```
/*

CDE/Hygain Rotor Controller
Emulates Yeasu GS-232A Azimuth Controller
Written by Glen Popiel, KW5GP


 This program is free software: you can redistribute it and/or modify
 it under the terms of the version 3 GNU General Public License as
 published by the Free Software Foundation.

 This program is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 GNU General Public License for more details.

 You should have received a copy of the GNU General Public License
 along with this program.  If not, see <http://www.gnu.org/licenses/>.

 */

// Debug Mode must be off to communicate with Ham Radio Deluxe
#define debug_mode 0  // Set to 1 for debug data on Serial Port

#include <Wire.h>  // Include the I2C Communication Library
#include <EEPROM.h>  // Include the EEPROM Library
#include "ADS1115.h" // Include the ADS1115.h Library (Library Updated to
fix errors)
#include "I2Cdev.h"  // Include I2CDev.h Library (Library Updated to fix
errors)

ADS1115 adc; // Define the ADS1115 as adc

#define rotate_left 4  // Define Rotate Left as Pin 4
#define rotate_right 5  // Define Rotate Right as Pin 5
#define brake 6  // Define the Brake pin

#define BAUD_RATE 9600  // Set the Serial Port Baud rate to 9600

#define EEPROM_ID_BYTE 1   // EEPROM ID to validate EEPROM data location
#define EEPROM_ID  55  // EEPROM ID Value
#define EEPROM_AZ_CAL_0 2    // Azimuth Zero Calibration EEPROM location
#define EEPROM_AZ_CAL_MAX 4  // Azimuth Max Calibration Data EEPROM
location

#define AZ_CAL_0_DEFAULT 0        // Preset the Azimuth Zero Calibration
Point to 0
#define AZ_CAL_MAX_DEFAULT 7000        // Preset the Azimuth Max
Calibration Point to 27000

#define AZ_Tolerance 2  // Set the Azimuth Accuracy Tolerance

//variables
byte inByte = 0;  // incoming serial byte
```

```
byte serial_buffer[50];  // incoming serial byte buffer
int serial_buffer_index = 0;  // The index pointer variable for the
Serial buffer
int set_AZ;  // Azimuth set value
int current_AZ;  // Current Azimuth raw value
String Serial_Send_Data; // Data to send to Serial Port
int AZ_0;  // Azimuth Zero Value from EEPROM
int AZ_MAX; // Azimuth Max Value from EEPROM
int AZ_Degrees; // mapped AZ ADC value to Degrees
String Requested_AZ; // RS232 Requested Azimuth - M and short W command
int AZ_To; // Requested AZ Move
int AZ_Distance; // Distance to move AZ

void setup()
{
  pinMode(rotate_left, OUTPUT); // Define the Control Pins as Outputs
  pinMode(rotate_right, OUTPUT);
  pinMode(brake, OUTPUT);

  digitalWrite(rotate_left, LOW);  // Turn off all the relays just to be
sure
  digitalWrite(rotate_right, LOW);
  digitalWrite(brake, LOW);

  Serial.begin(BAUD_RATE); // initialize serial communication

  Wire.begin();  // join I2C bus

  adc.initialize(); // initialize ADS1115 16 bit A/D chip

  Wire.beginTransmission(0x48); // Begin direct ADC communication
  Wire.write(0x1);    //  Connect to adc and send two bytes - Set Config
Reg to all Ones
  Wire.write(0x7F);  //  MSB
  Wire.write(0xFF);    //  LSB
  Wire.endTransmission();  // End the direct ADC Communication

  adc.setMode(ADS1115_MODE_CONTINUOUS); // Set the ADC to free running
conversion mode
  adc.setGain(ADS1115_PGA_0P256);  // set the ADC gain to 0.256 Volt
range, 0.007813 Volts/step
  adc.setRate(ADS1115_RATE_32);  // set ADC sample rate to 32 samples per
second
  adc.setMultiplexer(ADS1115_MUX_P0_N1);  // Set the ADC to AN0+ AN1
Differential Mode

  set_AZ = -1;  // Preset the Azimuth and Elevation Move Variables

  read_eeprom_cal_data();  // Read the Azimuth and Elevation Calibration
Values from EEPROM

}  // End Setup Loop

void loop()
```

```
{
  check_serial();  // Check the Serial Port for Data
  check_move();  // Check to see if executing move command
} // End Main Loop

// Functions

void read_eeprom_cal_data()  // Function to Read the Azimuth and
Elevation Calibration Data
{
  if (EEPROM.read(EEPROM_ID_BYTE) == EEPROM_ID) // Verify the EEPROM has
valid data
  {
    if (debug_mode) // If in Debug Mode Print the Calibration Values
    {
      Serial.println("Read EEPROM Calibration Data Valid ID");
      Serial.println((EEPROM.read(EEPROM_AZ_CAL_0) * 256) +
EEPROM.read(EEPROM_AZ_CAL_0 + 1),DEC);
      Serial.println((EEPROM.read(EEPROM_AZ_CAL_MAX) * 256) +
EEPROM.read(EEPROM_AZ_CAL_MAX + 1),DEC);
    }

    AZ_0 = (EEPROM.read(EEPROM_AZ_CAL_0)*256) +
EEPROM.read(EEPROM_AZ_CAL_0 + 1);  // Read the Azimuth Zero Calibration
Value from EEPROM
    AZ_MAX = (EEPROM.read(EEPROM_AZ_CAL_MAX)*256) +
EEPROM.read(EEPROM_AZ_CAL_MAX + 1);  // Read the Azimuth Maximum
Calibration Value from EEPROM

  } else {  // initialize eeprom to default values
  if (debug_mode)
  {
    Serial.println("Read EEPROM Calibration Data Invalid ID - setting to
defaults");
  }
  AZ_0 = AZ_CAL_0_DEFAULT;  // Set the Calibration To Default Values
  AZ_MAX = AZ_CAL_MAX_DEFAULT;
  write_eeprom_cal_data();  // Write the Default Values to EEPROM
  }
}

void write_eeprom_cal_data() // Function to Write the Calibration Values
to EEPROM
{
  Serial.println("Writing EEPROM Calibration Data");

  EEPROM.write(EEPROM_ID_BYTE,EEPROM_ID);  //   Write the EEPROM ID
  EEPROM.write(EEPROM_AZ_CAL_0,highByte(AZ_0));  // Write the Azimuth
Zero Calibration High Order Byte
  EEPROM.write(EEPROM_AZ_CAL_0 + 1,lowByte(AZ_0));    // Write the
Azimuth Zero Calibration Low Order Byte
  EEPROM.write(EEPROM_AZ_CAL_MAX,highByte(AZ_MAX));  // Write the Azimuth
Max Calibration High Order Byte
```

```
  EEPROM.write(EEPROM_AZ_CAL_MAX + 1,lowByte(AZ_MAX));  // Write the
Azimuth Max Calibration Low Order Byte
}

void check_serial() // Function to check for data on the Serial port
{
  if (Serial.available() > 0) // Get the Serial Data if available
  {
    inByte = Serial.read();  // Get the Serial Data

   // You may need to uncomment the following line if your PC software
   // will not communicate properly with the controller
   //   Serial.print(char(inByte));  // Echo back to the PC

    if (inByte == 10)  // ignore Line Feeds
    {
    return;
    }
    if (inByte !=13)  // Add to buffer if not CR
    {
      serial_buffer[serial_buffer_index] = inByte;

      if (debug_mode) // Print the Character received if in Debug mode
      {
        Serial.print("Received = ");
        Serial.println(serial_buffer[serial_buffer_index]);
      }

      serial_buffer_index++;  // Increment the Serial Buffer pointer

    } else {  // It's a Carriage Return, execute command

    if ((serial_buffer[0] > 96) && (serial_buffer[0] < 123))  //If first
character of command is lowercase, convert to uppercase
    {
      serial_buffer[0] = serial_buffer[0] - 32;
    }

    switch (serial_buffer[0]) {  // Decode first character of command

      case 65:  // A Command - Stop the Azimuth Rotation

        if (debug_mode) {Serial.println("A Command Received");}
        az_rotate_stop();
        break;

      case 67:      // C - return current azimuth

        if (debug_mode)   // Return the Buffer Index Pointer in Debug
Mode
        {
          Serial.println("C Command Received");
          Serial.println(serial_buffer_index);
        }
```

```
  send_current_az();  // Return Azimuth if C Command
  break;

case 70:  // F - Set the Max Calibration

  if (debug_mode)
  {
    Serial.println("F Command Received");
    Serial.println(serial_buffer_index);
  }
  set_max_az_cal();  // F - Set the Max Azimuth Calibration
  break;

case 76:  // L - Rotate Azimuth CCW

  if (debug_mode)
  {
    Serial.println("L Command Received");
  }
  rotate_az_ccw();  // Call the Rotate Azimuth CCW Function
  break;

case 77:  // M - Rotate to Set Point

  if (debug_mode)
  {
    Serial.println("M Command Received");
  }
  rotate_to();  // Call the Rotate to Set Point Command
  break;

case 79:  // O - Set Zero Calibration

  if (debug_mode)
  {
    Serial.println("O Command Received");
    Serial.println(serial_buffer_index);
  }
  set_0_az_cal();  // O - Set the Azimuth Zero Calibration
  break;

case 82:  // R - Rotate Azimuth CW

  if (debug_mode)
  {
    Serial.println("R Command Received");
  }
  rotate_az_cw();  // Call the Rotate Azimuth CW Function
  break;

case 83:  // S - Stop All Rotation

  if (debug_mode)
  {
```

```
          Serial.println("S Command Received");
        }
        az_rotate_stop();  // Call the Stop Azimith Rotation Function
        break;

      }

      serial_buffer_index = 0;  // Clear the Serial Buffer and Reset the
Buffer Index Pointer
      serial_buffer[0] = 0;
    }
  }
}

void send_current_az() // Send the Current Azimuth Function
{
  read_adc();  // Read the ADC

  // Map Azimuth to degrees
  if (debug_mode)
  {
    Serial.println(current_AZ);
  }
  AZ_Degrees = map(current_AZ, AZ_0, AZ_MAX, 0, 360);  // Map the Current
Azimuth to Degrees
  if (AZ_Degrees > 180) // Correction Since Azimuth Reading starts at
Meter Center Point
  {
    AZ_Degrees = AZ_Degrees - 180;
  } else {
    AZ_Degrees = AZ_Degrees + 180;
  }
  if (debug_mode)
  {
    Serial.println(AZ_Degrees);
  }
  // Send it back via serial
  Serial_Send_Data = "";
  if (AZ_Degrees < 100)  // pad with 0's if needed
  {
    Serial_Send_Data = "0";
  }
  if (AZ_Degrees < 10)
  {
    Serial_Send_Data = "00";
  }
  Serial_Send_Data = "+0" + Serial_Send_Data + String(AZ_Degrees);  //
Send the Azimuth in Degrees
  Serial.println(Serial_Send_Data);  // Return value via RS-232 port
}

void set_max_az_cal() // Set the Max Azimuth Calibration Function
{
  Serial.println("Calibrate Max AZ Function");
```

```
  read_adc();  // Read the ADC

  // save current az and el values to EEPROM - Zero Calibration
  Serial.println(current_AZ);

  AZ_MAX = current_AZ;  // Set the Azimuth Maximum Calibration to Current
Azimuth Reading
  write_eeprom_cal_data();  // Write the Calibration Data to EEPROM
  Serial.println("Max Azimuth Calibration Complete");
}

void rotate_az_ccw() // Function to Rotate Azimuth CCW
{
  digitalWrite(brake, HIGH);  // Release the brake
  digitalWrite(rotate_left, HIGH);  // Set the Rotate Left Pin High
  digitalWrite(rotate_right, LOW);  // Make sure the Rotate Right Pin is
Low
}

void rotate_az_cw() // Function to Rotate Azimuth CW
{
  digitalWrite(brake, HIGH);  // Release the brake
  digitalWrite(rotate_right, HIGH);    // Set the Rotate Right Pin High
  digitalWrite(rotate_left, LOW);    // Make sure the Rotate Left Pin Low
}

void az_rotate_stop() // Function to Stop Azimuth Rotation
{
  digitalWrite(rotate_right, LOW);  // Turn off the Rotate Right Pin
  digitalWrite(rotate_left, LOW);   // Turn off the Rotate Left Pin
  set_AZ = -1;
  delay(1000);
  digitalWrite(brake, LOW);  // Engage the brake
}

void rotate_to() // Function to Rotate to Set Point
{
  if (debug_mode)
  {
    Serial.println("M Command -  Rotate Azimuth To Function");
  }
  // Decode Command - Format Mxxx - xxx = Degrees to Move to
  if (debug_mode)
  {
    Serial.println(serial_buffer_index);
  }
  if (serial_buffer_index == 4)  // Verify the Command is the proper
length
  {
    if (debug_mode)
    {
      Serial.println("Value in [1] to [3]?");
    }
```

```
    Requested_AZ = (String(char(serial_buffer[1])) +
String(char(serial_buffer[2])) + String(char(serial_buffer[3]))) ;  //
Decode the Azimuth Value
    AZ_To = (Requested_AZ.toInt()); // AZ Degrees to Move to as integer
    if (AZ_To <0) // Make sure we don't go below 0 degrees
    {
      AZ_To = 0;
    }
    if (AZ_To >360) // Make sure we don't go over 360 degrees
    {
      AZ_To = 360;
    }
    if (AZ_To > 180) // Adjust for Meter starting at Center
    {
      AZ_To = AZ_To - 180;
    } else {
      AZ_To = AZ_To + 180;
    }
    if (debug_mode)
    {
      Serial.println(Requested_AZ);
      Serial.println(AZ_To);
    }

    // set the move flag and start
    read_adc();  // Read the ADC

    // Map it to degrees
    if (debug_mode)
    {
      Serial.println(current_AZ);
    }
    AZ_Degrees = map(current_AZ, AZ_0, AZ_MAX, 0, 360);  // Map the
Azimuth Value to Degrees
    if (debug_mode)
    {
      Serial.println(AZ_Degrees);
    }
    AZ_Distance = AZ_To - AZ_Degrees;  // Figure out far we have to move
    set_AZ = AZ_To;
    if (abs(AZ_Distance) <= AZ_Tolerance)  // No move needed if we're
within the Tolerance Range
    {
      az_rotate_stop();  // Stop the Azimuth Rotation
      set_AZ = -1;  // Turn off the Move Command
    } else {  // Move Azimuth - figure out which way
      if (AZ_Distance > 0)   //We need to move CW
      {
        rotate_az_cw();  // If the distance is positive, move CW
      } else {
        rotate_az_ccw();  // Otherwise, move counterclockwise
      }
    }
  }
```

```
}

void set_0_az_cal() // Set Azimuth Zero Calibration
{
  Serial.println("Calibrate Zero Function");

  read_adc();  // Read the ADC
  // save current Azimuth value to EEPROM - Zero Calibration
  Serial.println(current_AZ);

  AZ_0 = current_AZ;  // Set the Azimuth Zero Calibration to current
position
  write_eeprom_cal_data();  // Write the Calibration Data to EEPROM
  Serial.println("Zero Azimuth Calibration Complete");
}

void read_adc() // Function to read the ADC
{
 if (debug_mode)
 {
   Serial.println("Read ADC Function  ");
 }

  int RotorValue;  // Variable to store the rotor value
  adc.setRate(ADS1115_RATE_32); // Set the ADC rate to 32 samples/sec
  adc.setGain(ADS1115_PGA_0P256); // Set the ADC gain to 0.007813 Volts
  adc.setMultiplexer(ADS1115_MUX_P0_N1);  // Set the ADC to Channel 0
AN0+ AN1 Differential Mode
  delay(100); // adc settling delay

  current_AZ = adc.getDifferential();  // Read ADC Channel 0 and 1
Differentially
}

void check_move() // Check to see if we've been commanded to move
{
  if (set_AZ != -1) {   // We're moving - check and stop as needed
    read_adc();  // Read the ADC
    // Map AZ to degrees
    if (debug_mode)
    {
      Serial.println(current_AZ);
    }
    AZ_Degrees = map(current_AZ, AZ_0, AZ_MAX, 0, 360);  // Map the
Current Azimuth reading to Degrees

    if (debug_mode)
    {
      Serial.println(AZ_Degrees);
    }

    if (set_AZ != -1) // If Azimuth is moving
    {
```

```
      AZ_Distance = set_AZ - AZ_Degrees;  // Check how far we have to
move
      if (abs(AZ_Distance) <= AZ_Tolerance)  // No move needed if we're
within the tolerance range
      {
        az_rotate_stop();  // Stop the Azimuth Rotation
        set_AZ = -1;  // Turn off the Azimuth Move Command
      } else {  // Move Azimuth - figure out which way
        if (AZ_Distance > 0)   //We need to move CW
        {
          rotate_az_cw();  // Rotate CW if positive
        } else {
          rotate_az_ccw();  // Rotate CCW if negative
        }
      }
    }
  }
}
```