**Training Claymore Report**



A picture containing indoor, wall, table, floor

Description generated with very high confidence

Sam Skinner and Sam Malhotra

**Summary**

This device is a training claymore that flashes lights and makes noise when triggered. A claymore is a manually triggered directional explosive device used by the U.S. military. It uses C-4 explosives to launch steel ball bearings into a wide spray like a shotgun. Baylor Army ROTC uses training claymore during tactical exercises, but these claymores are just blocks of wood and do not do anything when “triggered.” In order to enhance the training by making a claymore that actually had effect when triggered, this device was created.

This device mimics a real claymore but uses lights and sound to simulate the explosion instead of exploding. The device also does not use wire connecting to a physical trigger. Instead, it uses wireless capabilities with a webpage that allows the user to select from a manual trigger and motion sensing. Using a cell phone, the user can connect to a BeagleBone Black Wireless through the BeagleBone’s Wi-Fi. From there, the user can open a webpage and select whether to trigger the claymore manually or use motion sensing to trigger it. When it is triggered, 4 LED panels flash and 4 small magnetic buzzers emit sound for ten seconds.

**Objectives**

1. Portable: The device will be used in a field environment. It needs to be carried to the training location with just the use of a backpack.
2. Effects on “enemy”: When the device is triggered, it should be noticeable to everyone in the area, and it should be mildly shocking to the “enemy” in the training scenario.
3. Triggered from a webpage: The device should be wirelessly triggered by connecting to the BeagleBone’s Wi-Fi and opening a webpage with the trigger method and a button for the manual trigger.
4. Durable: The device needs to be used in harsh conditions including rain and being roughly handled. The device should continue to operate regardless of the environment it is in.

**Hardware Platform**

The BeagleBone Black Wireless (BBBw) is the hardware platform for this device. Although it has much higher battery consumption than a microcontroller, the BBBw has built in Wi-Fi capabilities that microcontrollers do not. This device requires the ability to connect wirelessly to the BBBw and to connect to a webpage using the BBBw’s Wi-Fi. Power consumption should not be an issue because the device only needs to be powered for a couple of hours at a time. Besides the webpage, the inputs and outputs of the device are fairly simple and do not take much extra effort to connect to the BBBw.

**Hardware Design**

Figure 1 below shows the hardware design of our entire device.

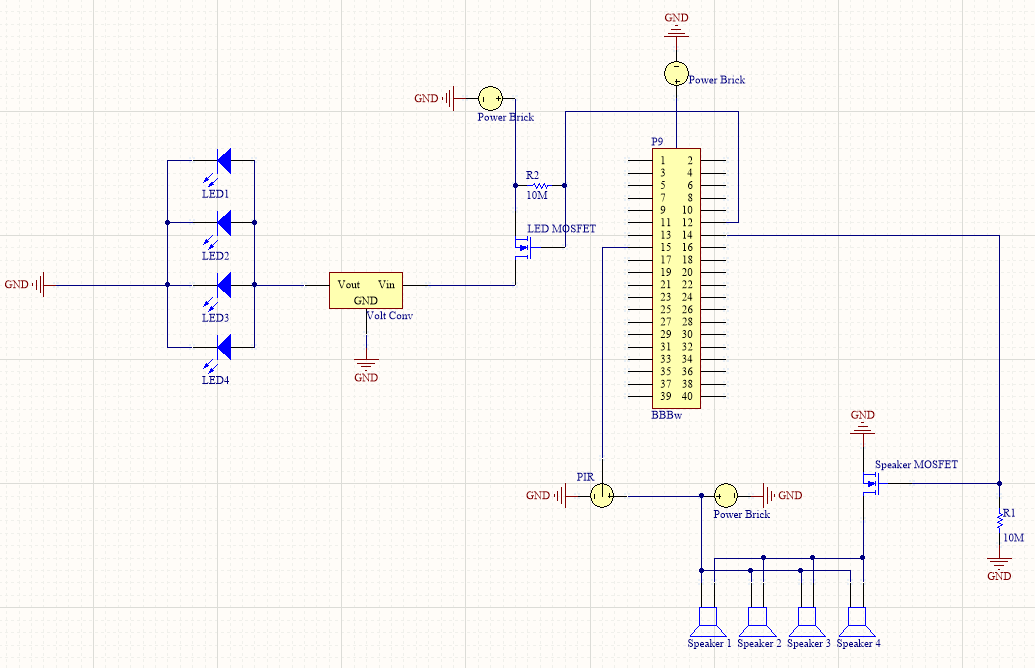


Figure 1 - Hardware design schematic

All components on this project were powered by a power brick that was capable of providing 4.8V out of each of its three power ports simultaneously. This power source was chosen for its ability to provide up to 22Ah as well as its potential to power multiple systems at the same time. In this case, the brick powered a BBBw, the LED system, and the PIR and magnetic buzzer system. The power brick is also very easy to charge; requiring only a wall plug to micro-USB similar to common cell phone chargers today. The BBBw received power directly from the power brick through power port 1.

The four LED panels were connected in parallel and received power from the power brick through power port 2. However, to achieve full brightness of the LEDs, a voltage converter was used between the power brick and LEDs to boost the voltage from 4.8V to 12V. A PSMN2R0-30PL MOSFET acted as the switch between the power brick and LEDs and was controlled by the P9.12 pin on the BBBw. The LEDs pulled a combined current of about 5A, so the purpose of the MOSFET was to isolate the LEDs from the BBBw due to current and voltage restrictions for the BBBw.

In a similar fashion, the PIR and magnetic buzzers received their power from power port 3. Again, a MOSFET was used as a switch while also isolating the BBBw from dangerously high currents required by the magnetic buzzers.

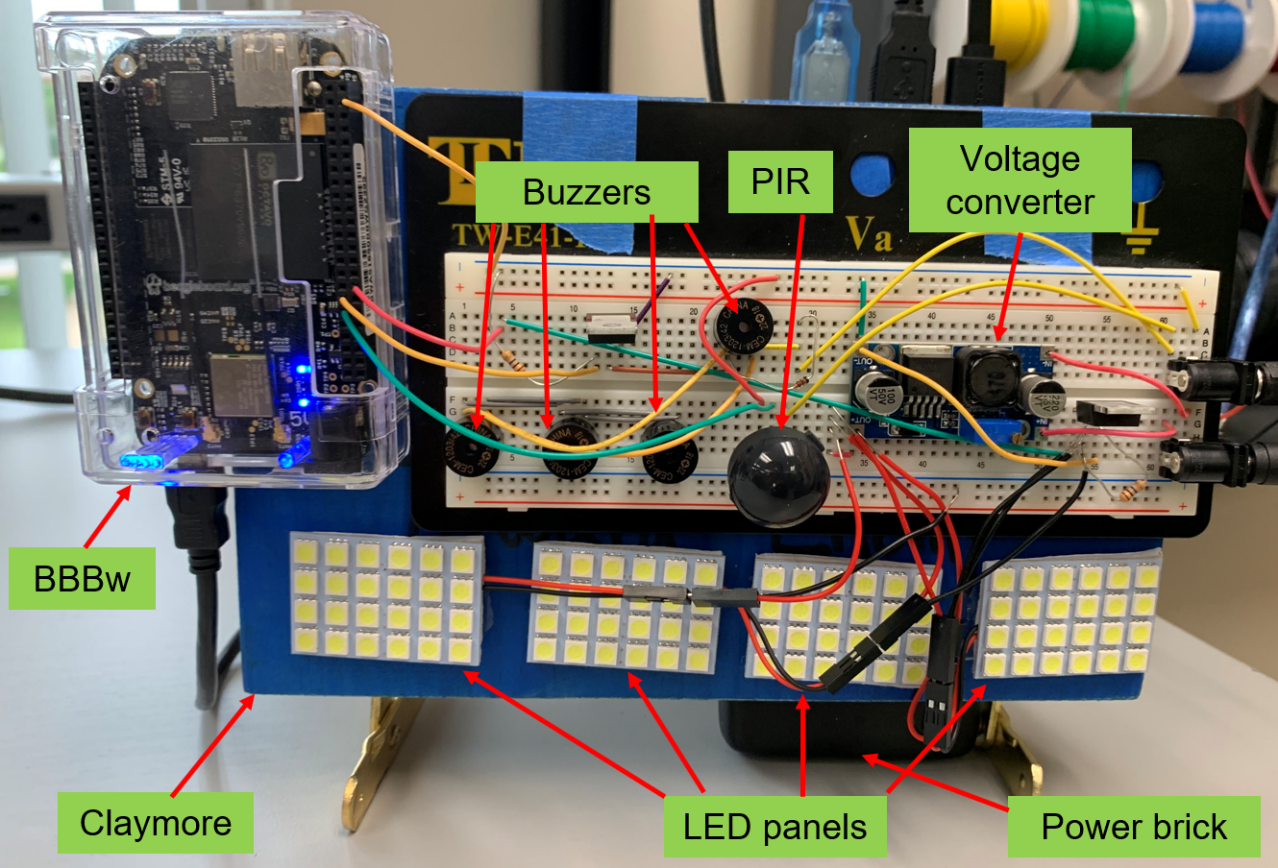


Figure 2 - Hardware implementation

A brief summary of each hardware component is included below, along with part specifications:

1. **Beaglebone Black wireless (BBBw)**

* Has PWM capabilities to power the magnetic buzzers
* GPIO pins were used to toggle the MOSFETs
* Built in WIFI capability which was used to host the web interface

1. **Claymore**

* Made out of wood and steel components
* Simulates approximate dimensions and weight of actual M18 Claymore
  + 8 in x 1 in x 5 in
  + 3 lbs

1. **Magnetic Buzzer -** CEM-1203(42)
   * Four of these buzzers were connected in parallel pulling a combined 140mA
   * Each buzzer was capable of outputting 95dB
2. **4.8V Power brick** **-** RAVPower Portable Charger

* Capable of providing 22000mAh
* Supplies up to 5.8A
* Has three output ports

# LED Panels - Everbright 5-Pack White 5050 24SMD Led Panel

# 288 Lumens output by each panel

# 24 LEDs on each panel for a combined total of 96 LEDs on the Claymore

# Has adhesive on the reverse side for easy installation and replacement

# Voltage Converter - eBoot Boost Converter Module XL6009 DC to DC 3.0-30 V to 5-35 V

* Easy Adjustment knob
* Boosts power brick voltage form 4.8V to 12V to be used by the LEDs
* Capable of boosting voltage to optimal levels for the LED panels

1. **PIR sensor –** EKMC2603112K VZ type long range detection

* Compact component
* 12m detection range, which was the furthest distance we could find within our budget
* Very low power (pulls 0.17mA)

**Software Platform**

This device runs 3 different software languages: C, JavaScript and HTML. The code in C runs the actual processes that the claymore does when it is triggered. The JavaScript and HTML create the webpage and pass information from it to the C code. C was chosen because all the effects that needed to happen already had code written. This code just needed to be slightly changed to fit the application. JavaScript and HTML were chosen because they were the simplest languages to create a webpage for this device.

**Software Design**

Figure 3 shows the high-level software design for this device. The HTML code creates a simple webpage displayed in figure 4. From this webpage, the user has three options to click on. When the user clicks on one of the buttons, the JavaScript code runs. If ‘manual’ or ‘motion sensing’ is selected, the JavaScript code writes a value of ‘0’ or ‘1’ respectively to a text file. The C code then reads this text file and determines which code to run based on the text file’s contents. If the trigger method is ‘manual,’ the C code polls a different text file to see if the manual ‘TRIGGER’ has been pressed. Once the trigger is pressed, the C code sends a signal to the lights and buzzers to alternate for 10 seconds at a frequency of 10 Hz. The lights are turned on and off by outputting high or low to the pin that connects to the MOSFET to the lights. The buzzers are turned on and off by outputting a PWM signal to them. After the 10 seconds have passed, the text file containing the manual trigger is rewritten to ‘0’ and the claymore can be activated again.

If ‘motion’ is chosen instead of ‘manual,’ the C code polls the pin connected to the PIR sensor. If the PIR sensor sees movement, it brings the pin that it is attached to high. The C code will then send the same signals to the lights and buzzers as after the manual trigger is pressed. After the 10 seconds have passed, the claymore is reset to the manual trigger method.



Figure 3 – High-level software design

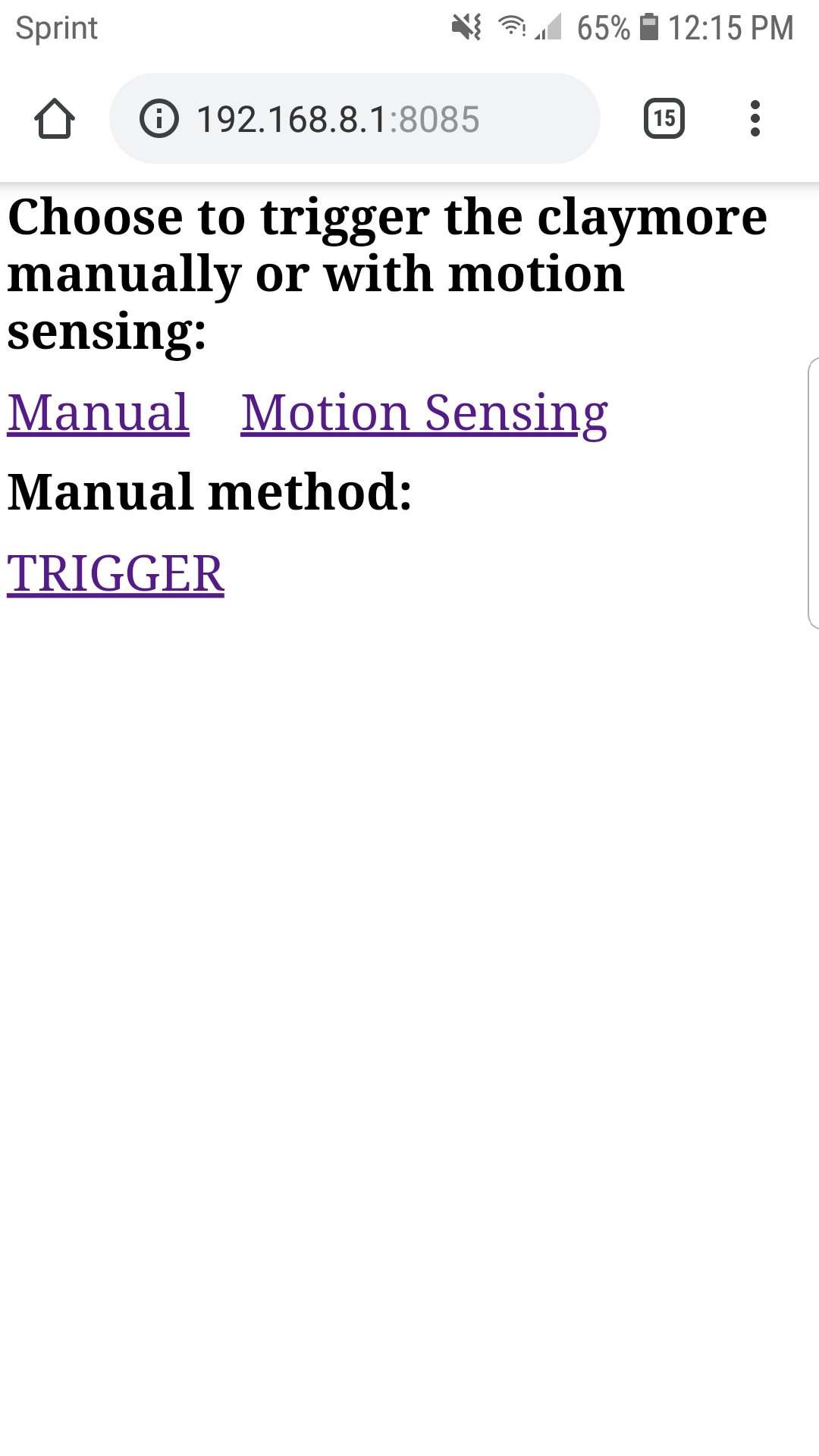


Figure 4 – Claymore webpage

# Test Results

This project successfully met our design criteria and fulfills its operating parameters. The webpage is easy to navigate and allows the user to successfully choose the trigger method (manual or PIR) and then trigger the device. The LEDs and magnetic buzzers provide a noticeable alarm that alerts the operator and anyone else with sight and sound of the system that it has been triggered. Through trial and error, it was estimated that the WIFI signal output by the BBBw was functional out to about 50m. This is more than enough range for the claymore to be useful in a training environment. It was also estimated that the PIR sensor was capable of being triggered by targets at a max range of 14m. Based on the rated angle of detection and range of the sensor, it was calculated that the PIR had an active detection area of 189 square feet on the ground in front of it.

**Lessons Learned**

1. Buy parts early: Much of the time leading up to the completion of the project was wasted because there were no parts for us to work with. If we had bought parts sooner, we could have had much more time to make the end product more polished.
2. Start unfamiliar tasks first: Making a webpage was totally new to us, so it took a much longer time to finish that section of the code than the C code. If we had started with the webpage, we would have realized sooner that it would be best to write all the code in JavaScript instead of having to use text files as an intermediary between JavaScript and C.
3. Pay attention to the little parts too: We had to splice some wires to make a USB to power plug connector because we did not think ahead far enough to realize that we would need to buy them.
4. Don’t stare at the lights: They’re bright.

**Final Notes**

Running on startup: In order to get the webpage to run as soon as the BeagleBone boots, put BUAROTC\_claymore.html in the autorun folder of cloud9. The other code files need to be in a folder under cloud9 named “claymore”. In order to make the C code run on startup, add the follow lines of code to rc.local:

sleep 30

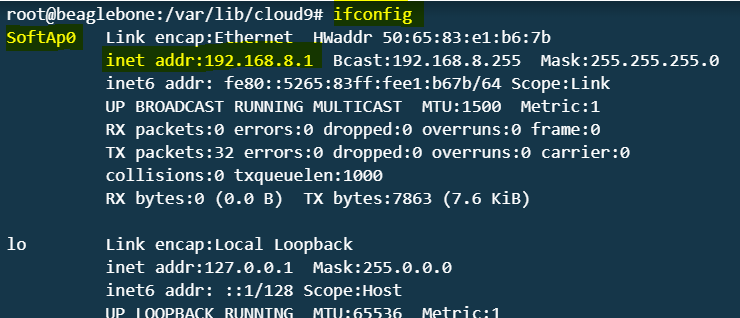
cd /var/lib/cloud9/claymore/

./claymore &

The sleep command gives the BeagleBone extra time to set up its pins before running the code. Without the sleep, the code sometimes will not work.

Connecting to the webpage:

Find the BeagleBone’s internet address for the Wi-Fi using the command ifconfig



Connect to the BeagleBone Wi-Fi. The password is BeagleBone

Open and internet browser and enter the internet address followed by port 8085 into the address bar (e.g. 192.168.8.1:8085)

Making the claymore: The specifications and dimensions for making a claymore can be found in the Army Publishing Directorate under TC 3-22.23. Here is the link to that document: <https://github.com/SamuelSkinner/Claymore/blob/master/datasheets/tc3_22x23.pdf>

**Code**

BUAROTC\_claymore

|  |
| --- |
| <html> |
|  | <head> |
|  | <script src = "/socket.io/socket.io.js" > </script> |
|  | <script> |
|  | var socket = io.connect(); |
|  | function manual() { |
|  | socket.emit('method\_choice', '0'); |
|  | } |
|  | function motion() { |
|  | socket.emit('method\_choice', '1'); |
|  | } |
|  | function explode() { |
|  | socket.emit('explosion', '1'); |
|  | } |
|  | </script> |
|  | </head> |
|  |  |
|  | <body> |
|  | <h1 style="font-size:30pt;">Choose to trigger the claymore manually or with motion sensing:</h1> |
|  | <a href="" onClick="manual();" style="font-size:32pt;">Manual</a> |
|  | &nbsp;&nbsp;&nbsp; |
|  | <a href="" onClick="motion();" style="font-size:32pt;">Motion Sensing</a> |
|  | <h1 style="font-size:30pt;">Manual method:</h1> |
|  | <a href="" onClick="explode();" style="font-size:32pt;">TRIGGER</a> |
|  | </body> |
|  | </html> |

BUAROTC\_claymore.js

|  |
| --- |
| #!/usr/bin/env node |
|  | var b = require('bonescript'); |
|  | var app = require('http').createServer(handler); |
|  | var io = require('/home/debian/web\_example/node\_modules/socket.io').listen(app); |
|  | var fs = require('fs'); |
|  |  |
|  | var htmlPage = '/var/lib/cloud9/claymore/BUAROTC\_claymore.html'; |
|  | app.listen(8085); |
|  |  |
|  | var method = 0; |
|  | var boom = 0; |
|  | var res = 0; |
|  |  |
|  |  |
|  | function handler(req, res) { |
|  | fs.readFile(htmlPage, function(err, data) { |
|  | if (err) { |
|  | res.writeHead(500); |
|  | return res.end('Error loading file: ' + htmlPage); |
|  | } |
|  | res.writeHead(200); |
|  | res.end(data); |
|  | }); |
|  | } |
|  |  |
|  | function onConnect(socket) { |
|  | socket.on('method\_choice', handleMethodChoice); |
|  | socket.on('explosion', handleExplosion); |
|  | } |
|  |  |
|  | function handleMethodChoice(message) { |
|  | method = message; |
|  | fs.writeFile('/var/lib/cloud9/claymore/Method.txt', method, (err) => { |
|  | if (err) throw err; |
|  | }); |
|  | } |
|  |  |
|  | function handleExplosion(message) { |
|  | boom = message; |
|  | fs.writeFile('/var/lib/cloud9/claymore/Trigger.txt', boom, (err) => { |
|  | if (err) throw err; |
|  | }); |
|  | } |
|  |  |
|  | io.sockets.on('connection', onConnect); |

main.c

|  |
| --- |
| #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <BBBiolib.h> |
|  | #include <signal.h> |
|  |  |
|  | void sig\_handler(int signo); |
|  | void initialize\_pwm(); |
|  | void motion(); |
|  | void manual(); |
|  |  |
|  | void main() |
|  | { |
|  | if (signal(SIGINT, sig\_handler) == SIG\_ERR) |
|  | printf("\ncan't catch SIGINT\n"); |
|  | else |
|  | printf("Successfully set up SIGINT handler\n"); |
|  |  |
|  | iolib\_init(); |
|  | initialize\_pwm(); |
|  |  |
|  | iolib\_setdir(9,15, BBBIO\_DIR\_OUT); |
|  | iolib\_setdir(9,12, BBBIO\_DIR\_IN); |
|  |  |
|  | while (1) |
|  | { |
|  | FILE \*fp; |
|  | int buff; |
|  | fp = fopen("/var/lib/cloud9/claymore/Method.txt", "r"); |
|  | fscanf(fp, "%d", &buff); |
|  | fclose(fp); |
|  | int method = buff; |
|  | if (method == 0) |
|  | { |
|  | manual(); |
|  | } |
|  | else if (method == 1) |
|  | { |
|  | motion(); |
|  | } |
|  | usleep(1000); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | void sig\_handler(int signo) |
|  | { |
|  | system("echo 0 > /sys/class/pwm/pwmchip2/pwm0/duty\_cycle"); |
|  | pin\_low(9,15); |
|  | iolib\_free(); |
|  | exit(0); |
|  | } |
|  |  |
|  |  |
|  | void motion() |
|  | { |
|  | int count = 0; |
|  | int even = 1; |
|  |  |
|  | if (is\_high(9,12)) |
|  | { |
|  | while (count < 200) |
|  | { |
|  | if (even == 1) |
|  | { |
|  | pin\_high(9,15); |
|  | system("echo 0 > /sys/class/pwm/pwmchip2/pwm0/duty\_cycle"); |
|  | even = 0; |
|  | } |
|  | else if (even == 0) |
|  | { |
|  | pin\_low(9,15); |
|  | system("echo 350000 > /sys/class/pwm/pwmchip2/pwm0/duty\_cycle"); |
|  | even = 1; |
|  | } |
|  | count ++; |
|  | usleep(50000); |
|  | pin\_low(9,15); |
|  | system("echo 0 > /sys/class/pwm/pwmchip2/pwm0/duty\_cycle"); |
|  | } |
|  | FILE \*fp; |
|  | int buff; |
|  | fp = fopen("/var/lib/cloud9/claymore/Method.txt", "w"); |
|  | fputc('0', fp); |
|  | fclose(fp); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | void manual() |
|  | { |
|  | FILE \*fp; |
|  | int buff; |
|  | fp = fopen("/var/lib/cloud9/claymore/Trigger.txt", "r"); |
|  | fscanf(fp, "%d", &buff); |
|  | fclose(fp); |
|  |  |
|  | int boomboom = buff; |
|  | int count = 0; |
|  | int even = 1; |
|  |  |
|  | if (boomboom == 1) |
|  | { |
|  | while (count < 200) |
|  | { |
|  | if (even == 1) |
|  | { |
|  | pin\_high(9,15); |
|  | system("echo 0 > /sys/class/pwm/pwmchip2/pwm0/duty\_cycle"); |
|  | even = 0; |
|  | } |
|  | else if (even == 0) |
|  | { |
|  | pin\_low(9,15); |
|  | system("echo 350000 > /sys/class/pwm/pwmchip2/pwm0/duty\_cycle"); |
|  | even = 1; |
|  | } |
|  | count ++; |
|  | usleep(50000); |
|  | pin\_low(9,15); |
|  | system("echo 0 > /sys/class/pwm/pwmchip2/pwm0/duty\_cycle"); |
|  | } |
|  | } |
|  | fp = fopen("/var/lib/cloud9/claymore/Trigger.txt", "w"); |
|  | fputc('0', fp); |
|  | fclose(fp); |
|  | } |
|  |  |
|  |  |
|  | void initialize\_pwm() |
|  | { |
|  | FILE \*fp; |
|  | char \*mode = "r+b"; |
|  |  |
|  | // Configure P9\_14 as a PWM pin |
|  | system("config-pin P9.14 pwm"); |
|  |  |
|  | // Wait for 10ms to allow changes to take effect |
|  | usleep(10000); |
|  |  |
|  | // Check to see if the PWM0 directory has been created by attempting |
|  | // to open the pwm0/enable file. If the file exists, then you should |
|  | // not echo 0 to the export file. If it does not exist, then you need |
|  | // to echo 0 to the export file to create the pwm0 directory |
|  | fp = fopen("/sys/class/pwm/pwmchip2/pwm0/enable", mode); |
|  |  |
|  | if (fp == NULL) |
|  | { |
|  | printf("Creating PWM0 for the first time\n"); |
|  | system("echo 0 > /sys/class/pwm/pwmchip2/export"); |
|  | // Wait for 10ms to allow changes to take effect |
|  | usleep(10000); |
|  | } |
|  | else |
|  | { |
|  | fclose(fp); |
|  | } |
|  |  |
|  | // Turn the pwm on |
|  | system("echo 1 > /sys/class/pwm/pwmchip2/pwm0/enable"); |
|  | // Wait for 10ms to allow changes to take effect |
|  | usleep(10000); |
|  |  |
|  | // Set the PWM period to 500000 |
|  | system("echo 500000 > /sys/class/pwm/pwmchip2/pwm0/period"); |
|  | // Wait for 10ms to allow changes to take effect |
|  | usleep(10000); |
|  | } |