```
/*
  CDE Rotor 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/>.

 */

#define debug_mode 1  // Enables Debug Mode - Sends Debug output to
Serial Monitor

#include <Wire.h>    //I2C Library
#include <LiquidCrystal_I2C.h>  // Liquid Crystal I2C Library
#include <EEPROM.h>  // Include EEPROM Library
#include "ADS1115.h" // Include ADS1115 ADC Library - Library Updated to
fix errors
#include "I2Cdev.h"  // Include I2Cdev Library - Library Updated to fix
errors
#include "Timer.h"  // Timer Library so we can do timed interrupts

#define lcd_end 16 // set width of LCD
#define lcd_address 0x27 // I2C LCD Address
#define lcd_lines 2 // Number of lines on LCD

const int comm_speed = 9600;  // Set the Serial Monitor Baud Rate

#define left 2   // Assign Left (Counter Clockwise) Relay to Pin 3
#define right 3  // Assign Right (Clockwise) Relay to Pin 2
#define brake 4  // Assign Brake Relay to Pin 4
#define rotate_CW 5  // Assign CCW switch to Pin 5
#define rotate_CCW 6   // Assign CW switch to Pin 6
#define cal_zero 7    // Assign Zero Calibrate switch to Pin 7
#define cal_360 8     // Assign 360 Calibrate Switch to Pin 8
#define red 9  // Assign the Red LED to Pin 9
#define blue 10  // Assign the Blue LED to Pin 10
#define green 11  // Assign the Green LED to Pin 11

#define adc_gain 0x04   // Set ADC Gain to 0 to 0.512 volts

#define brake_delay 3000  // Set the Brake Delay to 3 seconds

#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
#define AZ_CAL_MAX_DEFAULT 27000

int current_AZ;  // Variable for current Azimuth ADC Value
int AZ_Degrees;  // Variable for Current Azimuth in Degrees
boolean moving = false;  // Variable to let us know if the rotor is
moving
boolean timing = false;  // Variable to let us know that rotation timing
has started

int set_AZ;  // Azimuth set value
int AZ_0;  // Azimuth Zero Value from EEPROM
int AZ_MAX; // Azimuth Max Value from EEPROM

int tickEvent;
int derived_degrees;  // Integer variable for position calculation while
moving
float calculated_degrees;  // Variable for position calculation while
moving

float left_rotate_speed = 1.8341;  // Left rotational speed in degrees
per 250ms
float right_rotate_speed = 1.8369;  // Right rotational speed in degrees
per 250ms
String direction;

LiquidCrystal_I2C lcd(lcd_address,lcd_end,lcd_lines);  // set the LCD I2C
address to 0x27 for a 16 chars and 2 line display

Timer t;  // Create Timer object as t

ADS1115 adc; // Define as adc

void setup()
{
  // initialize the digital pins

  Serial.begin(comm_speed);  // Start the Serial port

  lcd.init();  // initialize the LCD
  lcd.backlight();  // Turn on the LCD backlight
  lcd.home();  // Set the cursor to line 0, column 0

  lcd.print("Rotor Controller");  // Display startup message
  lcd.setCursor(4,1);
  lcd.print("by KW5GP");

  pinMode(right, OUTPUT);  // Define the Input and Output pins
  pinMode(left, OUTPUT);
```

```
  pinMode(brake, OUTPUT);
  pinMode(rotate_CCW, INPUT);
  pinMode(rotate_CW, INPUT);
  pinMode(cal_zero, INPUT);
  pinMode(cal_360, INPUT);
  digitalWrite(rotate_CCW, HIGH);  // Enable the internal pullup
resistors
  digitalWrite(rotate_CW, HIGH);
  digitalWrite(cal_zero, HIGH);
  digitalWrite(cal_360, HIGH);
  pinMode(red, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(green, OUTPUT);

  read_eeprom_cal_data();  // Read the EEPROM calibration data

  if (debug_mode)  // Display the calibration data when in debug mode
  {
    Serial.print("Calbration Values - Zero = ");Serial.print(AZ_0);
Serial.print(" Max = "); Serial.println(AZ_MAX);
  }

  ledOff();  // Turn off LED

  adc.setRate(ADS1115_RATE_128); // Set the ADC sample speed to 128
samples/sec
  delay(100);  // wait for the ADC to process command

  adc.setGain(adc_gain);  // Set the ADC gain
  delay(100);  // wait for the ADC to process command

  adc.setMultiplexer(ADS1115_MUX_P0_NG);  // Set the ADC Channel 0 to
single ended mode
  delay(100); // adc settling delay

  delay(5000);  //Wait 5 seconds then clear the startup message
  lcd.clear();

}  // End Setup Loop

void loop()
{
  t.update();  // Update the Interrupt handler

  if (!moving)  // Read the ADC normally as long as the motor or brake
are not energized
  {
    // Read ADC and display position when we're not moving
    read_adc();  // Read the Anaolog to Digital Converter - value is
placed in current_AZ
    AZ_Degrees = map(current_AZ, AZ_0, AZ_MAX, 0, 360);  // Map the
current_AZ to calibrated degrees
```

```
    if (debug_mode) {Serial.println(AZ_Degrees);}  // Send position to
Serial Monitor in debug mode

    lcd.setCursor(6,0);  // Display the current position on the LCD
    if (AZ_Degrees <0)  // Set position to 0 degrees if below 0
    {
      AZ_Degrees = 0;
    }
    if (AZ_Degrees > 360)  // Set position to 360 degrees if above 360
    {
      AZ_Degrees = 360;
    }
    if (AZ_Degrees < 100)  // Adjust spacing on LCD
    {
      lcd.print(" ");
    }
    lcd.print(AZ_Degrees);  // Display the position on the LCD
    lcd.print(char(223));  // Add the degree symbol
    lcd.print("   ");
  } else {
    // We're moving, ADC reading is useless - too much noise
    if (!timing)  // Check to see if we're already timing the event
    {
        // We're just  starting to move - start the interrupt timer
        tickEvent = t.every(250, Tick);  // Set to interrupt every 250ms
        if (debug_mode)  // Send Interrupt start message in debug mode
        {
          Serial.print("250ms second tick started id=");
          Serial.println(tickEvent);
        }
        timing = true;  // Indicates we're timing the move
        calculated_degrees = AZ_Degrees;  // Set the starting point for
the move
    }
  }

  // Read the rotate switch
  if (digitalRead(rotate_CCW) == LOW || digitalRead(rotate_CW) == LOW)
// Check if the Rotate Switch is activated
  {
    if (digitalRead(rotate_CW) == LOW && !moving)  // Check for Move
Right (CW) Switch
    {
      // Rotate CW
      move_right();
    }
    if (digitalRead(rotate_CCW) == LOW && !moving)  // Check for Move
Left (CCW) Switch
    {
      // Rotate CCW
      move_left();
    }
  } else {
    if (moving)  // If we were moving, time to stop
```

```
    {
      moving = false;  // Turn off the moving flag
      ledRed();  // Turn the LED Red to indicate Brake Cycle
      all_stop();  // Stop Rotation
    }
  }

  // Read the Zero Degree Calibrate Switch
  if (digitalRead(cal_zero) == LOW)
  {
    //  Cal Zero switch pressed
    AZ_0 = current_AZ;  // Set the current position as the zero
calibration point
    write_eeprom_cal_data();  // Write the Calibration data to the EEPROM
    if (debug_mode)  // Display Calibration Data in debug mode
    {
      Serial.print("Zero Azimuth Calibration Complete -  Zero = ");
      Serial.print(AZ_0); Serial.print(" Max = ");
Serial.println(AZ_MAX);
    }
  }

  // Read the 360 Degree Calibrate Switch
  if (digitalRead(cal_360) == LOW)
  {
    //  Cal 360 switch pressed
    AZ_MAX = current_AZ -25;  // Adjust top end a bit to allow for jitter
    write_eeprom_cal_data();  // Write the Calikbration Data to the
EEPROM
    if (debug_mode)  // Display Calibration Data in debug mode
    {
      Serial.println("Max Azimuth Calibration Complete - Zero = ");
      Serial.print(AZ_0); Serial.print(" Max = ");
Serial.println(AZ_MAX);
    }
  }
}  // End Main Loop


// Relay Off function - stops motion, delays 3 seconds then turns off
Brake Relay
void all_stop()
{
  lcd.setCursor(0,1);  // Display Braking message on LCD
  lcd.print("    Braking     ");
  digitalWrite(right, LOW);  // Turn off CW Relay
  digitalWrite(left, LOW);  // Turn off CCW Relay
  timing = false;  // turn off the timing flag
  direction = "S";  // Set direction to S (Stop)
  t.stop(tickEvent);  // Turn off the Timer interrupts
  delay(brake_delay);  // Wait for rotor to stop
  digitalWrite(brake, LOW);  // Engage the Rotor Brake
  lcd.setCursor(0,1);  // Clear the Braking Message
  lcd.print("                ");
```

```
  ledOff();  // Turn off the LED
}

void move_left()  // Turn the rotor Left (CCW)
{
  ledGreen();  // Turn on the Green LED
  moving = true;  // set the moving flag
  direction = "L";  // set the direction flag to "L" (Left)
  lcd.setCursor(0,1);  // display the left arrow on the LCD
  lcd.print(char(127));
  digitalWrite(brake, HIGH);  // Release the Brake
  digitalWrite(left, HIGH);  // Energize the Left Drive Relay
}

void move_right()  // Turn the rotor Right (CW)
{
  ledBlue();  // Turn on the Blue LED
  moving = true;  // set the moving flag
  direction = "R";  // set the direction flag to "R" (Right)
  lcd.setCursor(15,1);  // display the right arrow on the LCD
  lcd.print(char(126));
  digitalWrite(brake, HIGH);  // Release the Brake
  digitalWrite(right, HIGH);  // Energize the Right Drive Relay
}

// LED Off function
void ledOff()
{
  digitalWrite(red, LOW);  // Set all RBG LED pins High (Off)
  digitalWrite(green, LOW);
  digitalWrite(blue, LOW);
}

// RED LED ON function
void ledRed()
{
  digitalWrite(red, HIGH);  // Turn on the RGB Red LED On
  digitalWrite(green, LOW);
  digitalWrite(blue, LOW);
}

// Green LED ON function
void ledGreen()
{
  digitalWrite(red, LOW);
  digitalWrite(green, HIGH);  // Turn on the RGB Green LED On
  digitalWrite(blue, LOW);
}

// Blue LED ON function
void ledBlue()
{
  digitalWrite(red, LOW);
  digitalWrite(green, LOW);
```

```
  digitalWrite(blue, HIGH);  // Turn on the RGB Blue LED On
}

// White LED function
void ledWhite()  // Turn on all LEDs to get white
{
  digitalWrite(red, HIGH);
  digitalWrite(green, HIGH);
  digitalWrite(blue, HIGH);
}

// Read the A/D Converter
void read_adc()
{
 if (debug_mode) {Serial.print("Read ADC Function  ");}  // display ADC
read status in debug mode

  adc.setRate(ADS1115_RATE_128);  // Set the ADC sample rate to 128
samples/second
  delay(10);  // Wait for ADC to settle

  adc.setGain(adc_gain);  // Set the ADC gain
  delay(10);  // Wait for ADC to settle

  adc.setMultiplexer(ADS1115_MUX_P0_NG);  // Set the ADC to single-ended
mode
  delay(100);  // Wait for ADC to settle and start sampling

  current_AZ = adc.getDiff0();  // Read ADC channel 0

  if (debug_mode) {Serial.println(current_AZ);}  // Display ADC value in
debug mode
}

void read_eeprom_cal_data()  // Read the EEPROM Calibration data
{
  if (EEPROM.read(EEPROM_ID_BYTE) == EEPROM_ID)   // Verify the EEPROM
has valid data
  {
    if (debug_mode) // Display the Calibration data in debug mode
    {
      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);  // Set the Zero degree Calibration
Point
    AZ_MAX = (EEPROM.read(EEPROM_AZ_CAL_MAX)*256) +
EEPROM.read(EEPROM_AZ_CAL_MAX + 1);  // Set the 360 degree Calibration
Point
```

```
  } else {  // EEPROM has no Calibration data - initialize eeprom to
default values
  if (debug_mode)
  {
    // Send status message in debug mode
    Serial.println("Read EEPROM Calibration Data Invalid ID - setting to
defaults");
  }
  AZ_0 = AZ_CAL_0_DEFAULT;  // Set the Calibration data to default values
  AZ_MAX = AZ_CAL_MAX_DEFAULT;
  write_eeprom_cal_data();  // Write the data to the EEPROM
  }
}

void write_eeprom_cal_data() // Write the Calibration data to the EEPROM
{
    if (debug_mode)
  {
    Serial.println("Writing EEPROM Calibration Data");  // Display status
in debug mode
  }

  EEPROM.write(EEPROM_ID_BYTE,EEPROM_ID);  // Write the EEPROM ID to the
EEPROM
  EEPROM.write(EEPROM_AZ_CAL_0,highByte(AZ_0));  // Write Zero
Calibration Data High Order Byte
  EEPROM.write(EEPROM_AZ_CAL_0 + 1,lowByte(AZ_0));  // Write Zero
Calibration Data Low Order Byte
  EEPROM.write(EEPROM_AZ_CAL_MAX,highByte(AZ_MAX));  // Write 360
Calibration Data High Order Byte
  EEPROM.write(EEPROM_AZ_CAL_MAX + 1,lowByte(AZ_MAX));  // Write 360
Calibration Data Low Order Byte
}

void Tick()  // Timer Interrupt Handler
{
  if (debug_mode)  // Display Interrupt information in debug mode
  {
    Serial.print("250ms second tick: millis()=");
    Serial.print(millis());
    Serial.print("   ");
  }
  // We're timing the rotation - Add the estimated distance traveled
  // to the current calculated positon
  if (direction == "R")
  {
    calculated_degrees = calculated_degrees + right_rotate_speed;  //
Increase when we move right
  } else {
    calculated_degrees = calculated_degrees - left_rotate_speed;  //
Decrease when we move left
  }
  if (debug_mode)  // Display Calculated Rotation information in debug
mode
```

```
  {
    Serial.print("   ");
    Serial.print("Rotating ");
    Serial.print(direction);
    Serial.print("   ");
    Serial.println(calculated_degrees);
  }
  lcd.setCursor(6,0);  // Update the LCD with the estimated position
  derived_degrees = (int)calculated_degrees;
  if (derived_degrees < 0)  // Set to Zero if we calculate below zero
  {
    derived_degrees = 0;
  }
  if (derived_degrees > 360)  // Set to 360 if we calculate above 360
  {
    derived_degrees = 360;
  }
  if (derived_degrees < 100)
    {
      lcd.print(" ");
    }
    lcd.print(derived_degrees);  // Display the position on the LCD
    lcd.print(char(223));  // Add the degree symbol
    lcd.print("   ");

}
```