**Wireless Weight Measurement System**

**EE537 Final Report**

**Fall 2023**

**By**

**Team 1 (T.E.D.)**

**Blake Wingard** ([cwingard@uab.edu](mailto:cwingard@uab.edu))

**Gaurav Lunagariya** ([glunagar@uab.edu](mailto:glunagar@uab.edu))

**Joe Wong** ([jawong1@uab.edu](mailto:jawong1@uab.edu))

**Abstract:**

This report describes the details of the *Wireless Weight Measurement System*. It will go over the software implementation for both Bluetooth and Cloud on an ESP32 board, and it will go over the hardware interface with the HX711 amplifier module. This report will also go over the design process for this project.

Table of Contents

[Section A – Project Introduction 3](#_Toc152078463)

[1. Project Description 3](#_Toc152078464)

[2. Constraint Analysis 3](#_Toc152078465)

[3. Hardware Interface 3](#_Toc152078466)

[4. Software Interface 3](#_Toc152078467)

[5. Power Constraints and Efficiency 3](#_Toc152078468)

[6. Cost Constraints 4](#_Toc152078469)

[7. Component Section Rationale 4](#_Toc152078470)

[Section B – Detail Project Description 5](#_Toc152078471)

[1. Significant System Components 5](#_Toc152078472)

[2. Project Description 5](#_Toc152078473)

[3. Results 6](#_Toc152078474)

[4. Future Improvements 6](#_Toc152078475)

[5. Team Organization and Labor Distribution 6](#_Toc152078476)

[Summery 6](#_Toc152078477)

[Details 6](#_Toc152078478)

[Section C – Additional Documents and Datasheets 8](#_Toc152078479)

[a. Code 8](#_Toc152078480)

[b. circuit schematic 26](#_Toc152078481)

[c. List of libraries 26](#_Toc152078482)

# Section A – Project Introduction

## 1. Project Description

Design, develop and fabricate basic prototype of a weight measurement  
device that it has wireless communication capability. The communication between scale and  
remote device (smartphone or tablet) should accomplish by using a private WIFI/Bluetooth  
network.

## 2. Constraint Analysis

### Minimum requirements:

1. Device should have simple display to show weight in Kg or LB
2. Device should have a simple, user-friendly auto calibration
3. Device should have option for automatic Tare cancellation
4. Device should have wireless communication for a laptop, tablet or for a smartphone; iPhone iPad is preferred.
5. Device should have a simple GUI for easy use and save measurements or send the measurements over “messages”, or with an email client.
6. Program code must have one interrupt routine.
7. For programming language/library, will be a native programming language Python, sketch/C# can be used.

### Constraints:

1. Overall cost should not exceed $250.
2. Arduino and its clones should be used.

## 3. Hardware Interface

The controller for the scale is the ESP32 which has internals for Bluetooth and Wi-Fi communication. The weight is communicated with the ESP32 through serial communication with an HX711 amplifier. An LCD display is controlled via I2C bus on the ESP32. There are three physical buttons used for taring the device, switching units, and waking the device from sleep if needed.

## 

## 4. Software Interface

The controller for the scale is the ESP32 which has internals for Bluetooth and Wi-Fi communication. Bluetooth LE is used for communications with the Android app. Wi-Fi is used for communicating with a ThingSpeak Cloud database. The Android app can set the units of the scale for user customization.

## 5. Power Constraints and Efficiency

There were no power constraints. The scale is being powered via ESP32. The software does contain a sleep functionality the turns off the LCD display to conserve some power.

## 6. Cost Constraints

The project utilizes only components provided by the professor. No additional expenses were necessary beyond cut-to-length wires. The following list is an estimate of the cost for one system:

Table 1: Cost Analysis

|  |  |  |  |
| --- | --- | --- | --- |
| Item | Cost/Item | Quantity | Cost/system |
| ESP32 | $ 9.99 | 1 | $ 9.99 |
| HX711 | $ 10.95 | 1 | $ 10.95 |
| Load Cell | $ 18.22 | 4 | $ 72.88 |
| LCD | $ 12.99 | 1 | $ 12.99 |
| 3.3k resistor | $ 0.06 | 3 | $ 0.18 |
| Switch | $ 0.45 | 3 | $ 1.35 |
| scale frame | $ 50.00 | 1 | $ 50.00 |
| Total Cost |  |  | $ 158.34 |

## 7. Component Section Rationale

### Component 1: ESP32

The ESP32 was chosen as the main controller due to its built-in Bluetooth LE and Wi-Fi capabilities.

### Component 2: HX711

The HX711 was chosen because it provides an easier interface with strain gauges.

### Component 3: 20×4 2004 LCD Module with I2C Serial Interface

This LCS module was chosen due to the I2C interface. The I2C interface significantly reduces the number of pins needed to drive the module.

# Section B – Detail Project Description

## 1. Significant System Components

### Component 1: ESP32

Main controller for the scale. Provides Bluetooth and Wi-Fi internally for use in the project. Handles the reading of the HX711 load cell amplifier and displays it to the LCD module. Also provides error messages (if any) to the LCD module.

### Component 2: HX711

The load cell amplifier for reading strain sensors. The amplifier reports the data as serial data to the ESP32.

### Component 3: 20×4 2004 LCD Module with I2C Serial Interface

The LCD module for displaying the weight, and any error messages. Utilizes a simple I2C bus to drive the LCD saving pins on the ESP32.

## 2. Project Description

The main purpose of this project was to design a wireless scale. The scale must be capable of supporting 500 lbs (227 kg). The scale must provide the data to both an Android app and ThingSpeak cloud database. The following design steps are listed below:

1. Test LCD communication between the ESP32 and LCD module.
2. Test communication between the ESP32 and HX711 amplifier.
3. Test Bluetooth LE communication between ESP32 and random BLE terminal app
4. Develop prototype Android app using MIT app inventor.
5. Test BLE communication with app to ESP32.
6. Test BLE communication with ESP32 to app.
7. Set up a button for changing the units.
8. Test sending different weight measurements to the app.
9. Build scale.
10. Polish app.
11. Calibrate the HX711 sensor
12. Test reading users weight and reporting it to app.
13. Test using the app to set the unit.
14. Set up a button for taring the scale.
15. Set up a timer interrupt for taring the scale if the user does not stay on scale long enough.
16. Set up a timer interrupt to put the device to sleep.
17. Set up a button to wake the device.
18. Set a threshold parameter to read from the HX711. If it is passed, wake the device.
19. Set up ThingSpeak channel for weight saving.
20. Set up Wi-Fi on the ESP32.
21. Test Sending weight to ThingSpeak database.
22. Develop calibration code for obtaining calibration information. This is a separate file from the main.

## 3. Results

The project met nearly all the design requirements. Some exceptions include the IOS app. This was due to being unable to develop for IOS without a Mac. Another exception is the weight does not get saved. This was due to a later requirement change to place it on the cloud. Any weight saving can be handled via ThingSpeak.

The scale is capable of being calibrated easily by the user with simple instructions loaded directly onto the device. The scale is capable of reading and displaying the user’s weight. The scale can communicate with an Android app for displaying the weight and changing the units. The scale is also capable of communicating the weight with ThingSpeak for cloud storage potential.

## 4. Future Improvements

An IOS app could be developed for the scale in addition to the Android app. This app would require an iPhone/iPad for testing and a Mac for development.

The Android app could include a history functionality for saving the user’s weight. This idea was scrapped due to the addition of cloud communication.

The sleep functionality could be modified to put the device to sleep properly (ESP32 in low power mode). This couldn’t be done since the wake functionality depends on reading the HX711. This improvement could potentially be implemented with a button that becomes depressed after some load on the scale.

## 5. Team Organization and Labor Distribution

### Summery

* Blake Wingard
  + App Developer
  + Arduino Developer
  + Project Manager
* Gaurav Lunagariya
  + Hardware Designer
  + Cloud Researcher
  + Schematic Designer
* Joe Wong
  + Arduino Developer
  + Hardware Designer
  + Technical Writer

### Details

While each member contributed in some form to each step, this report will highlight the individual’s major contribution towards the project.

#### Blake Wingard

Blake Wingard contributed mainly to the Android app development. Adding features such as user profiles, simple Bluetooth connection, and customization of the app itself. Blake contributed to developing the scale’s hardware and software either directly or by guidance. Blake was responsible for designing the communication protocol between devices. Blake took on the role of managing the project to assist everyone in staying on track.

#### Gaurav Lunagariya

Gaurav Lunagariya contributed heavily to the hardware design. Gaurav was responsible for a lot of the initial implementations on the ESP32 for Bluetooth and Wi-Fi communication. Gaurav was a phenomenal asset in getting the cloud service up and running for the project.

#### Joe Wong

Joe Wong focused on the Arduino/ESP32 development side. Joe was able to take the individual test pieces and weave them together into a cohesive system. Joe was responsible for putting together the elegant calibration package to aid in setting up a scale. Joe also cleaned up the final breadboard layout using cut-to-length wires to achieve as pristine a finish as possible for a prototype.

# Section C – Additional Documents and Datasheets

## a. Code

### ESPSoftware.ino:

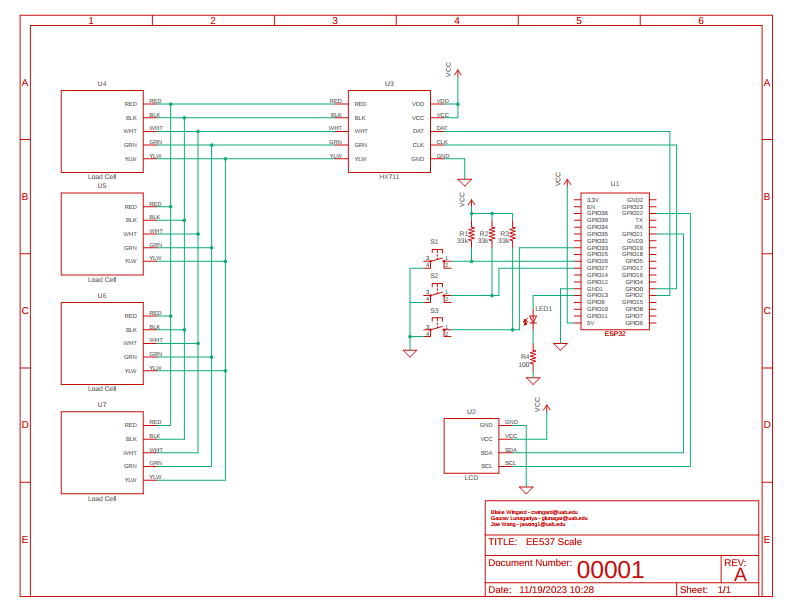
1. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
2. *\* File: ESPSoftware.ino \**
3. *\* Group: The Embedded Dragon (T.E.D.) \**
4. *\* Members: Blake Wingard - cwingard@uab.edu \**
5. ***\* Gaurav Lunagariya - glunagar@uab.edu \****
6. *\* Joe Wong - jawong1@uab.edu \**
7. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
8. *\* Desc: The purpose of this file is to drive the \**
9. *\* scale's functionality. There are a few items \**
10. ***\* that need to be set before the code will \****
11. *\* function properly. \**
12. *\* 1: Set WiFi config (username/password) \**
13. *\* 2: Set ThinkSpeak (Api/channel) \**
14. *\* 3: Set calibration factor for the scale. \**
15. ***\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\****
16. *\* Note: If you do not have a calibration factor, \**
17. *\* please load ESPCalibration.ino first. \**
18. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
20. **//=============================================================================**
21. // SETUP (DO THIS BEFORE LOADING)
22. // Scale settings
23. #define calibration\_factor\_custom <factor>
25. **// WiFi settings**
26. #define NETWORK\_NAME "<WiFi-Name>"
27. #define NETWORK\_PASS "<WiFi-Password>"
29. // ThingSpeak settings
30. **#define API\_KEY "<API-KEY>"**
31. #define CHANNEL\_ID <channel-ID>
33. //=============================================================================
34. // INCLUDES & DEFINES:
35. **// General includes:**
36. #include "HX711.h"
37. #include <Wire.h>
38. #include <LiquidCrystal\_I2C.h>
39. #include <BLEDevice.h>
40. **#include <BLEServer.h>**
41. #include <BLEUtils.h>
42. #include <BLE2902.h>
43. #include <ThingSpeak.h>
44. #include <WiFi.h>
46. // LCD\_I2C pin defines:
47. #define SDA\_PIN SDA
48. #define SDA\_PORT PORTC
49. #define SCL\_PIN SCL
50. **#define SCL\_PORT PORTC**
52. // Load Cell pin defines:
53. #define LOADCELL\_DOUT\_PIN 2
54. #define LOADCELL\_SCK\_PIN 0
56. // Interrupt pin defines:
57. #define BUTTON\_PIN\_1 26
58. #define BUTTON\_PIN\_2 27
59. #define BUTTON\_PIN\_3 33
60. **#define LED\_PIN 13**
62. // BLE defines - UUID Generator: https://www.uuidgenerator.net/
63. #define SERVICE\_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
64. #define CHARACTERISTIC\_UUID\_1 "1c95d5e3-d8f7-413a-bf3d-7a2e5d7be87e"
66. // Serial port settings:
67. #define BAUD\_RATE 115200
69. // Timer settings
70. **#define TIME\_CONDITION 15000 // 15000ms == 15s**
71. #define WAIT\_FOR\_WEIGHING 5000 // 5000ms == 5s
72. #define WAIT\_FOR\_IDLE 2000000 // 2000000us == 2s
73. #define WAIT\_FOR\_DEEP\_SLEEP 2000000 // 2000000us == 2s
74. #define LCD\_PRINT\_DELAY 1000 // 1000ms == 1s
75. **#define CONNECTION\_DELAY 60000 // 60000ms == 60s**
77. // Scale settings
78. #define WEIGHT\_THRESHOLD 2
80. **// DEBUG INFO**
81. #define SERIAL\_DEBUG // Serial debug information
82. // #define LCD\_DEBUG // LCD debug information
83. #define WIFI\_DEBUG
84. #define BUTTON1\_DEBUG // Button1 enabled
85. **#define BUTTON2\_DEBUG // Button2 enabled**
86. #define BUTTON3\_DEBUG // Button3 enabled
87. //=============================================================================
88. // TYPEDEFS
89. typedef enum unitType\_t {
90. **imperial,**
91. metric
92. } unitType\_t;
94. typedef enum state\_t {
95. **idle,**
96. active,
97. done,
98. asleep,
99. calibrating,
100. **deepSleep**
101. } state\_t;
102. //=============================================================================
103. // FUNCTION PROTOTYPES
104. void setUnits(unitType\_t unitType);
105. **void IRAM\_ATTR onIdle(void);**
106. void IRAM\_ATTR onSleep(void);
107. //=============================================================================
108. // VARIABLES AND OBJECTS SETUP:
109. // Interrupt set up:
110. **#ifdef BUTTON1\_DEBUG**
111. volatile bool button1Pressed = false; // Flag to indicate button 1 press
112. #endif
113. #ifdef BUTTON2\_DEBUG
114. volatile bool button2Pressed = false; // Flag to indicate button 2 press
115. **#endif**
116. #ifdef BUTTON3\_DEBUG
117. volatile bool button3Pressed = false; // Flag to indicate button 3 press
118. #endif
120. **// Timer interrupts**
121. hw\_timer\_t\* idleTimer = NULL;
122. hw\_timer\_t\* deepSleepTimer = NULL;
124. // State variables
125. **#ifdef BUTTON1\_DEBUG**
126. bool button1State = false; // State associated with button 1 press
127. #endif
128. unitType\_t unitType = imperial;
129. state\_t state = idle;
131. // Non-blocking delay setup:
132. unsigned long timeSinceLastPrint = millis();
133. unsigned long timeSinceLastStateChange = millis();
134. unsigned long timeSinceLastConnection = millis();
136. // Load cell sensor
137. HX711 scale;
139. // LCD
140. **// lcd(addr,en,rw,rs,d4,d5,d6,d7,bl, blpol)**
141. LiquidCrystal\_I2C lcd(0x3F, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
143. // BLE - pointers setup:
144. BLEServer\* server = NULL;
145. **BLECharacteristic\* characteristic = NULL;**
147. // BLE - callback for when a client connects or disconnects
148. class MyServerCallbacks : public BLEServerCallbacks {
149. void onConnect(BLEServer\* server) {
150. **// Determine which units to use from the phone**
151. std::string rxValue = characteristic->getValue();
153. if (rxValue[0] == 'K') {
154. unitType = metric;
155. **} else if (rxValue[0] == 'L') {**
156. unitType = imperial;
157. }
159. setUnits(unitType);
161. #ifdef SERIAL\_DEBUG
162. Serial.print("Message from the phone (getValue): ");
163. Serial.println(rxValue.c\_str());
164. #endif
165. **}**
167. void onDisconnect(BLEServer\* server) {
168. // give the bluetooth stack the chance to get things ready
169. delay(500);
170. **server->startAdvertising();**
171. #ifdef SERIAL\_DEBUG
172. Serial.println("start advertising");
173. #endif
174. }
175. **};**
177. // Cloud
178. WiFiClient client;
180. **//=============================================================================**
181. // SETUP FUNCTION:
182. void setup() {
184. #ifdef SERIAL\_DEBUG
185. **Serial.begin(BAUD\_RATE);**
186. #endif
188. // BLE - Setup:
189. BLEDevice::init("T.E.D.");
191. server = BLEDevice::createServer();
192. server->setCallbacks(new MyServerCallbacks());
194. BLEService\* Service = server->createService(SERVICE\_UUID);
196. characteristic = Service->createCharacteristic( CHARACTERISTIC\_UUID\_1,
197. BLECharacteristic::PROPERTY\_READ |
198. BLECharacteristic::PROPERTY\_WRITE |
199. BLECharacteristic::PROPERTY\_NOTIFY);
201. Service->start();
203. // BLE - Start advertising server
204. BLEAdvertising\* Advertising = BLEDevice::getAdvertising();
205. **Advertising->addServiceUUID(SERVICE\_UUID);**
206. Advertising->setScanResponse(false);
207. Advertising->setMinPreferred(0x0); // set value to 0x00 to not advertise this parameter
208. BLEDevice::startAdvertising();
209. #ifdef SERIAL\_DEBUG
210. **Serial.println("Waiting a client connection to notify...");**
211. #endif
213. // Scale Set Up:
214. scale.begin(LOADCELL\_DOUT\_PIN, LOADCELL\_SCK\_PIN);
215. **scale.set\_scale(calibration\_factor\_custom);**
216. scale.tare();
218. // LCD Set Up & Verification:
219. lcd.begin(20, 4);
220. **#ifdef LCD\_DEBUG**
221. for (int i = 0; i < 3; i++) { // 3 quick blinks of backlight
222. lcd.backlight();
223. delay(200);
224. lcd.noBacklight();
225. **delay(200);**
226. }
227. lcd.backlight();
228. lcd.clear();
229. lcd.write("Testing 1.. 2..");
230. **#else**
231. lcd.backlight();
232. lcd.clear();
233. lcdPrintIdle();
234. #endif
236. // Interrupts Set Up:
237. #ifdef BUTTON1\_DEBUG
238. // Set button pin with a pull-up resistor
239. pinMode(BUTTON\_PIN\_1, INPUT\_PULLUP);
240. **// Triggers on a falling edge (button press)**
241. attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN\_1), button1Press, FALLING);
242. #endif
244. #ifdef BUTTON2\_DEBUG
245. **// Set button pin with a pull-up resistor**
246. pinMode(BUTTON\_PIN\_2, INPUT\_PULLUP);
247. // Triggers on a falling edge (button press)
248. attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN\_2), button2Press, FALLING);
249. #endif
251. #ifdef BUTTON3\_DEBUG
252. // Set button pin with a pull-up resistor
253. pinMode(BUTTON\_PIN\_3, INPUT\_PULLUP);
254. // Triggers on a falling edge (button press)
255. **attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN\_3), button3Press, FALLING);**
256. #endif
258. // LED Pin Set Up:
259. #ifdef BUTTON1\_DEBUG
260. **pinMode(LED\_PIN, OUTPUT);**
261. #endif
263. // TIMER interrupts
264. // TIMER<n>, prescaller = 80, counting up
265. **idleTimer = timerBegin(0, 80, true);**
266. deepSleepTimer = timerBegin(2, 80, true);
268. // Attach the interrupts to the timers
269. timerAttachInterrupt(idleTimer, &onIdle, true);
270. **timerAttachInterrupt(deepSleepTimer, &onSleep, true);**
272. // Set when the event should fire (time in us)
273. timerAlarmWrite(idleTimer, WAIT\_FOR\_IDLE, true);
274. timerAlarmWrite(deepSleepTimer, WAIT\_FOR\_DEEP\_SLEEP, true);
276. // Enable generation of timer events
277. timerAlarmEnable(idleTimer);
278. timerAlarmEnable(deepSleepTimer);
280. **// Stop the timer for now**
281. timerStop(idleTimer);
282. timerStop(deepSleepTimer);
284. // Cloud
285. **WiFi.mode(WIFI\_STA);**
286. WiFi.begin(NETWORK\_NAME, NETWORK\_PASS);
287. ThingSpeak.begin(client);
288. timeSinceLastConnection = millis();
289. }
290. **//=============================================================================**
291. // LOOP:
292. void loop() {
293. unsigned long timeNow = millis();
295. **// Check WiFi connection status**
296. if (WiFi.status() != WL\_CONNECTED &&
297. timeNow - timeSinceLastConnection > CONNECTION\_DELAY) {
298. timeSinceLastConnection = timeNow;
299. #if defined SERIAL\_DEBUG && defined WIFI\_DEBUG
300. **Serial.println("Attempting to connect");**
301. #endif
302. WiFi.begin(NETWORK\_NAME, NETWORK\_PASS);
303. #if defined SERIAL\_DEBUG && defined WIFI\_DEBUG
304. if (WiFi.status() == WL\_CONNECTED)
305. **Serial.println("Connected");**
306. #endif
307. }
308. #ifdef LCD\_DEBUG
309. if (timeNow - timeSinceLastPrint > LCD\_PRINT\_DELAY) {
311. // Scale Output via LCD
312. lcd.clear();
313. lcd.setCursor(0, 0);
314. lcd.print("Reading: ");
315. **lcd.print(int(scale.get\_units()), 1);**
316. lcdPrintUnitSuffix(unitType);
318. lcd.setCursor(0, 2);
319. lcd.print("Raw Data: ");
320. **lcdPrintRawData(unitType);**
322. // Resets timer
323. timeSinceLastPrint = timeNow;
324. }
325. **#endif**
327. // State management
328. int readWeight = (int)scale.get\_units();
330. **// Check if the weight value is below the threshold**
331. if (readWeight < WEIGHT\_THRESHOLD) {
332. // idle action
333. if (state == active) {
334. timeSinceLastStateChange = timeNow;
335. **state = calibrating;**
336. // calibrate
337. scale.tare();
338. #ifndef LCD\_DEBUG
339. lcdPrintCalibrating();
340. **timerRestart(idleTimer);**
341. timerStart(idleTimer);
342. #endif
343. }
345. **// done weighing**
346. if (state == done) {
347. timeSinceLastStateChange = timeNow;
348. state = idle;
349. #ifndef LCD\_DEBUG
350. **lcdPrintIdle();**
351. #endif
352. }
353. } else {
354. // wake from sleep
355. **if (state == asleep || state == deepSleep) {**
356. wakeDevice();
357. timeSinceLastStateChange = timeNow;
358. state = active;
359. }
361. // active action
362. if (state == idle) {
363. timeSinceLastStateChange = timeNow;
364. state = active;
365. **}**
367. #ifndef LCD\_DEBUG
368. // flash weight every second until done
369. if (state == active && timeNow - timeSinceLastPrint > LCD\_PRINT\_DELAY) {
370. **// Scale Output via LCD**
371. lcd.clear();
372. lcd.setCursor(0, 0);
373. lcd.print(int(scale.get\_units()), 1);
374. lcdPrintUnitSuffix(unitType);
375. **timeSinceLastPrint = timeNow;**
376. }
377. #endif
378. }
380. **// sleep action**
381. if (state == idle && timeNow - timeSinceLastStateChange >= TIME\_CONDITION) {
382. timeSinceLastStateChange = timeNow;
383. sleepDevice();
384. state = asleep;
385. **timerRestart(deepSleepTimer);**
386. timerStart(deepSleepTimer);
387. } else if (state == idle) {
388. lcdPrintIdle();
389. timerStop(idleTimer);
390. **}**
392. // deep sleep
393. if (state == deepSleep) {
394. lcd.noBacklight();
395. **lcd.noDisplay();**
396. timerStop(deepSleepTimer);
397. }
399. // wait for user to stay on scale for 5 seconds
400. **if (state == active &&**
401. timeNow - timeSinceLastStateChange >= WAIT\_FOR\_WEIGHING) {
402. timeSinceLastStateChange = timeNow;
403. state = done;
404. // display weight
405. **#ifndef LCD\_DEBUG**
406. lcdPrintWeight(readWeight, unitType);
407. #endif
408. // write to BLE
409. String txValue = "R";
410. **characteristic->setValue(txValue.c\_str());**
411. characteristic->notify();
412. delay(50);
414. txValue = String(readWeight);
415. **characteristic->setValue(txValue.c\_str());**
416. characteristic->notify();
418. // write to cloud
419. {
420. **int response = ThingSpeak.writeField(CHANNEL\_ID, 1, readWeight, API\_KEY);**
422. #if defined SERIAL\_DEBUG && defined WIFI\_DEBUG
423. if (response == 200) {
424. Serial.println("Message sent to cloud");
425. **} else {**
426. Serial.println("Other error");
427. lcd.setCursor(0, 1);
428. lcd.print("ERROR: ");
429. lcd.print(response);
430. **}**
431. #endif
432. }
433. }
435. **// Interrupts:**
436. // Toggle the sleep or wake when BUTTON 1 is pressed
437. #ifdef BUTTON1\_DEBUG
438. if (button1Pressed) {
439. button1State = !button1State; // toggle button 1 state
440. **digitalWrite(LED\_PIN, button1State); // visual of button 1 state**
442. // JUST EXPERIMENTING:
443. String txValue = "R";
444. characteristic->setValue(txValue.c\_str());
445. **characteristic->notify();**
446. delay(50);
448. txValue = String(scale.get\_units());
449. characteristic->setValue(txValue.c\_str());
450. **characteristic->notify();**
452. button1Pressed = false; // Reset the button1Pressed flag
453. }
454. #endif
456. // Toggle scale units between lbs & kgs when BUTTON 2 is pressed.
457. // Kilograms when unit state is true, else pounds when false.
458. #ifdef BUTTON2\_DEBUG
459. if (button2Pressed) {
460. **unitType = unitType == metric ? imperial : metric;**
461. setUnits(unitType);
462. button2Pressed = false; // Reset the button2Pressed flag
463. }
464. #endif
466. // Turns on LCD screen when BUTTON 3 is pressed
467. #ifdef BUTTON3\_DEBUG
468. if (button3Pressed) {
469. wakeDevice();
470. **button3Pressed = false; // Reset the button3Pressed flag**
471. }
472. #endif
473. }
475. **//=============================================================================**
476. // FUNCTIONS:
477. void button1Press() {
478. button1Pressed = true; // Set the flag to indicate button 1 press
479. }
481. #ifdef BUTTON2\_DEBUG
482. void button2Press() {
483. button2Pressed = true; // Set the flag to indicate button 2 press
484. }
485. **#endif**
487. void button3Press() {
488. button3Pressed = true; // Set the flag to indicate button 3 press
489. }
491. // Set the units for the scale
492. void setUnits(unitType\_t unitType) {
493. switch (unitType) {
494. case metric:
495. **scale.set\_scale(2.2 \* calibration\_factor\_custom);**
496. break;
497. case imperial:
498. default:
499. scale.set\_scale(calibration\_factor\_custom);
500. **break;**
501. }
502. }
504. // Print the units on the LCD
505. **void lcdPrintUnitSuffix(unitType\_t unitType) {**
506. switch (unitType) {
507. case metric:
508. lcd.print(" [kgs]");
509. break;
510. **case imperial:**
511. default:
512. lcd.print(" [lbs]");
513. break;
514. }
515. **}**
517. // Print raw data to LCD
518. void lcdPrintRawData(unitType\_t unitType) {
519. switch (unitType) {
520. **case metric:**
521. lcd.print(abs(scale.read()));
522. break;
523. case imperial:
524. default:
525. **lcd.print(abs(scale.read()));**
526. break;
527. }
528. }
530. **// Start putting the device to sleep**
531. void sleepDevice(void) {
532. #ifdef SERIAL\_DEBUG
533. Serial.println("Going to sleep now...");
534. #endif
535. **lcd.clear();**
536. lcd.setCursor(0, 0);
537. lcd.print("Sleeping...");
538. }
540. **// Wake the device**
541. void wakeDevice(void) {
542. #ifdef SERIAL\_DEBUG
543. Serial.println("I'm awake!");
544. #endif
545. **lcd.backlight();**
546. lcd.display();
547. }
549. // Print idle message to LCD
550. **void lcdPrintIdle(void) {**
551. lcd.clear();
552. lcd.setCursor(0, 0);
553. lcd.print("Waiting...");
554. }
556. // Print Calibration message to LCD
557. void lcdPrintCalibrating(void) {
558. lcd.clear();
559. lcd.setCursor(0, 0);
560. **lcd.print("Calibrating...");**
561. }
563. // Print weight to LCD
564. void lcdPrintWeight(int weight, unitType\_t unitType) {
565. **lcd.clear();**
566. lcd.setCursor(0, 0);
567. lcd.print(weight);
568. lcdPrintUnitSuffix(unitType);
569. }
571. // Idle interrupt
572. void IRAM\_ATTR onIdle(void) {
573. state = idle;
574. }
576. // sleep interrupt
577. void IRAM\_ATTR onSleep(void) {
578. state = deepSleep;
579. }

### ESPCalibration.ino:

1. */\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
2. *\* File: ESPCalibration.ino \**
3. *\* Group: The Embedded Dragon (T.E.D.) \**
4. *\* Members: Blake Wingard - cwingard@uab.edu \**
5. ***\* Gaurav Lunagariya - glunagar@uab.edu \****
6. *\* Joe Wong - jawong1@uab.edu \**
7. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**
8. *\* Desc: The purpose of this file to calibrate the \**
9. *\* HX711 for an attached scale. The instructions to \**
10. ***\* calibrate the sensor are displayed in serial and \****
11. *\* on the LCD screen. \**
12. *\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*
14. // Includes
15. **#include "HX711.h"**
16. #include <Wire.h>
17. #include <LiquidCrystal\_I2C.h>
19. // LCD
20. **// lcd(addr,en,rw,rs,d4,d5,d6,d7,bl, blpol)**
21. LiquidCrystal\_I2C lcd(0x3F, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);
23. // LCD\_I2C pin defines:
24. #define SDA\_PIN SDA
25. **#define SDA\_PORT PORTC**
26. #define SCL\_PIN SCL
27. #define SCL\_PORT PORTC
29. // HX711 circuit wiring
30. **#define LOADCELL\_DOUT\_PIN 2**
31. #define LOADCELL\_SCK\_PIN 0
33. // Interrupt pin defines:
34. #define BUTTON\_PIN\_1 26
35. **#define BUTTON\_PIN\_2 27**
36. #define BUTTON\_PIN\_3 33
38. // Serial defines
39. #define BAUD\_RATE 115200
41. volatile bool buttonPressed = false;
43. HX711 scale;
45. **void setup() {**
46. Serial.begin(BAUD\_RATE);
48. // Scale Set Up:
49. scale.begin(LOADCELL\_DOUT\_PIN, LOADCELL\_SCK\_PIN);
51. // LCD Set Up:
52. lcd.begin(20, 4);
53. lcd.backlight();
54. lcd.clear();
56. // Interrupts Set Up:
57. // Set button pins with a pull-up resistor
58. pinMode(BUTTON\_PIN\_1, INPUT\_PULLUP);
59. pinMode(BUTTON\_PIN\_2, INPUT\_PULLUP);
60. **pinMode(BUTTON\_PIN\_3, INPUT\_PULLUP);**
62. // Triggers on a falling edge (button press)
63. attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN\_1), buttonPress, FALLING);
64. attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN\_2), buttonPress, FALLING);
65. **attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN\_3), buttonPress, FALLING);**
66. }
68. void loop() {
70. **if (scale.is\_ready()) {**
71. // set scale to factory default
72. Serial.println("Tare... remove any weights from the scale.");
73. Serial.println("Press a button when done...");
75. **lcd.clear();**
76. lcd.setCursor(0, 0);
77. lcd.print("Tare...");
78. lcd.setCursor(0, 1);
79. lcd.print("remove any weights");
80. **lcd.setCursor(4, 3);**
81. lcd.print("Press button");
83. // wait for user to press a button
84. buttonPressed = false; // Reset the buttonPressed flag
85. **while (!buttonPressed);**
87. // clear the scale
88. scale.set\_scale();
90. **// display current readings**
91. long reading = scale.get\_units(10);
92. Serial.print("Pre-tare reading: ");
93. Serial.println(reading);
94. Serial.println("Press a button when done...");
96. lcd.clear();
97. lcd.setCursor(0, 0);
98. lcd.print("Pre-tare reading: ");
99. lcd.setCursor(0, 1);
100. **lcd.print(int(reading), 1);**
101. lcd.setCursor(4, 3);
102. lcd.print("Press button");

105. **// wait for user to press a button**
106. buttonPressed = false; // Reset the buttonPressed flag
107. while (!buttonPressed);
109. // tare scale
110. **scale.tare();**
112. Serial.println("Tare done...");
113. Serial.println("Place a known weight on the scale...");
114. Serial.println("Press a button when done...");
116. lcd.clear();
117. lcd.setCursor(0, 0);
118. lcd.print("Tare done...");
119. lcd.setCursor(0, 1);
120. **lcd.print("Place weight");**
121. lcd.setCursor(4, 3);
122. lcd.print("Press button");
124. // wait for user to press a button
125. **delay(10); // crude debounce**
126. buttonPressed = false; // Reset the buttonPressed flag
127. while (!buttonPressed)
128. ;
130. **// report reading**
131. reading = scale.get\_units(10);
133. Serial.print("Result: ");
134. Serial.println(reading);
135. **Serial.println("Calibration factor will be the (reading)/(known weight)");**
137. lcd.clear();
138. lcd.setCursor(0, 0);
139. lcd.print("Calibration factor:");
140. **lcd.setCursor(3, 1);**
141. lcd.print("result/weight");
142. lcd.setCursor(4, 3);
143. lcd.print("Press button");
145. **// wait for user to press a button**
146. delay(10); // crude debounce
147. buttonPressed = false; // Reset the buttonPressed flag
148. while (!buttonPressed);
150. **lcd.clear();**
151. lcd.setCursor(0, 0);
152. lcd.print("Result: ");
153. lcd.setCursor(1, 1);
154. lcd.print(int(reading), 1);
155. **lcd.setCursor(4, 2);**
156. lcd.print("Press button");
157. lcd.setCursor(5, 3);
158. lcd.print("to restart");
160. **// wait for user to press a button**
161. delay(10); // crude debounce
162. buttonPressed = false; // Reset the buttonPressed flag
163. while (!buttonPressed);
164. } else {
165. **Serial.println("HX711 not found.");**
166. }
167. delay(1000);
168. }
170. **// FUNCTIONS:**
171. void buttonPress() {
172. buttonPressed = true;
173. }

## b. circuit schematic

****

## c. List of libraries

Load Cell Library – HX711: <https://github.com/bogde/HX711>

LCD Library - LiquidCrystal\_I2C: <https://github.com/ribasco/new-liquidcrystal>