Ball Hopper – Digital Electronics

White Paper **Updated – March 19, 2023**

Kerem Oktay, Group Portugal, ECE, University of BC, Vancouver, BC, Canada

Abstract

A pulse counter-decoder system counting the encoder A and B signals coming from a built-in encoder of a motor is designed. The system consists of a quadrature decoder, HCTL-2022, and two PLDs, Atmel ATF16V8B, that are programmed as 4 to 1 multiplexers. Decoder reads the raw encoder signals coming from the motor and converts them to a single 32-bit count value. Multiplexers are used to minimize wiring while keeping maximum bits for the count signal. C code is developed to read the signal and convert it to motor angle value for later usage.

In this paper, Section 1 defines the sub-system behavior and identifies the RCGs to achieve the behavior. Section 2 explains the Sub-System design by going through the parts of the system and the design process using tools such as WinCUPL/WinSim and Altium CircuitMaker. Finally, Section 3 goes over the tests executed to verify the designed system satisfies the RCGs.

Nomenclature

RCG	Requirement, Constraint, Goal
IC	Integrated Circuit
MHz	Mega Hertz
CW/CCW	Clockwise/Counter-Clockwise
MCU	Microcontroller Unit
CPR	Counts Per Revolution
PCB	Printed Circuit Board
DRC	Design Rule Check
MUX	Multiplexer
PLD	Programmable Logic Device
PWM	Pulse Width Modulation
POC-VID	Proof of Concept Video

1. RCG Identification

In this project a DC motor with a built-in encoder is used. A digital electronics sub-system is designed to ensure proper reading and converting of the pulses coming from the encoder. The system must read the encoder A and B signals, count the pulses while keeping track of the direction and convert the count value into motor angle. From this defined functionality of the system, the following 5 RCGs for related issues are identified as shown in Table 1.

Issue	Requirement	Constraint	Goal
Pulse Counting	The system must not miss more than 10 pulses per 100 pulses.	The clock frequency of the decoder IC cannot exceed 33MHz.	Minimize the number of missed pulses.
Motor Direction Detection	The system must realize when the motor is rotating in CW and CCW directions.	Negative directions must be registered as 2's complement values.	Minimize the usage of different logic for the direction detection.
Wiring/Bit Usage	Pulses must be counted with at least 10 bits to avoid quick overflows.	Atmel ATF16V8Bs (PLDS) have a set number of Input and Output pins.	Maximize the bits for counting and minimize external wiring of the MCU.
Pulse Conversion into Degrees	Angle value resolution must be at least the same as the resolution CPR of the encoder.	The encoder has a set value of 64 CPR.	Maximize the resolution of the degree values.
Daughter Board	The PCB must be smaller than 6x6 cm.	The PCB cannot violate any DRC points.	Minimize the size of the PCB.

Table 1: Digital Electronics Sub-System RCGs

2. Sub-System Design

This sub-system consists of HCTL-2022, a Quadrature Decoder / Counter Interface IC, and 2 PLDs. This section first goes over the characteristics of the quadrature decoder IC. Afterwards, it explains the high level and 4-1 MUX design. Finally, it presents the integration of HCTL-2022 and 4-1 MUXs, the PCB design and the C code developed for degree value conversion.

2.1. HCTL-2022 Characteristics

HCTL-2022 counts the pulses of an encoder given the A and B signals as input. It counts with 32 bits and can run up to 33 MHz clock frequency. It has tri-state buffers to output a single byte (8 bits) of the count value. The pinout of the IC is shown at Figure 1. This diagram is obtained from the datasheet of the IC [1]. Pins D0-D7 are the 8 bits of a selected byte. Pins SEL1 and SEL2 are selection signals for outputting certain bytes of the count value. OE pin is used to enable/disable the tri-state buffers and RST pin is used to reset the IC. CHA and CHB are the input signals fed from the encoder of the motor. Other pins are not used in this design.

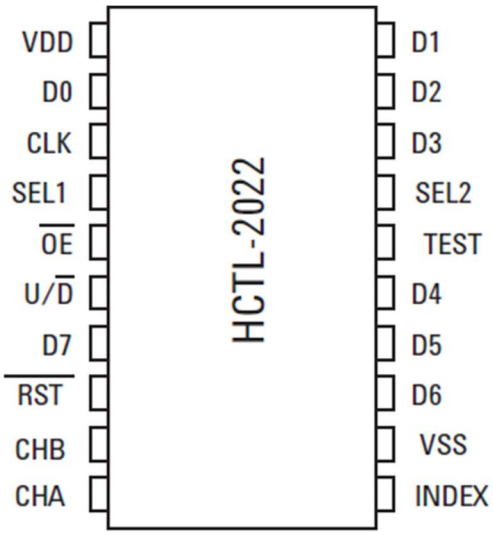

Figure 1: Pinout of HCTL-2022

Due to the digital noise filtering, high clock frequency, 32 bits up and down counting and latch capabilities; HCTL-2022 is chosen in this design. These features address the Pulse Counting, Motor Direction Detection and Wiring/Bit Usage RCGs.

2.2. High Level Decoder Design

The high-level decoder system design is shown at Figure 2. The system is provided encoder A and B signals, and control signals for HCTL-2022 and register block. The chosen register block is a 4-1 MUX. The count value is fed back to an MCU.

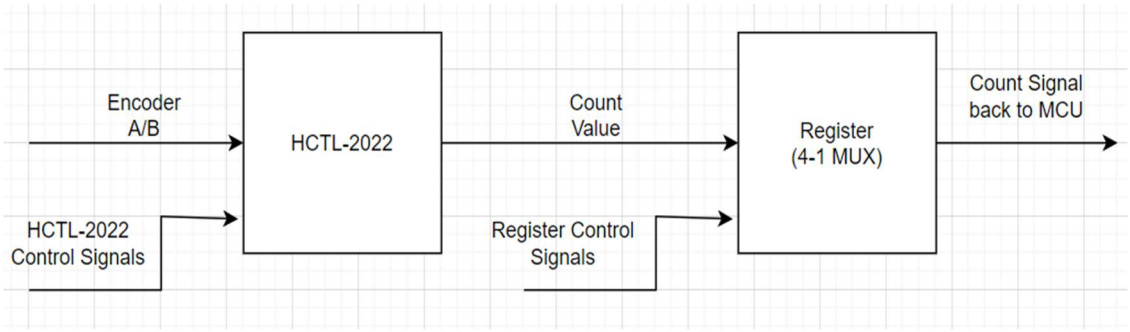

Figure 2: High-Level Decoder System Design

Since HCTL-2022 needs 8 bits for its output and 4 bits for its control, using it alone requires 12 bits that needs to be externally wired to MCU board. By using two 4-1 MUXs, there are only 2 bits for output and 2 bits for MUX control in addition to the 4 bits for HCTL-2022 control. This adds to 8 bits which satisfies the goal of "Wiring/Bit Usage" RCG. This way a single byte is read in 4 pairs of bits.

2.3. 4-1 MUX Design

Atmel ATF16V8B model PLDs are used to design the 4-1 MUXs. The truth table for a 4-1 MUX is shown at Table 2. S1 and S2 are selection signals. Q0 is the output of the MUX and D0-D3 are the input signals.

S1	**S2**	**Q0**
0	0	D0
1	0	D1
0	1	D2
1	1	D3

Table 2: 4-1 MUX Truth Table

By observing the truth table, the following circuit schematics in Figure 3 is built using logic gates. S1 and S2 are the same signals as on the Truth Table, I0-I3 correspond to D0-D3 and Y corresponds to Q0.

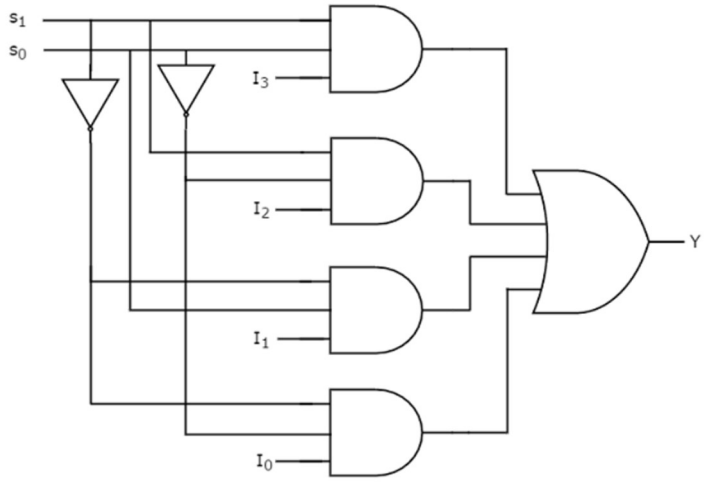

Figure 3: 4-1 MUX Circuit Schematic with Logic Gates [2]

The logic equation written in CUPL language in WinCUPL is as follows (1). The variables are the same signals as shown on the Truth Table.

Q0 = (D0 & !S1 & !S2) # (D1 & S1 & !S2) # (D2 & !S1 & S2) # (D3 & S1 & S2); (1)

The design is simulated using WinSim and the following waveform is obtained shown at Figure 4. The waveform matches with the desired Truth Table.

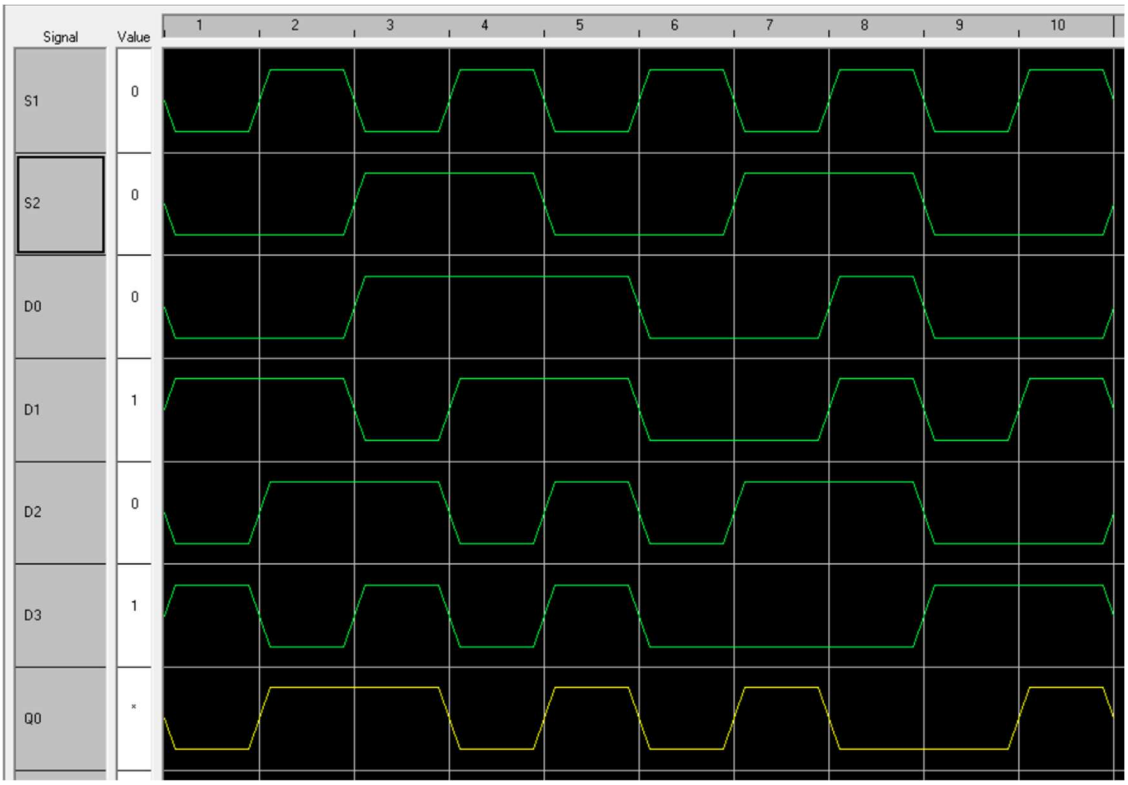

Figure 4: 4-1 MUX Simulated Waveform

2.4. HCTL-2022 IC and 4-1 MUX Integration

HCTL-2022 and 4-1 MUXs (PLDs) are integrated by simply connecting the D0-D3 outputs of the HCTL-2022 to inputs of one of the MUXs and D4-D7 outputs to the inputs of the other MUX. All control signals are fed from an MCU and output signals BIT0 and BIT1 are fed back to the MCU. The schematic prepared with Altium CircuitMaker is shown in Figure 5. Connectors are added for wiring between the motherboard and a crystal is added to provide clock for HCTL-2022.

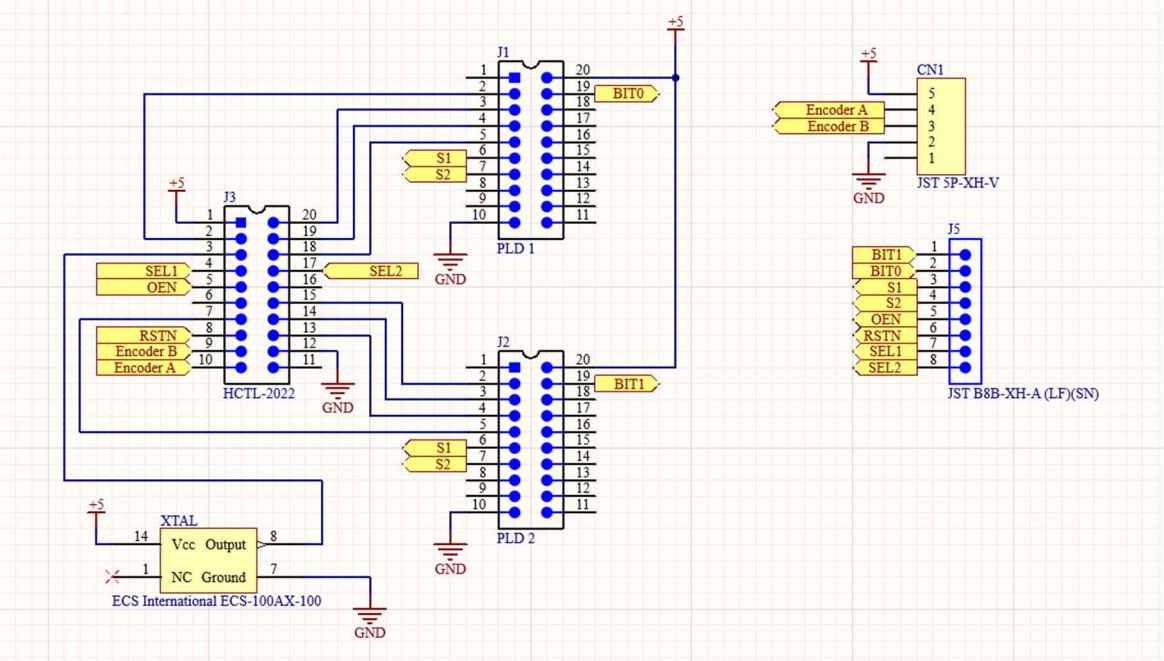

Figure 5: Decoder System Circuit Schematic

2.5. Daughter Board PCB Design

Using Altium CircuitMaker a daughter board called Pulse Counter is designed. The board consists of two-layers with no ground-plane. As the HCTL-2022 and ATF16V8B devices are through hole devices, 20-pin DIP sockets are put in their places. XH type connectors are used to connect the daughter board to motor driver board and motherboard. Figures 6 and 7 show the PCB schematics and 3D view of the Pulse Counter PCB.

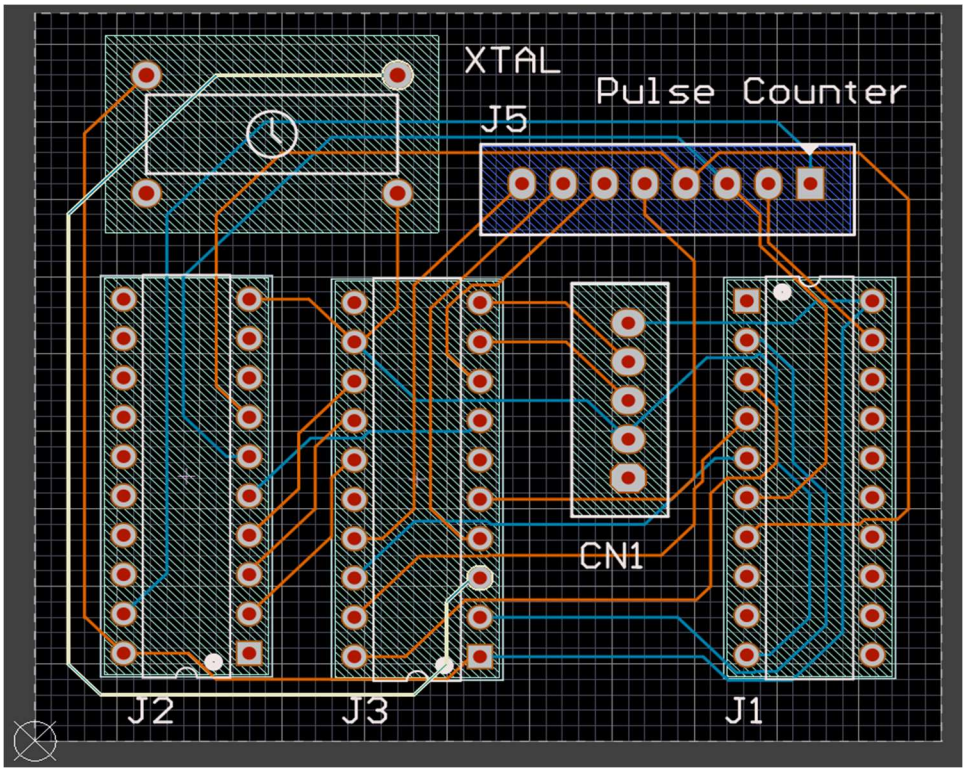

Figure 6: Pulse Counter PCB Schematics

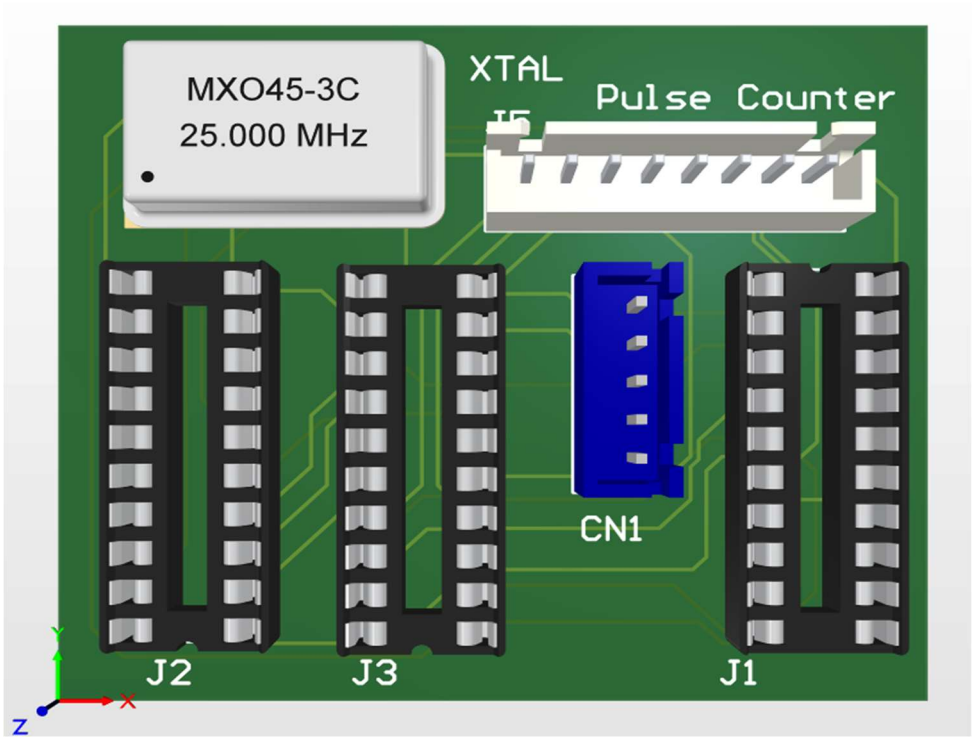

Figure 7: Pulse Counter PCB 3D Top View

2.6. C Code Design for Degree Conversion

As the signals coming from the decoder system are raw bits that carry the pulse count, a software routine is necessary to convert this value into degrees of the motor. In this report the C code logic is explained through an Arduino UNO board that is used for testing the system.

The routine is shown on the Flow Diagram on Figure 8. By setting SEL1 and SEL2 to corresponding HIGH and LOW values as described in page 6 of the HCTL-2022 datasheet and disabling the OE signal, certain bytes are registered [1]. Then by setting S1 and S2 signals for the 4-1 MUXs, bytes are read in 4 pairs of bits.

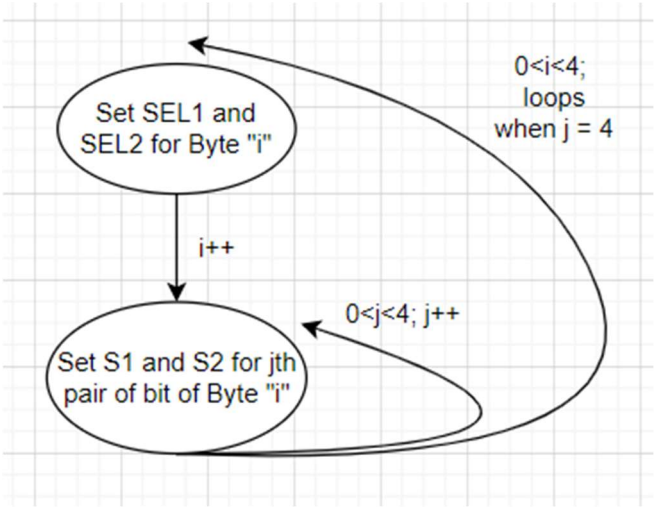

Figure 8: Software Routine Flow Diagram

After getting all 4 bytes, these bytes are combined into a single 32 bit variable. This byte is the count value. As there is 64 CPR, it means every 360 degrees the decoder has counted 64 pulses. Therefore by multiplying the count value by 360/64, the angle value of the motor is obtained.

Please refer to the Appendix A for the full C code developed.

3. Testing of the Design

The 5 RCGs listed in the first section are individually tested or considered during the design process. The tests executed and their results are explained below.

i) Pulse Counting

After resetting the quadrature decoder and motor position, the motor driver is fed with constant 5V input. This case is assumed as the highest speed as there is no load present to slow the motor down and in terms of a PWM signal this corresponds to 100% duty cycle.

The motor is run for a certain amount of time and then stopped. The position of the motor is brought to the initial position by aligning the marker lines. The angle value (a value between 0 and 360 degrees that resets after every rotation) is seen to be 0 degrees again. This indicates no pulses were missed as any missed pulse would create an offset between the initial position and angle value of 0 degrees.

This test shows that the RCG is met. Please refer to POC-VID1 for the test being run.

ii) Motor Direction Detection

HCTL-2022 counts the pulses in both directions by increasing or decreasing the same count value. By using variable types in the C code that detect 2's complement, negative values are detectable as well. The code does not take separate actions for two cases which means the logic is minimalized.

This shows that the RCG is met. Please refer to POC-VID2 for the test being run.

iii) Wiring/Bit Usage

HCTL-2022 counts with 32 bits which is above the 10 bits requirement. Two 4-1 MUXs in PLDs also minimize the external wire usage while keeping the same number of 32 bits counting. HCTL-2022 requires 4 wires. Using the MUXs only adds 4 more wires making a total of 8 wires.

This shows that the RCG is met during the design process.

iv) Pulse Conversion into Degrees

As every pulse count is registered properly by the MCU, the resolution is the same as resolution of the built-in encoder. Due to this value being set, it is also the maximum achievable resolution.

This shows that the RCG is met. Please refer to either POC-VID1 or POC-VID2 to see the angle values being reported through the serial port.

v) Daughter Board

After completing the PCB layout, “Run DRC” feature is used to verify the PCB design. While designing the board, the board size is first set to the Requirement value. After completing the design small changes are made to decrease the size of the board. The current final board is 5.5cmx4.7cm.

This shows that the RCG is met during the design process. Please refer to the Appendix B for the generated DRC document by the Altium CircuitMaker.

References

[1] "HCTL-2032,2022.pdf," . Retrieved March 19, 2023, from https://media.digikey.com/pdf/Data%20Sheets/Avago%20PDFs/HCTL-2032,2022.pdf

[2] "Digital Circuits - Multiplexers," . Retrieved March 19, 2023, from https://www.tutorialspoint.com/digital_circuits/digital_circuits_multiplexers.htm

Appendix A. C Code for Angle Value Conversion

```
// Control Signals
int RSTN = 2;
int OEN  = 3;
int SEL1 = 4;
int SEL2 = 5;
int S1   = 6;
int S2   = 7;

// Counter Signals
int BIT0 = 8;
int BIT1 = 9;

volatile long count = 0;

/*
  The algorithm below is based on the Example Routine given in the IC datasheet
*/

void setup()
{
  // Assign Output Pins
  pinMode(RSTN,OUTPUT);
  pinMode(OEN,OUTPUT);
  pinMode(SEL1,OUTPUT);
  pinMode(SEL2,OUTPUT);
  pinMode(S1,OUTPUT);
  pinMode(S2,OUTPUT);

  // Assign Input Pins
```

```
 pinMode(BIT0,INPUT);
 pinMode(BIT1,INPUT);

 // Initial Decoder Reset
 digitalWrite(RSTN,LOW);
 delay(15);
 digitalWrite(RSTN,HIGH);

 // Initial Control Signals
 digitalWrite(OEN,LOW);
 digitalWrite(SEL1,LOW);
 digitalWrite(SEL2,LOW);
 digitalWrite(S1,LOW);
 digitalWrite(S2,LOW);

 // Serial Port for Debug Purposes
 Serial.begin(9600);
}

byte getByte()
{
byte capture1 = 0; // Initialize a byte variable to read the first capture
for(int i=0; i<4; i++){

 switch(i){
   case 0:
     digitalWrite(S1,LOW);
     digitalWrite(S2,LOW);
     break;
   case 1:
```

```
    digitalWrite(S1,HIGH);
    digitalWrite(S2,LOW);
    break;
   case 2:
    digitalWrite(S1,LOW);
    digitalWrite(S2,HIGH);
    break;
   case 3:
    digitalWrite(S1,HIGH);
    digitalWrite(S2,HIGH);
    break;
   default:
    digitalWrite(S1,LOW);
    digitalWrite(S2,LOW);
    break;
  }

  capture1 |= (digitalRead(BIT0)<<i);
  capture1 |= (digitalRead(BIT1)<<(i+4));
}

byte capture2 = 0; // Initialize a byte variable to read the second capture
for(int i=0; i<4; i++){

  switch(i){
   case 0:
    digitalWrite(S1,LOW);
    digitalWrite(S2,LOW);
    break;
   case 1:
```

```
    digitalWrite(S1,HIGH);
    digitalWrite(S2,LOW);
    break;
   case 2:
    digitalWrite(S1,LOW);
    digitalWrite(S2,HIGH);
    break;
   case 3:
    digitalWrite(S1,HIGH);
    digitalWrite(S2,HIGH);
    break;
   default:
    digitalWrite(S1,LOW);
    digitalWrite(S2,LOW);
    break;
  }

  capture2 |= (digitalRead(BIT0)<<i);
  capture2 |= (digitalRead(BIT1)<<(i+4));
}

if (capture2 == capture1){  // If the first and second capture match then return the result
  byte byteCapture = capture2;
  return byteCapture;
}
else getByte(); // If the first and second capture DO NOT match, try again to capture
}

long combine4Bytes(byte byte4, byte byte3, byte byte2, byte byte1)
{
```

```
  long combine = 0;
  combine |= (((long)byte4 << 24) | ((long)byte3 << 16) | ((long)byte2 << 8) | ((long)byte1 << 0));
  return combine;
}

void getPulseCount()
{
  delay(25);
  digitalWrite(OEN, HIGH); // Disable OE
  digitalWrite(OEN, LOW); // Enable OE

  // SEL1 = 0 and SEL2 = 1 gets 4th Byte
  digitalWrite(SEL1, LOW);
  digitalWrite(SEL2, HIGH);
  byte Byte4 = getByte();

  // SEL1 = 1 and SEL2 = 1 gets 3rd Byte
  digitalWrite(SEL1, HIGH);
  digitalWrite(SEL2, HIGH);
  byte Byte3 = getByte();

  // SEL1 = 0 and SEL2 = 0 gets 2nd Byte
  digitalWrite(SEL1, LOW);
  digitalWrite(SEL2, LOW);
  byte Byte2 = getByte();

  // SEL1 = 1 and SEL2 = 0 gets 1st Byte
  digitalWrite(SEL1, HIGH);
  digitalWrite(SEL2, LOW);
  byte Byte1 = getByte();
```

```
  digitalWrite(OEN, HIGH); // disable OE
  delay(25);
  count = combine4Bytes(Byte4,Byte3,Byte2,Byte1);
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(25);
  digitalWrite(OEN, HIGH); // Disable OE

  getPulseCount();

  float angle = (count % 64)*(360.0/64.0) ;
  float angle_abs = (count)*(360.0/64.0) ;

  Serial.print("Counter: ");
  Serial.println(count);

  Serial.print("Angle: ");
  Serial.println(angle);

  Serial.print("Absolute Angle: ");
  Serial.println(angle_abs);

  Serial.println("\n");
}
```

Appendix B. Daughter Board DRC Document

ALTIUM CIRCUITMAKER

Design Rule Verification Report

Date: 19-Mar-23
Time: 3:15:12 AM
Elapsed Time: 00:00:01
Filename: v2.CMPcbDoc

Warnings: 0
Rule Violations: 0

Summary

Warnings	Count
Total	**0**

Rule Violations	Count
Clearance Constraint (Gap=0.254mm) (All),(All)	0
Short-Circuit Constraint (Allowed=No) (All),(All)	0
Un-Routed Net Constraint ((All))	0
Modified Polygon (Allow modified: No), (Allow shelved: No)	0
Width Constraint (Min=0.254mm) (Max=0.254mm) (Preferred=0.254mm) (All)	0
Power Plane Connect Rule(Relief Connect)(Expansion=0.508mm) (Conductor Width=0.254mm) (Air Gap=0.254mm) (Entries=4) (All)	0
Hole Size Constraint (Min=0.025mm) (Max=2.54mm) (All)	0
Hole To Hole Clearance (Gap=0.254mm) (All),(All)	0
Minimum Solder Mask Sliver (Gap=0.254mm) (All),(All)	0
Silk To Solder Mask (Clearance=0.254mm) (IsPad),(All)	0
Silk to Silk (Clearance=0.254mm) (All),(All)	0
Net Antennae (Tolerance=0mm) (All)	0
Height Constraint (Min=0mm) (Max=25.4mm) (Prefered=12.7mm) (All)	0
Total	**0**