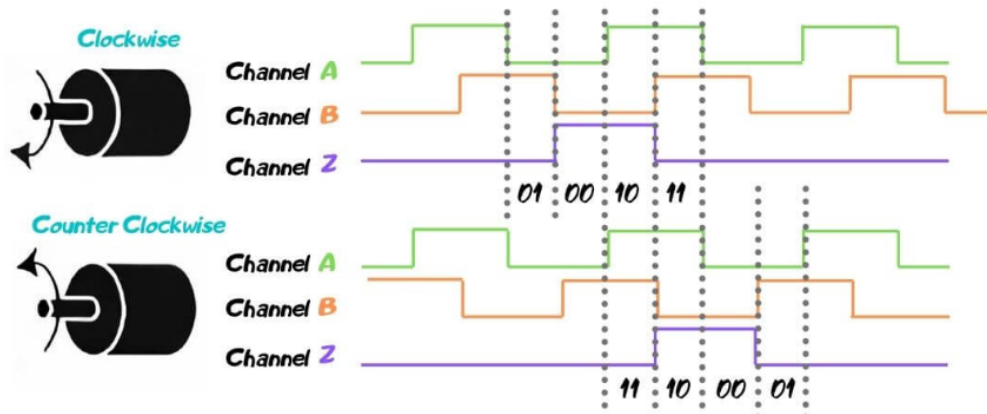


Figure 1: Quadrature Encoder Channels [1]

Figure 1 demonstrate the behaviour of these channels under clockwise and counter-clockwise motion. The A and B channels are separated by  $90^\circ$  of phase, where in clockwise motion A will lead B and vice versa. This results in four different states which indicate the direction of motion and hence, "a quadrature encoder with a resolution of 100ppr would actually produce 400 different states for each revolution" [2]. The Z channel provides an output pulse once per revolution indicating when the encoder disc crosses a fixed origin point. This channel is typically used to reset a counter. Hence, these signals can be used to create a finite state machine to count the pulses per revolution:

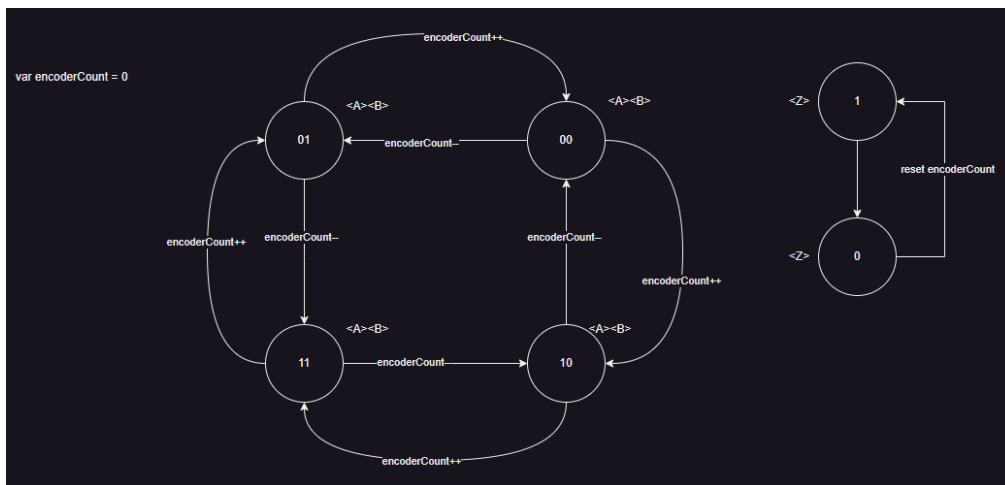


Figure 2: FSM for Quadrature Encoder

## 1.1 Motivation

Quadrature encoders are used to measure the direction, velocity and position of motor shafts in important autonomous processes. These encoders are subject to challenging conditions and general wear and tear. With their importance, it is important to ensure the encoders are working properly; however, testing kits are costly, so there is a need for a low cost system capable of testing quadrature encoders

This report describes the intricacies of the system built capable of measuring the working conditions of quadrature encoders and provides detailed, yet simple, instructions for use.

## 1.2 Aim

The aim of this project was to design a low cost, embedded system capable of testing the working condition of quadrature encoders. This involved measuring the pulses per revolution, individual channel counts (A, B and Z), relative position of motion and rpm/frequency of rotation. The system shows these measurements on a compact screen, and is comprehensible to someone with a trades-persons education level.

## 1.3 Scope

The scope of the project covers:

- Embedded system design.
- Measuring quadrature encoder pulses per revolution, individual channel counts, relative position of motion and rpm/frequency of rotation.
- PCB Design
- LCD screen interface design

## 2 Design

This section lays out the important design details for the encoder test kit.

### 2.1 Hardware

The encoders used for this project are **Allen Bradley 847H** encoders. These have a resolution of 1024 pulses per revolution, a 12V power requirement, and the following pinout:

#### Connector Pins and Signal Availability

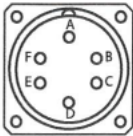
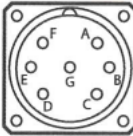
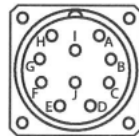
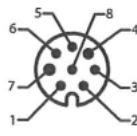
Signal Name	MS 6-pin	MS 7-pin	MS 10-pin	M12 8-pin	Cable Wire Colors
V DC	B	D	D	8	Red
Common	A	F	F	7	Black with red band
A output	E	A	A	2	White
AN output	-	-	H	1	Black with white band
B output	D	B	B	4	Blue
BN output	-	-	I	3	Black with blue band
Z output	C	C	C	6	Green
ZN output	-	-	J	5	Black with green band
Zero set input	-	E	E	-	Yellow
Case	F	G	G	-	Black
Encoder connectors/ cable view					-
Recommended mating cable part number	845-CA-A-*	845-CA-B-*	845-CA-C-*	889D-F8FB-*	(Attached to encoder)

Figure 3: Allen Bradley 847H Encoder Pinout

Thus, the hardware was designed to meet the following requirements:

- Supply 12V power to encoder
- Receive 8 signals from encoder
- Process signals on embedded microcontroller
- Display measurements on compact screen

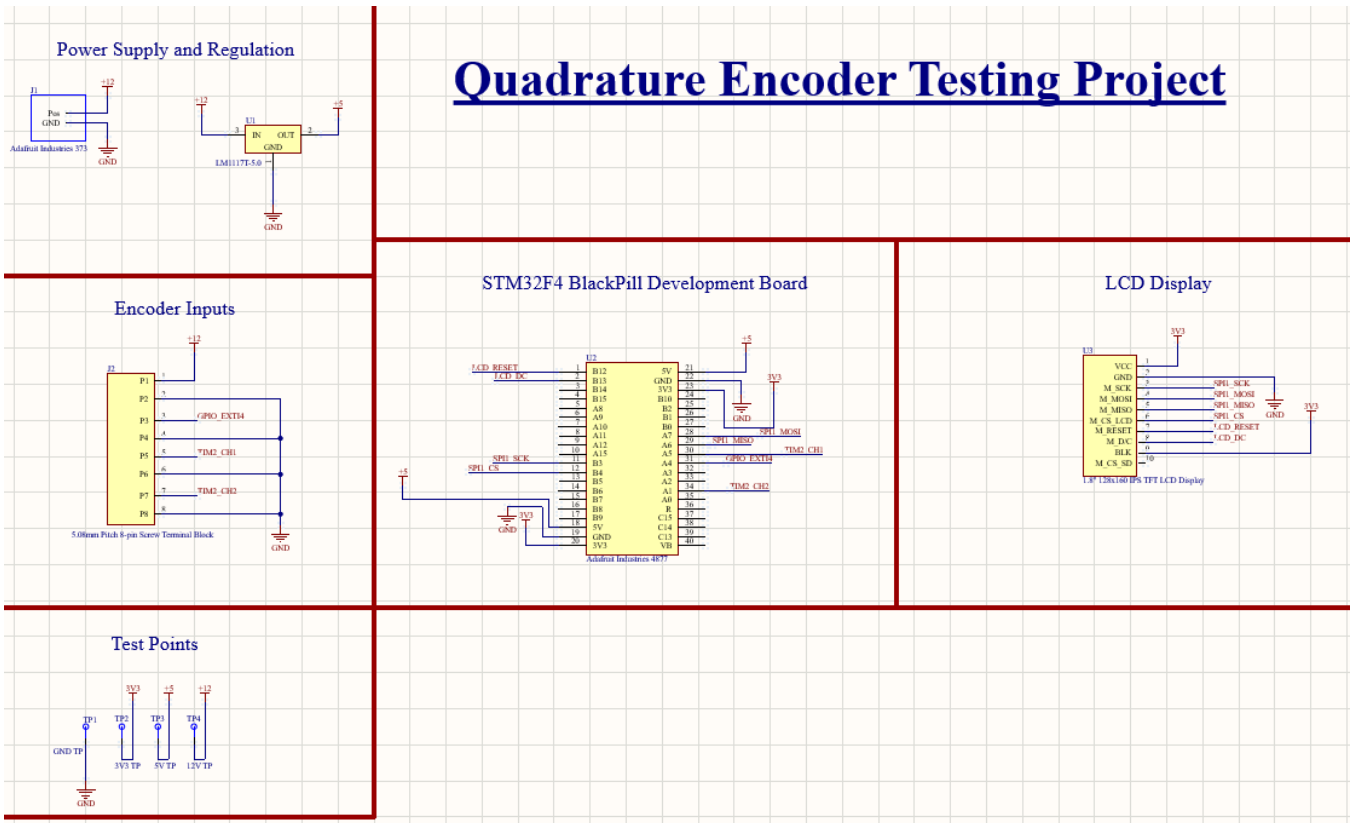


Figure 4: Hardware Schematic

A 12V, 1A DC plug supplies power to the board, facilitating proper power supply to the encoder, as well as, a regulated 5V supply to an embedded microcontroller. The STM32F4 BlackPill Development Board is utilised in the design for its native encoder timer support and through-hole mounting which allowed reliable soldering and agile development. An 8-pin plug and socket is used for board to wire connection for the encoder inputs/outputs. The measurements calculated from the encoder channels are displayed on a 1.8" 128x160 LCD screen. The display utilises an SPI interface for communication with an embedded microcontroller which allowed for fast update rates.

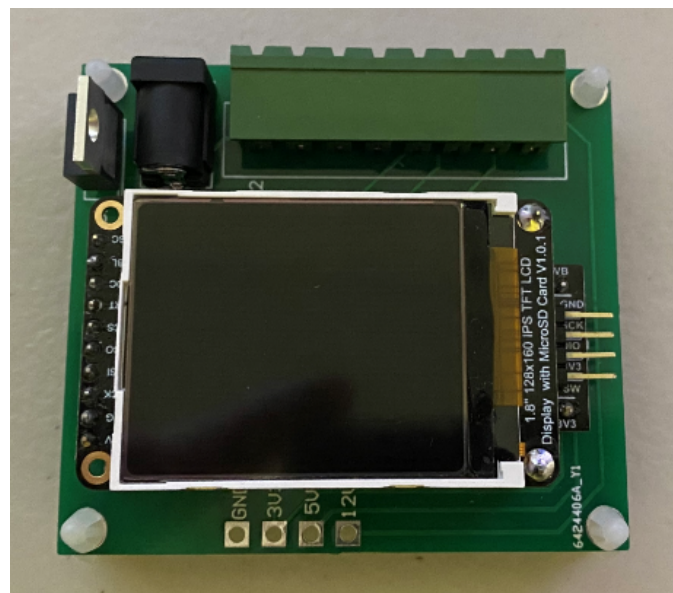


Figure 5: PCB Layout

Figure 5 shows the layout of the designed PCB. Inputs into the board (i.e. plug and socket and DC plug) sit on the edges of the board while the screen sits centrally over the microcontroller. The PCB is designed such that it can be mounted within a box. Mounting holes are drilled in each corner for solid installation while the positioning of the components allows for all important features to be accessed with well-positioned holes in the box. Assembly and Schematic drawings can be accessed from the project [GitHub](#).

### 2.1.1 Bill of Materials

Part	Cost	Quantity
STM32F411 BlackPill Development Board	\$29.85	1
LM1117T-5.0	\$2.74	1
2.1mm DC Jack	\$2.10	1
10pos 2.54mm header	\$8.11	1
1pos header	\$0.63	4
1.8" 128x160 LCD Display	\$14.95	1
Terminal Block Plug 8pos 5.08mm	\$14.49	1
Terminal Block Socket 8pos 5.08 mm	\$4.45	1
<b>Total Cost</b>	<b>\$79.21</b>	

Table 1: Bill of Materials

Links to these components can be found on the project [GitHub](#).

## 2.2 Software

The software utilises FreeRTOS to delegate tasks. Figure 6 describes the implementation of FreeRTOS in the design:

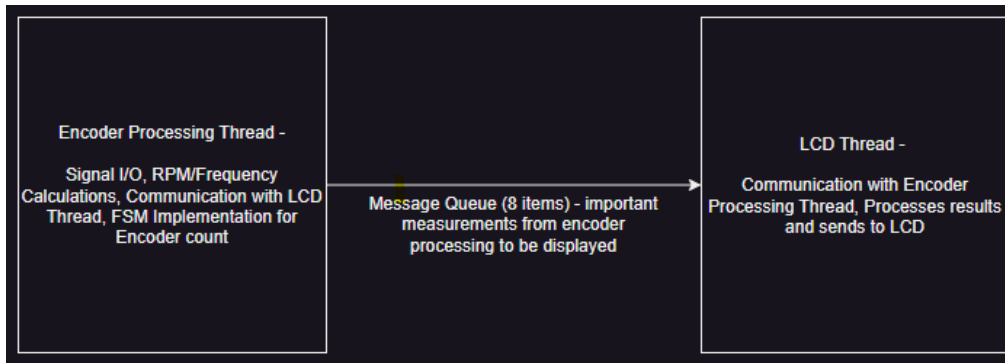


Figure 6: FreeRTOS Implementation

### 2.2.1 Encoder Processing

To process the encoder channels, the STM32 native encoder timer implementation is utilised. Using this timer, a register can be accessed in the thread which gives the x4 resolution of the encoder count (i.e. the timer increments/decrements for every rising edge on both A and B channels depending on rotation direction). For the **Allen Bradley 847H Encoders**, this means the 1024ppr, would give 4096ppr with x4 resolution. This timer is then reset when the Z channel pulse raises an interrupt. It should be noted this timer counts from 0 to 65536. Hence, it was necessary to convert this count to a *user-friendly* value.

```

1 //Adjust timer value for user friendly value
2 if (timerCount <= 10000) {
3     timerCountPrint = timerCount;
4 } else if (timerCount >= 50000) {
5     timerCountPrint = -(65536 - timerCount);
6 }

```

Listing 1: Adjusting timer count to interpretable value



This bounds the encoder count from  $(-1) \times \text{pulses per rotation} \leq \text{encoder count} \leq \text{pulses per rotation}$ . Counts for which are less than zero indicate consistent counter clock wise movement whereas counts which are greater than zero indicate consistent clock wise movement.

This *user-friendly* timer count is saved when an interrupt is raised by the Z channel. This saved value is equivalent to the pulses per revolution (x4 resolution).

In the thread, two structs are maintained which carry the previous and current values of important measures:

```
1 typedef struct __attribute__((packed)) _encoderInstance {
2     int64_t timerCount;
3     double rpm;
4     double freq;
5     uint8_t rotationDirection;
6     uint32_t aCount;
7     uint32_t bCount;
8     uint32_t zCount;
9     int pulsesPerRotation;
10 } encoderInstance_t;
```

Listing 2: Struct Definition

The direction of travel is calculated within the code by comparing previous and current values of the encoder timer count as such:

```
1 //Check if the timer has changed value, set rotation direction appropriately
2 if ((timerCount - prevTimerCount) > 0) {
3     //clockwise rotation
4     stillTime = HAL_GetTick();
5     rotation = CW_ROTATION;
6 } else if ((timerCount - prevTimerCount) < 0) {
7     //counter clockwise rotation
8     stillTime = HAL_GetTick();
9     rotation = CCW_ROTATION;
10 } else {
11     //no rotation, no velocity
12     if ((HAL_GetTick() - stillTime) >= 500) {
13         rotation = NO_ROTATION;
14     }
15 }
```

Listing 3: Direction Calculation

Knowing the direction of travel means we can interpolate the individual counts for the A and B channels:

```
1 //Calculate the counts for A and B channels
2 if (timerCountPrint > 0) {
3     aChannel = (int) ceil(timerCountPrint / 2.00);
4     bChannel = timerCountPrint - aChannel;
5 } else if (timerCountPrint < 0) {
6     aChannel = (int) ceil(timerCountPrint / 2.00);
7     bChannel = timerCountPrint - aChannel;
8 } else {
9     aChannel = 0;
10    bChannel = 0;
11 }
```

Listing 4: Channel A and B Calculation

Finally, the rotations per minute, frequency and relative degree of rotation is calculated as such:

$$rpm = \frac{\text{pulses per second} * 60}{\text{pulses per rotation}}$$

$$f = \frac{1}{2\pi} * rpm$$

$$degrees = \frac{\text{encoder count} * 360}{\text{pulses per rotation}}$$

### 2.2.2 LCD Display

Figure 7 shows the designed display:

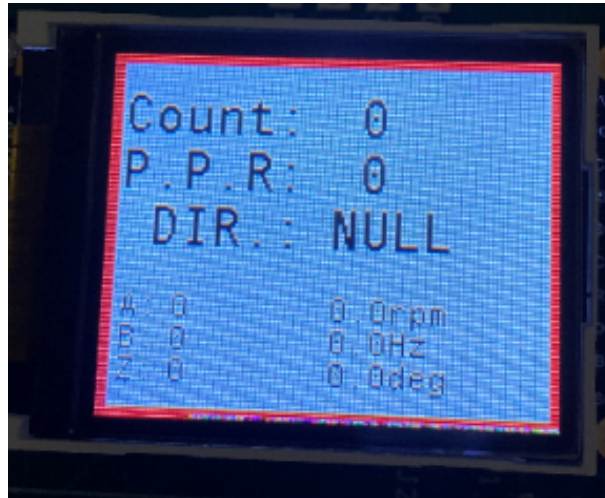


Figure 7: LCD Display

The displayed measures are:

- Count → current encoder timer count (x4 resolution)
- P.P.R → pulses per revolution (x4 resolution)
- DIR. → direction of rotation (NULL, CW, CCW)
- A, B, Z → individual channel counts (x4 resolution)
- Rotations per minute
- Frequency
- Degrees of rotation relative to Z channel pulse

The screen has been implemented with a refresh rate of 25Hz; however, RPM and frequency are only calculated every 0.5 seconds. A full version of the software can be accessed from the project [GitHub](#).

## 3 Instructions

### 3.1 Use Instructions

These instructions outlines how to use the device to test the working condition of quadrature encoders.

1. Use a 12V, 1A DC Power Supply (2.1mm plug). Its important not to supply more than the recommended voltage as it will fry the voltage regulator.

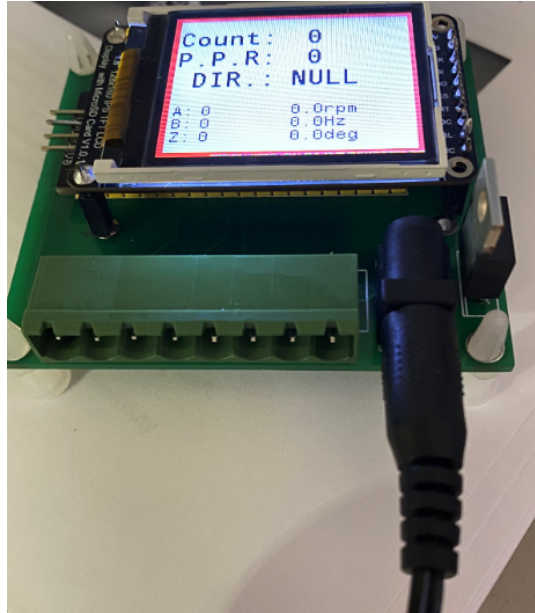


Figure 8: Power Supply

2. Connect the encoder to the 8-pin plug as such:

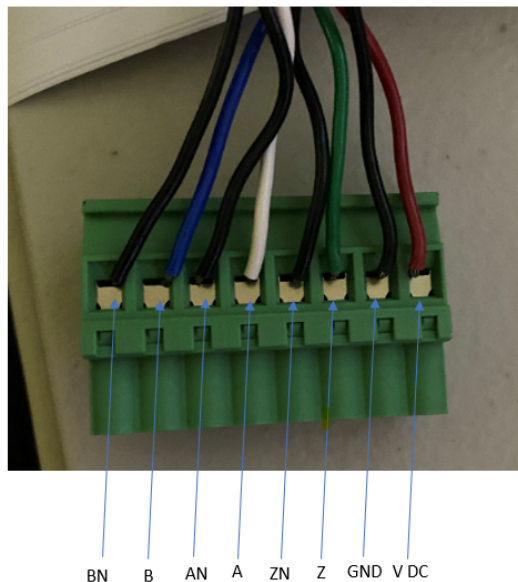


Figure 9: Connection to 8-pin Plug

Typically, encoders will follow the same colour scheme as listed in Figure 3. In practice, it is not essential to connect the complementary channels (i.e. AN, BN, ZN) as shown in Figure 9 as they are all internally connected to ground. As long as they are connected to the ground pins, the device will function properly.

3. Connect the 8-pin plug to the socket connection on the PCB

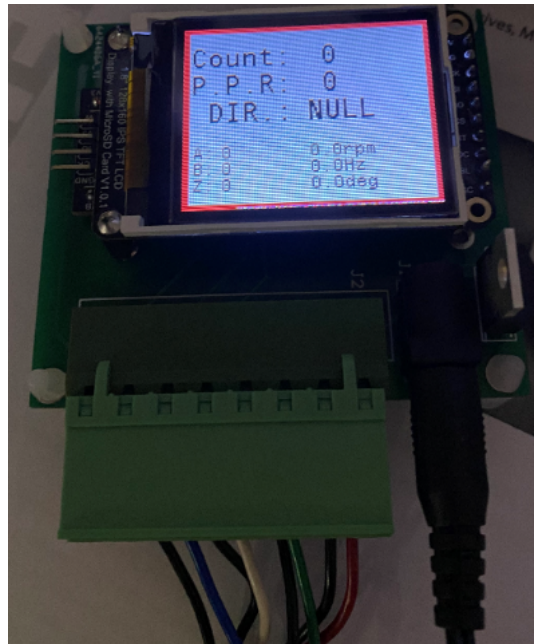


Figure 10: Plug and Socket Connection

4. Upon rotating the shaft of the encoder, the LCD display will now update with the appropriate measurements

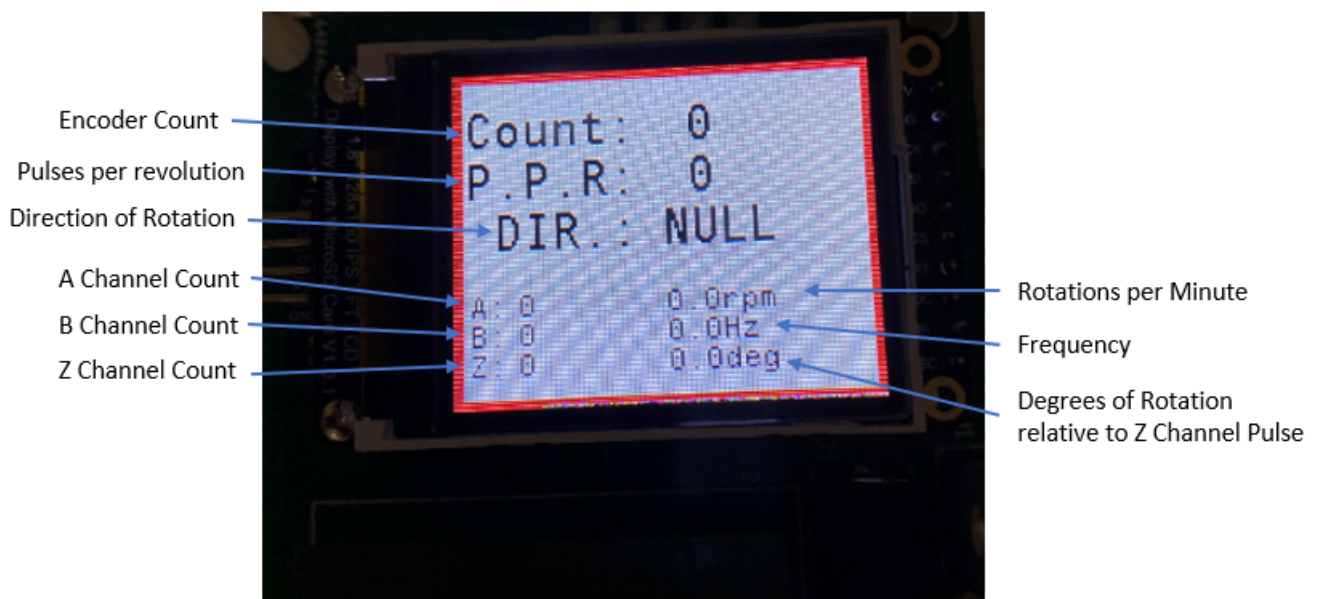


Figure 11: Display Measurements

Note: counts in x4 resolution.

5. It is a requirement for correct measurements that the encoder completes at least one full revolution in the direction of rotation starting from the origin point (i.e. origin meaning when the Z channel count ticks over). This point is set by the encoder itself and not the system.

### 3.2 Troubleshooting

**RPM, Frequency or Degrees shows 'error'.** This is not uncommon and is actually a purposeful implementation in the code. Referring back to the equations for RPM, Frequency and Degrees, we can see these values are heavily influenced by the pulses per revolution. So, when the pulses per rotation is extremely low ( $< 100$ ) and extremely high ( $> 10000$ ), these values (RPM, frequency and degrees) are set to 'error'. This stops the display from showing absurdly large floating point numbers, and also incorrect values for these parameters. This error would happen under three circumstances. (1) the encoder is not properly functioning, (2) the encoder has not been connected properly and (3) the encoder has not completed one full revolution in the direction of rotation from the origin point.

**Encoder counts ticks down for clockwise motion or vice-versa.** This would indicate that the A and B channels are swapped. Switch these channels on the 8-pin plug, power reset and the counter should operate correctly.

**Device doesn't seem to be powered up.** Use the test points on the side of the PCB to test 12V, 5V and 3.3V power.



Figure 12: PCB Test Points

If 5V and 3.3V test points are 0V or at unexpected voltage levels, this would most likely indicate a problem with the voltage regulator. Test the regulator and if appropriate, seek a replacement. If the 12V and 5V pins are acceptable but 3.3V is not, this would indicate a problem with the microcontroller. A replacement controller would need to be sourced. If all pins are low (including 12V) there is an issue with the power source and/or DC power jack. Seek a replacement.

### 3.3 Build Instructions

These instructions lay out the details on how to build and flash the software from source onto the embedded microcontroller.

1. Clone the [GitHub](#) repository.

```
1 git clone https://github.com/ryanlederhose/Quadrature-Encoder-Test-Kit.git
```

2. Ensure [STM32CubeIDE](#) is downloaded on your personal device.
3. Start a new STM32 Project from an Existing STM32CubeMX Configuration File (\*.ioc)
4. Use the .ioc file from

```
1 ./Quadrature-Encoder-Test-Kit/embedded/quadrature-decoder/quadrature-decoder.ioc
```

5. Replace the Core folder from the project with the Core folder from:

```
1 ./Quadrature-Encoder-Test-Kit/embedded/quadrature-decoder/Core/
```

6. Connect your debugger (e.g. [STLink](#)) as such.

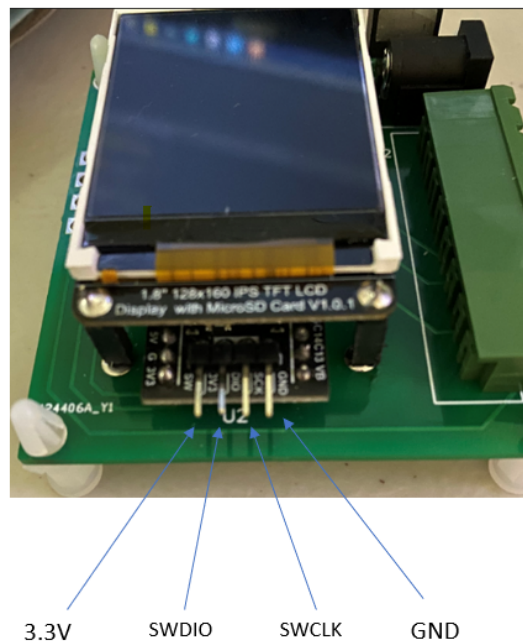


Figure 13: Debug Pinout

7. Run the debugger software on [STM32CubeIDE](#)

## References

- [1] E. Encoder. (2020) Quadrature encoders tutorial. how does it work? [Online]. Available: <https://eltra-encoder.eu/news/quadrature-encoder>
- [2] J. Kimbrell, “Fundamentals of industrial encoder sensing technologies, motion detection theory and methods, and signal output styles,” in *AutomationDirect*.