Automatic Score Tracking Cornhole Game

Written By:

Harrison Overturf
Mondona Behroozian
Daniel Hurwitz

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2020

Table of Contents

Section *Page*

List of Tables, Figures, and Code Snips

Code Snips *Page*

Acknowledgments

This project was sponsored by Cypress Semiconductor. We want to thank all the contacts at Cypress for the time, knowledge, and guidance provided throughout development of this project. We want to mention the names of people who helped throughout this project.

- Dr. Dale Dolan - for advising the team and providing direction throughout the year.
- Patrick Kane - for inspiring the project and providing cornhole boards and BLE Mesh equipment, as well as providing contacts.
- Greg Landry - for taking time to meet with us virtually to help debug certain firmware issues.
- Mark Saunders - for answering technical questions and providing guidance.
- Dr. Tali Freed - for providing additional information on RFID.
- Chuck Bland - for additional guidance throughout the process.
- John Planck - for answering questions regarding code.

Abstract

The Automated Score Tracking (AST) Cornhole Game introduces wireless communication and modularity to the game of cornhole. AST eliminates the responsibility for teams to manually keep track of their score from round to round. Users easily connect to other AST game boards by pairing the devices via bluetooth. The game begins once the boards pair. Players toss their bean bags, with points being awarded as per usual with one point being given to bags landing on the board and three points being given to bags that land through the hole. AST detects all possible outcomes for a bean bag toss including a “sink” where a bag lands in the hole located on the board, a “hit” where a bag lands and stays on the board, or a “miss” where the bag misses the board completely. This system reduces scoring errors made by participants by continually tracking the progress of the game, allowing users to make changes to the scores as necessary. This system makes the game of cornhole simpler and more enjoyable.

Chapter 1. Introduction

Automated Score Tracking (AST) for Cornhole is a wireless system that allows for ease of game flow. Cypress Semiconductor’s new Bluetooth Low Energy (BLE) technology inspired the idea. The CYBT-213043-Mesh evaluation module allows for users to receive real time score data, from round to round. The CYW20819 Bluetooth Module located on the mesh evaluation board integrates the use of components on the module to reduce the use of external components [1]. The bluetooth module allows for the two game boards to communicate scores for each team per round. The system first needs to accurately award points to each team in order for the game boards to communicate the scores . The system uses passive RFID to detect a “hit” which awards 1 point to the team, a “miss” which awards 0 points to the team, or “sink” which awards 3 points to the team [2]. The implementation of RFID embedded in each team's bean bags along with RFID sensors located on the game boards allows for the display of scores [3]. Once the system has determined the points and communicated using BLE, each team can view their scores on the CySmart app that Cypress Semiconductor developed for the Mesh evaluation modules. The level 0 and level 1 block diagrams for the system are shown in Figures 1 and 2 respectively.

The motivation behind AST is to innovate the classic game of cornhole. The purpose of this project is to improve the game play for users without changing the rules. The objective of classic cornhole is for each team to earn points each round, the team that reaches 21 points first wins [4]. The problem that AST resolves is the manual score-keeping each team has to do from round to round. AST eliminates the hassle of remembering each team's score while using the “Cancellation Scoring” method during the game. “Cancellation Scoring” is a point subtracting method where one player’s points cancel the points of another player each round, for the calculation of points of each team [4]. With the implementation of AST, no player has to remember their score or use the cancellation scoring method. AST improves efficiency and user enjoyment.

While on the surface the gameplay and feel will remain unchanged the user will now be able to interact with the game via phone or other peripheral device to view game score and statistics in real time due to the implementation of low energy bluetooth mesh. The system creation intends to act as an example implementation for Cypress Semiconductors latest technologies. The CYBT-213043-Mesh is Cypress’ latest consumer FPGA designed to make the implementation of bluetooth mesh easy and accessible to hobbyists and enthusiasts. The system uses the evaluation boards to pair the two games boards to each other and to other optional modules such as the companion app and score boards. The overall schedule for the project as well as a detailed breakdown of the tasks done to complete the project are shown in Chapter 2 Project Planning.

Chapter 2. Project Planning

2.1 Customer Needs Assessment

Automated Score Tracking Cornhole Game uses new Cypress Semiconductor technology. Cypress is looking for different ways to demonstrate the capabilities of their new EZ-BT Mesh Evaluation Kit. Cypress Semiconductor is our customer and our project implements their new technology and showcases the features of the EZ-BT Mesh Evaluation Kit. Our customer is looking for a modular way for standard cornhole boards to communicate wirelessly and maintain track of each team’s score throughout a game of cornhole. The system accurately tracks “hits”, “sinks”, and “misses” during the game. While implementing automated score tracking, the game rules remain the same, to avoid confusion for players. The functionality of the cornhole game remains the same for players but adds an ease of use with these customer needs.

2.2 Requirements and Specifications

The requirements that surround the AST Cornhole Game revolve around creating an experience that implements the technology in a way that does not change the play of the game. To achieve this, the game must be able to play in its normal environment with its usual game flow. To start this means that all the technology must fit underneath the boards so that they do not interfere with the users play. The boards must also be internally powered so that there are no extra cables to interfere with play. Common cornhole game environments include: grass fields, the beach, and both paved and unpaved roads. Encased circuits and sensors allow for the AST boards to function in all environments and an enclosure ensures dust and water resistance. Table I shown below summarizes these requirements and specifications.

TABLE I

Automated Score Tracking Cornhole Game Requirements and Specifications

Marketing Requirements	**Engineering Specifications**	**Justification**
1	2 feet x 4 feet	Must not exceed regulation corn hole size
2, 3	CYBT-213043 Mesh Evaluation Boards	The cornhole boards will communicate through Cypress BLE technology
2	Boards will operate without external power	Boards should be portable
4	Automatic Score Tracking	The boards will be able to identify all possible outcomes of a toss
5	Enclosed housing for electronics (IP51)	To protect boards in common use cases
4	User intervention possible	To be able to adjust if system error occurs
2, 3, 4	Mobile Phone application	Develop the ability to interface with mobile applications

Marketing Requirements

1. Fits inside a standard size cornhole set
2. Wireless/portable boards
3. Showcase Implementation of new Cypress technologies
4. Minimal user intervention
5. Water and dust resistant

TABLE II

Automated Score Tracking Cornhole Game Deliverables

Delivery Date	Deliverable Description
Dec 6th 2019	Design Review
March 6th 2020	EE 461 demo
March 13th 2020	EE 461 report
June 5th 2020	EE 462 demo
June 5th 2020	Cypress Semiconductor Design Review
June 9th 2020	EE 462 Report

2.3 Functional Decomposition

TABLE III

Automated Score Tracking Cornhole Game Level 0 Functional Decomposition

Module	Automatic Score Tracking
Inputs	- Hit Sensor: RFID - Sink Sensor: RFID - Miss Sensor: RFID - User Input: CySmart App - Power: 3V battery (x2) (CR2032)
Outputs	- Scores: Team1 and Team2 individual scores at the end of each round - CySmart App: Team1 and Team2 individual scores at the end of each round
Functionality	Detect each team's toss as a "hit" on the board using the "hit" sensor data. Detect each team's toss as a "sink" in the hole using the PIR sensor data. After detecting each team's toss input, calculate the score and output Team 1 score and Team 2 score. Teams view their score on the CySmart app, if scores are incorrect users can change the score using the user input.

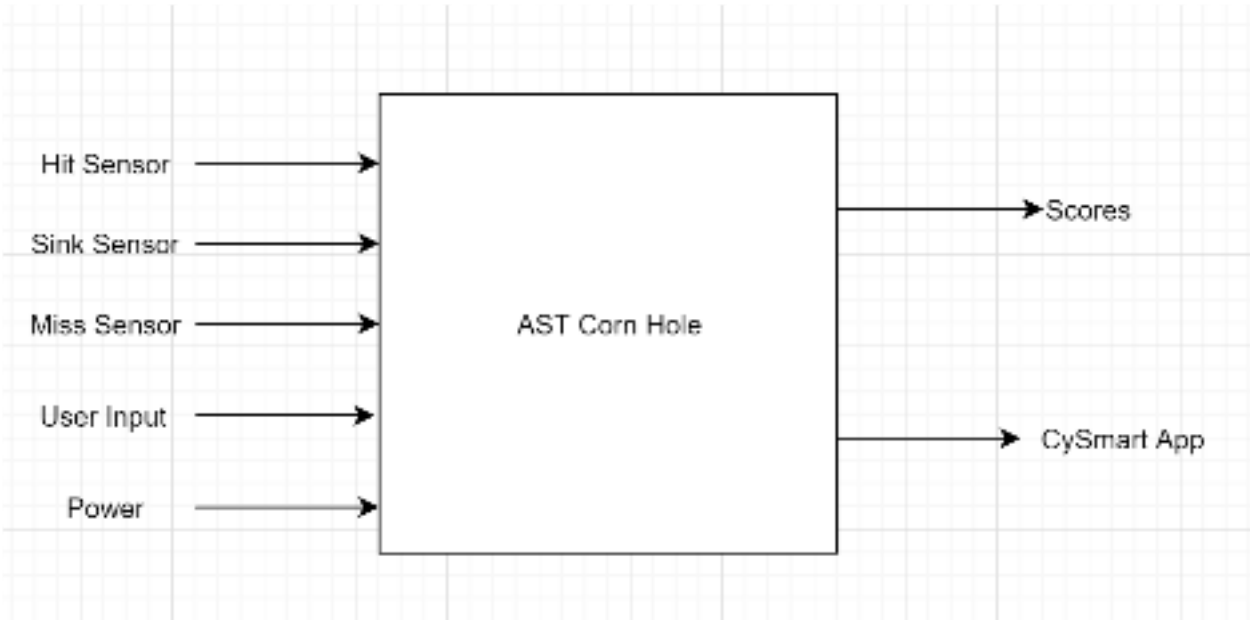

Figure 1. Block Diagram Level 0

TABLE IV

Automated Score Tracking Cornhole Game Level 1 Functional Decomposition RFID Module

Module	RFID
Inputs	- Hit Sensor: RFID - Sink Sensor: RFID - Miss Sensor: RFID
Outputs	- Bean bag position data: On board, in hole, off board
Functionality	Each bean bag will contain an RFID Chip which allows for the detection of each bag's position, either on, off, or in the hole of the game board. The position of each bag correlates to a “hit”, “miss”, or “sink”. The RFID reader on the board will then send the bean bag position data to the CYBT-213043 Mesh Evaluation Board.

TABLE V

Automated Score Tracking Cornhole Game Level 1 Functional Decomposition CYBT-213043 Module

Module	CYBT-213043 Mesh Evaluation Board
Inputs	- Bean bag position data: RFID - CySmart App: User Input - Power Supply: 3V
Outputs	- Scores: Team1 and Team2 individual scores at the end of each round - CySmart App: Team1 and Team2 individual scores at the end of each round
Functionality	The Mesh Evaluation Board is the module that ties everything together. The board is powered wirelessly using a 3V battery. It takes in position data to calculate the scores of each team. The board also takes in the user input from the CySmart App to correct any mistakes in the scores. Once all calculations have been done, the Mesh Evaluation Board communicates both teams' scores onto an external display and through the CySmart App for users to view their scores at the end of each round.

TABLE VI

Automated Score Tracking Cornhole Game Level 1 Functional Decomposition Power Supply Module

Module	Power Supply
Inputs	- Battery: 3V x 2
Outputs	- DC Voltage: 3V
Functionality	The power supply uses two 3V batteries to provide a 3V DC Voltage to the CYBT-213043 Mesh Evaluation Board.

TABLE VII

Automated Score Tracking Cornhole Game Level 1 Functional Decomposition CySmart App Module

Module	CySmart App
Inputs	- User Input: User corrected score - CYBT-213043: “hit”, “miss”, “sink” data for each team
Outputs	- CySmart App: Updates the user corrected score in the app
Functionality	The CySmart app allows for user inputs if any mistakes have been displayed in the score for each team. The user can manually use the app to correct their team's score. Once the user has changed their score, the app will update and output the appropriate number.

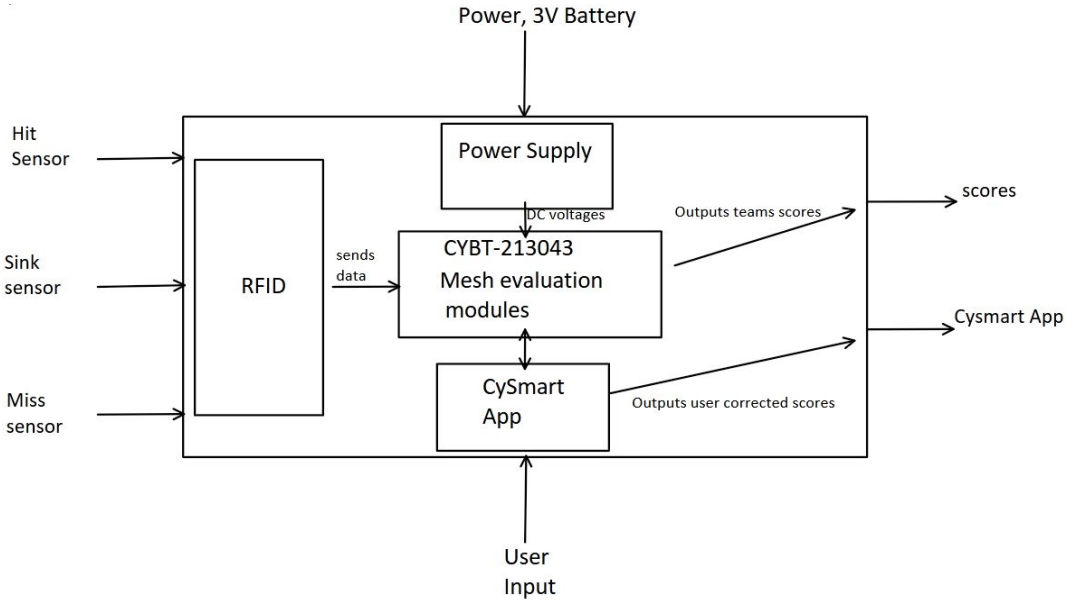

Figure 2: Block Diagram Level 1

Table VIII

Automated Score Tracking Gantt Table

TASK NAME	START DATE	END DATE	START ON DAY*	DURATION* (WORK DAYS)	TEAM MEMBER
Project Plan					
Abstract (Proposal) V1	9/19	9/23	0	4	Harrison
Requirements and Specifications (V1)	9/25	10/7	6	12	Mondy
Block Diagram	10/9	10/14	20	5	Daniel
Literature Search	9/25	10/21	6	26	Harrison
Gantt Chart	10/23	10/28	34	5	Mondy
Cost Estimates	10/23	10/28	34	5	Daniel
ABET Sr. Project Analysis	10/25	11/4	36	10	Harrison
Requirements and Specifications (V2)	11/6	11/11	48	5	Mondy
Research RFID, PIR, Vibration Sensing	11/11	11/22	53	11	Harrison
Report V1	9/23	11/15	4	53	Daniel
Design Review	11/21	12/6	63	15	Harrison
Order Components	12/2	12/16	74	14	Daniel
Report V2	11/29	12/9	71	10	Mondy
EE 461					
Design Hit Sensor Implementation Method 1	1/6	1/20	109	14	Mondy

Design Sink Sensor Implementation Method 1	1/6	1/20	109	14	Daniel
Design Miss Sensor Implementation Method 1	1/13	1/27	116	14	Harrison
Design User Input Implementation (button)	1/13	1/27	116	14	Mondy
Design User Input Implementation (Cysmart App)	1/13	2/5	116	23	Daniel
Design Build Test Score Outputs	1/13	2/3	116	21	
Design Review 1	1/6	2/3	109	28	Harrison
Design Hit Sensor Implementation Method 2	2/3	2/17	137	14	Mondy
Design Sink Sensor Implementation Method 2	2/3	2/17	137	14	Daniel
Design Miss Sensor Implementation Method 2	2/3	2/24	137	21	Harrison
Design User input Implementations (Buttons and App)	2/10	2/24	144	14	Mondy
Design Review 2	2/10	2/24	144	14	Daniel
Decide on Method 1 or 2 for each input	2/25	3/2	159	6	Harrison
Build and Test Prototype	3/2	3/9	165	7	Daniel
EE 461 Demo	1/6	3/6	109	60	Mondy
EE 461 Report	2/28	3/13	162	14	Daniel
Prototype Model for Cypress Conference	1/6	3/22	109	76	Harrison
EE 462					
Design Review 3	3/29	3/30	192	1	Daniel
Decide on Improvements to Project	3/30	4/6	193	7	Harrison
EE 462 Demo	4/1	5/31	195	60	Daniel
ABET Sr. Project Analysis	5/17	5/31	241	14	Mondy
Senior Project Expo Poster	5/17	5/31	241	14	Harrison
EE 462 Report	5/22	6/5	246	14	Daniel

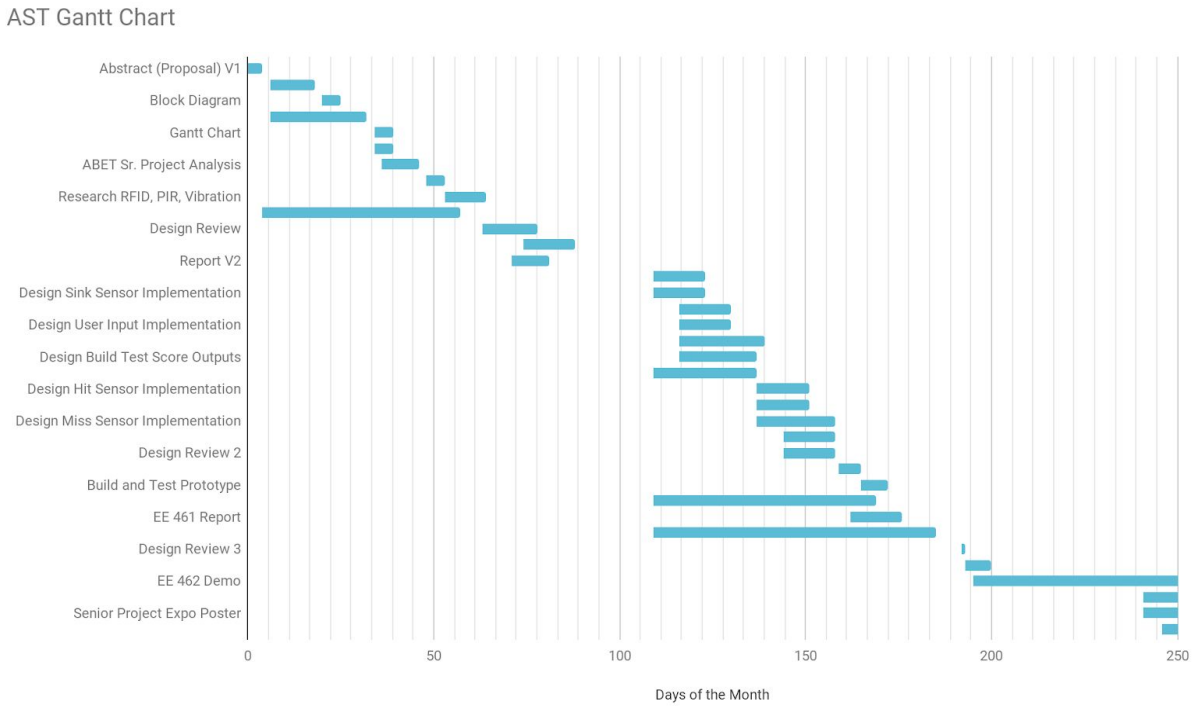

Figure 3: AST Gantt Chart

Table IX

Automated Score Tracking Cost Estimate Table

Expense	Cost Analysis	Justification
Triple-Axis Accelerometer - ±2/4/8g @ 14-bit - MMA8451	$Cost_a$ = \$6.00 $Cost_m$= \$7.95 $Cost_b$ = \$9.00 Cost = $\frac{6 + (4)7.95 + 9}{6}$ **Cost = \$7.80**	Method 1 for detecting hit case.
CR2032 Battery 10pack	$Cost_a$ = \$2.00 $Cost_m$= \$3.95 $Cost_b$ = \$5.00 Cost = $\frac{2 + (4)3.95 + 5}{6}$ **Cost = \$3.80**	Method 1 of powering the system wirelessly.
IZOKEE RFID Kit 3pack	$Cost_a$ = \$8.95	Method 1 for detecting sink case

	$Cost_m$= $10.99 $Cost_b$ = $12.95 Cost = $\frac{8.95 + (4)10.99 + 12.95}{6}$ **Cost = $10.97**	
RFID Sticker NFC tag	$Cost_a$ = $12.00 $Cost_m$= $14.98 $Cost_b$ = $17.50 Cost = $\frac{12 + (4)14.98 + 17.50}{6}$ **Cost = $14.90**	To be put inside corn hole bags for sink detection
Adhesive Velcro Tape	$Cost_a$ = $3.50 $Cost_m$= $5.36 $Cost_b$ = $7.00 Cost = $\frac{3.5 + (4)5.36 + 7}{6}$ **Cost = $5.32**	Method of securing system to cornhole boards safely and easily
Labor	$Cost_m$=120 hrs $Cost_a$=100 hrs $Cost_b$=150 hrs Cost= $\frac{100+4(120)+150}{6}$ Cost=121.66 hrs per quarter Assuming same amount of work per quarter: Total Hrs=3*121.66 Total Hrs=365 hrs Typical hourly engineering intern wage: $30/hr Cost of Labor = $30/hr * 365 Hr Cost of Labor = $10950	EE 449 project says it is around 120 man hours of work. Let this represent out most realistic cost. If we are able to shave off 20 hrs of work that would be the most optimistic cost. If an additional 30 hrs are required to finish that would be the most pessimistic cost.
Total=	**$10,992.79**	

Chapter 3. System Design

3.1 RFID

Radio Frequency Identification (RFID) is the method we chose to detect bean bag tosses for each team. Passive RFID is an accurate, inexpensive, and battery free way to detect interactions between objects. This method seemed the most plausible over other options such as image detection or pressure sensing, when considering cost, time, and accuracy.

3.1.1 RFID Reader

For the purposes of cornhole, the RFID reader needs to be able to detect multiple bags at the same time, instantly. In order to meet this requirement, we considered a couple different options. The two options considered are high-frequency (HF) RFID and ultra high-frequency (UHF) RFID. HF has an operating frequency of 13.56 MHz and UHF has an operating frequency between 860MHz - 960 MHz. Both HF and UHF have the ability to read tags simultaneously, UHF can read more at once, but both met this requirement. We ultimately chose UHF as the best option due to the way we decided to assign the scores of a “hit”, “miss”, and “sink”. Part of the design is to use aluminum foil to block or attenuate signals from the tags to the reader in cases of a “hit” or “miss”. UHF is suited for this because its radio waves reflect off metal, allowing signals to be blocked with aluminum foil. HF prevents the radio waves from reflecting off metal, making it not ideal for the design. So the UHF RFID reader we chose for this project is the Sparkfun Simultaneous RFID Reader. This reader has a ThingMagic M6E UHF RFID Reader, and the board has an Arduino shield footprint so it can directly be connected to an Arduino-compatible board or microcontroller.

3.1.2 Passive RFID Tags

The RFID tags need to complement the reader. Since the reader we chose is UHF the tags also need to be UHF tags. The Sparkfun Simultaneous RFID Reader suggests to use UHF EPCglobal Gen2 tags, which is what we decided to use. Each tag has a Truly Unique ID (TID), so each one can be differentiated from one another by the reader. The TID is necessary in order to keep score for each team and identify how many points are awarded based on the position of the tag in relation to the cornhole board. The tags will be placed inside each bean bag.

3.1.3 Antenna

The RFID reader has a built in antenna on the board, however it only has a minimum amount of range. So to increase that range we need an external antenna. Like the tags, the antenna also needs to complement the reader, so we need a UHF RFID antenna. Antennas can either be linear or circular polarized. At the same gain as linear antennas, circular antennas have a shorter read range because they have to split the power between two planes rather than one. So to find an adequate circular polarized antenna with the proper gain is more expensive than a linear polarized antenna. We chose to use a linear vertical polarized UHF RFID antenna with a gain of 6dBi.

So one issue that comes up with choosing a linear polarized antenna over a circular polarized antenna is the antenna can only read one plane. So if the tag lands horizontally in relation to the antenna, the tag can be missed. To resolve this issue, we placed two tags in a cross and grouped them as one. So if one tag lands horizontally, the other will be vertical and can be seen by the antenna. The antenna sits centered, underneath the cornhole board.

3.1.4 Aluminum signal blocking and attenuation

UHF RFID radio waves reflect off of metal. We used this knowledge to design a way to intentionally block and attenuate signals. The metal we decided to use is aluminum because it is inexpensive and easily accessible.

To fully block signals from bags that miss the board (“miss”), we need to line the sides of the cornhole board with sheets of solid aluminum foil. Since the RFID reader and antenna are located underneath the cornhole board, any signal off the board will be blocked. Aluminum can also be used to attenuate the signal strength. To attenuate the signals from bags that land on top of the board (“hit”), we need to line the back of the board with aluminum mesh. Aluminum mesh is a woven aluminum wire, so there are gaps in the mesh. These gaps allow some signal to get through the mesh, which results in attenuated signals. The attenuated signal strength is characterized into a range, so any bag that lands and has a signal strength within this range is considered a “hit” and will be assigned 1 point. Since the reader and antenna are located under the cornhole board, any bags that land in the hole (“sink”) will be close to the reader with no aluminum blocking or attenuating the signal. The non-attenuated signal strength is characterized into a range as well, so any bag that lands within this range is considered a “sink" and is assigned 3 points. So the signal strength will be stronger for the bags that land in the hole, compared to the bags that land on the board.

3.2 BLE Mesh

Bluetooth Low Energy (BLE) Mesh is a network of wireless communication between many-to-many devices. BLE Mesh allows for many nodes to communicate messages to each other, this ability allows for the range of communication to broaden and increase the application for different types of uses. Cypress Semiconductor developed BLE Mesh devices and kits, which this project implements to demonstrate the uses for.

3.2.1 CYBT-213043-Eval Kit

The CYBT-213043-Eval Kit is an evaluation kit that allows for the user to develop bluetooth mesh networks and apply different applications. The CYBT-213043-02 module on the board is an integrated, programmable bluetooth module that enables BLE Mesh design. The eval kit is compatible with arduino shields, such as the Sparkfun Simultaneous UHF RFID Reader. The eval kit plugs into the RFID reader so the eval kit can act as a microcontroller to the RFID reader.

For the application of bluetooth communicated data, the project requires two CYBT-213043-Eval kits. One eval kit has the RFID reader attached to it while the other will be connected to an LCD scoreboard. The node containing the eval board and RFID reader is the sensor while the node containing the other eval board with the LCD is the scoreboard. The eval board in the sensor node is the server and the eval board in the scoreboard node is the client (more will be explained on server and client below). The client receives the sensor data, collected by the RFID reader, through bluetooth from the server. To understand the flow between the two nodes, please see Figure 4, below.

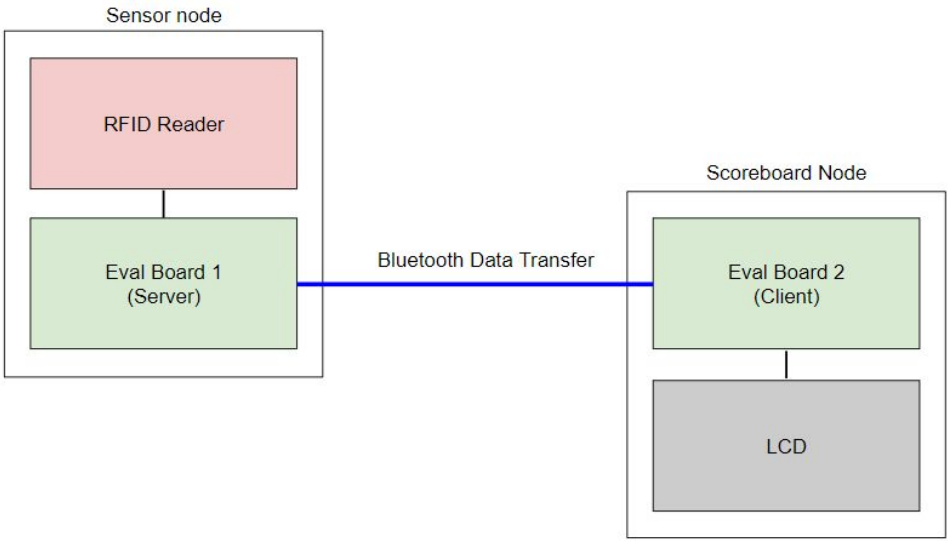

Figure 4: Data Flow from Sensor Server to Sensor Client Node

3.2.2 CYBT-213043-Mesh Kit

The CYBT-213043-Mesh Kit is also an evaluation kit, similar to the eval board. The mesh kit has the same CYBT-213043-02 bluetooth module as the eval kit, mentioned above. The difference between the two kits is the mesh kit includes many components such as sensors, RGB LEDs, and a user switch. These extra features implemented on the board allows the user to try multiple applications while integrating them in a bluetooth network.

For the application of bluetooth communicated data, the project requires one CYBT-213043-Mesh kit. The mesh kit will utilize the user switch feature on the board to tell the server that it is the end of a round, and it is time to collect data. The mesh board will send the server a “get” message, basically telling the sensor node that it is time to collect data. In this case the mesh board will be acting as a simple on-off client node, where it sends a get message but ignores the response back. In our design, the server needs to be told when to retrieve the data from the RFID Reader and send it to the client; this could have been done multiple ways, but we decided to use the mesh board to tell the server to retrieve the data to demonstrate the abilities of bluetooth mesh. To understand the complete flow between all three nodes, see Figure 5, below.

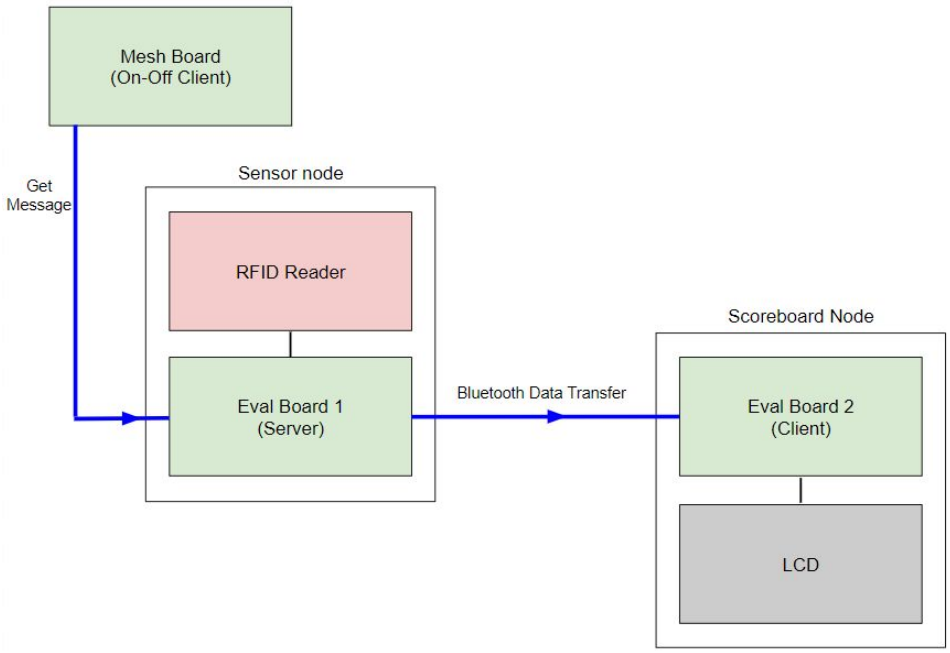

Figure 5: Data Flow from On-Off Client Node to Sensor Server

3.3 Score Board

The score board will display both teams' scores. The scores will update at the end of each round, upon the button press on the mesh board, from a user. The button press indicates the end of a round at which the data collection and bluetooth data transfer will occur, as mentioned above. The client eval board will receive the sensor data via bluetooth from the server eval board. This sensor data will then be displayed on a scoreboard.

3.3.1 LCD

There are many different types of displays, but two options were considered for this design: seven segment display and liquid crystal display (LCD). For ease of use and aesthetic purposes we believe that the LCD is a better fit for the design. An LCD allows us to easily display both teams' scores on a single display, and indicate which score belongs to each team.

As mentioned above, the LCD is connected to the client eval board and receives the current score data for each team, from that board. The LCD will update the current score upon the user button press on the mesh board, at the end of each round.

3.4 Design Revisions

The components mentioned in this section talk about how we revised the design of the system to better fit our timeline and capabilities after some trial and error runs with the original design.

3.4.1 Arduino

After starting to read documentation and learn more about bluetooth, mesh, and the CYBT-213043 Eval and Mesh kits, our group decided that it would be more time efficient to split the processing between the CYBT-213043 boards and an Arduino Uno.

We originally started using an arduino to test the functionality of the RFID reader because we are comfortable with arduino. Once we had that code working on the arduino, we tried to transfer it into the Cypress IDE, Modustoolbox, in order to program the CYBT-213043 boards. There needed to be a few tweaks to the code due to the difference in Arduino IDE and Modustoolboc IDE. However, we came across some issues. We tried a few different things, but it came to the point that our group decided to split the processing between an arduino and the CYBT-213043 boards in order to meet the scope of our timeline for the project. This decision was made because our group was new to the topics of bluetooth and mesh, we believe that our focus should be on the transfer of data through bluetooth rather than the processing of the RFID data.

Now, the RFID reader is plugged into the Arduino Uno, which processes the sensor data from the reader. The arduino is hardwired to the server eval board; so now the server eval board receives the sensor data from the arduino. The server eval board will take in the sensor data and send it via bluetooth to the client eval board. With this design, the focus for the CYBT-213043 Eval boards is just on sending and receiving the sensor data. To understand the new flow, please see Figure 6, below.

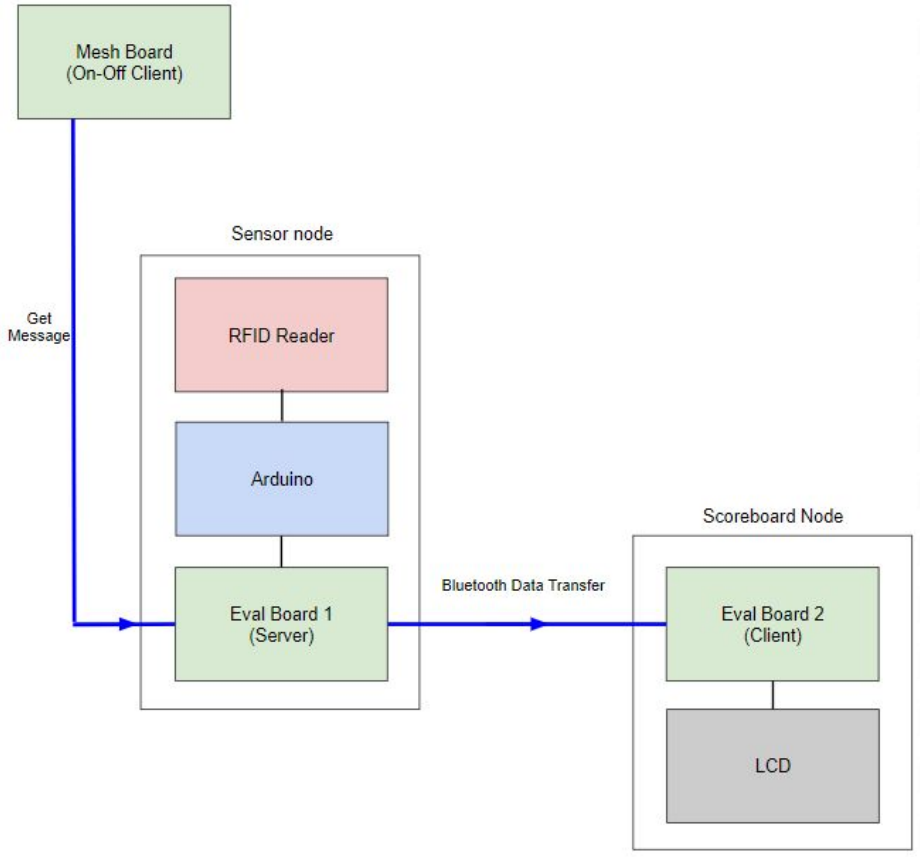

Figure 6: Dataflow from On-Off Client to Sensor Server with Arduino

3.4.2 CySmart

Originally the CySmart app was intended to allow user intervention if the score was miscalculated. However, the capabilities of the app are limited to lighting and weren't suited for our sensor based application. So this idea was eliminated.

Chapter 4. Development and Construction

4.1 Hardware

To meet the design requirement for all hardware to fit inside a standard 2x4 foot cornhole set, all hardware was assembled within the cornhole board.

4.1.1 Sensor Node

The RFID reader is connected to the arduino. The external antenna is connected to the RFID reader. To connect the external antenna, on the reader you need to clear the solder from the trace antenna and solder closed the jumper to the u.FL connector. This allows for the M6E Nano module on the reader to connect to the external antenna rather than the onboard antenna. See Figure 7, below.

Figure 7: RFID Reader Antenna Select Solder Jump

4.1.2 Cornhole Board Aluminum

In order to block signals from bean bags tossed outside of the board, the four sides of the board are lined with aluminum foil down to the ground as shown in Figure 8.

Figure 8: Aluminum foil board siding

To attenuate the signal strength of bean bags landing on the surface of the cornhole board, we tried a few methods. The first method we tested was aluminum mesh. Lining the back of the board with this resulted in full blockage of the signal. The issue with the aluminum mesh seems to be the size of the gaps between the woven wire. The gaps are too small, not allowing any radio wave to make it through. Perhaps if we tried an aluminum mesh with larger gaps, the result would have been better.

The second method we tested was uniform holes in aluminum foil. This method resulted in signal attenuation, however the attenuation was not consistent enough to characterize a range. The issue with this method is the difficulty of cutting uniform circles in aluminum foil. The foil is flimsy and is hard to work with when it comes to cutting out circles.

The third method we tested was strips of aluminum foil, arranged in a diamond pattern. This method is similar to the uniform circle method, however, it is much easier to handle cutting strips of aluminum foil and arranging them in a diamond pattern. When tested, this method gave the most consistent results. Figure 9 shows the aluminum foil pattern used to attenuate the signal strength.

Figure 9: On Board Attenuation Pattern

4.2 Firmware Development

Two sensor models were used for development of firmware for the CYBT-213043 Eval boards. Cypress Semiconductor includes code snips of these models, and more, in Modustoolbox. These models define the functionality set up and include many functions that can be used to develop firmware for these boards. The sensor models are a method that bluetooth mesh uses to interface between sensors.

4.2.1 Sensor Server Model

The sensor server model defines messages that models can send and receive. For this project the server model opens up the state of the sensor so the client can read the sensor data. One of the eval boards is programmed to be the server and will receive the sensor data from the arduino.

Due to forced isolation, the group decided to keep the user button on the server eval board, rather than on the on-off client mesh board. This decision allowed us to move forward with the rest of the project.

The function, `hal_gpio_app_test_input`, initializes pins 1-5 on the eval board as the RFID data score input pins. The way the arduino is packaging the data is four bits for the score and one bit to signify the team. So pins 1-5 receive the data from the arduino. Pin 1 is the least significant bit of the RFID score data, Pin 4 is the most significant bit of the RFID score data, and Pin 5 is the team bit. The last two lines of the function configure the user defined switch on the eval board, and then initializes that switch as an interrupt.

```
void hal_gpio_app_test_input(void)
{
        uint8_t index = 0;
        //initializes pins 1-5 as RFID data score input pins
    wiced_hal_gpio_configure_pin(WICED_P01, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);//LSB RFID data bit
    wiced_hal_gpio_configure_pin(WICED_P02, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P03, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P04, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);//MSB RFID data bit
    wiced_hal_gpio_configure_pin(WICED_P05, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);//will serve as Team bit

 //automatically maps to correct pins for any kit
wiced_hal_gpio_configure_pin(WICED_GPIO_PIN_BUTTON,(GPIO_INPUT_ENABLE|GPIO_PULL_DOWN|GPIO_EN_INT_RISING_EDGE),GP
IO_PIN_OUTPUT_LOW);
    //initialized SW3 user defined switch as interrupt
    wiced_hal_gpio_register_pin_for_interrupt(WICED_GPIO_PIN_BUTTON, hal_gpio_app_interrrupt_handler, NULL);
}
```

Code Snip 1: Sensor Server hal_gpio_app_test_input Function

`Set_RFID_data` is a function that takes in the sensor data processed by the arduino and sets the value for `RFID_data`. This function takes in the current state of the input pins and uses bit masking to add the corresponding value to `RFID_data`. See Code Snip 2 for the full function, below.

```
void set_RFID_data(int8_t v, int8_t w, int8_t x, int8_t y, int8_t z) //Based on data received from Arduino, this
function sets RFID data
 {
 if(v == 1)
  {
                RFID_data+=1;
        }
        else
        {
                RFID_data+=0;
        }
  if(w == 1)
  {
                RFID_data+=2;
        }
        else
        {
                RFID_data+=0;
        }

  if(x == 1)
  {
                RFID_data+=4;
        }
        else
        {
                RFID_data+=0;
 }

        if(y == 1)
  {
                RFID_data+=8;
        }
        else
        {
                RFID_data+=0;
        }

  if(z == 1)
  {
                RFID_data+=16;
        }
        else
        {
                RFID_data+=0;
        }
  }
```

Code Snip 2: Sensor Server set_RFID_data Function

The function seen below is the interrupt handler for the button press. Once the button is pressed, signifying the end of a round, the code jumps to this function and the variables (a,b,c,d,e) store the current state on the GPIO pins specified. Once the current state is stored, the function `Set_RFID_data` is called and the variables will run through it, updating the value for `RFID_data`. Also at the bottom of this function, `mesh_sensor_RFID_current_score` needs to be set to the correct value, which is `RFID_data`, since that is the variable that is being updated. `Wiced_bt_mesh_model_sensor_server_data` needs to be called in the interrupt handler in order to send out a status message to the client.

An initial issue occured when we were setting `mesh_sensor_RFID_current_score = RFID_data` in the sensor report handler. However, we realized that the report handler function would never be called unless the sensor client requested a "get" operation, which we had not set up in the client. So by doing this, the `mesh_sensor_RFID_current_score` was never being set since the code didn't go into the sensor report handler in the server model. So we learned that `mesh_sensor_RFID_current_score` needs to be set in the interrupt handler, since we are using a button interrupt. The report handler could be ignored for our case, since we are not using the client to send a "get" message.

```
void hal_gpio_app_interrrupt_handler(void *data, uint8_t pin) //Jump to this function once the button is pushed
{
        a = wiced_hal_gpio_get_pin_input_status(WICED_P01); //These variables store the current state on the
GPIO pins
        b = wiced_hal_gpio_get_pin_input_status(WICED_P02);
        c = wiced_hal_gpio_get_pin_input_status(WICED_P03);
        d = wiced_hal_gpio_get_pin_input_status(WICED_P04);
        e = wiced_hal_gpio_get_pin_input_status(WICED_P05);
        set_RFID_data(a, b, c, d, e);

        mesh_sensor_RFID_current_score = RFID_data;
        wiced_bt_mesh_model_sensor_server_data(MESH_SENSOR_SERVER_ELEMENT_INDEX,
WICED_BT_MESH_PROPERTY_RFID_VALUE, NULL); //This function sends a status message to the mesh which contains RFID
data
}
```

Code Snip 3: Sensor Server hal_gpio_app_interrupt_handler Function

Since we are using the function `hal_gpio_app_test_input` to configure the button, the function table needs to be revised so `hal_gpio_app_test_input` is included. This is important because the function table tells the provisioner what functions to use. Without including this function, the provisioner will try to set up the button on its own.

```
wiced_bt_mesh_app_func_table_t wiced_bt_mesh_app_func_table =
{
    mesh_app_init,              // application initialization
        hal_gpio_app_test_input,    // Button processing
    NULL,                       // GATT connection status
    NULL,                       // attention processing
    mesh_app_notify_period_set, // notify period set
        NULL,                                           // WICED HCI command
    NULL,                          // LPN sleep
    NULL                        // factory reset
};
```

Code Snip 4: Sensor Server Function Table

To view the full sensor server code, please see Appendix B.

4.2.2 Sensor Client Model

The sensor client waits for a status message sent from the server. Once the status message is received the client reads the sensor data that the server sent via bluetooth.

The main purpose for the sensor client is to read the data from the sensor server, the data being read is the score. The client will only know that there is data to read once it receives a status message from the sensor server. In our code, the status message the client is looking for from the server is

WICED_BT_MESH_SENSOR_STATUS. When the server sends out this status message, the client will go into the mesh_sensor_client_message_handler; here it will find the case for the particular status message and execute the code under the case. Once the status message is received the function wiced_bt_mesh_sensor_status_data_t is called. This function allows for the sensor status data structure to be exchanged between the application and the sensor model. One of the properties of the function is the raw_value of the sensor status data. The raw_value is stored in data using the memcpy function. The value stored in data is then displayed on the client terminal in Modustoolbox, using WICED_BT_TRACE.

```
void mesh_sensor_client_message_handler(uint8_t element_idx, uint16_t addr, uint16_t event, void *p_data)
{
         WICED_BT_TRACE("*****data*****:\n"); //Displays trace statement on Terminal

         uint8_t data[100]={0}; //Used to store the received RFID data
                                                                    //Data that is received is added on to the
previous value stored in this array

#if defined HCI_CONTROL
    wiced_bt_mesh_hci_event_t *p_hci_event;


#endif
    WICED_BT_TRACE("sensor clt msg:%d\n", event);


    switch (event)
    {
    case WICED_BT_MESH_TX_COMPLETE:
        WICED_BT_TRACE("tx complete\n");
        break;

#if defined HCI_CONTROL
    case WICED_BT_MESH_SENSOR_DESCRIPTOR_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_desc_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_descriptor_status_data_t
*)p_data);
        }
        break;

    case WICED_BT_MESH_SENSOR_STATUS://This is the only case we are using because the sensor node will send a
STATUS message with RFID data
        memcpy(data, ((wiced_bt_mesh_sensor_status_data_t *)p_data)->raw_value,
((wiced_bt_mesh_sensor_status_data_t *)p_data)->prop_value_len); //Used to add newly received RFID data to
previous value
        WICED_BT_TRACE("data:%B\n", data); //Displays data in terminal
        break;
```

Code Snip 5: Sensor Client mesh_sensor_client_message_handler Function

The function table for the sensor client model only requires `mesh_app_init`, all other functionality is `NULL` since nothing else is required to be configured.

```
wiced_bt_mesh_app_func_table_t wiced_bt_mesh_app_func_table =
{
    mesh_app_init,          // application initialization
    NULL,                   // Default SDK platform button processing
    NULL,                   // GATT connection status
    NULL,                   // attention processing
    NULL,                   // notify period set
         NULL,                                  // WICED HCI command
    NULL,                   // LPN sleep
    NULL                    // factory reset
};
```

Code Snip 6: Sensor Client Function Table

To view the full sensor client code, please see Appendix C.

4.2.3 Provisioning

Once both the sensor server model and the sensor client model are programmed onto the eval boards, the two models need to be grouped into the same network in order to talk to each other.

There are a few ways to provision the models. The first way is through the built-in provisioner in Modustoolbox, ClientControlMesh. ClientControlMesh is an application that uses a separate evaluation board in order to run the Bluetooth Stack and talk to the other boards. The second way to provision is to use another application provided by Cypress, however this one is not built-in to Modustoolbox and will have to be searched for in file explorer under the workspace, reference Appendix F for file path. This second application is called MeshClient. MeshClient is similar to ClientControlMesh, but the main difference is MeshClient does not require an external board to provision; it uses the bluetooth radio on the computer to run the Bluetooth Stack. The third way to provision is to use the phone application, MeshApp.

We tried all three ways and came across some issues. The first application we used was ClientControlMesh. The recurring issue that surfaced with this application was everytime we created a network and tried to open it, the network could not open and the ClientControlMesh would shut down and close itself. We used a Macbook rather than a windows operating system and believe this could be what caused the issue. The second attempt to provision was with MeshApp. We found the issue here is that MeshApp was developed for lighting purposes, so it was only letting lighting status messages get through. Our status messages were not related to lighting so they were blocked, and the client never was able to go into its message handler. The third attempt to provision was with MeshClient, however MeshClient could only be used with Windows. Once we used MeshClient with a Windows operating system, we were able to provision smoothly and successfully see the data communicate from the server to the client.

4.2.4 Arduino Firmware

The Arduino in this system is responsible for allowing the RFID reader to communicate with the server sensor. Once triggered The arduino will trigger a two second long continuous read that will allow for the reader to detect all present tags near the sensor.

```
//Read sequence
  if (button_bool == 1){
    //Read tags for 2 seconds
    while (millis() - start_time <= 2000){
      read_tags();
    }
    //Trigger print and output sequence
    print_info = 1;
    if (print_info == 1){
      print_data();
      //reset appropriate variables
      print_info = 0;
      button_bool = 0;
      int EPClist[10] = {0,0,0,0,0,0,0,0,0,0};
      int rssilist[10] = {0,0,0,0,0,0,0,0,0,0};
    }
  }
```

Code Snip 7: RFID Read Sequence

During the read sequence the arduino will maintain two lists. The first being boolean based list called `EPClist` that will indicate which of the RFID tags have been read during the continuous read, the next being a list that will correspond to the information in `EPClist` and provide the rssi (receiver signal strength indicator) which we use to determine if a bag is on the board or in the hole as shown in Code Snip 8.

```
// Assign point to proper team base on bag position
    if ((bag_num) % 2 == 0){ //Even numbered bags correspond to team 1
      // Missed bags will not have an rssi value
      if (bag_rssi == 0){
        t1_score = t1_score;
      }
      // Signal of bags on the surface will be attenuated below -39 dbm
        t1_score = t1_score + 1;
      else if (bag_rssi < -39){
      }
      // Bags with rssi signal between 0 and -39 dmb will be classified as through the hole
      else{
        t1_score = t1_score + 3;
      }
    }
    else{ //Odd numbered bags correspond to team 2
      // Missed bags will not have an rssi value
      if (bag_rssi == 0){
        t2_score = t2_score;
      }
      // Signal of bags on the surface will be attenuated below -39 dbm
      else if (bag_rssi < -39){
        t2_score = t2_score + 1;
      }
      // Bags with rssi signal between 0 and -39 dmb will be classified as through the hole
      else{
        t2_score = t2_score + 3;
      }
    }
```

Code Snip 8: Score Assignment

Next, the Arduino will determine which team will receive points after the round and trigger the team bit to adjust accordingly. The team bit will be set to 1 for team 1 and 0 for team 2. This is shown in Code Snip 9 below.

```
//If team 1 scored higher than team 2 set team bit to 1
  if (t1_score > t2_score){
    digitalWrite(team_out, HIGH);
    score_diff = t1_score - t2_score;
    Serial.print("team 1: ");
  }
  //If team 2 scored higher than team 1 set team bit to 0
  else if (t2_score > t1_score){
    digitalWrite(team_out, LOW);
    score_diff = t2_score - t1_score;
    Serial.print("team 2: ");
  }
  //If each team score equal exit function
  else{
    return
  }
```

Code Snip 9: Team Assignment

Finally, the Arduino will set the output score pins high or low depending on the difference in score. The score pins represent a 4 bit binary number allowing to display the maximum score difference of 12.

```
//Send a score difference of 2
  else if (score_diff == 2){
    Serial.println("2");
    digitalWrite(score_0, LOW);
    digitalWrite(score_1, HIGH);
    digitalWrite(score_2, LOW);
    digitalWrite(score_3, LOW);
  }
```

Code Snip 10: Example Score Output

To view full Arduino code, see Appendix D.

Chapter 5: System Performance

This chapter overviews the final performance of the entire system. The final results are compared to the engineering requirements from the project plan section. Also the final expenses are reviewed.

5.1 System Requirement Check

This section takes the original engineering specifications and compares how well the final results accomplished meeting the specification.
*Due to forced isolation, certain specifications needed to be adjusted.

Table X: Engineering Specifications vs. Final Design Results

Engineering Specifications	Final Results
2 feet x 4 feet	The system fits within the cornhole board, with the exception of an external eval board used for the user button to indicate the end of a round.
CYBT-213043 Mesh Evaluation Boards	CYBT-213043 Eval Boards were used rather than the Mesh boards due to the availability of pins. The Eval and Mesh boards contain the same bluetooth module.
Boards will operate without external power	*
Automatic Score Tracking	The system is able to track score and send the data via bluetooth to display on the client terminal in Modustoolbox and updates upon button press.
Enclosed housing for electronics (IP51)	*
User intervention possible	*
Mobile Phone application	*

5.2 Expense Check

All parts that were used in the final system were documented and the final cost for each, along with the overall total cost can be seen below in Table XI. Please note that many items were donated to the project by Cypress Semiconductor and are indicated in the table below.

Table XI: Automated Score Tracking Senior Project Actual Cost

Automated Score Tracking Senior Project Actual Cost						
Part	Count	Actual Cost ($) each	Shipping ($)	Tax ($)	Total ($)	Donated
CYBT-213043 Eval Board	2	0.00	0.00	0.00	0.00	Yes
CYBT-213043 Mesh Kit	2	0.00	0.00	0.00	0.00	Yes
Custom Cornhole Game Set	1	0.00	0.00	0.00	0.00	Yes
Arduino Uno	1	0.00	0.00	0.00	0.00	No
SparkFun Simultaneous RFID Reader - M6E Nano	1	224.95	8.99	17.43	251.37	No
UHF RFID Antenna	1	37.95	9.36	3.66	50.97	No
UHF RFID Tag - Adhesive (Set of 5)	2	1.95	11.76	0.15	15.81	No
Aluminum Foil Roll	1	0.00	0.00	0.00	0.00	No
Overall Total					318.15	

5.3 Overall Performance

Below in Figure 10 we see one example of the information generated by the Arduino. Currently there are four bags recognizable to the system. Bag zero is detected with an RSSI of -38 dbm this indicates that it is through the hole, giving team 1 three points. Next we see bag one has an RSSI of -62 dbm indicating that it has landed on top of the game board, giving team 2 one point. Bag 2 does not have an RSSI value meaning that it landed off of the game board, not contributing any points to the assigned team. Following this, bag 3 is detected on the board, adding one point to team 2's score, while bag 4 has missed the board. Once all the bag data is received, the Arduino will calculate the total round score for each team and then calculate who this round will affect the game score. The last line in Figure 10 shows that the round will end with team one receiving one point to their overall score.

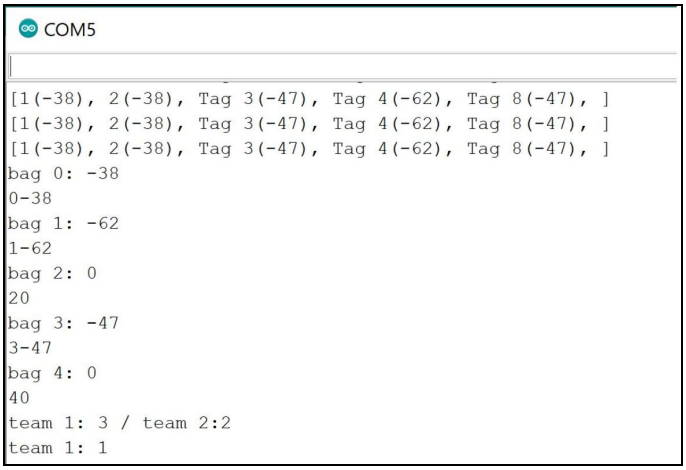

Figure 10: Round One Arduino Serial Monitor Output

Figure 11 and 12 show the data that has been received by the sensor client board. Figure 11 is the data before the Arduino sends it the new round data and Figure 12 is after receiving the new data. We can see that the data value goes from a hex value of 71 to 82. Using a programming calculator shown in Figure 13, we can see that the difference of these two data values gives you a binary result of 0001 0001. This aligns with the communication protocol that was assigned between the arduino and the mesh node (000[team indicator] [score difference]). The 1 indicates that team 1 will be gaining points and the 0001 indicates that they will be receiving 1 point. Figures 14, 15, and 16 shows this process again after another round of play ending with team 2 receiving one additional point.[1]

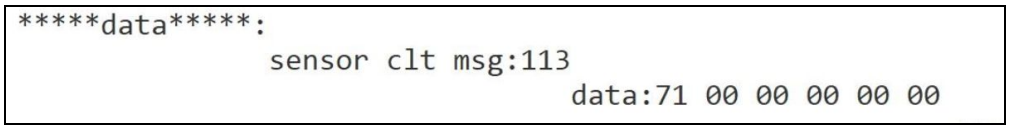

Figure 11: Previous Sensor Server Data

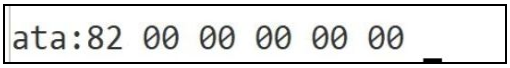

Figure 12: New Round Sensor Server Data

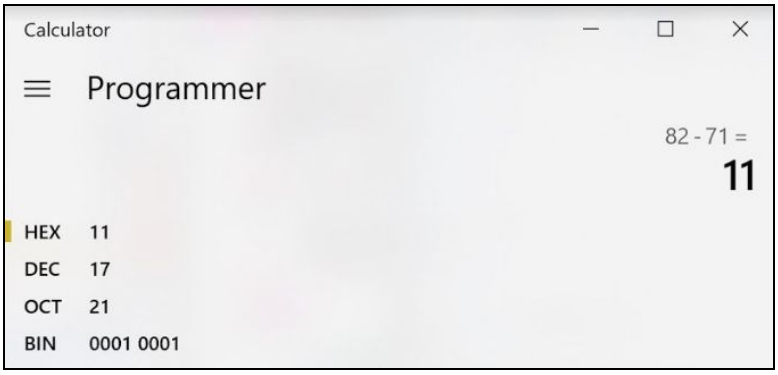

Figure 13: Round Difference Calculation

[1] Full demo and explanation of the entire system: https://youtu.be/vxVVcaOL13g

```
[3(-44), 4(-34), ]
[3(-45), 4(-34), ]
[3(-45), 4(-34), ]
bag 0: 0
00
bag 1: -45
1-45
bag 2: 0
20
bag 3: 0
30
bag 4: 0
40
team 1: 0 / team 2:1
team 2: 1
```

Figure 14: New Round Arduino Serial Output

```
*data*****:
          sensor clt msg:113
                              data:83 00 00 00 00 00
```

Figure 15: New Round Seneser Server Data

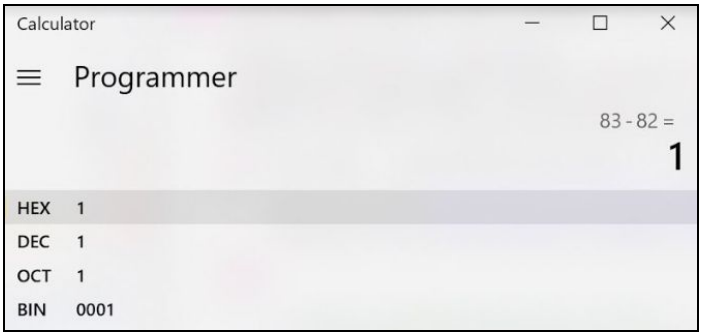

Figure 16: New Round Score Calculation

Chapter 6: Conclusion

6.1 Accomplishments

The Automated Score Tracking system for cornhole has a larger scope than was originally thought. However, the group was able to accomplish a working system in the end. The system is able to distinguish between the three outcomes of a bean bag toss, process that data with an arduino, and send it via bluetooth using the Cypress eval boards. There was definitely a lot to learn from this project, considering that we came in not knowing anything about bluetooth mesh or developing firmware. A lot of our time was used to read documentation to get a full understanding of the concepts. The biggest take away from the project is all the knowledge we gained about bluetooth mesh. We are happy that we have a system that works, but there is much room for improvement and advancements.

6.2 Issues

We want to highlight a few issues that we came across during the development of this project to inform future students about and how to handle them.

6.2.1 Phone MeshApp

MeshApp is a provisioning app for iPhone and Android. One detail we did not know was that MeshApp is used for provisioning lighting related models. We found that we were able to provision an example model that was related to lighting in MeshApp. However, when it came to our sensor models the status message from the sensor server was never making it to the sensor client because it seems MeshApp only allows lighting related messages to transmit. We suggest using MeshClient as the provisioning tool. As a reminder MeshClient can only be used on a Windows operating system.

6.2.2 Mac OS

One of our group members uses a Macbook, many things worked the same as it did on Windows. However, we noticed that the accessibility to certain things like MeshClient were not available for Mac OS. So we suggest having access to a Windows operating system to make sure everything runs smoothly.

6.2.3 Baud rate for terminal plug-in

Once we started testing our system we wanted to view and debug certain parts of the code. We downloaded a terminal plug-in, TM Term[2]. However, the terminal was not printed in english. This definitely was a baud rate issue. In the settings for the built-in terminal, there was a drop down menu to select from ten different baud rates. The baud rate for the boards is 115200, so this is what we set it to. This didn't help, so we tried every baud rate on the list but none of them worked. We talked to Cypress engineer, Greg, and he mentioned that the baud rate needs to be 921600; this is not one of the options on the drop down menu. So there is a way to change the baud rate from 921600 to 115200, so you can select the correct baud rate on the drop down menu. Please see Appendix E for instructions.

6.3 Future Improvement Opportunities

There is always room for improvement, and considering the scope of the project there can be a few large improvements made to make the project run smoother and accomplish the original goal to showcase Cypress' Bluetooth Mesh technology.

6.3.1 More Mesh Nodes

The goal is to showcase a mesh network, so the more nodes that are in the network, the more "meshy" it is. We included two nodes communicating to each other in a network. To improve, a button node can be added using a separate mesh board. This was in our original design plans, but time did not permit for us to include this third node in the network. Other nodes such as a motion sensor can use the PIR sensor on the mesh kit. It can be placed on the hole of the cornhole board and communicate with the sensor node to determine which team made a "sink" and will then flash an LED strip the color of the team that just scored. There are so many different possibilities with Bluetooth Mesh and the CYBT-213043 Mesh Kits can help bring the possibilities to reality.

[2] http://download.eclipse.org/releases/oxygen

6.3.2 Move data processing from arduino to CYBT-213043 Boards

The biggest improvement that can be made is moving the data processing from the Arduino to the CYBT-213043 Eval Boards. The eval board has an arduino layout so the RFID reader will plug right into it. The goal is to show the capabilities of the CYBT-213043 boards, so completely moving the processing to this board is a huge improvement.

References

[1] Cypress Semiconductor, “Ultra Low Power, BLE/BR/EDR Bluetooth 5.0 SoC,” CYW20819 datasheet, March 2018 [Revised May 2019].

[2] M. Schrag, *The Sports Rules Book*. 4th ed., Champaign, IL: Human Kinetics, 2018, pp. 86-89.

[3] J. Tang, and C. Chen. "A Billiards Track and Score Recording System by RFID Trigger," *Procedia Environmental Sciences*, vol. 11, no. PA,pp. 465–470, 2011.

[4] American Cornhole Organization. “Official Rules for the Sport of Cornhole” Internet: https://americancornhole.com/rules/, 2016 [Oct. 2019].

[5] M. Baert, J. Rossey, A. Shahid, and J. Hoebeke, “The Bluetooth Mesh Standard: An Overview and Experimental Evaluation,” *Sensors*, vol. 18, no. 8, p. 2409, Jul. 2018.

[6] D. Solda, et al. *Getting Started with Bluetooth Mesh.* Cypress Semiconductor Appl. Note AN227069.

[7] S. Bodapati, et al. “Systems and Methods to Detect Cross Reads in RFID Tags,” United States Patent 8854190, Oct. 7, 2014.

[8] A. Polo, et al. “Bluetooth low energy automation mesh network,” United States Patent 10440546, Oct. 8, 2019.

[9] A. Nikoukar, et al. “Low-Power Wireless for the Internet of Things: Standards and Applications,” *IEEE Access*, vol. 6, Nov. 2018, [DOI: 10.1109/ACCESS.2018.2879189]. Available: https://ieeexplore-ieee-org.ezproxy.lib.calpoly.edu/document/8528458/citations#citations. [Accessed Oct. 2019].

[10] S. M. Darroudi, R. Caldera-Sànchez and C. Gomez, "Bluetooth Mesh Energy Consumption: A Model," *Sensors (Basel, Switzerland),* vol. 19, *(5),* 2019. Available: http://ezproxy.lib.calpoly.edu/login?url=https://search-proquest-com.ezproxy.lib.calpoly.edu/docview/2193162340?accountid=10362. DOI: http://dx.doi.org.ezproxy.lib.calpoly.edu/10.3390/s19051238.

[11] M. Woolley, “Bluetooth Mesh Models: Technical Overview,” Version 1.0, March 2019.

[12] R. Heydon, J. Tanner, V. Zhodzishsky, et. al. “Mesh Model: Bluetooth Specification,” Revision v1.0, Jul. 13, 2017.

[13] *MeshClient and ClientControlMesh App User Guide*. San Jose, 2020 [Online]. Available: https://www.cypress.com/file/462491/download.

[14] *Simultaneous RFID Tag Reader Hookup Guide*. Sparkfun, 2020 [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Sparkfun%20PDFs/Simultaneous_RFID_Tag_Reader_HookupGuide_Web.pdf.

[15] "BLE Mesh", *cypress.com*, 2020. [Online]. Available: https://www.cypress.com/products/ble-mesh.

Appendix A. Senior Project Analysis

Project Title: Automated Score Tracking for the Game of Cornhole

Student's Name: Harrison Overturf, Daniel Hurwitz, Mondona Behroozian

Advisor's Name: Dale Dolan

Date: 11/18/19

1. Summary of Functional Requirements

Describe the overall capabilities or functions of your project or design. Describe what your project does. (Do *not* describe how you designed it).

The Automated Score Tracking (AST) Cornhole Game introduces wireless communication and modularity to the game of cornhole. AST eliminates the responsibility for teams to manually keep track of their score from round to round. Users easily connect to other AST game boards by pairing the devices via bluetooth. The game begins once the boards pair. Players toss their bean bags, with points being awarded as per usual with one point being given to bags landing on the board and three points being given to bags that land through the hole. AST detects all possible outcomes for a bean bag toss including a "sink" where a bag lands in the hole located on the board, a "hit" where a bag lands and stays on the board, or a "miss" where the bag misses the board completely. This system reduces scoring errors made by participants by continually tracking the progress of the game, allowing users to make changes to the scores as necessary. This system makes the game of cornhole simpler and more enjoyable.

2. Primary Constraints

Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors, or other issues that impacted your approach?

AST needs to detect all possible bean bag toss outcomes. One outcome that strikes a challenge to the implementation of the project is the "miss". If a player's bean bag does not land on the board, there needs to be a way to detect that outcome and factor it into the score for each round of play. For the system to properly process a "miss" we have considered a few different types of sensors for implementation. Another anticipated challenge is a case if the bean bags change position after they have been accounted for.

3. Economic

• What economic impacts result? Consider:

Human Capital – What people do.

Automating the score for cornhole allows people to have more social interaction during the game. Not having to worry about keeping track of a score makes people's lives easier, which makes the game more enjoyable.

Financial Capital – Monetary instruments.

Based on the cost estimates provided above the only financial capital needed for the completion of the AST project is the $200 stipend provided for each team member by the Cal Poly Electrical Engineering Department.

Manufactured or Real Capital – Made by people and their tools.

The AST project requires the CYBT-213043-Mesh evaluation module manufactured by Cypress Semiconductor to implement mesh functionality and allow the user to get real time score data on their mobile device. Each AST enabled system will consist of four mesh evaluation modules.

Natural Capital – The Earth's resources and bio-capacity.

The AST system requires two standard cornhole boards which are typically made from pine wood. The dimensions of the boards are listed below to show the amount of wood required to make the boards.

- (2) 24" x 48" pieces of 1/2" plywood for the surface
- (4) 2x4 x 48" for the frame
- (4) 2x4 x 21" for the frame

• When and where do costs and benefits accrue throughout the project's lifecycle?

The majority of costs will accrue during the research and development phase of the project in which external components will need to be purchased in order to successfully implement the AST system. The benefits will begin occurring once the system is completed and users are able to play the automated cornhole game.

• What inputs does the project require? How much does the project cost? Who pays?

The project inputs are as follows:

- CYBT-213043-Mesh evaluation kit (x4) provided by Cypress Semiconductor
- Cornhole Board(s) (x2) provided by Cypress Semiconductor
- Active RFID reader paid for using student stipend provided by Cal Poly EE department

- Passive RFID tags for bean bags paid for using student stipend provided by Cal Poly EE department
- Project Enclosure paid for using student stipend provided by Cal Poly EE department
- Extra External Parts also to be paid for using funds provided by Cal Poly EE department

Original estimated cost of component parts (as of the start of your project).

Cypress Semiconductor donated the cornhole game boards as well as the EZ-BT Mesh Evaluation boards. The additional components estimated cost is $42.79.

Actual final cost of component parts (at the end of your project)

Attach a final bill of materials for all components.

Additional equipment costs (any equipment needed for development?)

The only external equipment that we will need throughout the development of the AST corn hole game is a set of test board to use while testing different implementations of sensors to find the best methods.

• How much does the project earn? Who profits?

Estimates show the project will earn around $1,105,650 per year. These profits will go to Cypress Semiconductor while the users of the system will profit in entertainment. Cypress will also benefit from the marketing that this project will bring to their BLE Mesh enabled products. The company will profit monetary wise, while the users profit in entertainment.

• Timing

When do products emerge? How long do products exist? What maintenance or operation costs exist?

The American Cornhole League was founded in 2015, since then the popularity of the game of cornhole has increased. Now, the game has national tournaments and entertains outdoor gatherings. The beginning of summer is a popular time for people to purchase outdoor games which is a good time for the products release.The product will retain its quality and functionality if treated with care. The external battery life is the only maintenance needed after purchase.

Original estimated development time (as of the start of your project), as Gantt or Pert chart

The estimated total development time for the project is 360 man-hours.

Actual development time (at the end of your project), as Gantt or Pert chart

What happens after the project ends?

4. If manufactured on a commercial basis:

• Estimated number of devices sold per year

To get an estimate for the number of devices that would be sold a year we first have to look at how many sets of the base game are sold a year, estimates for this are within 150k-200k games per year. We then need to consider that we would be taking a small portion of this market, estimated at around 15%, which would give us an estimated number of devices sold per year of 22.5k-30k.

• Estimated manufacturing cost for each device

For the development of the board, we will need to use two of the Cypress CYBT-213014-MESH EZ-BT Module Mesh Evaluation boards. We are able to purchase four of these boards for $119.99 ($29.99 per board). Then for the rest of the electronics, we would estimate about an extra $20 for the sets. Finally an extra $18 for the wood needed to construct the boards themselves. In addition to the game boards we will also need to produce compatible tossing baggies so that they can interact properly with the board sensors, we estimate that the baggies will cost around $1.30-$2 a piece ($10.40-$16 for a complete set) to produce depending on what sensors need to be added. In the future, we would expect this price to go down as we negotiate prices of components in bulk. So before labor, the cost of manufacturing for each set of boards would be $113.98.

• Estimated purchase price for each device

After doing some competitive analysis we can see that a standard pre-assembled cornhole set is up for purchase at around $100-$120 and then sets with artwork range for about $200-$300. After researching a large toy company's profit margins, we see that the profit margins they have aimed for in recent years is 43%. we have to take into consideration the cost of labor, packaging, and distribution as well. So if we assume a standard distribution cost of $5 per item, a standard packaging cost of $5, and $30 an hour for a total of 5 hours to manufacture, as well as including the total cost to manufacture the set of boards. When we do the profit margin analysis, the total purchase price for the AST integrated cornhole game is $391.79. Please see math below:

$$(\$113.98+\$5+\$5+(\$30x5hr))(1.43) = \$319.79$$

• Estimated profit per year

The Revenue per year for AST integrated cornhole games would be
$26,250$ *units sold per year* $\times$ 319.79 *dollars per unit* = $ per year

The profit per year for AST integrated cornhole games would be
8,394,487.5 - (26,250 $\times$ (113.98 + 5 + 5 + (30*x*5))) = $1,202512.5 per year

• Estimated cost for user to operate device, per unit time (specify time interval)

After the initial purchase of the AST integrated cornhole sets the user will need to maintain the external battery life of the system which will cost around $3.80. The CR2032 battery will last up to 53 hours

when there is a continuous current of 3.1mA. On average, a game of cornhole lasts about an hour, so the users will need to change the battery about every 53 cycles.

5. Environmental

• Describe any environmental impacts associated with manufacturing or use, explain where they occur and quantify.

The batteries used for powering the system contain Lithium. The process to obtain Lithium impacts local towns by destroying land and depleting water. One way to limit this impact is to use rechargeable batteries. Another environmental impact is the use of wood to manufacture the cornhole game boards, which contributes to deforestation. The electrical components used for the system contribute to E-waste, one way to avoid this harmful impact is to use RoHS compliant components. RoHS compliant components restriction of the Use of certain Hazardous Substances in Electrical and Electronic Equipment.

• Which natural resources and ecosystem services does the project use directly and indirectly?

The natural resources the project uses are lithium and wood.

• Which natural resources and ecosystem services does the project improve or harm?

The project harms the forests through the use of wood to manufacture the game boards. The lithium mining affects humans that live in the local area due to the removal of water from land.

• How does the project impact other species?

6. Manufacturability

Describe any issues or challenges associated with manufacturing.

Manufacturing the devices is fairly straight forward. Once the BLE-mesh boards are flashed with the game code all that is needed to be done is for each device to be properly wired and attached to the physical cornhole boards. The baggies will also need to be sewn together with the needed sensors.

7. Sustainability

• Describe any issues or challenges associated with maintaining the completed device, or system.

The most important aspect of device maintenance will be changing the battery. Because the devices will have both dust and water resistance they should be fine in most conditions. But the user will be advised to store the boards indoors when they are not in use.

• Describe how the project impacts the sustainable use of resources.

The cornhole boards used in this project are made from wood while the rest of the materials associated with the external circuitry consist of various plastics, silicon, rubber, and metal. The wood used for the cornhole boards represent the largest environmental impact due to the fact that trees must be cut down to harvest the wood.

• Describe any upgrades that would improve the design of the project.

One potential upgrade to the system design would be to use only recycled wood for the conrhole boards. This would reduce the impact on the environment because instead of cutting down more trees for the conrhole boards, we could use recycled materials.

• Describe any issues or challenges associated with upgrading the design.

Using recycled wood would require the team to find a cheap and reliable source of recycled wood. This would also require us to build our own cornhole boards out of recycled materials which would add more time to the overall project as well as more cost for tools to build the boards.

8. Ethical

Describe ethical implications relating to the design, manufacture, use, or misuse of the project. Analyze using one or more ethical frameworks in addition to the IEEE Code of Ethics.

Characterization of system accuracy is a big concern in regards to the development of this project. Ideally the system would be able to report accurate scores 100% of the time but this may not be achievable. Much testing will be done in order to honestly and realistically state the scoring accuracy of the system to be in line with the third tenet of the IEEE Code of Ethics, “to be honest and realistic in stating claims or estimates based on available data.

9. Health and Safety

Describe any health and safety concerns associated with design, manufacture or use of the project.

Cornhole is typically played outside during fair weather days. Due to this, users should be aware of the health hazards associated with playing outside including things like exposure to the sun, air purity levels, as well as physical harm that can arise due to physical activity. Users who cannot play a normal game of cornhole without risking physical harm are not recommended to play with the AST version of cornhole.

10. Social and Political

• Describe social and political issues associated with design, manufacture, and use.

The AST system is considered a “luxury” item, meaning that one does not need to have it but having the system will make the game more enjoyable. The issue that comes across is the price of the product,

not everyone can afford it. The affordability of the product strikes a social issue to those in a lower economic class.

• Who does the project impact? Who are the direct and indirect stakeholders?

This project will impact those who use the system which will most likely be future Cal Poly electrical engineering students who will play with the cornhole boards. The boards used in this project have the Cypress Semiconductor logo on them which will lead to Cypress receiving more publicity amongst Cal Poly electrical engineering students who use the cornhole boards.

• How does the project benefit or harm various stakeholders?

The project will benefit the users of the system because they will be able to reduce stress and relax as they play cornhole without having to worry about the score. Cypress Semiconductor will benefit from the free marketing that the boards will generate for the company. The project may inspire other electrical engineering students to apply to jobs at Cypress due to the project.

• To what extent do stakeholders benefit equally? Pay equally? Does the project create any inequities?

All stakeholders benefit equally. Those who use the system to play cornhole will benefit from the physical activity and stress reduction due to playing the game. Cypress Semiconductor will benefit from the free publicity that the boards will generate for them.

• Consider various stakeholders' locations, communities, access to resources, economic power, knowledge, skills, and political power.

11. Development

Describe any new tools or techniques, used for either development or analysis that you learned independently during the course of your project.

Modus Toolbox will be used to develop the software that will implement the automatic score tracking functionality of the project. The project will also make use of RFID technology to identify who to give points to. Bluetooth Mesh will be utilized and experimented with during the development of this project.

Appendix B. Sensor Server Code

```
/*Finalized Sensor Server Code for Automated Cornhole
 * Copyright 2020, Cypress Semiconductor Corporation or a subsidiary of
 * Cypress Semiconductor Corporation. All Rights Reserved.
 *
 * This software, including source code, documentation and related
 * materials ("Software"), is owned by Cypress Semiconductor Corporation
 * or one of its subsidiaries ("Cypress") and is protected by and subject to
 * worldwide patent protection (United States and foreign),
 * United States copyright laws and international treaty provisions.
 * Therefore, you may use this Software only as provided in the license
 * agreement accompanying the software package from which you
 * obtained this Software ("EULA").
 * If no EULA applies, Cypress hereby grants you a personal, non-exclusive,
 * non-transferable license to copy, modify, and compile the Software
 * source code solely for use in connection with Cypress's
 * integrated circuit products. Any reproduction, modification, translation,
 * compilation, or representation of this Software except as specified
 * above is prohibited without the express written permission of Cypress.
 *
 * Disclaimer: THIS SOFTWARE IS PROVIDED AS-IS, WITH NO WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, NONINFRINGEMENT, IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress
 * reserves the right to make changes to the Software without notice. Cypress
 * does not assume any liability arising out of the application or use of the
 * Software or any product or circuit described in the Software. Cypress does
 * not authorize its products for use in any products where a malfunction or
 * failure of the Cypress product may reasonably be expected to result in
 * significant property damage, injury or death ("High Risk Product"). By
 * including Cypress's product in a High Risk Product, the manufacturer
 * of such system or application assumes all risk of such use and in doing
 * so agrees to indemnify Cypress against all liability.
 */

/** @file
 *
 *
 * This file shows how to create a device which publishes user_property level.
 */

#include "sparcommon.h"
#include "wiced_bt_dev.h"

#include "wiced_hal_gpio.h"

#include "wiced_platform.h"
#include "wiced_bt_stack.h"
#include "GeneratedSource/cycfg_pins.h"


#include "wiced_bt_uuid.h"
#include "wiced_bt_ble.h"
#include "wiced_bt_gatt.h"
#include "wiced_bt_mesh_models.h"
#include "wiced_bt_trace.h"
#include "wiced_timer.h"
#include "wiced_bt_mesh_app.h"

#ifdef HCI_CONTROL
#include "wiced_transport.h"
#include "hci_control_api.h"
#endif

#include "wiced_bt_cfg.h"
extern wiced_bt_cfg_settings_t wiced_bt_cfg_settings;

/*****************************************************
 *          Constants
 *****************************************************/
#define MESH_PID                0x3122
#define MESH_VID                0x0002
#define MESH_CACHE_REPLAY_SIZE  0x0008

//Change 1 and 2 are below
#define WICED_BT_MESH_PROPERTY_RFID_VALUE                           WICED_BT_MESH_PROPERTY_PRESENCE_DETECTED
        //define RFID Property Value using predefined presence detected property
```

```
#define WICED_BT_MESH_PROPERTY_LEN_PRESENT_RFID_TAGS        1//in terms of bytes

//Here I had to change the tolerance to unspeified becasue Ideally each bean bag will be seen as whole
numbers
//Also note here that temperature should actually say RFID
#define MESH_TEMPERATURE_SENSOR_NEGATIVE_TOLERANCE          WICED_BT_MESH_SENSOR_TOLERANCE_UNSPECIFIED
#define MESH_TEMPERATURE_SENSOR_POSITIVE_TOLERANCE          WICED_BT_MESH_SENSOR_TOLERANCE_UNSPECIFIED
#define MESH_TEMPERATURE_SENSOR_SAMPLING_FUNCTION       WICED_BT_MESH_SENSOR_SAMPLING_FUNCTION_UNKNOWN
#define MESH_TEMPERATURE_SENSOR_MEASUREMENT_PERIOD      WICED_BT_MESH_SENSOR_VAL_UNKNOWN
#define MESH_TEMPERATURE_SENSOR_UPDATE_INTERVAL         WICED_BT_MESH_SENSOR_VAL_UNKNOWN

/******************************************************
 *          Structures
 ******************************************************/

/******************************************************
 *          Function Prototypes
 ******************************************************/
static void mesh_app_init(wiced_bool_t is_provisioned);
static wiced_bool_t mesh_app_notify_period_set(uint8_t element_idx, uint16_t company_id, uint16_t model_id,
uint32_t period);
static uint32_t mesh_app_proc_rx_cmd(uint16_t opcode, uint8_t *p_data, uint32_t length);
static void mesh_sensor_server_restart_timer(wiced_bt_mesh_core_config_sensor_t *p_sensor);
static void mesh_sensor_server_report_handler(uint16_t event, uint8_t element_idx, void *p_get, void
*p_ref_data);
static void mesh_sensor_server_config_change_handler(uint8_t element_idx, uint16_t event, uint16_t
property_id, uint16_t setting_prop_id);
static void mesh_sensor_server_status_changed(uint8_t element_idx, uint8_t *p_data, uint32_t length);
static void mesh_sensor_server_send_column_status(wiced_bt_mesh_event_t *p_event,
wiced_bt_mesh_sensor_column_get_data_t *p_get_column);
static void mesh_sensor_server_send_series_status(wiced_bt_mesh_event_t *p_event,
wiced_bt_mesh_sensor_series_get_data_t* data);
static void mesh_sensor_server_process_cadence_changed(uint8_t element_idx, uint16_t property_id);
static void mesh_sensor_server_process_setting_changed(uint8_t element_idx, uint16_t property_id, uint16_t
setting_property_id);
static void mesh_sensor_publish_timer_callback(TIMER_PARAM_TYPE arg);

#ifdef HCI_CONTROL
static void mesh_sensor_hci_event_send_cadence_set(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_cadence_set_data_t *p_set);
static void mesh_sensor_hci_event_send_setting_set(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_setting_set_data_t *p_set);
#endif

//GPIO initialization functions 4/29/2020
static void hal_gpio_app_test_output(void);
static void hal_gpio_app_test_input(void);
void hal_gpio_app_interrrupt_handler(void *data, uint8_t port_pin);

int8_t RFID_data = 0;

int8_t a; //These variables store the state of the selected GPIO pins
int8_t b; //Data from Arduino is stored with these variables
int8_t c;
int8_t d;
int8_t e;
/******************************************************
 *          Variables Definitions
 ******************************************************/
uint8_t mesh_mfr_name[WICED_BT_MESH_PROPERTY_LEN_DEVICE_MANUFACTURER_NAME] = { 'C', 'y', 'p', 'r', 'e', 's',
's', 0 };
uint8_t mesh_model_num[WICED_BT_MESH_PROPERTY_LEN_DEVICE_MODEL_NUMBER]     = { '1', '2', '3', '4', 0, 0, 0,
0 };
uint8_t mesh_system_id[8]                                                  = { 0xbb, 0xb8, 0xa1, 0x80, 0x5f,
0x9f, 0x91, 0x71 };

int8_t             mesh_sensor_RFID_current_score = 0;      //start out with score difference of 0 between
the two teams

int8_t        mesh_sensor_sent_value = 0;           //
uint32_t      mesh_sensor_sent_time;                // time stamp when temperature was published
uint32_t      mesh_sensor_publish_period = 0;       // publish period in msec
uint32_t      mesh_sensor_fast_publish_period = 0;  // publish period in msec when values are outside of
limit
wiced_timer_t mesh_sensor_cadence_timer;
```

```
// Optional setting for the temperature sensor, the Total Device Runtime, in Time Hour 24 format
uint8_t mesh_temperature_sensor_setting0_val[] = { 0x01, 0x00, 0x00 };

// first set of series and column values
uint8_t raw_valuex_0[]                          = { 0x00, 0x01 };
uint8_t column_width0[]                         = { 0x01, 0x10 };
uint8_t raw_valuey0[]                           = { 0x00, 0x10 };

// second set of series and column values
uint8_t raw_valuex_1[]                          = { 0x00, 0x10 };
uint8_t column_width1[]                         = { 0x01, 0x10 };
uint8_t raw_valuey1[]                           = { 0x00, 0x20 };

wiced_bt_mesh_core_config_model_t mesh_element1_models[] =
{
    WICED_BT_MESH_DEVICE,
    WICED_BT_MESH_MODEL_SENSOR_SERVER,
};
#define MESH_APP_NUM_MODELS  (sizeof(mesh_element1_models) / sizeof(wiced_bt_mesh_core_config_model_t))

wiced_bt_mesh_sensor_config_setting_t sensor_settings[] =
{
    {
        .setting_property_id = WICED_BT_MESH_PROPERTY_TOTAL_DEVICE_RUNTIME,
        .access              = WICED_BT_MESH_SENSOR_SETTING_READABLE_AND_WRITABLE,
        .value_len           = 3,
        .val                 = mesh_temperature_sensor_setting0_val
    },
};

wiced_bt_mesh_sensor_config_column_data_t   mesh_temperature_sensor_columns[] =
{
    {
        .raw_valuex = raw_valuex_0,
        .column_width = column_width0,
        .raw_valuey = raw_valuey0,
    },
    {
        .raw_valuex = raw_valuex_1,
        .column_width = column_width1,
        .raw_valuey = raw_valuey1
    },
};

wiced_bt_mesh_core_config_sensor_t mesh_element1_sensors[] =
{
    {

         .property_id = WICED_BT_MESH_PROPERTY_RFID_VALUE, //Set to our defined RFID property value
                 .prop_value_len = WICED_BT_MESH_PROPERTY_LEN_PRESENT_RFID_TAGS,

        .descriptor =
        {
            .positive_tolerance = MESH_TEMPERATURE_SENSOR_POSITIVE_TOLERANCE, //all of these have been
changed to correpsond to RFID stuff and is shown starting on line 75
            .negative_tolerance = MESH_TEMPERATURE_SENSOR_NEGATIVE_TOLERANCE,
            .sampling_function  = MESH_TEMPERATURE_SENSOR_SAMPLING_FUNCTION,
            .measurement_period = MESH_TEMPERATURE_SENSOR_MEASUREMENT_PERIOD,
            .update_interval    = MESH_TEMPERATURE_SENSOR_UPDATE_INTERVAL,
        },
                .data = (uint8_t *)&mesh_sensor_RFID_current_score, //our current score starts out at 0
        .cadence =
        {
            // Value 1 indicates that cadence does not change depending on the measurements
            .fast_cadence_period_divisor = 1,            // Value of the divisor
            .trigger_type_percentage     = WICED_FALSE,
            .trigger_delta_down          = 0,
            .trigger_delta_up            = 0,
            .min_interval                = (1 << 0x0C),  // ~4 seconds
            .fast_cadence_low            = 0,
            .fast_cadence_high           = 0,
        },

                .num_series     = 0,
                .series_columns = NULL,
                .num_settings   = 0,
                .settings       = NULL,
```

```
    },
};


#define MESH_APP_NUM_PROPERTIES (sizeof(mesh_element1_properties) /
sizeof(wiced_bt_mesh_core_config_property_t))

#define MESH_SENSOR_SERVER_ELEMENT_INDEX   0
#define MESH_RFID_SENSOR_INDEX      0 //I replacedd all instances of MESH_TEMP_SENSOR_INDEX with this


wiced_bt_mesh_core_config_element_t mesh_elements[] =
{
    {
        .location = MESH_ELEM_LOC_MAIN,                                     // location description as defined
in the GATT Bluetooth Namespace Descriptors section of the Bluetooth SIG Assigned Numbers
        .default_transition_time = MESH_DEFAULT_TRANSITION_TIME_IN_MS,   // Default transition time for
models of the element in milliseconds
        .onpowerup_state = WICED_BT_MESH_ON_POWER_UP_STATE_RESTORE,      // Default element behavior on
power up
        .default_level = 0,                                                 // Default value of the variable
controlled on this element (for example power, lightness, temperature, hue...)
        .range_min = 1,                                                     // Minimum value of the variable
controlled on this element (for example power, lightness, temperature, hue...)
                .range_max = 0xffff,                                                // Maximum value of the
variable controlled on this element (for example power, lightness, temperature, hue...)
                .move_rollover = 0,                                                  // If true when level gets
to range_max during move operation, it switches to min, otherwise move stops.
        .properties_num = 0,                                                // Number of properties in the
array models
        .properties = NULL,                                                 // Array of properties in the
element.
        .sensors_num = 1,                                                   // Number of properties in the
array models
        .sensors = mesh_element1_sensors,                                   // Array of properties in the
element.
        .models_num = MESH_APP_NUM_MODELS,                                  // Number of models in the array
models
        .models = mesh_element1_models,                                     // Array of models located in that
element. Model data is defined by structure wiced_bt_mesh_core_config_model_t
    },
};

wiced_bt_mesh_core_config_t  mesh_config =
{
    .company_id         = MESH_COMPANY_ID_CYPRESS,                  // Company identifier assigned by the
Bluetooth SIG
    .product_id         = MESH_PID,                                 // Vendor-assigned product identifier
    .vendor_id          = MESH_VID,                                 // Vendor-assigned product version
identifier
    .replay_cache_size  = MESH_CACHE_REPLAY_SIZE,                   // Number of replay protection entries,
i.e. maximum number of mesh devices that can send application messages to this device.
#if defined(LOW_POWER_NODE) && (LOW_POWER_NODE == 1)
    .features           = WICED_BT_MESH_CORE_FEATURE_BIT_LOW_POWER, // A bit field indicating the device
features. In Low Power mode no Relay, no Proxy and no Friend
    .friend_cfg         =                                           // Empty Configuration of the Friend
Feature
    {
        .receive_window = 0,                                        // Receive Window value in milliseconds
supported by the Friend node.
        .cache_buf_len  = 0,                                        // Length of the buffer for the cache
        .max_lpn_num    = 0                                         // Max number of Low Power Nodes with
established friendship. Must be > 0 if Friend feature is supported.
    },
    .low_power          =                                           // Configuration of the Low Power
Feature
    {
        .rssi_factor           = 2,                                 // contribution of the RSSI measured by
the Friend node used in Friend Offer Delay calculations.
        .receive_window_factor = 2,                                 // contribution of the supported Receive
Window used in Friend Offer Delay calculations.
        .min_cache_size_log    = 3,                                 // minimum number of messages that the
Friend node can store in its Friend Cache.
        .receive_delay         = 100,                               // Receive delay in 1 ms units to be
requested by the Low Power node.
        .poll_timeout          = 36000                              // Poll timeout in 100ms units to be
```

```
requested by the Low Power node.
    },
#else
    .features           = WICED_BT_MESH_CORE_FEATURE_BIT_FRIEND | WICED_BT_MESH_CORE_FEATURE_BIT_RELAY |
WICED_BT_MESH_CORE_FEATURE_BIT_GATT_PROXY_SERVER,   // In Friend mode support friend, relay
    .friend_cfg         =                                       // Configuration of the Friend
Feature(Receive Window in Ms, messages cache)
    {
        .receive_window        = 20,
        .cache_buf_len         = 300,                           // Length of the buffer for the cache
        .max_lpn_num           = 4                              // Max number of Low Power Nodes with
established friendship. Must be > 0 if Friend feature is supported.
    },
    .low_power          =                                       // Configuration of the Low Power
Feature
    {
        .rssi_factor           = 0,                             // contribution of the RSSI measured by
the Friend node used in Friend Offer Delay calculations.
        .receive_window_factor = 0,                             // contribution of the supported Receive
Window used in Friend Offer Delay calculations.
        .min_cache_size_log    = 0,                             // minimum number of messages that the
Friend node can store in its Friend Cache.
        .receive_delay         = 0,                             // Receive delay in 1 ms units to be
requested by the Low Power node.
        .poll_timeout          = 0                              // Poll timeout in 100ms units to be
requested by the Low Power node.
    },
#endif
    .gatt_client_only          = WICED_FALSE,                   // Can connect to mesh over GATT or ADV
    .elements_num  = (uint8_t)(sizeof(mesh_elements) / sizeof(mesh_elements[0])),   // number of elements on
this device
    .elements      = mesh_elements                              // Array of elements for this device
};

/*
 * Mesh application library will call into application functions if provided by the application.
 */
wiced_bt_mesh_app_func_table_t wiced_bt_mesh_app_func_table =
{
    mesh_app_init,              // application initialization
        hal_gpio_app_test_input,    // Button processing                    //Tells provisioner to use this
function to set up the button
    NULL,                       // GATT connection status
    NULL,                       // attention processing
    mesh_app_notify_period_set, // notify period set
        NULL,                                           // WICED HCI command
    NULL,                         // LPN sleep
    NULL                        // factory reset
};

/******************************************************
 *               Function Definitions
 ******************************************************/
void mesh_app_init(wiced_bool_t is_provisioned)
{
#if 0
    // Set Debug trace level for mesh_models_lib and mesh_provisioner_lib
    wiced_bt_mesh_models_set_trace_level(WICED_BT_MESH_CORE_TRACE_INFO);
#endif
#if 0
    // Set Debug trace level for all modules but Info level for CORE_AES_CCM module
    wiced_bt_mesh_core_set_trace_level(WICED_BT_MESH_CORE_TRACE_FID_ALL, WICED_BT_MESH_CORE_TRACE_DEBUG);
    wiced_bt_mesh_core_set_trace_level(WICED_BT_MESH_CORE_TRACE_FID_CORE_AES_CCM,
WICED_BT_MESH_CORE_TRACE_INFO);
#endif

    wiced_bt_cfg_settings.device_name = (uint8_t *)"RFID Sensor";
    wiced_bt_cfg_settings.gatt_cfg.appearance = APPEARANCE_GENERIC_TAG;
    // Adv Data is fixed. Spec allows to put URI, Name, Appearance and Tx Power in the Scan Response Data.
    if (!is_provisioned)
    {
        wiced_bt_ble_advert_elem_t  adv_elem[3];
        uint8_t                     buf[2];
        uint8_t                     num_elem = 0;
        adv_elem[num_elem].advert_type = BTM_BLE_ADVERT_TYPE_NAME_COMPLETE;
        adv_elem[num_elem].len = (uint16_t)strlen((const char*)wiced_bt_cfg_settings.device_name);
        adv_elem[num_elem].p_data = wiced_bt_cfg_settings.device_name;
```

```
        num_elem++;

        adv_elem[num_elem].advert_type = BTM_BLE_ADVERT_TYPE_APPEARANCE;
        adv_elem[num_elem].len = 2;
        buf[0] = (uint8_t)wiced_bt_cfg_settings.gatt_cfg.appearance;
        buf[1] = (uint8_t)(wiced_bt_cfg_settings.gatt_cfg.appearance >> 8);
        adv_elem[num_elem].p_data = buf;
        num_elem++;

        wiced_bt_mesh_set_raw_scan_response_data(num_elem, adv_elem);
    }

    if (!is_provisioned)
        return;


    hal_gpio_app_test_output();//These output pins are only used for setting LEDs for testing purposes



    // initialize the cadence timer.  Need a timer for each element because each sensor model can be
    // configured for different publication period.  This app has only one sensor.
    wiced_init_timer(&mesh_sensor_cadence_timer, &mesh_sensor_publish_timer_callback,
(TIMER_PARAM_TYPE)&mesh_config.elements[MESH_SENSOR_SERVER_ELEMENT_INDEX].sensors[MESH_RFID_SENSOR_INDEX ],
WICED_MILLI_SECONDS_TIMER);

    wiced_bt_mesh_model_sensor_server_init(MESH_SENSOR_SERVER_ELEMENT_INDEX,
mesh_sensor_server_report_handler, mesh_sensor_server_config_change_handler, is_provisioned);
}

/*
 * New publication period is set.
 * The period may need to be adjusted based on the divisor.
 */
wiced_bool_t mesh_app_notify_period_set(uint8_t element_idx, uint16_t company_id, uint16_t model_id,
uint32_t period)
{
    if ((element_idx != MESH_RFID_SENSOR_INDEX ) || (company_id != MESH_COMPANY_ID_BT_SIG) || (model_id !=
WICED_BT_MESH_CORE_MODEL_ID_SENSOR_SRV))
    {
        return WICED_FALSE;
    }
    mesh_sensor_publish_period = period;
    WICED_BT_TRACE("Sensor data send period:%dms\n", mesh_sensor_publish_period);
    mesh_sensor_server_restart_timer(&mesh_config.elements[element_idx].sensors[MESH_RFID_SENSOR_INDEX ]);
    return WICED_TRUE;
}

void set_pin_output(void)
{
        wiced_hal_gpio_set_pin_output(WICED_P08, GPIO_PIN_OUTPUT_HIGH); //this is the bit to tell the
arduino we are ready for data
}

void disable_pin_output(void)
{
        wiced_hal_gpio_set_pin_output(WICED_P08, GPIO_PIN_OUTPUT_LOW); //disable this bit after we've
received data
}

void  hal_gpio_app_test_output(void) //we want to call this function at the top inside app_init
{
    uint8_t index = 0;


    wiced_hal_gpio_configure_pin(WICED_P08, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);


}

//Placing all GPIO functions here...

void hal_gpio_app_test_input(void)
{
        uint8_t index = 0;
        //initializes pins 1-5 as RFID data score input pins
    wiced_hal_gpio_configure_pin(WICED_P01, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);//LSB RFID data bit
```

```
    wiced_hal_gpio_configure_pin(WICED_P02, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P03, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P04, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);//MSB RFID data bit
    wiced_hal_gpio_configure_pin(WICED_P05, GPIO_INPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);//will serve as Team bit
    //automatically maps to correct pins for any kit

wiced_hal_gpio_configure_pin(WICED_GPIO_PIN_BUTTON,(GPIO_INPUT_ENABLE|GPIO_PULL_DOWN|GPIO_EN_INT_RISING_EDGE
),GPIO_PIN_OUTPUT_LOW);
    //initialized SW3 user defined switch as interrupt
    wiced_hal_gpio_register_pin_for_interrupt(WICED_GPIO_PIN_BUTTON, hal_gpio_app_interrrupt_handler, NULL);
}


void set_RFID_data(int8_t v, int8_t w, int8_t x, int8_t y, int8_t z) //Based on data received from Arduino,
this function sets RFID data
 {
 if(v == 1)
  {
                RFID_data+=1;
        }
        else
        {
                RFID_data+=0;
        }
  if(w == 1)
  {
                RFID_data+=2;
        }
        else
        {
                RFID_data+=0;
        }

  if(x == 1)
  {
                RFID_data+=4;
        }
        else
        {
                RFID_data+=0;
 }

        if(y == 1)
  {
                RFID_data+=8;
        }
        else
        {
                RFID_data+=0;
        }

  if(z == 1)
  {
                RFID_data+=16;
        }
        else
        {
                RFID_data+=0;
        }
  }


void hal_gpio_app_interrrupt_handler(void *data, uint8_t pin) //Jump to this function once the button is
pushed
{

        a = wiced_hal_gpio_get_pin_input_status(WICED_P01); //These variables store the current state on the
GPIO pins
        b = wiced_hal_gpio_get_pin_input_status(WICED_P02);
        c = wiced_hal_gpio_get_pin_input_status(WICED_P03);
        d = wiced_hal_gpio_get_pin_input_status(WICED_P04);
        e = wiced_hal_gpio_get_pin_input_status(WICED_P05);
        set_RFID_data(a, b, c, d, e);

        mesh_sensor_RFID_current_score = RFID_data;
        wiced_bt_mesh_model_sensor_server_data(MESH_SENSOR_SERVER_ELEMENT_INDEX,
WICED_BT_MESH_PROPERTY_RFID_VALUE, NULL); //This function sends a status message to the mesh which contains
```

```
RFID data
}

/*
 * Start periodic timer depending on the publication period, fast cadence divisor and minimum interval
 */
void mesh_sensor_server_restart_timer(wiced_bt_mesh_core_config_sensor_t *p_sensor)
{
    // If there are no specific cadence settings, publish every publish period.
    uint32_t timeout = mesh_sensor_publish_period;

    wiced_stop_timer(&mesh_sensor_cadence_timer);
    if (mesh_sensor_publish_period == 0)
    {

        if ((p_sensor->cadence.min_interval != 0) &&
            ((p_sensor->cadence.trigger_delta_up != 0) || (p_sensor->cadence.trigger_delta_down != 0)))
        {
            timeout = p_sensor->cadence.min_interval;
        }
        else
        {
            WICED_BT_TRACE("sensor restart timer period:%d\n", mesh_sensor_publish_period);
            return;
        }
    }
    else
    {
        // If fast cadence period divisor is set, we need to check temperature more
        // often than publication period.  Publish if measurement is in specified range
        if (p_sensor->cadence.fast_cadence_period_divisor > 1)
        {
            mesh_sensor_fast_publish_period = mesh_sensor_publish_period /
p_sensor->cadence.fast_cadence_period_divisor;
            timeout = mesh_sensor_fast_publish_period;
        }
        else
        {
            mesh_sensor_fast_publish_period = 0;
        }
        // The thermistor is not interrupt driven.  If client configured sensor to send notification when
        // the value changes, we may need to check value more often not to miss the trigger.
        // The cadence.min_interval can be used because we do not need to send data more often than that.
        if ((p_sensor->cadence.min_interval < timeout) &&
            ((p_sensor->cadence.trigger_delta_up != 0) || (p_sensor->cadence.trigger_delta_down != 0)))
        {
            timeout = p_sensor->cadence.min_interval;
        }
    }
    WICED_BT_TRACE("sensor restart timer:%d\n", timeout);
    wiced_start_timer(&mesh_sensor_cadence_timer, timeout);
}

/*
 * Process the configuration changes set by the Sensor Client.
 */
void mesh_sensor_server_config_change_handler(uint8_t element_idx, uint16_t event, uint16_t property_id,
uint16_t setting_property_id)
{
#if defined HCI_CONTROL
    wiced_bt_mesh_hci_event_t *p_hci_event;
#endif
    WICED_BT_TRACE("mesh_sensor_server_config_change_handler msg: %d\n", event);

    switch (event)
    {

    case WICED_BT_MESH_SENSOR_CADENCE_SET:
#if defined HCI_CONTROL
//        if ((p_hci_event = wiced_bt_mesh_create_hci_event(p_event)) != NULL)
//            mesh_sensor_hci_event_send_cadence_set(p_hci_event, (wiced_bt_mesh_sensor_cadence_set_data_t
*)p_data);
#endif
        mesh_sensor_server_process_cadence_changed(element_idx, property_id);
        break;

    case WICED_BT_MESH_SENSOR_SETTING_SET:
```

```
#if defined HCI_CONTROL
//        if ((p_hci_event = wiced_bt_mesh_create_hci_event(p_event)) != NULL)
//            mesh_sensor_hci_event_send_setting_set(p_hci_event, (wiced_bt_mesh_sensor_setting_set_data_t
*)p_data);
#endif
        mesh_sensor_server_process_setting_changed(element_idx, property_id, setting_property_id);
        break;
    }
}

/*
 * Process get request from Sensor Client and respond with sensor data
 */
void mesh_sensor_server_report_handler(uint16_t event, uint8_t element_idx, void *p_get, void *p_ref_data)
{
         WICED_BT_TRACE("*****got here*****:\n");
    wiced_bt_mesh_sensor_get_t *p_sensor_get = (wiced_bt_mesh_sensor_get_t *)p_get;
    WICED_BT_TRACE("mesh_sensor_server_report_handler msg: %d\n", event);

    switch (event)
    {
    case WICED_BT_MESH_SENSOR_GET:
        // tell mesh models library that data is ready to be shipped out, the library will get data from
mesh_config
         //here we need to set an output bit to signal Arduino that we are ready for RFID data
         set_pin_output();
         mesh_sensor_RFID_current_score = RFID_data;//this is how we package the data
        wiced_bt_mesh_model_sensor_server_data(element_idx, p_sensor_get->property_id, p_ref_data);
        disable_pin_output();
        break;

    case WICED_BT_MESH_SENSOR_COLUMN_GET:
        mesh_sensor_server_send_column_status(wiced_bt_mesh_create_reply_event(p_ref_data),
(wiced_bt_mesh_sensor_column_get_data_t *)p_get);
        break;

    case WICED_BT_MESH_SENSOR_SERIES_GET:
        mesh_sensor_server_send_series_status(wiced_bt_mesh_create_reply_event(p_ref_data),
(wiced_bt_mesh_sensor_series_get_data_t *)p_get);
        break;

    default:
        WICED_BT_TRACE("unknown\n");
        break;
    }
}

/*
 * Process cadence change
 */
void mesh_sensor_server_process_cadence_changed(uint8_t element_idx, uint16_t property_id)
{
    wiced_bt_mesh_core_config_sensor_t *p_sensor;
    p_sensor = &mesh_config.elements[element_idx].sensors[MESH_RFID_SENSOR_INDEX ];

    WICED_BT_TRACE("cadence changed property id:%04x\n", property_id);
    WICED_BT_TRACE("Fast cadence period divisor:%d\n", p_sensor->cadence.fast_cadence_period_divisor);
    WICED_BT_TRACE("Is trigger type percent:%d\n", p_sensor->cadence.trigger_type_percentage);
    WICED_BT_TRACE("Trigger delta up:%d\n", p_sensor->cadence.trigger_delta_up);
    WICED_BT_TRACE("Trigger delta down:%d\n", p_sensor->cadence.trigger_delta_down);
    WICED_BT_TRACE("Min Interval:%d\n", p_sensor->cadence.min_interval);
    WICED_BT_TRACE("Fast cadence low:%d\n", p_sensor->cadence.fast_cadence_low);
    WICED_BT_TRACE("Fast cadence high:%d\n", p_sensor->cadence.fast_cadence_high);

    mesh_sensor_server_restart_timer(p_sensor);
}

/*
 * Publication timer callback.  Need to send data if publish period expired, or
 * if value has changed more than specified in the triggers, or if value is in range
 * of fast cadence values.
 */
void mesh_sensor_publish_timer_callback(TIMER_PARAM_TYPE arg)
{
//    wiced_bt_mesh_event_t *p_event;
//    wiced_bt_mesh_core_config_sensor_t *p_sensor = (wiced_bt_mesh_core_config_sensor_t *)arg;
//    wiced_bool_t pub_needed = WICED_FALSE;
```

```
//    uint32_t cur_time = wiced_bt_mesh_core_get_tick_count();
//
//    if ((cur_time - mesh_sensor_sent_time) < p_sensor->cadence.min_interval)
//    {
//        WICED_BT_TRACE("time since last pub:%d interval:%d\n", cur_time - mesh_sensor_sent_time,
p_sensor->cadence.min_interval);
//        wiced_start_timer(&mesh_sensor_cadence_timer, p_sensor->cadence.min_interval - cur_time +
mesh_sensor_sent_time);
//    }
//    else
//    {
//        // check if publication timer expired
//        if ((mesh_sensor_publish_period != 0) && (cur_time - mesh_sensor_sent_time >=
mesh_sensor_publish_period))
//        {
//            WICED_BT_TRACE("Pub needed period\n");
//            pub_needed = WICED_TRUE;
//        }
//        // still need to send if publication timer has not expired, but triggers are configured, and value
//        // changed too much
//        if (!pub_needed && ((p_sensor->cadence.trigger_delta_up != 0) ||
(p_sensor->cadence.trigger_delta_down != 0)))
//        {
//            if (!p_sensor->cadence.trigger_type_percentage)
//            {
//                WICED_BT_TRACE("Native cur value:%d sent:%d delta:%d/%d\n",
//                        mesh_sensor_RFID_current_score, mesh_sensor_sent_value,
p_sensor->cadence.trigger_delta_up, p_sensor->cadence.trigger_delta_down);
//
//                if (((p_sensor->cadence.trigger_delta_up != 0)   && (mesh_sensor_RFID_current_score >=
(mesh_sensor_sent_value + p_sensor->cadence.trigger_delta_up))) ||
//                    ((p_sensor->cadence.trigger_delta_down != 0) && (mesh_sensor_RFID_current_score <=
(mesh_sensor_sent_value - p_sensor->cadence.trigger_delta_down))))
//                {
//                    WICED_BT_TRACE("Pub needed native value\n");
//                    pub_needed = WICED_TRUE;
//                }
//            }
//            else
//            {
//                // need to calculate percentage of the increase or decrease.  The deltas are in 0.01%.
//                if ((p_sensor->cadence.trigger_delta_up != 0) && (mesh_sensor_RFID_current_score >
mesh_sensor_sent_value))
//                {
//                    WICED_BT_TRACE("Delta up:%d\n", ((uint32_t)(mesh_sensor_RFID_current_score -
mesh_sensor_sent_value) * 10000 / mesh_sensor_RFID_current_score));
//                    if (((uint32_t)(mesh_sensor_RFID_current_score - mesh_sensor_sent_value) * 10000 /
mesh_sensor_RFID_current_score) > p_sensor->cadence.trigger_delta_up)
//                    {
//                        WICED_BT_TRACE("Pub needed percent delta up:%d\n",
((mesh_sensor_RFID_current_score - mesh_sensor_sent_value) * 10000 / mesh_sensor_RFID_current_score));
//                        pub_needed = WICED_TRUE;
//                    }
//                }
//                else if ((p_sensor->cadence.trigger_delta_down != 0) && (mesh_sensor_RFID_current_score <
mesh_sensor_sent_value))
//                {
//                    WICED_BT_TRACE("Delta down:%d\n", ((uint32_t)(mesh_sensor_sent_value -
mesh_sensor_RFID_current_score) * 10000 / mesh_sensor_RFID_current_score));
//                    if (((uint32_t)(mesh_sensor_sent_value - mesh_sensor_RFID_current_score) * 10000 /
mesh_sensor_RFID_current_score) > p_sensor->cadence.trigger_delta_down)
//                    {
//                        WICED_BT_TRACE("Pub needed percent delta down:%d\n", ((mesh_sensor_sent_value -
mesh_sensor_RFID_current_score) * 10000 / mesh_sensor_RFID_current_score));
//                        pub_needed = WICED_TRUE;
//                    }
//                }
//            }
//        }
//        // may still need to send if fast publication is configured
//        if (!pub_needed && (mesh_sensor_fast_publish_period != 0))
//        {
//            // check if fast publish period expired
//            if (cur_time - mesh_sensor_sent_time >= mesh_sensor_fast_publish_period)
//            {
//                // if cadence high is more than cadence low, to publish, the value should be in range
//                if (p_sensor->cadence.fast_cadence_high >= p_sensor->cadence.fast_cadence_low)
```

```
//                {
//                    if ((mesh_sensor_RFID_current_score >= p_sensor->cadence.fast_cadence_low) &&
//                        (mesh_sensor_RFID_current_score <= p_sensor->cadence.fast_cadence_high))
//                    {
//                        WICED_BT_TRACE("Pub needed in range\n");
//                        pub_needed = WICED_TRUE;
//                    }
//                }
//                else if (p_sensor->cadence.fast_cadence_high < p_sensor->cadence.fast_cadence_low)
//                {
//                    if ((mesh_sensor_RFID_current_score > p_sensor->cadence.fast_cadence_low) ||
//                        (mesh_sensor_RFID_current_score < p_sensor->cadence.fast_cadence_high))
//                    {
//                        WICED_BT_TRACE("Pub needed out of range\n");
//                        pub_needed = WICED_TRUE;
//                    }
//                }
//            }
//        }
//        /*
//        if (!pub_needed)
//        {
//           if (((p_sensor->cadence.trigger_delta_up == 0) && (mesh_sensor_RFID_current_score >
mesh_sensor_sent_value)) ||
//               ((p_sensor->cadence.trigger_delta_down == 0) && (mesh_sensor_RFID_current_score <
mesh_sensor_sent_value)))
//            {
//               WICED_BT_TRACE("Pub needed new value no deltas\n");
//               pub_needed = WICED_TRUE;
//            }
//        }
//        */
//        if (pub_needed)
//        {
//            mesh_sensor_sent_value  = mesh_sensor_RFID_current_score;
//            mesh_sensor_sent_time   = cur_time;
//
//            WICED_BT_TRACE("Pub value:%d time:%d\n", mesh_sensor_sent_value, mesh_sensor_sent_time);
//            wiced_bt_mesh_model_sensor_server_data(MESH_SENSOR_SERVER_ELEMENT_INDEX,
WICED_BT_MESH_PROPERTY_PRESENT_AMBIENT_TEMPERATURE, NULL);
//        }
//        mesh_sensor_server_restart_timer(p_sensor);
//    }
}


/*
 * Send Sensor Series Status message to the Sensor Client
 */
void mesh_sensor_server_send_series_status(wiced_bt_mesh_event_t *p_event,
wiced_bt_mesh_sensor_series_get_data_t* data)
{
    uint8_t i,j,k=0;
    uint8_t element_idx = p_event->element_idx;
    wiced_bt_mesh_sensor_series_status_data_t series_data;
    uint8_t end_idx;
    wiced_bool_t copy_flag = WICED_FALSE;

    series_data.property_id = data->property_id;

    for (i = 0; i < mesh_config.elements[element_idx].sensors_num; i++)
    {
        if (mesh_config.elements[element_idx].sensors[i].property_id == data->property_id)
        {
            series_data.prop_value_len = mesh_config.elements[element_idx].sensors[i].prop_value_len;

            end_idx = (data->end_index > mesh_config.elements[element_idx].sensors[i].num_series) ?
                          mesh_config.elements[element_idx].sensors[i].num_series : data->end_index;
            for (j = data->start_index; j < end_idx; j++, k++)
            {
                memcpy(series_data.column_list[k].raw_valuex,
mesh_config.elements[element_idx].sensors[i].series_columns[j].raw_valuex,  series_data.prop_value_len);
                memcpy(series_data.column_list[k].raw_valuey,
mesh_config.elements[element_idx].sensors[i].series_columns[j].raw_valuey,  series_data.prop_value_len);
                memcpy(series_data.column_list[k].column_width,
mesh_config.elements[element_idx].sensors[i].series_columns[j].column_width,  series_data.prop_value_len);
            }
```

```
            series_data.no_of_columns = k;
            wiced_bt_mesh_model_sensor_server_series_status_send(p_event, &series_data);
            return;
        }
    }
}

/*
 * Send Sensor Column Status message to the Sensor Client
 */
void mesh_sensor_server_send_column_status(wiced_bt_mesh_event_t *p_event,
wiced_bt_mesh_sensor_column_get_data_t *p_get_column)
{
    wiced_bt_mesh_sensor_column_status_data_t column_status;
    uint8_t element_idx = p_event->element_idx;
    uint8_t num_sensor = mesh_config.elements[element_idx].sensors_num;
    uint16_t property_id;
    uint16_t prop_val_len;
    uint8_t i;
    uint8_t j;

    WICED_BT_TRACE("%s\n", __FUNCTION__);

    for (i = 0; i < num_sensor; i++)
    {
        if (mesh_config.elements[element_idx].sensors[i].property_id == p_get_column->property_id)
        {
            property_id  = mesh_config.elements[element_idx].sensors[i].property_id;
            prop_val_len = mesh_config.elements[element_idx].sensors[i].prop_value_len;
            for (j = 0; j < mesh_config.elements[element_idx].sensors[i].num_series; j++)
            {
                if (memcmp(p_get_column->raw_valuex,
mesh_config.elements[element_idx].sensors[i].series_columns[j].raw_valuex, prop_val_len) == 0)
                {
                    column_status.property_id = property_id;
                    column_status.prop_value_len = prop_val_len;
                    column_status.is_column_present = WICED_TRUE;
                    memcpy(column_status.column_data.raw_valuex,
mesh_config.elements[element_idx].sensors[i].series_columns[j].raw_valuex, prop_val_len);
                    memcpy(column_status.column_data.raw_valuey,
mesh_config.elements[element_idx].sensors[i].series_columns[j].raw_valuey, prop_val_len);
                    memcpy(column_status.column_data.column_width,
mesh_config.elements[element_idx].sensors[i].series_columns[j].column_width, prop_val_len);
                }
            }
            wiced_bt_mesh_model_sensor_server_column_status_send(p_event, &column_status);
        }
    }
}


/*
 * Process setting change.  Library already copied the new value to the mesh_config.  Add additional
processing here if needed.
 */
void mesh_sensor_server_process_setting_changed(uint8_t element_idx, uint16_t property_id, uint16_t
setting_property_id)
{
    WICED_BT_TRACE("setting changed, prop_id:%x, setting prop_id:%x\n", property_id, setting_property_id);
}


/*
 * Send Sensor Status event
 */
void mesh_sensor_server_status_changed(uint8_t element_idx, uint8_t *p_data, uint32_t length)
{
    uint16_t property_id;
    uint16_t prop_value_len;
    uint32_t cur_time = wiced_bt_mesh_core_get_tick_count();
    wiced_bt_mesh_core_config_sensor_t *p_sensor;

    STREAM_TO_UINT16(property_id, p_data);
    STREAM_TO_UINT16(prop_value_len, p_data);

    if ((length >= 5) && (element_idx == 0) && (property_id == WICED_BT_MESH_PROPERTY_RFID_VALUE) &&
(prop_value_len == 1)) //where it says WICED_BT_MESH_PROPERTY_RFID_VALUE it used to say
```

```
MESH_TEMP_SENSOR_PROPERTY_ID
    {
        mesh_sensor_RFID_current_score = p_data[0];
        WICED_BT_TRACE("new temp:%d\n", mesh_sensor_RFID_current_score);

        p_sensor = &mesh_config.elements[element_idx].sensors[MESH_RFID_SENSOR_INDEX ];

        // Cannot send pubs more often than cadence.min_interval
        if ((cur_time - mesh_sensor_sent_time) < p_sensor->cadence.min_interval)
        {
            WICED_BT_TRACE("Not enough time since last pub\n");

            // if timer is running, the value will be sent, when needed, otherwise, start the time.
            wiced_start_timer(&mesh_sensor_cadence_timer, p_sensor->cadence.min_interval +
mesh_sensor_sent_time - cur_time);
        }
        else
        {
            // the timer callback function sends value change notification if it is appropriate
            mesh_sensor_publish_timer_callback((TIMER_PARAM_TYPE)p_sensor);
        }
    }
    else
    {
        WICED_BT_TRACE("sensor server bad params idx:%d prop:%04x len:%d\n", element_idx, property_id,
prop_value_len);
    }
}

/*
 * In 2 chip solutions MCU can send commands to change user_property state.
 */
uint32_t mesh_app_proc_rx_cmd(uint16_t opcode, uint8_t *p_data, uint32_t length)
{
#ifdef HCI_CONTROL
    uint8_t element_idx;
    element_idx = wiced_bt_mesh_get_element_idx_from_wiced_hci(&p_data, &length);
    switch (opcode)
    {
    case HCI_CONTROL_MESH_COMMAND_SENSOR_SET:
        mesh_sensor_server_status_changed(element_idx, p_data, length);
        break;
    }
#endif
    return WICED_TRUE;
}


#ifdef HCI_CONTROL
/*
 * Send Sensor Cadence Set event over transport
 */
void mesh_sensor_hci_event_send_cadence_set(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_cadence_set_data_t *p_set)
{
    uint8_t *p = p_hci_event->data;
    uint8_t flag_trigger_type;

    WICED_BT_TRACE("mesh_sensor_hci_event_send_cadence_set:\n");

    UINT16_TO_STREAM(p, p_set->property_id);
    UINT8_TO_STREAM(p, p_set->prop_value_len);
    UINT16_TO_STREAM(p, p_set->cadence_data.fast_cadence_period_divisor);
    UINT8_TO_STREAM(p, p_set->cadence_data.trigger_type ? 0x01 : 0x00);
    UINT32_TO_STREAM(p, p_set->cadence_data.trigger_delta_down);
    UINT32_TO_STREAM(p, p_set->cadence_data.trigger_delta_up);
    UINT32_TO_STREAM(p, p_set->cadence_data.min_interval);
    UINT32_TO_STREAM(p, p_set->cadence_data.fast_cadence_low);
    UINT32_TO_STREAM(p, p_set->cadence_data.fast_cadence_high);

    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_CADENCE_SET, (uint8_t *)p_hci_event, (uint16_t)(p
- (uint8_t *)p_hci_event));
}

/*
 * Send Sensor Setting Set event over transport
 */
```

```
void mesh_sensor_hci_event_send_setting_set(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_setting_set_data_t *p_set)
{
    uint8_t *p = p_hci_event->data;

    WICED_BT_TRACE("mesh_sensor_hci_event_send_setting_get:\n");

    UINT16_TO_STREAM(p, p_set->property_id);
    UINT16_TO_STREAM(p, p_set->setting_property_id);
    UINT8_TO_STREAM(p, p_set->prop_value_len);
    ARRAY_TO_STREAM(p, p_set->setting_raw_val, p_set->prop_value_len);

    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_SETTING_SET, (uint8_t *)p_hci_event, (uint16_t)(p
- (uint8_t *)p_hci_event));
}

#endif



    switch (event)
    {
    case WICED_BT_MESH_TX_COMPLETE:
        WICED_BT_TRACE("tx complete\n");
        break;

#if defined HCI_CONTROL
    case WICED_BT_MESH_SENSOR_DESCRIPTOR_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_desc_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_descriptor_status_data_t
*)p_data);
        }
        break;

    case WICED_BT_MESH_SENSOR_STATUS://This is the only case we are using because the sensor node will send
a STATUS message with RFID data
        memcpy(data, ((wiced_bt_mesh_sensor_status_data_t *)p_data)->raw_value,
((wiced_bt_mesh_sensor_status_data_t *)p_data)->prop_value_len); //Used to add newly received RFID data to
previous value
        WICED_BT_TRACE("data:%B\n", data); //Displays data in terminal
        break;
```

Appendix C. Sensor Client Code

```
/*
 * Copyright 2020, Cypress Semiconductor Corporation or a subsidiary of
 * Cypress Semiconductor Corporation. All Rights Reserved.
 *
 * This software, including source code, documentation and related
 * materials ("Software"), is owned by Cypress Semiconductor Corporation
 * or one of its subsidiaries ("Cypress") and is protected by and subject to
 * worldwide patent protection (United States and foreign),
 * United States copyright laws and international treaty provisions.
 * Therefore, you may use this Software only as provided in the license
 * agreement accompanying the software package from which you
 * obtained this Software ("EULA").
 * If no EULA applies, Cypress hereby grants you a personal, non-exclusive,
 * non-transferable license to copy, modify, and compile the Software
 * source code solely for use in connection with Cypress's
 * integrated circuit products. Any reproduction, modification, translation,
 * compilation, or representation of this Software except as specified
 * above is prohibited without the express written permission of Cypress.
 *
 * Disclaimer: THIS SOFTWARE IS PROVIDED AS-IS, WITH NO WARRANTY OF ANY KIND,
 * EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, NONINFRINGEMENT, IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress
 * reserves the right to make changes to the Software without notice. Cypress
 * does not assume any liability arising out of the application or use of the
 * Software or any product or circuit described in the Software. Cypress does
 * not authorize its products for use in any products where a malfunction or
 * failure of the Cypress product may reasonably be expected to result in
 * significant property damage, injury or death ("High Risk Product"). By
 * including Cypress's product in a High Risk Product, the manufacturer
 * of such system or application assumes all risk of such use and in doing
 * so agrees to indemnify Cypress against all liability.
 */

/** @file
 *
 *
 * This file shows how to create a device which implements mesh user sensor client.
 */

//#include "cy_device_headers.h"
#include "cycfg.h"
#include "stdio.h"


#include "sparcommon.h"
#include "wiced_bt_dev.h"

#include "wiced_hal_gpio.h"

#include "wiced_platform.h"
#include "wiced_bt_stack.h"
#include "GeneratedSource/cycfg_pins.h"

#include "wiced_bt_ble.h"
#include "wiced_bt_gatt.h"
#include "wiced_bt_mesh_models.h"
#include "wiced_bt_trace.h"
#include "wiced_bt_mesh_app.h"

#ifdef HCI_CONTROL
#include "wiced_transport.h"
#include "hci_control_api.h"
#endif

#include "wiced_bt_cfg.h"
extern wiced_bt_cfg_settings_t wiced_bt_cfg_settings;

/*****************************************************
 *          Constants
 *****************************************************/
#define MESH_PID                0x3021
#define MESH_VID                0x0002
#define MESH_CACHE_REPLAY_SIZE  0x0008

#define SETTING_PROPERTY_ID                         (0x2001)
```

```
//Create RFID property value and assign to Presence Detected property
#define WICED_BT_MESH_PROPERTY_RFID_VALUE  WICED_BT_MESH_PROPERTY_PRESENCE_DETECTED


/******************************************************
 *          Structures
 ******************************************************/

/******************************************************
 *          Function Prototypes
 ******************************************************/
static void mesh_app_init(wiced_bool_t is_provisioned);
static uint32_t mesh_app_proc_rx_cmd(uint16_t opcode, uint8_t *p_data, uint32_t length);
static void mesh_sensor_client_message_handler(uint8_t element_idx, uint16_t addr, uint16_t event, void
*p_data);
static void mesh_sensor_descriptor_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_column_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_cadence_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_series_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_cadence_set(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_setting_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_setting_set(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_settings_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length);
static void mesh_sensor_series_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_series_status_data_t *p_data);
static void mesh_sensor_desc_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_descriptor_status_data_t *p_data);
static void mesh_sensor_data_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_status_data_t *p_data);
static void mesh_sensor_column_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_column_status_data_t *p_data);
static void mesh_sensor_cadence_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_cadence_status_data_t *cadence_status );
static void mesh_sensor_setting_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_setting_status_data_t *p_data);
static void mesh_sensor_settings_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_settings_status_data_t *p_data);
static void hal_gpio_app_test_output(void);
static void set_test_pin(void);


/******************************************************
 *          Variables Definitions
 ******************************************************/
uint8_t mesh_mfr_name[WICED_BT_MESH_PROPERTY_LEN_DEVICE_MANUFACTURER_NAME] = { 'C', 'y', 'p', 'r', 'e', 's',
's', 0 };
uint8_t mesh_model_num[WICED_BT_MESH_PROPERTY_LEN_DEVICE_MODEL_NUMBER]     = { '1', '2', '3', '4', 0, 0, 0,
0 };
uint8_t mesh_system_id[8]                                                  = { 0xbb, 0xb8, 0xa1, 0x80, 0x5f,
0x9f, 0x91, 0x71 };

uint8_t RFID_data =0; //RFID_data stores the score for a given read cycle

wiced_bt_mesh_core_config_model_t   mesh_element1_models[] =
{
        WICED_BT_MESH_DEVICE,
        WICED_BT_MESH_MODEL_SENSOR_CLIENT,
};
#define MESH_APP_NUM_MODELS  (sizeof(mesh_element1_models) / sizeof(wiced_bt_mesh_core_config_model_t))

#define MESH_SENSOR_CLIENT_ELEMENT_INDEX   0

wiced_bt_mesh_core_config_property_t mesh_element1_properties[] =
{
    { WICED_BT_MESH_PROPERTY_MOTION_SENSED,               0, 0, 1, NULL},
        { WICED_BT_MESH_PROPERTY_PRESENT_AMBIENT_LIGHT_LEVEL, 0, 0, 3, NULL},
        { WICED_BT_MESH_PROPERTY_TOTAL_LIGHT_EXPOSURE_TIME,   0, 0, 3, NULL},
    { SETTING_PROPERTY_ID,                                0, 0, 2, NULL},

        //2nd change below

        { WICED_BT_MESH_PROPERTY_RFID_VALUE,                  0, 0, 1, NULL}, //here I'm defining a property
for the RFID value

                                                //the 1 that I'm using defines the max length of the
```

```
property value which is our score length in bytes

};

wiced_bt_mesh_core_config_element_t mesh_elements[] =
{
    {
        .location = MESH_ELEM_LOC_MAIN,                                 // location description as defined
in the GATT Bluetooth Namespace Descriptors section of the Bluetooth SIG Assigned Numbers
        .default_transition_time = MESH_DEFAULT_TRANSITION_TIME_IN_MS,  // Default transition time for
models of the element in milliseconds
        .onpowerup_state = WICED_BT_MESH_ON_POWER_UP_STATE_RESTORE,     // Default element behavior on power
up
        .properties_num = 5,                                            // Number of properties in the array
models
        .properties = mesh_element1_properties,                         // Array of properties in the
element.
        .sensors_num = 0,                                               // Number of sensors in the sensor
array
        .sensors = NULL,                                                // Array of sensors of that element
        .models_num = MESH_APP_NUM_MODELS,                              // Number of models in the array
models
        .models = mesh_element1_models,                                 // Array of models located in that
element. Model data is defined by structure wiced_bt_mesh_core_config_model_t
    },
};

uint8_t mesh_num_elements = 1;

wiced_bt_mesh_core_config_t  mesh_config =
{
    .company_id         = MESH_COMPANY_ID_CYPRESS,                  // Company identifier assigned by the
Bluetooth SIG
    .product_id         = MESH_PID,                                 // Vendor-assigned product identifier
    .vendor_id          = MESH_VID,                                 // Vendor-assigned product version
identifier
    .replay_cache_size  = MESH_CACHE_REPLAY_SIZE,                   // Number of replay protection entries,
i.e. maximum number of mesh devices that can send application messages to this device.
#if defined(LOW_POWER_NODE) && (LOW_POWER_NODE == 1)
    .features           = WICED_BT_MESH_CORE_FEATURE_BIT_LOW_POWER, // A bit field indicating the device
features. In Low Power mode no Relay, no Proxy and no Friend
    .friend_cfg         =                                           // Empty Configuration of the Friend
Feature
    {
        .receive_window = 0,                                        // Receive Window value in milliseconds
supported by the Friend node.
        .cache_buf_len  = 0,                                        // Length of the buffer for the cache
        .max_lpn_num    = 0                                         // Max number of Low Power Nodes with
established friendship. Must be > 0 if Friend feature is supported.
    },
    .low_power          =                                           // Configuration of the Low Power
Feature
    {
        .rssi_factor           = 2,                                 // contribution of the RSSI measured by
the Friend node used in Friend Offer Delay calculations.
        .receive_window_factor = 2,                                 // contribution of the supported Receive
Window used in Friend Offer Delay calculations.
        .min_cache_size_log    = 3,                                 // minimum number of messages that the
Friend node can store in its Friend Cache.
        .receive_delay         = 100,                               // Receive delay in 1 ms units to be
requested by the Low Power node.
        .poll_timeout          = 36000                              // Poll timeout in 100ms units to be
requested by the Low Power node.
    },
#else
    .features = WICED_BT_MESH_CORE_FEATURE_BIT_FRIEND | WICED_BT_MESH_CORE_FEATURE_BIT_RELAY |
WICED_BT_MESH_CORE_FEATURE_BIT_GATT_PROXY_SERVER,   // Supports Friend, Relay and GATT Proxy
    .friend_cfg         =                                           // Configuration of the Friend
Feature(Receive Window in Ms, messages cache)
    {
        .receive_window        = 20,
        .cache_buf_len         = 300,                               // Length of the buffer for the cache
        .max_lpn_num           = 4                                  // Max number of Low Power Nodes with
established friendship. Must be > 0 if Friend feature is supported.
    },
    .low_power          =                                           // Configuration of the Low Power
Feature
```

```
    {
        .rssi_factor           = 0,                                    // contribution of the RSSI measured by
the Friend node used in Friend Offer Delay calculations.
        .receive_window_factor = 0,                                    // contribution of the supported Receive
Window used in Friend Offer Delay calculations.
        .min_cache_size_log    = 0,                                    // minimum number of messages that the
Friend node can store in its Friend Cache.
        .receive_delay         = 0,                                    // Receive delay in 1 ms units to be
requested by the Low Power node.
        .poll_timeout          = 0                                     // Poll timeout in 100ms units to be
requested by the Low Power node.
    },
#endif
    .gatt_client_only          = WICED_FALSE,                          // Can connect to mesh over GATT or ADV
    .elements_num  = (uint8_t)(sizeof(mesh_elements) / sizeof(mesh_elements[0])),   // number of elements on
this device
    .elements      = mesh_elements                                     // Array of elements for this device
};

/*
 * Mesh application library will call into application functions if provided by the application.
 */
wiced_bt_mesh_app_func_table_t wiced_bt_mesh_app_func_table =
{
    mesh_app_init,          // application initialization
    NULL,                   // Default SDK platform button processing
    NULL,                   // GATT connection status
    NULL,                   // attention processing
    NULL,                   // notify period set
         NULL,                                      // WICED HCI command
    NULL,                   // LPN sleep
    NULL                    // factory reset
};

/******************************************************
 *               Function Definitions
 ******************************************************/
void mesh_app_init(wiced_bool_t is_provisioned)
{
#if 0
    // Set Debug trace level for mesh_models_lib and mesh_provisioner_lib
    wiced_bt_mesh_models_set_trace_level(WICED_BT_MESH_CORE_TRACE_INFO);
#endif
#if 0
    // Set Debug trace level for all modules but Info level for CORE_AES_CCM module
    wiced_bt_mesh_core_set_trace_level(WICED_BT_MESH_CORE_TRACE_FID_ALL, WICED_BT_MESH_CORE_TRACE_DEBUG);
    wiced_bt_mesh_core_set_trace_level(WICED_BT_MESH_CORE_TRACE_FID_CORE_AES_CCM,
WICED_BT_MESH_CORE_TRACE_INFO);
#endif

    wiced_bt_cfg_settings.device_name = (uint8_t *)"Sensor Client";
    wiced_bt_cfg_settings.gatt_cfg.appearance = APPEARANCE_GENERIC_TAG;
    // Adv Data is fixed. Spec allows to put URI, Name, Appearance and Tx Power in the Scan Response Data.
    if (!is_provisioned)
    {
        wiced_bt_ble_advert_elem_t  adv_elem[3];
        uint8_t                     buf[2];
        uint8_t                     num_elem = 0;
        adv_elem[num_elem].advert_type = BTM_BLE_ADVERT_TYPE_NAME_COMPLETE;
        adv_elem[num_elem].len = (uint16_t)strlen((const char*)wiced_bt_cfg_settings.device_name);
        adv_elem[num_elem].p_data = wiced_bt_cfg_settings.device_name;
        num_elem++;

        adv_elem[num_elem].advert_type = BTM_BLE_ADVERT_TYPE_APPEARANCE;
        adv_elem[num_elem].len = 2;
        buf[0] = (uint8_t)wiced_bt_cfg_settings.gatt_cfg.appearance;
        buf[1] = (uint8_t)(wiced_bt_cfg_settings.gatt_cfg.appearance >> 8);
        adv_elem[num_elem].p_data = buf;
        num_elem++;

        wiced_bt_mesh_set_raw_scan_response_data(num_elem, adv_elem);

    }

    hal_gpio_app_test_output(); //enable output bits which will be sent to LED screen
    // register with the library to receive parsed data
    wiced_bt_mesh_model_sensor_client_init(MESH_SENSOR_CLIENT_ELEMENT_INDEX,
```

```
mesh_sensor_client_message_handler, is_provisioned);
}



void set_output_pins(void) //This function assigns output pins based on the received RFID data
{                                                          //This is meant to light LEDs corresponding to
the value of the score received and for testing purposes
        uint8_t a=0;//LSB
        uint8_t b=0;
        uint8_t c=0;
        uint8_t d=0;
        uint8_t e=0;//MSB
        if (RFID_data & 1)
        {
                a=1;
        }
        else
        {
                a=0;
        }

        if (RFID_data & 2)
        {
                b=1;
        }
        else
        {
                b=0;
        }

        if (RFID_data & 4)
        {
                c=1;
        }
        else
        {
                c=0;
        }

        if (RFID_data & 8)
        {
                d=1;
        }
        else
        {
                d=0;
        }

        if (RFID_data & 16)
        {
                e=1;
        }
        else
        {
                e=0;
        }


        wiced_hal_gpio_set_pin_output(WICED_P08, a ); //This sets GPIO pins on the CYBT213043 EVAL boards
        wiced_hal_gpio_set_pin_output(WICED_P12, b );
        wiced_hal_gpio_set_pin_output(WICED_P13, c );
        wiced_hal_gpio_set_pin_output(WICED_P14, d );
        wiced_hal_gpio_set_pin_output(WICED_P15, e );
}

void hal_gpio_app_test_output(void) //Enables GPIO pins to be outputs
{
    uint8_t index = 0;

    wiced_hal_gpio_configure_pin(WICED_P08, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P12, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P13, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P14, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);
    wiced_hal_gpio_configure_pin(WICED_P15, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);

    wiced_hal_gpio_configure_pin(WICED_P01, GPIO_OUTPUT_ENABLE, GPIO_PIN_OUTPUT_LOW);//this is strictly for
```

```
testing purposes


}

void set_test_pin(void) //Place this function in places to set an LED to tell you where you're at in the
code
{
        wiced_hal_gpio_set_pin_output(WICED_P01, GPIO_PIN_OUTPUT_HIGH);
}

/*
 * Process event received from the sensor Server.
 */
void mesh_sensor_client_message_handler(uint8_t element_idx, uint16_t addr, uint16_t event, void *p_data)
{
        WICED_BT_TRACE("*****data*****:\n"); //Displays trace statement on Terminal

        uint8_t data[100]={0}; //Used to store the received RFID data
                                                                //Data that is received is added on to the
previous value stored in this array

#if defined HCI_CONTROL
    wiced_bt_mesh_hci_event_t *p_hci_event;


#endif
    WICED_BT_TRACE("sensor clt msg:%d\n", event);


    switch (event)
    {
    case WICED_BT_MESH_TX_COMPLETE:
        WICED_BT_TRACE("tx complete\n");
        break;

#if defined HCI_CONTROL
    case WICED_BT_MESH_SENSOR_DESCRIPTOR_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_desc_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_descriptor_status_data_t
*)p_data);
        }
        break;

    case WICED_BT_MESH_SENSOR_STATUS://This is the only case we are using because the sensor node will send
a STATUS message with RFID data
        memcpy(data, ((wiced_bt_mesh_sensor_status_data_t *)p_data)->raw_value,
((wiced_bt_mesh_sensor_status_data_t *)p_data)->prop_value_len); //Used to add newly received RFID data to
previous value
        WICED_BT_TRACE("data:%B\n", data); //Displays data in terminal
        break;

    case WICED_BT_MESH_SENSOR_COLUMN_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_column_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_column_status_data_t
*)p_data);
        }
        break;

    case WICED_BT_MESH_SENSOR_SERIES_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_series_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_series_status_data_t
*)p_data);
        }
        break;

    case WICED_BT_MESH_SENSOR_CADENCE_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_cadence_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_cadence_status_data_t
```

```
*)p_data);
        }
        break;

    case WICED_BT_MESH_SENSOR_SETTINGS_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_settings_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_settings_status_data_t
*)p_data);
        }
        break;

    case WICED_BT_MESH_SENSOR_SETTING_STATUS:
        if ((p_hci_event = wiced_bt_mesh_alloc_hci_event(element_idx)) != NULL)
        {
            p_hci_event->src = addr;
            mesh_sensor_setting_hci_event_send(p_hci_event, (wiced_bt_mesh_sensor_setting_status_data_t
*)p_data);
        }
        break;
#endif
    default:
        WICED_BT_TRACE("not processed\n");
        break;
    }
}


/*
 * In 2 chip solutions MCU can send commands to change sensor state.
 */
uint32_t mesh_app_proc_rx_cmd(uint16_t opcode, uint8_t *p_data, uint32_t length)
{
#ifdef HCI_CONTROL
    wiced_bt_mesh_event_t *p_event;

    switch (opcode)
    {
    case HCI_CONTROL_MESH_COMMAND_SENSOR_DESCRIPTOR_GET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_GET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_COLUMN_GET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_SERIES_GET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_CADENCE_GET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_CADENCE_SET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_SETTING_GET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_SETTING_SET:
    case HCI_CONTROL_MESH_COMMAND_SENSOR_SETTINGS_GET:
        break;

    default:
        return WICED_FALSE;
    }
    p_event = wiced_bt_mesh_create_event_from_wiced_hci(opcode, MESH_COMPANY_ID_BT_SIG,
WICED_BT_MESH_CORE_MODEL_ID_SENSOR_CLNT, &p_data, &length);
    if (p_event == NULL)
    {
        WICED_BT_TRACE("bad hdr\n");
        return WICED_TRUE;
    }
    switch (opcode)
    {
    //sensor client messages
    case HCI_CONTROL_MESH_COMMAND_SENSOR_DESCRIPTOR_GET:
        mesh_sensor_descriptor_get(p_event, p_data, length);
        break;

    case HCI_CONTROL_MESH_COMMAND_SENSOR_GET:
        mesh_sensor_get(p_event, p_data, length);
        break;

    case HCI_CONTROL_MESH_COMMAND_SENSOR_COLUMN_GET:
        mesh_sensor_column_get(p_event, p_data, length);
        break;

    case HCI_CONTROL_MESH_COMMAND_SENSOR_SERIES_GET:
        mesh_sensor_series_get(p_event, p_data, length);
```

```
        break;

    //sensor setup server messages
    case HCI_CONTROL_MESH_COMMAND_SENSOR_CADENCE_GET:
        mesh_sensor_cadence_get(p_event, p_data, length);
        break;

    case HCI_CONTROL_MESH_COMMAND_SENSOR_CADENCE_SET:
        mesh_sensor_cadence_set(p_event, p_data, length);
        break;

    case HCI_CONTROL_MESH_COMMAND_SENSOR_SETTING_GET:
        mesh_sensor_setting_get(p_event, p_data, length);
        break;

    case HCI_CONTROL_MESH_COMMAND_SENSOR_SETTING_SET:
        mesh_sensor_setting_set(p_event, p_data, length);
        break;

    case HCI_CONTROL_MESH_COMMAND_SENSOR_SETTINGS_GET:
        mesh_sensor_settings_get(p_event, p_data, length);
        break;
    }
#endif

    return WICED_TRUE;
}

/*
 * Send sensor descriptor get command
 */
void mesh_sensor_descriptor_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_get_t get_data;

    WICED_BT_TRACE("mesh_sensor_descriptor_get\n");

    if (length == 2)
    {
        STREAM_TO_UINT16(get_data.property_id, p_data);
    }
    else
    {
        get_data.property_id = 0;
    }
    wiced_bt_mesh_model_sensor_client_descriptor_send_get(p_event, &get_data);
}

/*
 * Send sensor get command
 */
void mesh_sensor_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_get_t get_data;

    WICED_BT_TRACE("mesh_sensor_get\n");

    if (length == 2)
    {
        STREAM_TO_UINT16(get_data.property_id, p_data);
    }
    else
    {
        get_data.property_id = 0;
    }
    wiced_bt_mesh_model_sensor_client_sensor_send_get(p_event, &get_data);
    //set_test_pin();
}

/*
 * Send column get command
 */
void mesh_sensor_column_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_column_get_data_t get_data;
    int i;
```

```
    STREAM_TO_UINT16(get_data.property_id, p_data);
    STREAM_TO_UINT8(get_data.prop_value_len, p_data);
    STREAM_TO_ARRAY(get_data.raw_valuex, p_data, get_data.prop_value_len);

    WICED_BT_TRACE("sensor column get\n");
    for(i = 0; i < get_data.prop_value_len; i++)
        WICED_BT_TRACE(" %02x", get_data.raw_valuex[i]);
    WICED_BT_TRACE("\n");

    wiced_bt_mesh_model_sensor_client_sensor_column_send_get(p_event, &get_data);
}

/*
 * Send series get command
 */
void mesh_sensor_series_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_series_get_data_t get_data;
    uint8_t header;
         int i;

    memset(&get_data, 0, sizeof(wiced_bt_mesh_sensor_series_get_data_t));

    header = 2 * sizeof(uint16_t);

    WICED_BT_TRACE("mesh_sensor_series_get\n");
    if ((length - header) == 2)
    {
        WICED_BT_TRACE("\n property id only\n");
        STREAM_TO_UINT16(get_data.property_id, p_data);
        get_data.start_index = 0x00;
        get_data.end_index = 0xFF;
    }
    else
    {
        WICED_BT_TRACE("\n property id \n");
        STREAM_TO_UINT16(get_data.property_id, p_data);
        STREAM_TO_UINT8(get_data.prop_value_len, p_data);
        STREAM_TO_ARRAY(get_data.raw_valuex1, p_data, get_data.prop_value_len);
        STREAM_TO_ARRAY(get_data.raw_valuex2, p_data, get_data.prop_value_len);
    }
    for(i = 0; i < get_data.prop_value_len; i++)
        WICED_BT_TRACE(" %02x",get_data.raw_valuex1[i]);
    WICED_BT_TRACE("\n");

    for(i = 0; i < get_data.prop_value_len; i++)
        WICED_BT_TRACE(" %02x",get_data.raw_valuex2[i]);
    WICED_BT_TRACE("\n");

    wiced_bt_mesh_model_sensor_client_sensor_series_send_get(p_event, &get_data);
}

/*
 * Send cadence get command
 */
void mesh_sensor_cadence_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_get_t get_data;

    if (length == 2)
    {
        STREAM_TO_UINT16(get_data.property_id, p_data);
    }
    else
    {
        get_data.property_id = 0;
    }
    wiced_bt_mesh_model_sensor_client_sensor_cadence_send_get(p_event, &get_data);
}

/*
 * Send cadence set command
 */

void mesh_sensor_cadence_set(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t data_len)
{
    wiced_bt_mesh_sensor_cadence_set_data_t cadence_set;
```

```
    WICED_BT_TRACE("sensor cadence set\n");

    STREAM_TO_UINT16(cadence_set.property_id, p_data);
    STREAM_TO_UINT8(cadence_set.prop_value_len, p_data);
    STREAM_TO_UINT16(cadence_set.cadence_data.fast_cadence_period_divisor, p_data);
    STREAM_TO_UINT8(cadence_set.cadence_data.trigger_type, p_data);
    STREAM_TO_UINT32(cadence_set.cadence_data.trigger_delta_down, p_data);
    STREAM_TO_UINT32(cadence_set.cadence_data.trigger_delta_up, p_data);
    STREAM_TO_UINT32(cadence_set.cadence_data.min_interval, p_data);
    STREAM_TO_UINT32(cadence_set.cadence_data.fast_cadence_low, p_data);
    STREAM_TO_UINT32(cadence_set.cadence_data.fast_cadence_high, p_data);

    wiced_bt_mesh_model_sensor_client_sensor_cadence_send_set(p_event, &cadence_set);
}

/*
 * Send setting get command
 */
void mesh_sensor_setting_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_setting_get_data_t get_data;

    WICED_BT_TRACE("sensor setting get\n");

    STREAM_TO_UINT16(get_data.property_id, p_data);
    STREAM_TO_UINT16(get_data.setting_property_id, p_data);

    wiced_bt_mesh_model_sensor_client_sensor_setting_send_get(p_event, &get_data);
}

/*
 * Send setting set command
 */
void mesh_sensor_setting_set(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_setting_set_data_t set_data;

    WICED_BT_TRACE("sensor setting set\n");

    memset(&set_data, 0, sizeof(set_data));

    STREAM_TO_UINT16(set_data.property_id, p_data);
    STREAM_TO_UINT16(set_data.setting_property_id, p_data);
    STREAM_TO_UINT8(set_data.prop_value_len, p_data);
    STREAM_TO_ARRAY(set_data.setting_raw_val, p_data, set_data.prop_value_len);

    wiced_bt_mesh_model_sensor_client_sensor_setting_send_set(p_event, &set_data);
}

/*
 * Send settings get command
 */
void mesh_sensor_settings_get(wiced_bt_mesh_event_t *p_event, uint8_t *p_data, uint32_t length)
{
    wiced_bt_mesh_sensor_get_t get_data;

    STREAM_TO_UINT16(get_data.property_id, p_data);

    WICED_BT_TRACE("sensor settings get prop:%04x\n", get_data.property_id);

    wiced_bt_mesh_model_sensor_client_sensor_settings_send_get(p_event, &get_data);
}

#ifdef HCI_CONTROL
/*
 * Send Descriptor Status event over transport
 */
void mesh_sensor_desc_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_descriptor_status_data_t *p_data)
{
    int i;
    uint8_t *p = p_hci_event->data;

    WICED_BT_TRACE("mesh_sensor_desc_hci_event_send: num descriptors %x\n", p_data->num_descriptors);
    if (p_data->num_descriptors != 0)
    {
```

```
        for (i = 0; i < p_data->num_descriptors; i++)
        {
            WICED_BT_TRACE("property_id :  %x\n", p_data->descriptor_list[i].property_id);
            WICED_BT_TRACE("positive_tolerance :  %x\n", p_data->descriptor_list[i].positive_tolerance);
            WICED_BT_TRACE("negative_tolerance :  %x\n", p_data->descriptor_list[i].negative_tolerance);
            WICED_BT_TRACE("sampling_function :  %x\n", p_data->descriptor_list[i].sampling_function);
            WICED_BT_TRACE("measurement_period :  %x\n", p_data->descriptor_list[i].measurement_period);
            WICED_BT_TRACE("update_interval :  %x\n", p_data->descriptor_list[i].update_interval);
            UINT16_TO_STREAM(p, p_data->descriptor_list[i].property_id);
            UINT16_TO_STREAM(p, p_data->descriptor_list[i].positive_tolerance);
            UINT16_TO_STREAM(p, p_data->descriptor_list[i].negative_tolerance);
            UINT8_TO_STREAM(p, p_data->descriptor_list[i].sampling_function);
            UINT8_TO_STREAM(p, p_data->descriptor_list[i].measurement_period);
            UINT8_TO_STREAM(p, p_data->descriptor_list[i].update_interval);
        }
    }
    else
    {
        WICED_BT_TRACE("mesh_sensor_desc_get : no descriptor present for property ID\n");
    }

    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_DESCRIPTOR_STATUS, (uint8_t *)p_hci_event,
(uint16_t)(p - (uint8_t *)p_hci_event));
}

/*
 * Send sensor Status event over transport
 */
void mesh_sensor_data_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_status_data_t *p_data)
{
    uint8_t *p = p_hci_event->data;
    int i, j;

    WICED_BT_TRACE("property_id:%04x\n", p_data->property_id);
    WICED_BT_TRACE("prop_value_len:%d\n", p_data->prop_value_len);
    WICED_BT_TRACE("Raw val");

    for (j = 0; j < p_data->prop_value_len; j++)
        WICED_BT_TRACE(" %02x", p_data->raw_value[j]);
    WICED_BT_TRACE("\n");

    UINT16_TO_STREAM(p, p_data->property_id);
    UINT8_TO_STREAM(p, p_data->prop_value_len);
    memcpy(p, p_data->raw_value, p_data->prop_value_len);
    p = p + p_data->prop_value_len;

    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_STATUS, (uint8_t *)p_hci_event, (uint16_t)(p -
(uint8_t *)p_hci_event));
}

/*
 * Send sensor column status event over transport
 */
void mesh_sensor_column_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_column_status_data_t *p_data)
{
    uint8_t *p = p_hci_event->data;
    int i;

    UINT16_TO_STREAM(p, p_data->property_id);
    UINT8_TO_STREAM(p, p_data->prop_value_len);

    ARRAY_TO_STREAM(p, p_data->column_data.raw_valuex, p_data->prop_value_len);
    ARRAY_TO_STREAM(p, p_data->column_data.column_width, p_data->prop_value_len);
    ARRAY_TO_STREAM(p, p_data->column_data.raw_valuey, p_data->prop_value_len);

    WICED_BT_TRACE(" property_id:%x\n",p_data->property_id);
    WICED_BT_TRACE(" prop_value_len:%x\n",p_data->prop_value_len);
    WICED_BT_TRACE("\n -----RAW VAL X----------- \n");
    for (i = 0; i < p_data->prop_value_len; i++)
        WICED_BT_TRACE(" %x ",p_data->column_data.raw_valuex[i]);
    WICED_BT_TRACE("\n -----COL WIDTH----------- \n");
    for (i = 0; i < p_data->prop_value_len; i++)
          WICED_BT_TRACE(" %x ",p_data->column_data.column_width[i]);
    WICED_BT_TRACE("\n -----RAW VAL Y----------- \n");
    for (i = 0; i < p_data->prop_value_len; i++)
```

```
            WICED_BT_TRACE(" %x ",p_data->column_data.raw_valuey[i]);
    WICED_BT_TRACE("\n ----------------------- \n");
    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_COLUMN_STATUS, (uint8_t *)p_hci_event,
(uint16_t)(p - (uint8_t *)p_hci_event));
}

/*
 * Send sensor series status event over transport
 */
void mesh_sensor_series_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_series_status_data_t *p_data)
{
    uint8_t *p = p_hci_event->data;
    int i, j;

    WICED_BT_TRACE(" property_id:%x val_len:%x no_columns:%x\n",p_data->property_id, p_data->prop_value_len,
p_data->no_of_columns);

    UINT16_TO_STREAM(p, p_data->property_id);
    UINT8_TO_STREAM(p, p_data->prop_value_len);
    UINT8_TO_STREAM(p, p_data->no_of_columns);

    for (i = 0; i < p_data->no_of_columns; i++)
    {
        ARRAY_TO_STREAM(p, p_data->column_list->raw_valuex, p_data->prop_value_len);
        ARRAY_TO_STREAM(p, p_data->column_list->column_width, p_data->prop_value_len);
        ARRAY_TO_STREAM(p, p_data->column_list->raw_valuey, p_data->prop_value_len);

        WICED_BT_TRACE("\n -----RAW VAL X\n");
        for (j=0; j <  p_data->prop_value_len; j++)
            WICED_BT_TRACE(" %x ", p_data->column_list[i].raw_valuex[j]);
        WICED_BT_TRACE("\n -----COL WIDTH\n");
        for (j=0; j < p_data->prop_value_len; j++)
               WICED_BT_TRACE(" %x ", p_data->column_list[i].column_width[j]);
        WICED_BT_TRACE("\n -----RAW VAL Y\n");
        for (j=0; j < p_data->prop_value_len; j++)
               WICED_BT_TRACE(" %x ", p_data->column_list[i].raw_valuey[j]);
    }
    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_SERIES_STATUS, (uint8_t *)p_hci_event,
(uint16_t)(p - (uint8_t *)p_hci_event));
}

/*
 * Send sensor cadence status event over transport
 */

void mesh_sensor_cadence_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_cadence_status_data_t *cadence_status)
{
    uint8_t *p = p_hci_event->data;

    UINT16_TO_STREAM(p, cadence_status->property_id);
    WICED_BT_TRACE(" property_id:%x\n",cadence_status->property_id);

    if (cadence_status->is_data_present)
    {
        WICED_BT_TRACE("fast_cadence_period_divisor:%x\n",
cadence_status->cadence_data.fast_cadence_period_divisor);
        WICED_BT_TRACE("trigger_type:%x\n", cadence_status->cadence_data.trigger_type);
        WICED_BT_TRACE("trigger_delta_down:%d\n", cadence_status->cadence_data.trigger_delta_down);
        WICED_BT_TRACE("trigger_delta_up:%d\n", cadence_status->cadence_data.trigger_delta_up);
        WICED_BT_TRACE("min interval:%x\n", cadence_status->cadence_data.min_interval);
        WICED_BT_TRACE("fast_cadence_high:%d\n", cadence_status->cadence_data.fast_cadence_high);
        WICED_BT_TRACE("fast_cadence_low:%d\n", cadence_status->cadence_data.fast_cadence_low);

        UINT16_TO_STREAM(p, cadence_status->cadence_data.fast_cadence_period_divisor);
        UINT8_TO_STREAM(p, cadence_status->cadence_data.trigger_type);
        UINT32_TO_STREAM(p, cadence_status->cadence_data.trigger_delta_down);
        UINT32_TO_STREAM(p, cadence_status->cadence_data.trigger_delta_up);
        UINT32_TO_STREAM(p, cadence_status->cadence_data.min_interval);
        UINT32_TO_STREAM(p, cadence_status->cadence_data.fast_cadence_low);
        UINT32_TO_STREAM(p, cadence_status->cadence_data.fast_cadence_high);
    }
    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_CADENCE_STATUS, (uint8_t *)p_hci_event,
(uint16_t)(p - (uint8_t *)p_hci_event));
}
```

```
/*
 * Send sensor setting status event over transport
 */
void mesh_sensor_setting_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_setting_status_data_t *p_data)
{
    uint8_t *p = p_hci_event->data;
    int i;

    UINT16_TO_STREAM(p, p_data->property_id);
    UINT8_TO_STREAM(p, p_data->setting.setting_property_id);
    UINT8_TO_STREAM(p, p_data->setting.access);
    UINT8_TO_STREAM(p, p_data->setting.value_len);
    ARRAY_TO_STREAM(p, p_data->setting.val, p_data->setting.value_len);

    WICED_BT_TRACE(" property_id:%x\n", p_data->property_id);
    WICED_BT_TRACE(" setting_property_id:%x\n",p_data->setting.setting_property_id);
    WICED_BT_TRACE(" access:%x\n",p_data->setting.access);
    WICED_BT_TRACE(" value_len:%x\n", p_data->setting.value_len);
    for (i = 0; i < p_data->setting.value_len; i++)
        WICED_BT_TRACE(" %x ",p_data->setting.val[i]);

    WICED_BT_TRACE("\n----------\n");
    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_SETTING_STATUS, (uint8_t *)p_hci_event,
(uint16_t)(p - (uint8_t *)p_hci_event));
}

/*
 * Send sensor settings status event over transport
 */
void mesh_sensor_settings_hci_event_send(wiced_bt_mesh_hci_event_t *p_hci_event,
wiced_bt_mesh_sensor_settings_status_data_t *p_data)
{
    uint8_t *p = p_hci_event->data;
    int i;

    UINT16_TO_STREAM(p, p_data->property_id);
    WICED_BT_TRACE(" property_id:%x\n", p_data->property_id);

    for (i = 0; i < p_data->num_setting_property_id; i++)
    {
        UINT16_TO_STREAM(p, p_data->setting_property_id_list[i]);
        WICED_BT_TRACE(" %x ", p_data->setting_property_id_list[i]);
    }
    mesh_transport_send_data(HCI_CONTROL_MESH_EVENT_SENSOR_SETTINGS_STATUS, (uint8_t *)p_hci_event,
(uint16_t)(p - (uint8_t *)p_hci_event));
}

#endif
```

Appendix D. Arduino Code

```
/*
  Reading multiple RFID tags, simultaneously!
  By: Nathan Seidle @ SparkFun Electronics
  Date: October 3rd, 2016
  https://github.com/sparkfun/Simultaneous_RFID_Tag_Reader

  Constantly reads and outputs any tags heard

  If using the Simultaneous RFID Tag Reader (SRTR) shield, make sure the serial slide
  switch is in the 'SW-UART' position
*/

#include <SoftwareSerial.h> //Used for transmitting to the device

SoftwareSerial softSerial(2, 3); //RX, TX

#include "SparkFun_UHF_RFID_Reader.h" //Library for controlling the M6E Nano module
RFID nano; //Create instance

int EPClist[10] = {0,0,0,0,0,0,0,0,0,0};
int rssilist[10] = {0,0,0,0,0,0,0,0,0,0};

int input_button = 11;

int team_out = 8; //arduino score output pins
int score_0 = 4;
int score_1 = 5;
int score_2 = 6;
int score_3 = 7;

int button_state = 0;

// EPC codes for RFID Tags
int tag1[10][12] = {{226,00,00,28,18,21,00,85,07,00,26,87},
{226,00,00,28,18,21,00,134,07,00,54,115},
{226,00,00,28,18,21,00,135,07,00,55,191},
{226,00,00,28,18,21,00,115,07,00,40,112},
{226,00,00,28,18,21,00,117,07,00,47,241},
{226,00,00,28,18,21,02,8,07,00,182,152},
{226,00,00,28,18,21,02,25,07,00,199,217},
{226,00,00,28,18,21,01,145,07,00,167,107},
{226,00,00,28,18,21,01,153,07,00,175,223},
{226,00,00,28,18,21,01,137,07,00,167,106}};

int start_time = 0;

boolean print_info = 1;
boolean button_bool = 0;

void setup()
{
  Serial.begin(115200);
  while (!Serial); //Wait for the serial port to come online

  if (setupNano(38400) == false) //Configure nano to run at 38400bps
  {
    Serial.println(F("Module failed to respond. Please check wiring."));
    while (1); //Freeze!
  }

  // assign output pins
  pinMode(input_button, INPUT);
  pinMode(team_out, OUTPUT);
  pinMode(score_0, OUTPUT);
  pinMode(score_1, OUTPUT);
  pinMode(score_2, OUTPUT);
  pinMode(score_3, OUTPUT);

  nano.setRegion(REGION_NORTHAMERICA); //Set to North America

  nano.setReadPower(2100); //5.00 dBm. Higher values may caues USB port to brown out
  //Max Read TX Power is 27.00 dBm and may cause temperature-limit throttling

  // Serial.println(F("Press a key to begin scanning for tags."));
  // while (!Serial.available()); //Wait for user to send a character
```

```
  Serial.read(); //Throw away the user's character
  start_time = millis();
  nano.startReading(); //Begin scanning for tags
}

void loop()
{
  button_state = digitalRead(input_button); //triggers rfid reader to start reading

  if (button_state == HIGH){
    button_bool = 1;
    start_time = millis();
  }

  if (button_bool == 1){
    while (millis() - start_time <= 2000){
      read_tags();
    }
    print_info = 1;
    if (print_info == 1){
      print_data();
      print_info = 0; //output rfid data to terminal and output pins
      button_bool = 0;
      int EPClist[10] = {0,0,0,0,0,0,0,0,0,0};
      int rssilist[10] = {0,0,0,0,0,0,0,0,0,0};
    }
  }
}

void print_data()
{
  int t1_score = 0;
  int t2_score = 0;
  int score_diff = 0;

  // display score information to serial monitor
  for (int i = 0; i < 10; i = i + 2){
    int bag_rssi = 0;
    if ((EPClist[i] == 1) & (EPClist[i+1] == 1)){
      if (rssilist[i] < rssilist[i+1]){
        bag_rssi = rssilist[i];
      }
      else{
        bag_rssi = rssilist[i+1];
      }
    }
    else if ((EPClist[i] == 1) & (EPClist[i+1] == 0)){
      bag_rssi = rssilist[i];
    }
    else if ((EPClist[i] == 0) & (EPClist[i+1] == 1)){
      bag_rssi = rssilist[i+1];
    }
    Serial.print("bag ");
    Serial.print((i+1)/2);
    Serial.print(": ");
    Serial.println(bag_rssi);
    int bag_num = (i+1)/2;
    Serial.print(bag_num);
    Serial.println(bag_rssi);
    if ((bag_num) % 2 == 0){
      if (bag_rssi == 0){
        t1_score = t1_score;
      }
      else if (bag_rssi < -39){
        t1_score = t1_score + 1;
      }
      else{
        t1_score = t1_score + 3;
      }
    }
    else{
      if (bag_rssi == 0){
        t2_score = t2_score;
      }
      else if (bag_rssi < -39){
        t2_score = t2_score + 1;
      }
```

```
    else{
      t2_score = t2_score + 3;
    }
  }
}
Serial.print("team 1: ");
Serial.print(t1_score);
Serial.print(" / team 2:");
Serial.println(t2_score);

//output score information to output pins
if (t1_score > t2_score){

  digitalWrite(team_out, HIGH);
  score_diff = t1_score - t2_score;
  Serial.print("team 1: ");
}
else if (t2_score > t1_score){
  digitalWrite(team_out, LOW);
  score_diff = t2_score - t1_score;
  Serial.print("team 2: ");
}
score_3 = HIGH;
delay(10);
score_3 = LOW;

if (score_diff == 0){
  Serial.println("0");
  digitalWrite(score_0, LOW);
  digitalWrite(score_1, LOW);
  digitalWrite(score_2, LOW);
  digitalWrite(score_3, LOW);
}
else if (score_diff == 1){
  Serial.println("1");
  digitalWrite(score_0, HIGH);
  digitalWrite(score_1, LOW);
  digitalWrite(score_2, LOW);
  digitalWrite(score_3, LOW);
}
else if (score_diff == 2){
  Serial.println("2");
  digitalWrite(score_0, LOW);
  digitalWrite(score_1, HIGH);
  digitalWrite(score_2, LOW);
  digitalWrite(score_3, LOW);
}
else if (score_diff == 3){
  Serial.println("3");
  digitalWrite(score_0, HIGH);
  digitalWrite(score_1, HIGH);
  digitalWrite(score_2, LOW);
  digitalWrite(score_3, LOW);
}
else if (score_diff == 4){
  Serial.println("4");
  digitalWrite(score_0, LOW);
  digitalWrite(score_1, LOW);
  digitalWrite(score_2, HIGH);
  digitalWrite(score_3, LOW);
}
else if (score_diff == 5){
  Serial.println("5");
  digitalWrite(score_0, HIGH);
  digitalWrite(score_1, LOW);
  digitalWrite(score_2, HIGH);
  digitalWrite(score_3, LOW);
}
else if (score_diff == 6){
  Serial.println("6");
  digitalWrite(score_0, LOW);
  digitalWrite(score_1, HIGH);
  digitalWrite(score_2, HIGH);
  digitalWrite(score_3, LOW);
}

print_info = 0;
```

```
}


void read_tags()
{
  if (nano.check() == true) //Check to see if any new data has come in from module
  {
    byte responseType = nano.parseResponse(); //Break response into tag ID, RSSI, frequency, and timestamp

    if (responseType == RESPONSE_IS_KEEPALIVE)
    {
      // Serial.println(F("Scanning"));
    }
    else if (responseType == RESPONSE_IS_TAGFOUND)
    {
      //If we have a full record we can pull out the fun bits
      int rssi = nano.getTagRSSI(); //Get the RSSI for this tag read
      long freq = nano.getTagFreq(); //Get the frequency this tag was detected at
      long timeStamp = nano.getTagTimestamp(); //Get the time this was read, (ms) since last keep-alive
message
      byte tagEPCBytes = nano.getTagEPCBytes(); //Get the number of bytes of EPC from response
      //Print EPC bytes, this is a subsection of bytes from the response/msg array
      int myEPC[12] = {0,0,0,0,0,0,0,0,0,0,0,0};
      for (byte x = 0 ; x < tagEPCBytes ; x++)
      {
          myEPC[x] = (nano.msg[31 + x]);
      }

      bool tag_true = true;
      for (int x = 0; x < 10; x++)
      {
        tag_true = true;
        for (int y = 0; y < 12; y++)
        {
          if (myEPC[y] != tag1[x][y])
          {
            tag_true = false;
          }
        }
        if (tag_true == true)
        {
            EPClist[x] = 1;
            rssilist[x] = rssi;
        }
      }
      //print current tag information during continous read
      Serial.print("[");
      for (int x = 0; x < 10; x++)
      {
        if (EPClist[x] == 1)
        {
          if (rssilist[x] < -45)
              Serial.print("Tag ");
              Serial.print(x+1);
              Serial.print("(");
              Serial.print(rssilist[x]);
              Serial.print(")");
              Serial.print(", ");
        }
      }
      Serial.println("]");
    }
    else
    {
      //Unknown response
      // Serial.print("Unknown error");
    }
  }
}

//Gracefully handles a reader that is already configured and already reading continuously
//Because Stream does not have a .begin() we have to do this outside the library
boolean setupNano(long baudRate)
{
  nano.begin(softSerial); //Tell the library to communicate over software serial port

  //Test to see if we are already connected to a module
```

```
  //This would be the case if the Arduino has been reprogrammed and the module has stayed powered
  softSerial.begin(baudRate); //For this test, assume module is already at our desired baud rate
  while (softSerial.isListening() == false); //Wait for port to open

  //About 200ms from power on the module will send its firmware version at 115200. We need to ignore this.
  while (softSerial.available()) softSerial.read();

  nano.getVersion();

  if (nano.msg[0] == ERROR_WRONG_OPCODE_RESPONSE)
  {
    //This happens if the baud rate is correct but the module is doing a ccontinuous read
    nano.stopReading();

    Serial.println(F("Module continuously reading. Asking it to stop..."));

    delay(1500);
  }
  else
  {
    //The module did not respond so assume it's just been powered on and communicating at 115200bps
    softSerial.begin(115200); //Start software serial at 115200

    nano.setBaud(baudRate); //Tell the module to go to the chosen baud rate. Ignore the response msg

    softSerial.begin(baudRate); //Start the software serial port, this time at user's chosen baud rate

    delay(250);
  }

  //Test the connection
  nano.getVersion();
  if (nano.msg[0] != ALL_GOOD) return (false); //Something is not right

  //The M6E has these settings no matter what
  nano.setTagProtocol(); //Set protocol to GEN2

  nano.setAntennaPort(); //Set TX/RX antenna ports to 1

  return (true); //We are ready to rock
}
```

Appendix E. Terminal Baud Rate Instructions

1. Follow this path in Modustoolbox 2.1: wiced_btsdk > dev_kit > libraries > btsdk_mesh > component > mesh_app_hci.c
2. In mesh_app_hci.c command find "921600" and change that number to 115200 (it should be for puart), save and build
3. Reprogram the boards

Appendix F. Instructions on how to open MeshClient on Windows

Follow this file path in File Explorer: users > mtw > wiced_btsdk > tools > btsdk-peer-apps-mesh > Windows > Meshclient > Release > x86 > MeshClient.exe

Note: mtw is the name of the workspace used