Stepper Temperature

Dan Blanchette

May 5, 2023

1 Source Code

1.1 main.cpp

```
/**
 * @file main.cpp
 * @author Dan Blanchette
 * @brief  This program will run a web sever on the ESP32 Core 0 (Uses Loop Function).
 * RTOS tasks for 2 I2C devices(temp/hum sensor, sunlight sensor, and the stepper motor on ESP32's processor
 * are ran on core 1.
 * Credit: James Lasso for help with RESTful and IoT server functionality.
 * @version 0.1
 * @date 2023-04-12
 *
 * @copyright Copyright (c) 2023
 *
 */
#include "devices.h"
#include <string.h>

Adafruit_TSL2591 tsl = Adafruit_TSL2591(2591);

void displaySensorDetails(void);

// Task handles
TaskHandle_t webServerTask;
TaskHandle_t iotServerTask;
TaskHandle_t RTOS_Tasks;

// HDC1080 Class Object
ClosedCube_HDC1080 hdc1080;

/******************WEB CLIENT VARIABLES******************/
// current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in miliseconds
const long timeoutTime = 2000;
```

```
36
37 // HTTPClient object
38 HTTPClient http;
39
40 // Queue Handles for stepper direction
41 static QueueHandle_t xStateQueue = NULL;
42 // Queue Handle for Sunlight Sensor
43 static QueueHandle_t xVisibleQueue = NULL;
44 static QueueHandle_t xInfraredQueue = NULL;
45 static QueueHandle_t xFullSpectQueue = NULL;
46 // Queue Handle for Temp/Hum Sensor
47 // celsius values
48 static QueueHandle_t xCtempQueue = NULL;
49 // fahrenheit values
50 static QueueHandle_t xFtempQueue = NULL;
51 static QueueHandle_t xRhQueue = NULL;
52
53 // Web Server Setup
54 WiFiServer server(80);
55 String header;
56
57 // Auxiliar variables to store current output state
58 String output13State = "off";
59
60 // detect server IP address
61 String serverDet = "http://52.23.160.25:5000/IOTAPI/DetectServer";
62 String serverReg = "http://52.23.160.25:5000/IOTAPI/
       RegisterWithServer";
63 String serverData = "http://52.23.160.25:5000//IOTAPI/IOTData";
64 String serverQueryForCommands = "http://52.23.160.25:5000//IOTAPI/
       QueryServerForCommands";
65
66 // Home WiFi credentials
67 // censor this before submitting or pushing to git
68 const char *ssid = "sending-stone";
69 const char *password = "4579W!$hThis#";
70
71 /**********************TSL2591**********************/
72 void configureSensor(void)
73 {
74   tsl.setGain(TSL2591_GAIN_MED);
75   tsl.setTiming(TSL2591_INTEGRATIONTIME_300MS);
76
77   Serial.println(F("------------------------------------"));
78   Serial.print(F("Gain:         "));
79   tsl2591Gain_t gain = tsl.getGain();
80   switch (gain)
81   {
82   case TSL2591_GAIN_LOW:
83     Serial.println(F("1x (Low)"));
84     break;
85   case TSL2591_GAIN_MED:
86     Serial.println(F("25x (Medium)"));
87     break;
88   case TSL2591_GAIN_HIGH:
89     Serial.println(F("428x (High)"));
90     break;
```

```
  case TSL2591_GAIN_MAX:
    Serial.println(F("9876x (Max)"));
    break;
  }
  Serial.print(F("Timing:       "));
  Serial.print((tsl.getTiming() + 1) * 100, DEC);
  Serial.println(F(" ms"));
  Serial.println(F("------------------------------------"));
  Serial.println(F(""));
}



/**
 * @brief   // Simple data read example. Just read the infrared, fullspecrtrum diode
  // or 'visible' (difference between the two) channels.
  // This can take 100-600 milliseconds! Uncomment whichever of the following you want to read
 *
 */
void simpleRead(void)
{
  uint16_t vis = tsl.getLuminosity(TSL2591_VISIBLE);
  uint16_t fs = tsl.getLuminosity(TSL2591_FULLSPECTRUM);
  uint16_t ir = tsl.getLuminosity(TSL2591_INFRARED);
  xQueueSend(xVisibleQueue, &vis, 0U);
  vTaskDelay(20 / portTICK_PERIOD_MS);
  xQueueSend(xInfraredQueue, &ir, 0U);
  vTaskDelay(20 / portTICK_PERIOD_MS);
  xQueueSend(xFullSpectQueue, &fs, 0U);
  vTaskDelay(20 / portTICK_PERIOD_MS);
}

/**
 * @brief   // More advanced data read example. Read 32 bits with top 16 bits IR, bottom 16 bits full spectrum
  // That way you can do whatever math and comparisons you want!
 *
 */
void advancedRead(void)
{

  uint32_t lum = tsl.getFullLuminosity();
  uint16_t ir, full;
  ir = lum >> 16;
  full = lum & 0xFFFF;
  Serial.print(F("[ "));
  Serial.print(millis());
  Serial.print(F(" ms ] "));
  Serial.print(F("IR: "));
  Serial.print(ir);
  Serial.print(F("  "));
  Serial.print(F("Full: "));
  Serial.print(full);
  Serial.print(F("  "));
  Serial.print(F("Visible: "));
```

```
  Serial.print(full - ir);
  Serial.print(F("  "));
  Serial.print(F("Lux: "));
  Serial.println(tsl.calculateLux(full, ir), 6);
}

/*******************RTOS TASKS***************************/

void Task_IoT_Server_Data(void *parameter)
{
  while (1)
  {

    printf("Attempting to POST data....\n");
    const String auth_code = "8fe0f80a4e9f7bf3";
    float cel_val, fahr_val, rh_val;
    float light = 0;
    String JSON, response, reply;
    // get the fahrenheit value from the sensor
    xQueueReceive(xFtempQueue, &fahr_val, 0U);
    // get the humidity from the sensor
    xQueueReceive(xRhQueue, &rh_val, 0U);

    http.begin("http://52.23.160.25:5000//IOTAPI/IOTData");
    http.addHeader("Content-Type", "application/json");
    // // Send POST code for registration
    // int postCode = http.POST("{\"key\":\"2436e8c114aa64ee\",\"
    iotid\":1001}");
    // String response = http.getString();
    // Serial.print("HTTP Response Code: ");
    // Serial.println(postCode);
    // Serial.println(response);

    // send the data
    JSON += "{\"auth_code\": \"";
    JSON += auth_code;
    JSON += "\", \"temperature\": ";
    JSON += fahr_val;
    JSON += ", \"humidity\": ";
    JSON += rh_val;
    JSON += ", \"light\": ";
    JSON += light;
    JSON += "}";

    Serial.println(JSON);
    printf("\n");

    response = http.POST(JSON);
    Serial.println(response);

    vTaskDelay(10000 / portTICK_PERIOD_MS);
  }
}

/**
 * @brief Stepper Motor Task
 *
```

```
 * @param parameter
 */
void Task_Stepper(void *parameter)
{
  while (1)
  {
    int rec_val;
    // NOTE: step_dir(int dir) has vTaskDelay(10 /
    portTICK_PERIOD_MS)
    // Flag Directions (True = CW, False = CCW)
    xQueueReceive(xStateQueue, &rec_val, 0U);
    step_dir(rec_val);
  }
}

/**
 * @brief Task that reads the temperature and relative humidity
    from the HDC1080 Sensor
 *
 * @param parameter
 */
void Task_HDC1080(void *parameter)
{
  while (1)
  {
    // convert celsius to Fahr
    float celsius = hdc1080.readTemperature();
    float fahr = ((celsius * 1.8) + 32);
    // get relative humidity
    float rh = hdc1080.readHumidity();

    xQueueSend(xCtempQueue, &celsius, 0U);
    xQueueSend(xFtempQueue, &fahr, 0U);
    xQueueSend(xRhQueue, &rh, 0U);

    // Serial.print("T=");
    // Serial.print(hdc1080.readTemperature());
    // Serial.println("C");

    // Serial.print("T=");
    // Serial.print(fahr);
    // Serial.print("F RH=");
    // Serial.print(rh);
    // Serial.println("%");
    // read once every 3 seconds per port tick
    vTaskDelay(3000 / portTICK_PERIOD_MS);
  }
}

/**
 * @brief Task Reads Sun Sensor Data TR2591
 *
 * @param Parameters
 */
void Task_sunSensor(void *Parameters)
{
  while (1)
```

```
  {
    simpleRead();
  }
}

/**
 * @brief Arduino Device Setup Function
 *
 */
void setup()
{
  Serial.begin(115200);
  /****************RTOS DEVICES***********/
  // Stepper Motor Setup
  setup_stepper();
  // D13 LED Setup
  d13_setup();

  /*****************HDC1080(Temp/Humidity Sensor) AND TSL2591(
    Sunlight Sensor)***************/

  // Enable communication with the I2C Bus
  Wire.begin(SDA, SCL);

  /********HDC1080*****************/
  // setup defaults for HDC1080
  hdc1080.begin(0x40);

  /************SETUP TSL2591******************/
  // displaySensorDetails();
  configureSensor();

  /**********QUEUE INSTANTIATION********/
  // stepper
  xStateQueue = xQueueCreate(1, sizeof(int));

  // photosensor
  xVisibleQueue = xQueueCreate(1, sizeof(uint16_t));
  xInfraredQueue = xQueueCreate(1, sizeof(uint16_t));
  xFullSpectQueue = xQueueCreate(1, sizeof(uint16_t));

  // temp/hum sensor
  xCtempQueue = xQueueCreate(1, sizeof(float));
  xFtempQueue = xQueueCreate(1, sizeof(float));
  xRhQueue = xQueueCreate(1, sizeof(float));

  /************** WIFI SETUP *****************/
  // Connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
  Serial.println(WiFi.localIP());

```

```
312   WiFiClient client;
313
314   // Start up an ESP32 Web Server
315   server.begin();
316
317   /*************IOT Setup****************/
318
319   // Begin new connection to cloud website
320   http.begin("http://52.23.160.25:5000/");
321   int detServer = http.begin(serverDet);
322   int regServer = http.begin(serverReg);
323   Serial.println("IoT Server Connection: 1 = connected, 0 = error
        connecting: ");
324   Serial.printf("Connection Status: %d\n", regServer);
325
326   // // Register Device with the server
327   // http.addHeader("Content-Type", "application/json");
328   // // Send POST code for registration
329   // int postCode = http.POST("{\"key\":\"2436e8c114aa64ee\",\"
        iotid\":1001}");
330   // String response = http.getString();
331   // Serial.print("HTTP Response Code: ");
332   // Serial.println(postCode);
333   // Serial.println(response);
334
335   /********************** Create RTOS Tasks
        ************************************************************
        */
336
337   // Web Server and IoT RTOS Server Tasks
338   xTaskCreatePinnedToCore(Task_IoT_Server_Data, "
        Task_IoT_Server_Data", 10000, NULL, 4, &iotServerTask,
        core_zero);
339
340   // RTOS Tasks for Connected Devices
341   xTaskCreatePinnedToCore(Task_Stepper, "Task_Stepper", 10000, NULL
        , 4, &RTOS_Tasks, core_one);
342   xTaskCreatePinnedToCore(Task_HDC1080, "Task_HDC1080", 10000, NULL
        , 3, &RTOS_Tasks, core_one);
343   xTaskCreatePinnedToCore(Task_sunSensor, "Task_sunSensor", 10000,
        NULL, 2, &RTOS_Tasks, core_one);
344 }
345
346 // Handles ESP32 local web client server requests and user/client
        interactions with the stepper motor
347 /**
348  * @brief Arduino Loop Function
349  *
350  */
351 void loop()
352 {
353   WiFiClient client = server.available(); // Listen for incoming
        clients
354
355   int step_direction;
356   uint16_t vis_val;
357   uint16_t ir_val;
```

```
358   uint16_t fullSpec_val;
359
360   if (client)
361   { // If a new client connects,
362     currentTime = millis();
363     previousTime = currentTime;
364     Serial.println("New Client."); // print a message out in the
      serial port
365     String currentLine = "";       // make a String to hold
      incoming data from the client
366     while (client.connected() && currentTime - previousTime <=
      timeoutTime)
367     { // loop while the client's connected
368       currentTime = millis();
369       if (client.available())
370       {                            // if there's bytes to read from
      the client,
371         char c = client.read(); // read a byte, then
372         Serial.write(c);        // print it out the serial monitor
373         header += c;
374         if (c == '\n')
375         { // if the byte is a newline character
376           // if the current line is blank, you got two newline
      characters in a row.
377           // that's the end of the client HTTP request, so send a
      response:
378           if (currentLine.length() == 0)
379           {
380             // HTTP headers always start with a response code (e.g.
       HTTP/1.1 200 OK)
381             // and a content-type so the client knows what's coming
      , then a blank line:
382             client.println("HTTP/1.1 200 OK");
383             client.println("Content-type:text/html");
384             client.println("Connection: close");
385             client.println();
386
387             // turns the GPIOs on and off
388             if (header.indexOf("GET /13/on") >= 0)
389             {
390               // Serial.println("GPIO 13 on");
391               output13State = "on";
392               // Serial.println("Sending 1 to Queue.. value: ");
393               step_direction = 1;
394               xQueueSend(xStateQueue, &step_direction, 0U);
395               delay(50);
396               // Turn D13 on to indicate CW motion on the stepper
397               digitalWrite(output13, HIGH);
398             }
399             else if (header.indexOf("GET /13/off") >= 0)
400             {
401               // Serial.println("GPIO 13 off");
402               output13State = "off";
403               // Serial.println("Sending 1 to Queue.. value: ");
404               step_direction = 0;
405               xQueueSend(xStateQueue, &step_direction, 0U);
406               delay(50);
```

```
              // Turn D13 off to indicate CCW motion on stepper
              digitalWrite(output13, LOW);
            }
            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content
=\"width=device-width, initial-scale=1\">");

            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-
size attributes to fit your preferences
            client.println("<style>html { font-family: Helvetica;
display: inline-block; margin: 0px auto; text-align: center;}")
;
            client.println(".button { background-color: #4CAF50;
border: none; color: white; padding: 16px 40px;");
            client.println("text-decoration: none; font-size: 30px;
 margin: 2px; cursor: pointer;}");
            client.println(".button2 {background-color: #555555;}</
style></head>");

            // Web Page Heading
            client.println("<body><h1>ESP32 Web Server</h1>");

            // Display current state, and ON/OFF buttons for GPIO
26
            client.println("<body><h2> Stepper Motor Direction </h2
>");
            client.println("<p>D13_LED = off: Stepper = CCW.");
            client.println("<p>When D13_LED = on: Stepper = CW. </p
>");
            client.println("<p>Default State: D13_LED off, Stepper
= CCW");
            client.println("<body><h3> D13 LED - State: " +
output13State + "</h3>");

            if (output13State == "off")
            {
              client.println("<p><a href=\"/13/on\"><button class
=\"button\">CW</button></a></p>");
            }
            else
            {
              client.println("<p><a href=\"/13/off\"><button class
=\"button button2\">CCW</button></a></p>");
            }

            client.println("<body><h3>TR2591 Photo Sensor Data</h3>
");
            // Get the Visible Light Reading From the Sensor and
update to webpage
            xQueueReceive(xVisibleQueue, &vis_val, 0U);
            client.print("<p>Visible: ");
            client.print(vis_val, DEC);
            client.println(" Lumen(s)</p>");
            // Get the Infrared Reading From the Sensor and update
```

```
    to webpage
448             xQueueReceive(xInfraredQueue, &ir_val, 0U);
449             client.print("<p>Infrared: ");
450             client.print(ir_val, DEC);
451             client.println(" micron(s)");
452             // Get the Full Spectrum Reading and update to webpage
453             xQueueReceive(xFullSpectQueue, &fullSpec_val, 0U);
454             client.print("<p>Full Spectrum: ");
455             client.print(fullSpec_val, DEC);
456             client.println(" Angstrom(s)");
457             // The HTTP response ends with another blank line
458             client.println();
459             // Break out of the while loop
460             break;
461           }
462           else
463           { // if you got a newline, then clear currentLine
464             currentLine = "";
465           }
466         }
467         else if (c != '\r')
468         {                    // if you got anything else but a
    carriage return character,
469           currentLine += c; // add it to the end of the currentLine
470         }
471       }
472     }
473     // Clear the header variable
474     header = "";
475     // Close the connection
476     client.stop();
477     Serial.println("Client disconnected.");
478     Serial.println("");
479   }
480 }
```

1.2 devices.h

```
1  #ifndef DEVICES_H
2  #define DEVICES_H
3  // ESP32 Web Server Libraries
4  #include <WiFi.h>
5  #include <HTTPClient.h>
6  #include <Wire.h>
7  #include "ClosedCube_HDC1080.h"
8  #include <Adafruit_Sensor.h>
9  #include "Adafruit_TSL2591.h"
10
11 // #include <WebServer.h>
12 #include <ESPAsyncWebServer.h>
13 // Device Libraries
14 #include <Arduino.h>
15
16
17 // I2C PINS
18 #define SCL 22
```

```
#define SDA 23


// Stepper PINS
#define STEP_IN1 15
#define STEP_IN2 12
#define STEP_IN3 4
#define STEP_IN4 5

//Assign output variabless to GPIO pins
#define output13  13

// ESP32 Core Assignment
// webserver core
static int core_zero = 0;
// RTOS core
static int core_one = 1;

// Setup Functions
void setup_HDC1080();
void setup_stepper();
void d13_setup();
// void setup_buttons();
void setup_light_sensor();

// Device Functions
void stepper_move(int step);
void step_dir(int direction);
void displaySensorDetails(void *parameters);




#endif
```

1.3 devices.cpp

```
#include "devices.h"


/**
 * @brief Set the up stepper motor pins
 *
 */
void setup_stepper()
{
   pinMode(STEP_IN1, OUTPUT);
   pinMode(STEP_IN2, OUTPUT);
   pinMode(STEP_IN3, OUTPUT);
   pinMode(STEP_IN4, OUTPUT);
}


/**
 * @brief Moves the stepper motor
 *
```

```
 * @param step picks a state. If a state does not exist defaults to
     reset/initial state
 */
void stepper_move(int step)
{
   switch(step)
   {
      case 1:
         digitalWrite(STEP_IN4, 1);
         digitalWrite(STEP_IN3, 0);
         digitalWrite(STEP_IN2, 0);
         digitalWrite(STEP_IN1, 0);
         break;

      case 2:
         digitalWrite(STEP_IN4, 1);
         digitalWrite(STEP_IN3, 1);
         digitalWrite(STEP_IN2, 0);
         digitalWrite(STEP_IN1, 0);
         break;

      case 3:
         digitalWrite(STEP_IN4, 0);
         digitalWrite(STEP_IN3, 1);
         digitalWrite(STEP_IN2, 0);
         digitalWrite(STEP_IN1, 0);
         break;

      case 4:
         digitalWrite(STEP_IN4, 0);
         digitalWrite(STEP_IN3, 1);
         digitalWrite(STEP_IN2, 1);
         digitalWrite(STEP_IN1, 0);
         break;

      case 5:
         digitalWrite(STEP_IN4, 0);
         digitalWrite(STEP_IN3, 0);
         digitalWrite(STEP_IN2, 1);
         digitalWrite(STEP_IN1, 0);
         break;

      case 6:
         digitalWrite(STEP_IN4, 0);
         digitalWrite(STEP_IN3, 0);
         digitalWrite(STEP_IN2, 1);
         digitalWrite(STEP_IN1, 1);
         break;

      case 7:
         digitalWrite(STEP_IN4, 0);
         digitalWrite(STEP_IN3, 0);
         digitalWrite(STEP_IN2, 0);
         digitalWrite(STEP_IN1, 1);
         break;

      case 8:
```

```
            digitalWrite(STEP_IN4, 1);
            digitalWrite(STEP_IN3, 0);
            digitalWrite(STEP_IN2, 0);
            digitalWrite(STEP_IN1, 1);
            break;

        default:
            digitalWrite(STEP_IN4, 1);
            digitalWrite(STEP_IN3, 0);
            digitalWrite(STEP_IN2, 0);
            digitalWrite(STEP_IN1, 1);
            break;
    }
}

/**
 * @brief This function will get the true for false flag from a button toggle on the ESP32 local webserver.
 * If true the stepper motor will rotate in a clockwise direction. Default is Counter Clockwise
 * @param dir
 *
 */
void step_dir(int direction)
{
    //CW
    if (direction == 1)
    {
        for (int i = 9; i > 1; i--)
        {
            stepper_move(i);
            vTaskDelay(10 / portTICK_PERIOD_MS);
        }
    }
    // CCW
    else if (direction == 0)
    {
        for (int i = 1; i < 9; i++)
        {
            stepper_move(i);
            vTaskDelay(10 / portTICK_PERIOD_MS);
        }
    }
}

/**
 * @brief test for web integration and device control
 *
 */
void d13_setup()
{
    pinMode(output13, OUTPUT);
   // pinMode(output27, OUTPUT);

    digitalWrite(output13, LOW);
    // digitalWrite(output27, LOW);
}
```