CSEE5590-0005/490-0005

IOT / Robot Programming

LAB ASSIGNMENT #1

Lesson Title: Smart IOT Monitoring Station using Adafruit Visualization

Lesson Description: Adafruit Visualization of the IOT sensors for professional Looks

Team 1: Ahmed, Cameron, Marajan, Syed Jawad

Contributions:

Ahmed: Coding and Visualization.

Cameron: Coding and Adafruit Visualization setup.

MaraJan: Board and sensor setup and visualization, Youtube videos.

Jawad: Board setup, visualizations, coding and Report creation.

Lab Report:

In this lab assignment the task was to make a weather station using IOT tools, like board, sensors, Arduino etc. The job of the sensors was to get the respective reading and send this to a visualization platform. Further there should be color indicators to distinguish between safe and unsafe readings and warning message based on certain conditions should also be displayed. Finally, there should be an on and off switch to control the circuit.

The sensors use in this lab assignment are pulse sensor, Temperature sensor, Light sensor, UV Sensor, Barometer sensor and the dust sensor. C++ code was used to control the board. There was a Wi-Fi module attached to the board to send the readings to the online visualization platform, Adafruit. Led lights were used as color indicator and they turn green for safe range and red for unsafe reading. Threshold values for set for different sensors to indicate safe and unsafe region.

The visualization platform used for this lab assignment is Adafruit. It is an open source platform which allows people to customize data visualization. Visualization interfaces were made for all the sensors on Adafruit visualization platform. All the data collected from sensors at the board were sent to their respective visualization interface at Adafruit visualization platform via WIFI module that was also attached to the board.

There were some challenges that were also faced in completing this lab. Initially, the visualization platform WIFI module was throwing some errors and it took a while to fix that issue. Also sending the data from each sensor one at a time was achieved quickly and accurately but sending all the data together to Adafruit was a challenge but that was accomplished as well.

The YouTube link for the assignment is:

For Arduino:

<https://youtu.be/bnB7ZJZPKVs>

For Visualization:

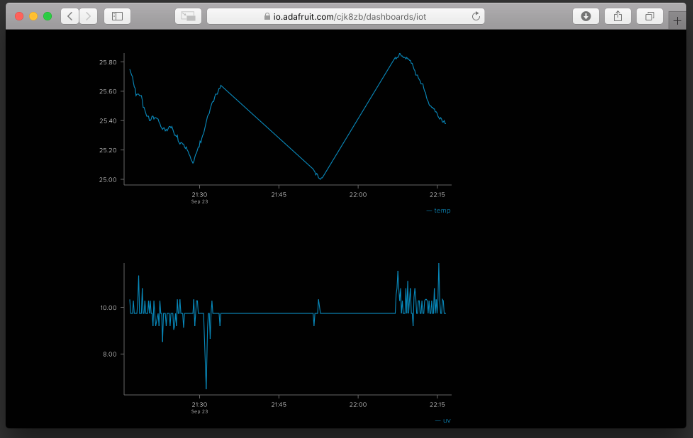
<https://youtu.be/gZCWY-B-V7w>

For Code:

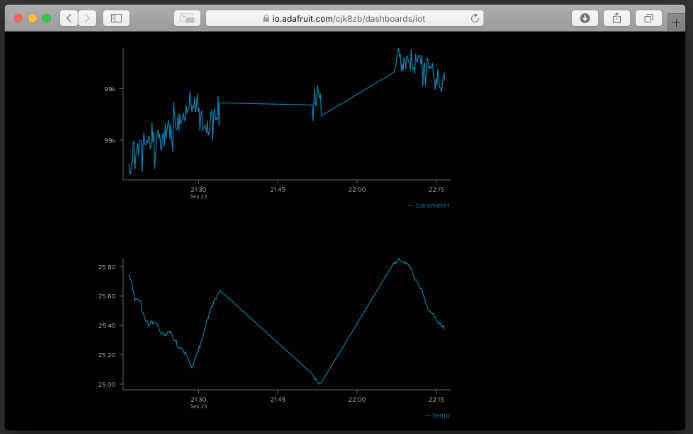
<https://youtu.be/o-RWdlD-z20>

The Screen shots for visualization are as follow:

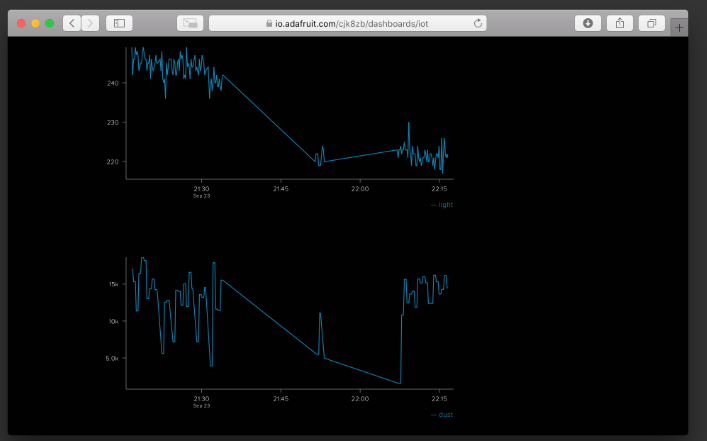
**Temperature and UV:**



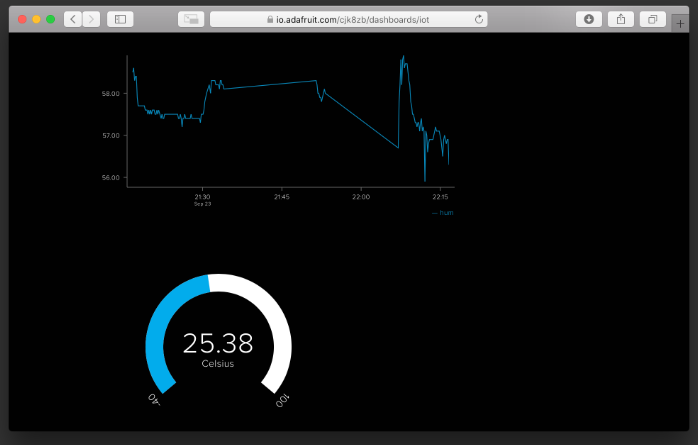
**Barometer and Temperature:**



**Light and Dust:**

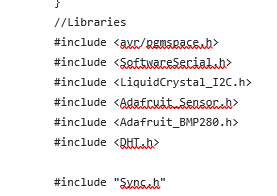


**Humidity and Temp:**

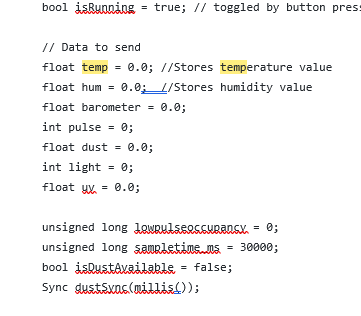


**Code Snippet:**

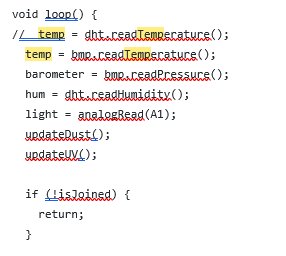
**Libraries:**

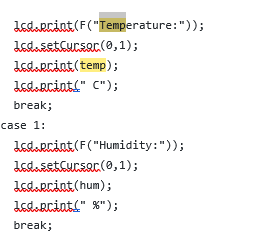


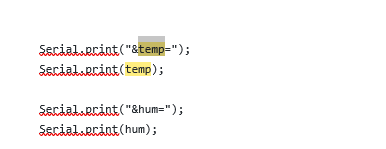
**Temperature, light and UV reading:**

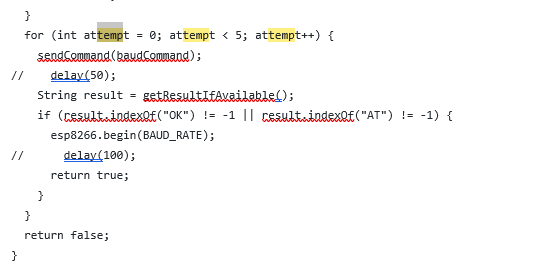






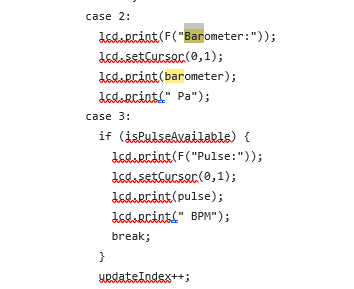


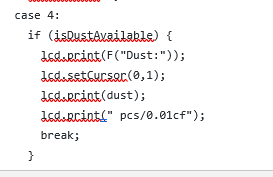


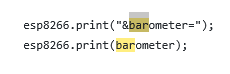


**Barometer and Dust readings:**

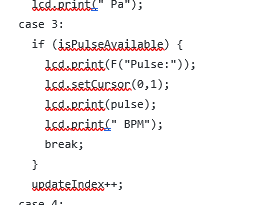


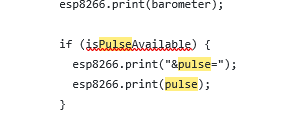


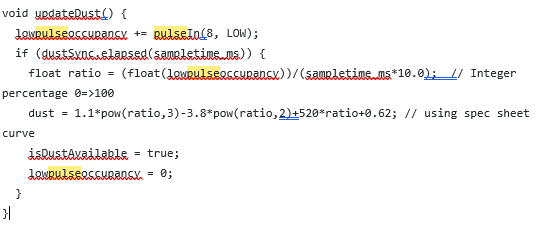




**Pulse Reading:**







**Code:**

|  |
| --- |
| #include "Sync.h" |
|  | #include <Arduino.h> |
|  |  |
|  | Sync::Sync() { |
|  | reset(); |
|  | } |
|  |  |
|  | Sync::Sync(unsigned long start) { |
|  | previous = start; |
|  | } |
|  |  |
|  | void Sync::reset() { |
|  | previous = 0; |
|  | } |
|  |  |
|  | bool Sync::elapsed(unsigned long timeout) { |
|  | unsigned long now = millis(); |
|  | return now - previous >= timeout ? (previous = now) : false; |
|  | } |
|  | //Libraries |
|  | #include <avr/pgmspace.h> |
|  | #include <SoftwareSerial.h> |
|  | #include <LiquidCrystal\_I2C.h> |
|  | #include <Adafruit\_Sensor.h> |
|  | #include <Adafruit\_BMP280.h> |
|  | #include <DHT.h> |
|  |  |
|  | #include "Sync.h" |
|  |  |
|  | //Constants |
|  | #define ssid "Wireless Network 2.4 GHz" |
|  | #define pass ".lozer12" |
|  | #define aioKey "041da268241949669fd13faa7c832296" |
|  |  |
|  | #define BUFFER\_SIZE 50 |
|  |  |
|  | // AT Commands -- store in flash |
|  | const char atCommand[] PROGMEM = "AT"; |
|  | const char echoOffCommand[] PROGMEM = "ATE0"; |
|  | const char baudCommand[] PROGMEM = "AT+UART\_CUR=9600,8,1,0,0"; |
|  | const char modeCommand[] PROGMEM = "AT+CWMODE=1"; |
|  | const char joinCommand[] PROGMEM = "AT+CWJAP=\"" ssid "\",\"" pass "\""; |
|  | const char startCommand[] PROGMEM = "AT+CIPSTART=\"TCP\",\"io.adafruit.com\",80"; |
|  | const char closeCommand[] PROGMEM = "AT+CIPCLOSE"; |
|  | const char sendDataCommand[] PROGMEM = "AT+CIPSENDEX=2048"; |
|  |  |
|  |  |
|  | // Variables |
|  | #define USE\_SOFTWARE\_SERIAL 1 |
|  |  |
|  | #if USE\_SOFTWARE\_SERIAL |
|  | SoftwareSerial esp8266(9,10); |
|  | #define BAUD\_RATE 9600 |
|  | #else |
|  | #define esp8266 Serial |
|  | #define BAUD\_RATE 115200 |
|  | #endif |
|  |  |
|  | LiquidCrystal\_I2C lcd(0x27,16,2); |
|  | Adafruit\_BMP280 bmp; // I2C |
|  | DHT dht(7, DHT22); //// Initialize DHT sensor for normal 16mhz Arduino |
|  |  |
|  | Sync updateSync; |
|  | Sync displaySync; |
|  | Sync ledSync; |
|  |  |
|  | bool isConnected = false; |
|  | bool isJoined = false; |
|  | bool isRunning = true; // toggled by button press; |
|  |  |
|  | // Data to send |
|  | float temp = 0.0; //Stores temperature value |
|  | float hum = 0.0; //Stores humidity value |
|  | float barometer = 0.0; |
|  | int pulse = 0; |
|  | float dust = 0.0; |
|  | int light = 0; |
|  | float uv = 0.0; |
|  |  |
|  | unsigned long lowpulseoccupancy = 0; |
|  | unsigned long sampletime\_ms = 30000; |
|  | bool isDustAvailable = false; |
|  | Sync dustSync(millis()); |
|  |  |
|  | int uvSampleCount = 0; |
|  | long uvSampleTotal = 0; |
|  | bool isUVAvailable = false; |
|  | Sync uvSync; |
|  |  |
|  | volatile boolean isPulseAvailable = false; // "True" when heartbeat is detected. "False" when not a "live beat". |
|  |  |
|  | int redLED = LOW; |
|  | int greenLED = LOW; |
|  |  |
|  | void setup() { |
|  | lcd.init(); |
|  | lcd.backlight(); |
|  | lcd.clear(); |
|  | lcd.setCursor(0,0); |
|  | lcd.print(F("Team 1")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(F("Assignment 1")); |
|  |  |
|  | // interruptSetup(); |
|  |  |
|  | bmp.begin(); |
|  | pinMode(A1, INPUT); // Light |
|  | pinMode(8, INPUT); // Dust |
|  | pinMode(2, OUTPUT); // Green LED |
|  | pinMode(3, OUTPUT); // Red LED |
|  |  |
|  | #if USE\_SOFTWARE\_SERIAL |
|  | Serial.begin(BAUD\_RATE); |
|  | setupSoftwareSerial(); |
|  | #else |
|  | //interruptSetup(); |
|  | #endif |
|  |  |
|  | esp8266.begin(BAUD\_RATE); |
|  | delay(1000); |
|  | lcd.clear(); |
|  | lcd.setCursor(0,0); |
|  | lcd.print(F("SSID: ")); |
|  | lcd.print(F(ssid)); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(F("Connecting...")); |
|  |  |
|  | sendCommand(echoOffCommand); |
|  | while(!readline().equals("OK")) {} |
|  |  |
|  | sendCommand(modeCommand); |
|  | while(!readline().equals("OK")) {} |
|  |  |
|  | sendCommand(joinCommand); |
|  |  |
|  | while(!isJoined) { |
|  | String result = readline(); |
|  | if (result.equals("OK")) { |
|  | isJoined = true; |
|  | lcd.setCursor(0,1); |
|  | lcd.print(F("Connected ")); |
|  | } else if (result.indexOf("FAIL") != -1 || result.indexOf("ERROR") != -1) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print(F("Not Connected")); |
|  | return; |
|  | } |
|  | } |
|  |  |
|  |  |
|  | } |
|  |  |
|  | void loop() { |
|  | // temp = dht.readTemperature(); |
|  | temp = bmp.readTemperature(); |
|  | barometer = bmp.readPressure(); |
|  | hum = dht.readHumidity(); |
|  | light = analogRead(A1); |
|  | updateDust(); |
|  | updateUV(); |
|  |  |
|  | if (!isJoined) { |
|  | return; |
|  | } |
|  |  |
|  | if (digitalRead(A3)==HIGH) { |
|  | digitalWrite(2, LOW); |
|  | digitalWrite(3, LOW); |
|  | lcd.clear(); |
|  | isRunning = !isRunning; |
|  | lcd.setCursor(0,0); |
|  | lcd.print(isRunning ? "Starting..." : "Stopping..."); |
|  | delay(2000); |
|  | lcd.clear(); |
|  | } |
|  |  |
|  | if (!isRunning) { |
|  | return; |
|  | } |
|  |  |
|  | if (ledSync.elapsed(500)) { |
|  | if (allGood()) { |
|  | greenLED = greenLED == HIGH ? LOW : HIGH; |
|  | redLED = LOW; |
|  | } else { |
|  | redLED = redLED == HIGH ? LOW : HIGH; |
|  | greenLED = LOW; |
|  | } |
|  | digitalWrite(2, greenLED); |
|  | digitalWrite(3, redLED); |
|  | } |
|  |  |
|  | if (updateSync.elapsed(10000)) { |
|  | updateData(); |
|  | } |
|  |  |
|  | if (displaySync.elapsed(1000)) { |
|  | updateLCD(); |
|  | } |
|  | } |
|  |  |
|  | bool allGood() { |
|  | return light >= 60; |
|  | } |
|  |  |
|  | int updateIndex = 0; |
|  |  |
|  | void updateLCD() { |
|  | lcd.clear(); |
|  | lcd.setCursor(0,0); |
|  |  |
|  | switch(updateIndex) { |
|  | case 0: |
|  | lcd.print(F("Temperature:")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(temp); |
|  | lcd.print(" C"); |
|  | break; |
|  | case 1: |
|  | lcd.print(F("Humidity:")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(hum); |
|  | lcd.print(" %"); |
|  | break; |
|  | case 2: |
|  | lcd.print(F("Barometer:")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(barometer); |
|  | lcd.print(" Pa"); |
|  | case 3: |
|  | if (isPulseAvailable) { |
|  | lcd.print(F("Pulse:")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(pulse); |
|  | lcd.print(" BPM"); |
|  | break; |
|  | } |
|  | updateIndex++; |
|  | case 4: |
|  | if (isDustAvailable) { |
|  | lcd.print(F("Dust:")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(dust); |
|  | lcd.print(" pcs/0.01cf"); |
|  | break; |
|  | } |
|  | updateIndex++; |
|  | case 5: |
|  | lcd.print(F("Light:")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(light); |
|  | lcd.print(" lx"); |
|  | break; |
|  | case 6: |
|  | if (isUVAvailable) { |
|  | lcd.print(F("UV Level:")); |
|  | lcd.setCursor(0,1); |
|  | lcd.print(uv); |
|  | break; |
|  | } |
|  | default: |
|  | updateIndex = -1; |
|  | } |
|  | updateIndex++; |
|  | if (updateIndex > 6) { |
|  | updateIndex = 0; |
|  | } |
|  | } |
|  |  |
|  | void updateDust() { |
|  | lowpulseoccupancy += pulseIn(8, LOW); |
|  | if (dustSync.elapsed(sampletime\_ms)) { |
|  | float ratio = (float(lowpulseoccupancy))/(sampletime\_ms\*10.0); // Integer percentage 0=>100 |
|  | dust = 1.1\*pow(ratio,3)-3.8\*pow(ratio,2)+520\*ratio+0.62; // using spec sheet curve |
|  | isDustAvailable = true; |
|  | lowpulseoccupancy = 0; |
|  | } |
|  | } |
|  |  |
|  | void updateUV() { |
|  | int sampleValue = analogRead(A0); |
|  | uvSampleCount++; |
|  | uvSampleTotal += sampleValue; |
|  |  |
|  | if (uvSync.elapsed(2000)) { |
|  | float averageUV = (float)uvSampleTotal / (float)uvSampleCount; |
|  | // Vsig is the value of voltage measured from the SIG pin of the Grove interface |
|  | uv = averageUV \* 4980.0 / 1023.0; |
|  | isUVAvailable = true; |
|  | uvSampleTotal = 0; |
|  | uvSampleCount = 0; |
|  | } |
|  | } |
|  |  |
|  | String readline(char until) { |
|  | char buff[BUFFER\_SIZE] = {0}; |
|  | int n = 0; |
|  |  |
|  | while(1) { |
|  | int in = esp8266.read(); |
|  | if (in >= 0) { |
|  | Serial.write(in); |
|  | if (in == until) { |
|  | buff[n] = in; |
|  | buff[n+1] = 0; |
|  | return String(buff); |
|  | } |
|  | if (in == '\n' && n > 0 && buff[n - 1] == '\r') { |
|  | //\r\n |
|  | if (n > 1) { |
|  | buff[n - 1] = 0; |
|  | return String(buff); |
|  | } |
|  | n = 0; |
|  | continue; |
|  | } |
|  | if (in == '\n') { |
|  | n = 0; |
|  | continue; |
|  | } |
|  | buff[n] = in; n++; |
|  | if (n == BUFFER\_SIZE - 2) { |
|  | buff[n + 1] = 0; |
|  | return String(buff); |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | String readline(void) { |
|  | return readline(0); |
|  | } |
|  |  |
|  | void sendCommand(const char \*msg) { |
|  | esp8266.println((\_\_FlashStringHelper\*)msg); |
|  | #if USE\_SOFTWARE\_SERIAL |
|  | Serial.println((\_\_FlashStringHelper\*)msg); |
|  | #endif |
|  | } |
|  |  |
|  | void updateData() { |
|  | lcd.clear(); |
|  | lcd.setCursor(0,0); |
|  | lcd.print("Sending Data"); |
|  | if (!isConnected) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Connecting"); |
|  | sendCommand(startCommand); |
|  | while(1) { |
|  | String result = readline(); |
|  | if (result.equals("OK") || result.equals("ALREADY CONNECTED")) { |
|  | isConnected = true; |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Connected "); |
|  | break; |
|  | } else if (result.equals("FAIL")|| result.equals("ERROR")) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Not Connected "); |
|  | delay(1000); |
|  | break; |
|  | } |
|  | } |
|  | if (!isConnected) { |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | sendCommand(sendDataCommand); |
|  |  |
|  | bool responded = false; |
|  | while(!responded) { |
|  | String result = readline('>'); |
|  | if (result.equals(">")) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Sending... "); |
|  | responded = true; |
|  | } else if (result.equals("FAIL") || result.equals("ERROR")) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Failed "); |
|  | sendCommand(closeCommand); |
|  | isConnected = false; |
|  | delay(1000); |
|  | return; |
|  | } |
|  | } |
|  |  |
|  | esp8266.print("GET /api/groups/default/send.json?x-aio-key="); |
|  | esp8266.print(aioKey); |
|  |  |
|  | esp8266.print("&temp="); |
|  | esp8266.print(temp); |
|  |  |
|  | esp8266.print("&hum="); |
|  | esp8266.print(hum); |
|  |  |
|  | esp8266.print("&barometer="); |
|  | esp8266.print(barometer); |
|  |  |
|  | if (isPulseAvailable) { |
|  | esp8266.print("&pulse="); |
|  | esp8266.print(pulse); |
|  | } |
|  |  |
|  | if (isDustAvailable) { |
|  | esp8266.print("&dust="); |
|  | esp8266.print(dust); |
|  | } |
|  |  |
|  | esp8266.print("&light="); |
|  | esp8266.print(light); |
|  |  |
|  | if (isUVAvailable) { |
|  | esp8266.print("&uv="); |
|  | esp8266.print(uv); |
|  | } |
|  |  |
|  | esp8266.println(" HTTP/1.0"); |
|  | esp8266.println("Host: io.adafruit.com"); |
|  | esp8266.println(); |
|  | esp8266.print("\\0"); // EOT |
|  |  |
|  | #if USE\_SOFTWARE\_SERIAL |
|  | Serial.print("GET /api/groups/default/send.json?x-aio-key="); |
|  | Serial.print(aioKey); |
|  |  |
|  | Serial.print("&temp="); |
|  | Serial.print(temp); |
|  |  |
|  | Serial.print("&hum="); |
|  | Serial.print(hum); |
|  |  |
|  | Serial.print("&barometer="); |
|  | Serial.print(barometer); |
|  |  |
|  | if (isPulseAvailable) { |
|  | Serial.print("&pulse="); |
|  | Serial.print(pulse); |
|  | } |
|  |  |
|  | if (isDustAvailable) { |
|  | Serial.print("&dust="); |
|  | Serial.print(dust); |
|  | } |
|  |  |
|  | Serial.print("&light="); |
|  | Serial.print(light); |
|  |  |
|  | if (isUVAvailable) { |
|  | Serial.print("&uv="); |
|  | Serial.print(uv); |
|  | } |
|  |  |
|  | Serial.println(" HTTP/1.0"); |
|  | Serial.println("Host: io.adafruit.com"); |
|  | Serial.println(); |
|  | Serial.print("\\0"); // EOT |
|  | #endif |
|  |  |
|  | responded = false; |
|  | while(!responded) { |
|  | String result = readline(); |
|  | if (result.equals("SEND OK")) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Receiving... "); |
|  | responded = true; |
|  | } else if (result.equals("SEND FAIL") || result.equals("ERROR")) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Error "); |
|  | responded = true; |
|  | sendCommand(closeCommand); |
|  | isConnected = false; |
|  | delay(1000); |
|  | } |
|  | } |
|  |  |
|  | bool closed = false; |
|  | while(!closed) { |
|  | String result = readline(); |
|  | if (result.equals("CLOSED")) { |
|  | lcd.setCursor(0,1); |
|  | lcd.print("Done "); |
|  | closed = true; |
|  | } |
|  | } |
|  | delay(500); |
|  | isConnected = false; |
|  | } |
|  |  |
|  |  |
|  | String getResultIfAvailable() { |
|  | while(esp8266.available()) { |
|  | String result = esp8266.readStringUntil('\n'); |
|  | result.trim(); |
|  | if (result[0] != 0) { |
|  | return result; |
|  | } |
|  | } |
|  | return ""; |
|  | } |
|  |  |
|  | String waitForResult() { |
|  | // Wait for data |
|  | int \_startMillis = millis(); |
|  | while(esp8266.available() == 0 && millis() - \_startMillis < 60000); |
|  | return getResultIfAvailable(); |
|  | } |
|  |  |
|  | bool setupSoftwareSerial() { |
|  | unsigned long current = 0; |
|  | for (int i = 0; i < 2; i++) { |
|  | for (int attempt = 0; attempt < 5; attempt++) { |
|  | unsigned long baud = i ? 115200 : 9600; |
|  | esp8266.begin(baud); |
|  | sendCommand(atCommand); |
|  | delay(1000); |
|  | if (getResultIfAvailable().indexOf("OK") != -1) { |
|  | current = baud; |
|  | break; |
|  | } |
|  | } |
|  | if (current) { break; } |
|  | } |
|  |  |
|  | Serial.print(F("Current baud is ")); |
|  | Serial.println(current); |
|  | esp8266.setTimeout(5000); |
|  | if (current == 0) { |
|  | return false; |
|  | } |
|  | if (current == BAUD\_RATE) { |
|  | return true; |
|  | } |
|  | for (int attempt = 0; attempt < 5; attempt++) { |
|  | sendCommand(baudCommand); |
|  | // delay(50); |
|  | String result = getResultIfAvailable(); |
|  | if (result.indexOf("OK") != -1 || result.indexOf("AT") != -1) { |
|  | esp8266.begin(BAUD\_RATE); |
|  | // delay(100); |
|  | return true; |
|  | } |
|  | } |
|  | return false; |
|  | } |
|  |  |
|  | void interruptSetup(){ |
|  | TCCR2A = 0x02; // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO INTO CTC MODE |
|  | TCCR2B = 0x06; // DON'T FORCE COMPARE, 256 PRESCALER |
|  | OCR2A = 0X7C; // SET THE TOP OF THE COUNT TO 124 FOR 500Hz SAMPLE RATE |
|  | TIMSK2 = 0x02; // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2 AND OCR2A |
|  | sei(); // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED |
|  | } |
|  |  |
|  | volatile unsigned long sampleCounter = 0; // used to determine pulse timing |
|  | volatile unsigned long lastBeatTime = 0; // used to find IBI |
|  | volatile int thresh = 525; // used to find instant moment of heart beat, seeded |
|  | volatile int IBI = 600; // int that holds the time interval between beats! Must be seeded! |
|  | volatile int P =512; // used to find peak in pulse wave, seeded |
|  | volatile int T = 512; // used to find trough in pulse wave, seeded |
|  | volatile boolean firstBeat = true; // used to seed rate array so we startup with reasonable BPM |
|  | volatile boolean secondBeat = false; // used to seed rate array so we startup with reasonable BPM |
|  | volatile int rate[10]; // array to hold last ten IBI values |
|  | volatile boolean QS = false; // becomes true when Arduino finds a beat. |
|  | volatile int amp = 100; // used to hold amplitude of pulse waveform, seeded |
|  |  |
|  | ISR(TIMER2\_COMPA\_vect){ // triggered when Timer2 counts to 124 |
|  | cli(); // disable interrupts while we do this |
|  | int Signal = analogRead(A2); // read the Pulse Sensor |
|  | sampleCounter += 2; // keep track of the time in mS |
|  | int N = sampleCounter - lastBeatTime; // monitor the time since the last beat to avoid noise |
|  |  |
|  | // find the peak and trough of the pulse wave |
|  | if(Signal < thresh && N > (IBI/5)\*3){ // avoid dichrotic noise by waiting 3/5 of last IBI |
|  | if (Signal < T){ // T is the trough |
|  | T = Signal; // keep track of lowest point in pulse wave |
|  | } |
|  | } |
|  |  |
|  | if(Signal > thresh && Signal > P){ // thresh condition helps avoid noise |
|  | P = Signal; // P is the peak |
|  | } // keep track of highest point in pulse wave |
|  |  |
|  | // NOW IT'S TIME TO LOOK FOR THE HEART BEAT |
|  | // signal surges up in value every time there is a pulse |
|  | if (N > 250){ // avoid high frequency noise |
|  | if ( (Signal > thresh) && (isPulseAvailable == false) && (N > (IBI/5)\*3) ){ |
|  | isPulseAvailable = true; // set the Pulse flag when there is a pulse |
|  | digitalWrite(13,HIGH); // turn on pin 13 LED |
|  | IBI = sampleCounter - lastBeatTime; // time between beats in mS |
|  | lastBeatTime = sampleCounter; // keep track of time for next pulse |
|  |  |
|  | if(secondBeat){ // if this is the second beat |
|  | secondBeat = false; // clear secondBeat flag |
|  | for(int i=0; i<=9; i++){ // seed the running total to get a realistic BPM at startup |
|  | rate[i] = IBI; |
|  | } |
|  | } |
|  |  |
|  | if(firstBeat){ // if it's the first time beat is found |
|  | firstBeat = false; // clear firstBeat flag |
|  | secondBeat = true; // set the second beat flag |
|  | sei(); // enable interrupts again |
|  | return; // IBI value is unreliable so discard it |
|  | } |
|  | word runningTotal = 0; // clear the runningTotal variable |
|  |  |
|  | for(int i=0; i<=8; i++){ // shift data in the rate array |
|  | rate[i] = rate[i+1]; // and drop the oldest IBI value |
|  | runningTotal += rate[i]; // add up the 9 oldest IBI values |
|  | } |
|  |  |
|  | rate[9] = IBI; // add the latest IBI to the rate array |
|  | runningTotal += rate[9]; // add the latest IBI to runningTotal |
|  | runningTotal /= 10; // average the last 10 IBI values |
|  | pulse = 60000/runningTotal; // how many beats can fit into a minute? that's BPM! |
|  | QS = true; // set Quantified Self flag |
|  | // QS FLAG IS NOT CLEARED INSIDE THIS ISR |
|  | } |
|  | } |
|  |  |
|  | if (Signal < thresh && isPulseAvailable == true){ // when the values are going down, the beat is over |
|  | digitalWrite(13,LOW); // turn off pin 13 LED |
|  | isPulseAvailable = false; // reset the Pulse flag so we can do it again |
|  | amp = P - T; // get amplitude of the pulse wave |
|  | thresh = amp/2 + T; // set thresh at 50% of the amplitude |
|  | P = thresh; // reset these for next time |
|  | T = thresh; |
|  | } |
|  |  |
|  | if (N > 2500){ // if 2.5 seconds go by without a beat |
|  | thresh = 512; // set thresh default |
|  | P = 512; // set P default |
|  | T = 512; // set T default |
|  | lastBeatTime = sampleCounter; // bring the lastBeatTime up to date |
|  | firstBeat = true; // set these to avoid noise |
|  | secondBeat = false; // when we get the heartbeat back |
|  | } |
|  |  |
|  | sei(); |
|  | // enable interrupts when youre done! |
|  | }// end isr |