IVEND – INFRARED DETECTION

Team Dec15-10

with Dr. Wang and Fawn Engineering

Tyler Westerberg
Tim Howell
Jacob Olstad
Steve Dorenkamp
Seth Schmidt

Contents

Project Design

The system we are dealing with in this project has two main parts, one composed of hardware, and the second composed of software. The hardware element is our PCB (see appendix B) boards. These boards are responsible for detecting when a product is vended and passes through the IR field. Once this is detected, our software element needs to alert the vending machine's separate on-board controller. The system should be able to detect an item with dimensions 1.912" x 3.029" x 0.028" or a sphere with diameter 0.338". The system operates by modulating one led at a time at 38 KHz. The system then uses the IR detectors to see if the IR light has crossed the bin. If the IR light is present the system then turns off the LED and moves on to the next. If the IR light is not present the system considers this a detection and will signal to the main vending machine controller that an object has been detected.

Hardware:

The boards that we developed for the new system has several key components. They have IR emitters and sensors for detecting a product and they have an LED (see appendix B) for notifying when a product has been detected. The IR emitters essentially create a light curtain that is detected by the sensors. Emitters for these boards were chosen so that they are capable of sending IR streams to each sensor, but that they are narrow enough to cut down the reflection from the tray. When the sensors do not detect an IR stream for a given amount of time, they will pull the output signal to active to indicate a product has been vended. When the output signal is pulled active, the LED will switch to the off position, indicating that the IR curtain has been broken. This light will remain off until the IR curtain is no longer blocked. The amount of emitters and sensors, as well as the configuration, was decided based upon that which will achieve the greatest results and keep costs low.

Firmware:

The firmware will not be notified when a product is vended, and therefore will operate continuously. If the IR light is seen by the detectors the firmware will consider this a no vend. If the IR light is not detected the system will communicate back that an additional vend is required. On startup the firmware will be responsible for calibrating the LEDs. This will be done by adjusting the voltage they see. The firmware will start at the lowest voltage available and step it up until the detectors can see the LEDs. Once this is accomplished the firmware will resume normal operation.

Hardware

Block Diagram

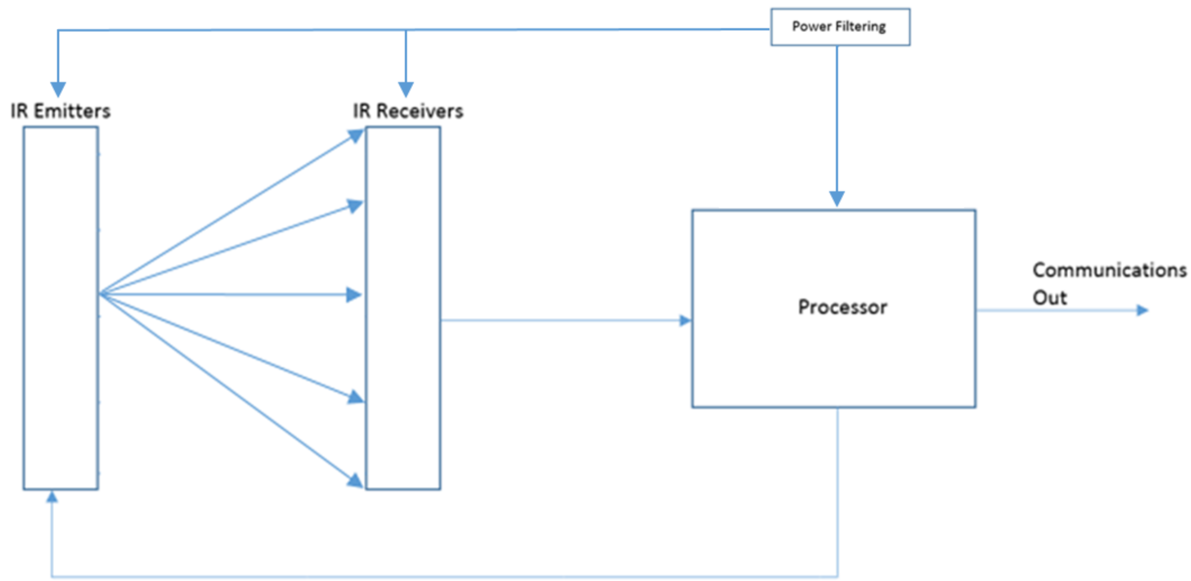

Figure 1 Block Diagram of the System

Schematic

See Appendix IV

PCB Boards

There are two, two layer PCB boards. The first board holds the 10 IR detectors, the processor, and the communication back to the main vending machine system. It also holds a visual indicator of when an object is detected. The second board holds the 10 IR LEDs. There is a cable that connects both boards.

LPC1111

The LPC1111 is the main processing unit for or project. It is responsible for modulating the IR LEDs and determining when an object has passed through the IR mesh based on sensor feedback. The LPC1111 will also blink an LED and pull a signal line low when an object is detected. The LPC1111 was chosen due to its low cost and high performance for the application. This should allow the system to grow in the future without the need for a new microcontroller.

Figure 2 LPC11xx

IR LEDs

The IR LEDs are modulated at 38 KHz and are responsible for creating an IR mesh that an object will break when falling through. They sit on the opposite end of the bin as the detectors and only one will be switched on at a time.

Figure 3 IR LED

RED LED

A single red LED will be present on the detection board so that operators can have a visual indication of when an object is detected. This LED will need to be prominently displayed so the operator can easily see it.

Detectors

The detectors are digital and designed to filter for the 38 KHz signal the IR LEDs are modulated at. We chose the TSSP4038 from Vishay to accomplish this as they are designed to do exactly that.

Figure 4 TSSP4038

Spacing

The detectors and emitters are spaced such as to reduce the area that an item would fall out of line of sight between an emitter detector pair. This was calculated and modeled in Matlab in order to find the ideal number of emitter detector pairs and their ideal spacing. (See Appendix A)

Input/Output

The hardware shall provide one open collector signal output ports. The only input to the board shall be a 5VDC power source.

Firmware

Block Diagram

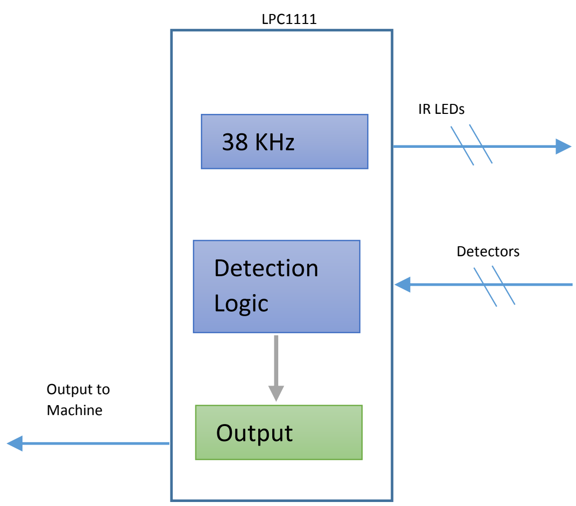

Figure 5 Firmware Block Diagram

Interface

The only human interface that is available is a single red LED that is kept on during normal operation and turned off if an object is detected.

Signal Output

The firmware shall turn on a single GPIO port that will be used to signal when an object is detected. If an object is detected the line is on for 150ms and then turned low again.

LED Modulation

The firmware is responsible for modulating the IR LEDs at a frequency of 38 KHz in order to filter out other IR interference. The LEDs are turned on one at a time in order to make sure that each detector can see each LED. This is accomplished by switching the GPIO (see appendix B) pins to turn on or off certain LEDs. Each pin shall than be switched in and out when needed.

Calibration

The firmware goes through a calibration on start up to set the ideal LED intensity. This is done using our calibration circuit (see Appendix A). The program will set the LEDs to the lowest intensity and check if all detectors can see all LEDs. If it is successful, this is the ideal intensity. If it is unsuccessful, it will increase the intensity slightly and start the check again. This will repeat until the ideal intensity is found.

Object Detection

The firmware processes the signals from the IR detectors in order to determine when an object is or is not blocking the path. This is accomplished by checking the output signal of each detector and verifying it is correct.

Fault Detection

On startup the firmware will be responsible of ensuring that all LEDs can be seen by each detector. If for any reason the firmware cannot accomplish this it will keep the output line low to signal that a fault has been detected.

Implementation Details

Numerous considerations were made while implementing new designs with the current ones. Many of the constraints played a factor to the implementation of the new designs. These constraints include:

- Cost
- Number of emitters and detectors
- Board dimensions
- System dimensions
- Retrofitting
- Reflections inside the system
- Components used
- Light intensity

Retrofitting machines with an updated product detection system was one of the first major concerns during implementation. The dynamics of the system changes with every model of machine, so while the dimensions of the PCBs could not change, the length from the emitters to the detectors would need to be considered. Since light intensity was a major concern, the varying lengths would most likely need a calibration system to compensate for the various systems.

Cost was also an important consideration during design implementations. If there was a need to upgrade the emitters and detectors, the current microprocessor would only support two more emitters and two more detectors due to the remaining GPIO pins available. Any additional emitters/detectors would result in the need to upgrade the microprocessor, which along with the increased cost of emitters and detectors, would also increase the overall cost of the PCBs.

Testing Process and Testing Results

The testing phase of this project was very rigorous. Our aim was to make the infrared matrix work with 100 percent success rate. Since it could take up to a few hundred product deliveries before an error occurred with the original system, much time and consideration was put into testing. It was very critical that our testing method was repeatable and quantitative.

This testing included:

1.) Finding out what components work best at detecting objects
2.) Determining if reflection inside the delivery bin was an issue
3.) Making sure the firmware code is properly checking vended items
4.) Examining outside light sources and testing whether certain lights interfere with the infrared matrix

Diagrams of our testing boards are shown below. We will use these boards to be able to test 360 unique points in the IR array for holes by placing a zip-tie in each whole three times and recording how many times it was detected.

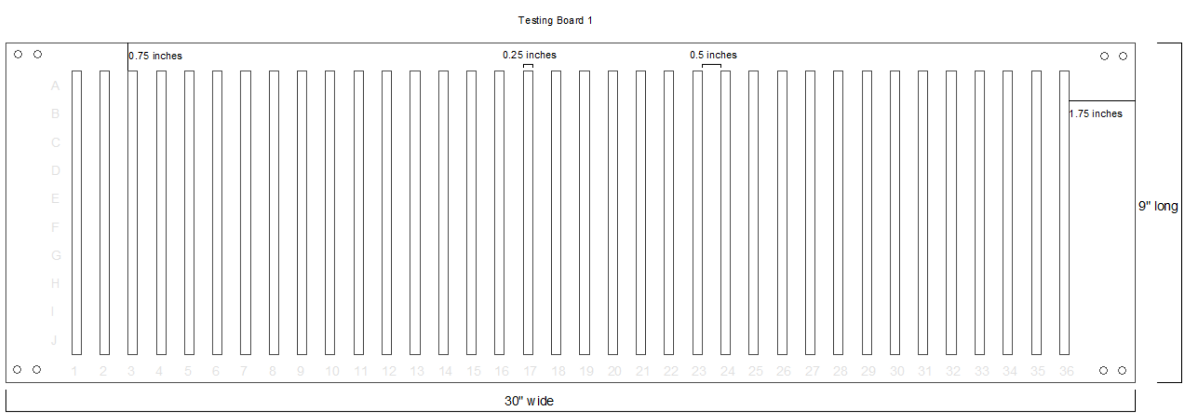

Figure 6 Testing Board 1

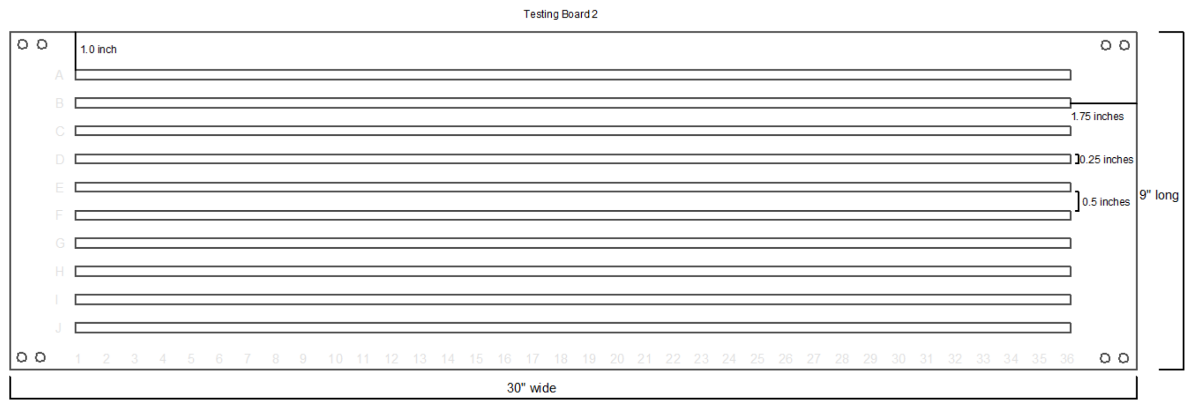

Figure 7 Testing Board 2

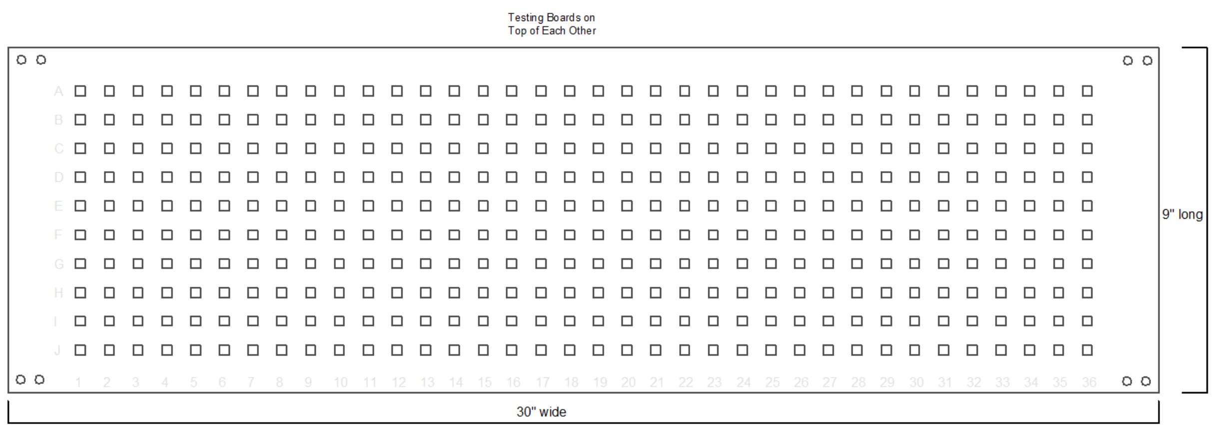

Figure 8 Testing Boards Together

With these testing methods in place we were able to test various different design ideas including simply decreasing the intensity of the LEDs, shrouding the shell of the bin in black cloth to reduce reflections, adding LEDs and detectors, etc. Some of these testing results are shown below:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
A	3	3	3	2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	2	0	0	0	0
B	0	0	0	0	3	2	3	3	2	0	0	0	0	3	2	3	3	0	0	0	2	3	2	0	0	0	0	3	3	3	3	3	3	2	0
C	3	3	3	3	2	2	2	3	3	2	0	3	2	0	2	3	3	2	3	1	2	0	3	0	3	3	0	3	0	0	0	3	3	3	3
D	0	0	0	0	0	2	1	2	3	3	0	0	0	1	1	3	3	3	3	3	1	0	3	0	0	3	3	3	1	0	2	0	0	0	0
E	0	3	2	2	2	2	0	1	0	0	2	0	3	2	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3	3
F	0	0	0	3	3	2	2	3	0	0	3	3	3	1	3	2	3	3	2	1	3	2	3	0	2	3	3	2	3	1	3	3	0	0	0
G	0	0	0	0	1	1	1	3	0	0	3	0	0	1	3	3	2	0	0	2	3	1	0	0	3	3	0	3	2	2	2	3	3	1	0
H	0	0	3	3	1	2	0	0	3	2	2	0	1	0	0	0	0	0	0	0	0	0	0	0	2	3	0	0	1	3	1	2	3	1	0

Figure 9 Test Results from Original iVend – 48.33% Detection

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
A	2	3	3	3	3	1	0	0	1	0	0	2	0	2	0	0	1	3	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3
B	0	0	0	0	0	3	0	3	0	0	2	2	0	1	2	1	3	3	3	3	1	0	0	0	0	3	3	2	3	3	3	3	0	0	0
C	3	3	3	3	3	3	3	3	2	1	3	0	0	2	0	0	1	3	3	3	3	3	3	3	2	3	3	1	1	3	2	3	3	3	3
D	0	0	0	3	3	3	0	3	3	3	3	3	3	3	3	1	2	3	2	3	2	3	3	3	3	3	2	3	3	3	1	0	0	0	0
E	0	0	0	1	3	3	1	1	3	3	3	0	0	3	3	3	3	2	3	3	3	3	3	0	0	2	3	3	3	3	2	3	3	3	0
F	0	1	3	3	3	3	3	3	3	3	3	3	2	3	3	3	0	0	0	0	3	0	3	3	3	3	3	3	3	1	0	3	3	3	0
G	0	0	0	0	0	3	3	3	3	3	3	3	3	3	3	0	0	0	0	0	2	2	3	3	3	3	3	3	3	3	0	0	0	0	0
H	3	3	3	3	3	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	3	3	3	3	3	3

Figure 10 Test Results from Original iVend w/ Reduced Intensity – 62.5% Detection

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
A	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	2	3	3	3	2	2	2	2	2	2	2
D	0	0	0	0	0	0	0	2	2	2	3	3	3	2	2	2	2	3	3	3	3	3	2	2	2	2	2	2	2	2	2	3	3	3	3
E	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F	0	0	0	0	0	0	0	0	0	0	0	0	2	3	3	2	2	2	3	3	2	2	2	2	2	2	3	3	3	3	3	3	3	3	2
G	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	3	3	2	2	2	2	2	2	2
H	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2

Figure 11 Test Results from iVend Prototype w/ 1 LED, 8 Detectors – 32.6% Detection

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
A	3	3	3	3	3	3	3	3	2	0	2	1	1	0	0	0	0	0	1	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
B	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
C	0	0	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
D	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
E	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
F	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
G	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
H	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Figure 12 Test Results from the iVend Redesign – 95.83% Detection

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
A	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
B	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
C	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
D	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
E	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
F	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
G	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
H	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Figure 13 Test Results from the updated iVend Redesign – 99.6% Detection

Our new design is not quite perfect, but some of that can be attributed to a manufacturer defect where the detectors were not spaced as designed. The third detector from the right is incorrect, as seen in figure 12. In figure 13 we changed the code to have 5 detectors reading at a time to allow us to lower the intensity on the emitters thereby increasing our detection in our outside rows. We also moved the misplaced detector to the right position. With the new code, and the corrected detector, we can see that the detection increases to approximately 99.6%.

Appendix I Operation Manual

Parts List

- Detector Board
- Emitter Board
- Detector/Emitter Interconnect Cable (Cable 1)
- Main Computer Board Cable (Cable 2)

Setup

Step 1: Connect one end of cable 1 to the 7-pin header on the detector board.

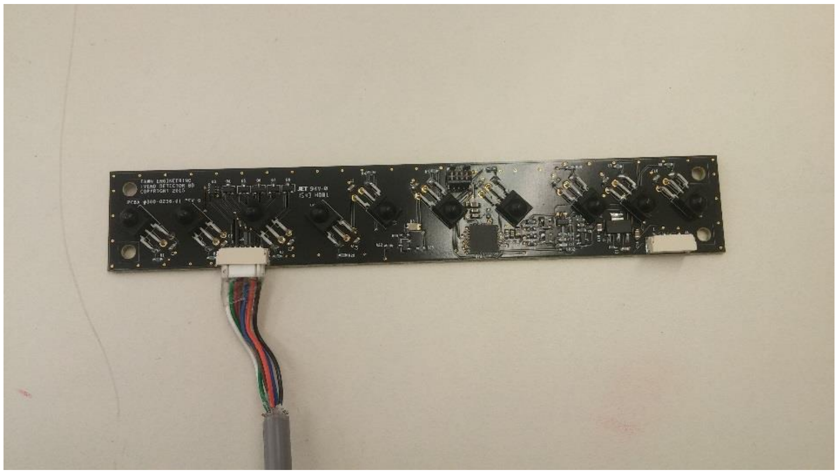

Figure 14 – Setup Step 1

Step 2: Connect the corresponding end of cable 2 to the 6-pin header of the detector board.

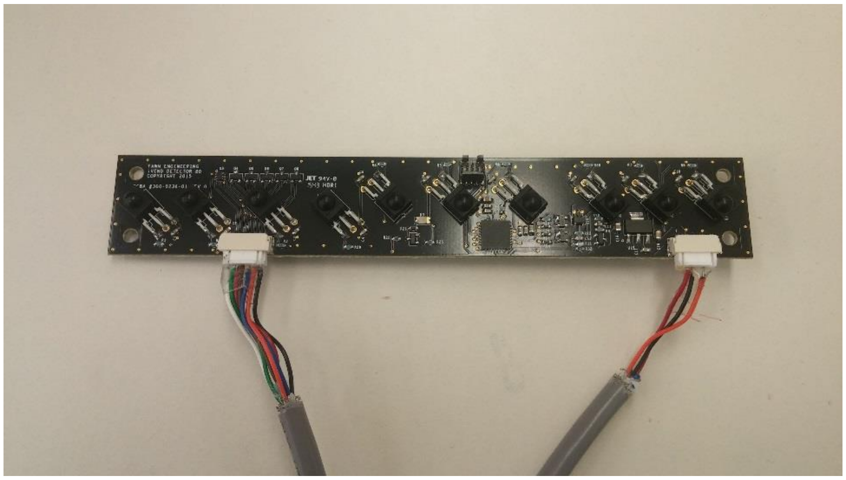

Figure 15 – Setup Step 2

Step 3: Mount the detector board on the white plastic stand-offs on the side of the bin closest to the main controller board, with the detectors facing into the bin.

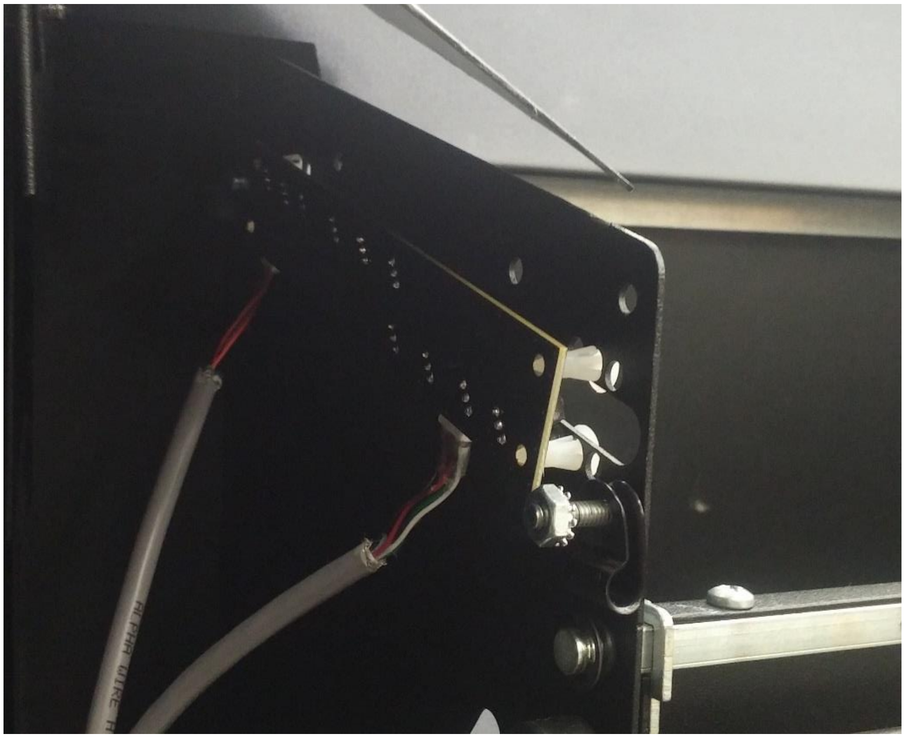

Figure 16 – Setup Step 3

Step 4: Run the loose end of cable 1 through the cable routing slot provided.

Figure 17 – Setup Step 4

Step 5: Connect the end of the cable you've just routed to the header on the emitter board.

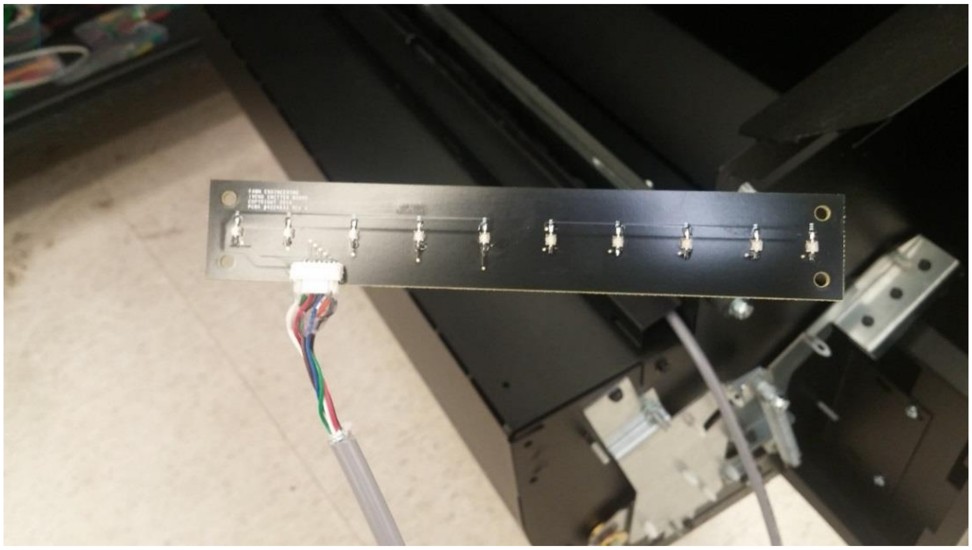

Figure 18 – Setup Step 5

Step 6: Mount the emitter board on the white plastic stand-offs across from the emitter board, with the emitters facing into the bin.

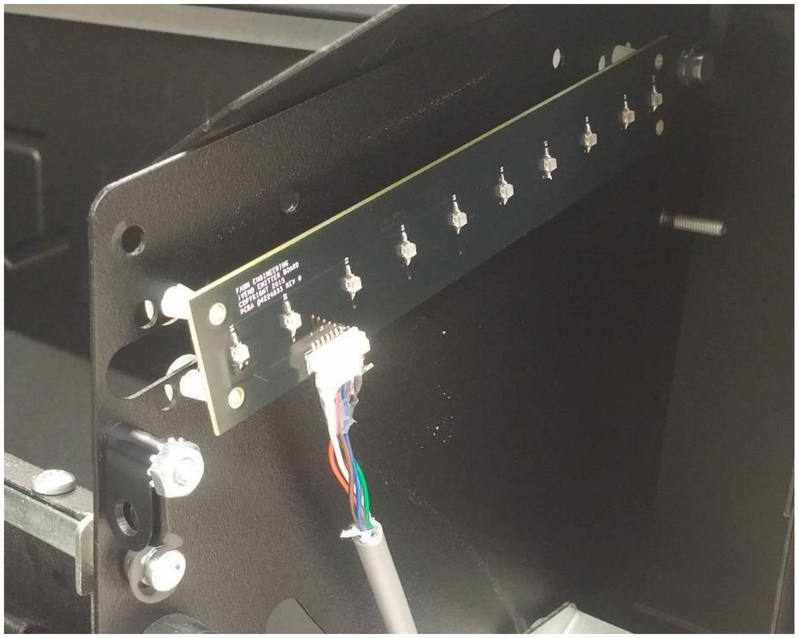

Figure 19 – Setup Step 6

Step 7: Connect the loose end of cable 2 to the header on the main controller board.

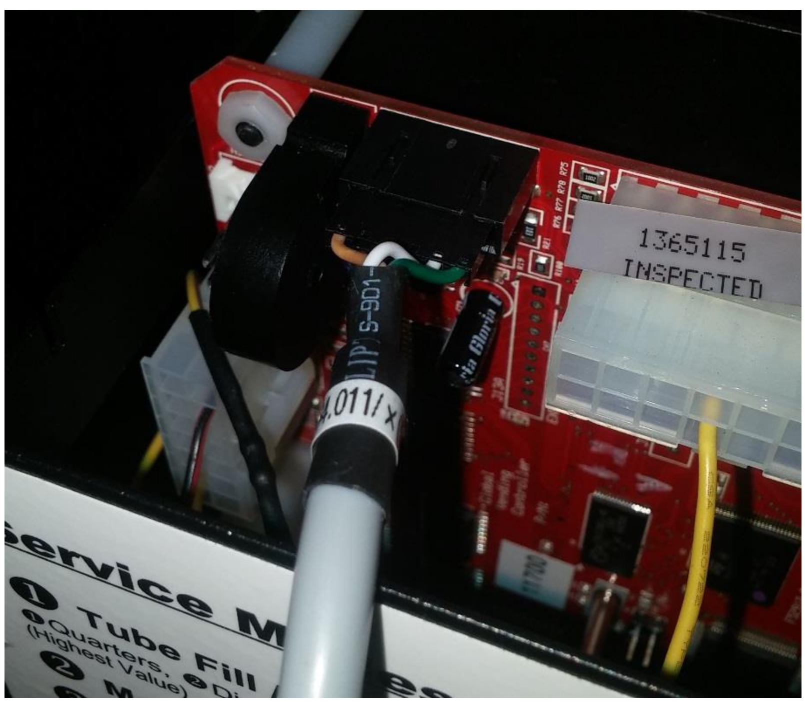

Figure 20 – Setup Step 7

When Setup is Complete

From this point on, the system should run automatically. You can verify the iVend is functioning by turning on the power and moving an object in and out of the detection zone and watching the red LED mounted on the detector board. The LED should turn off or blink when an object enters the field.

Appendix II: Alternative Versions:

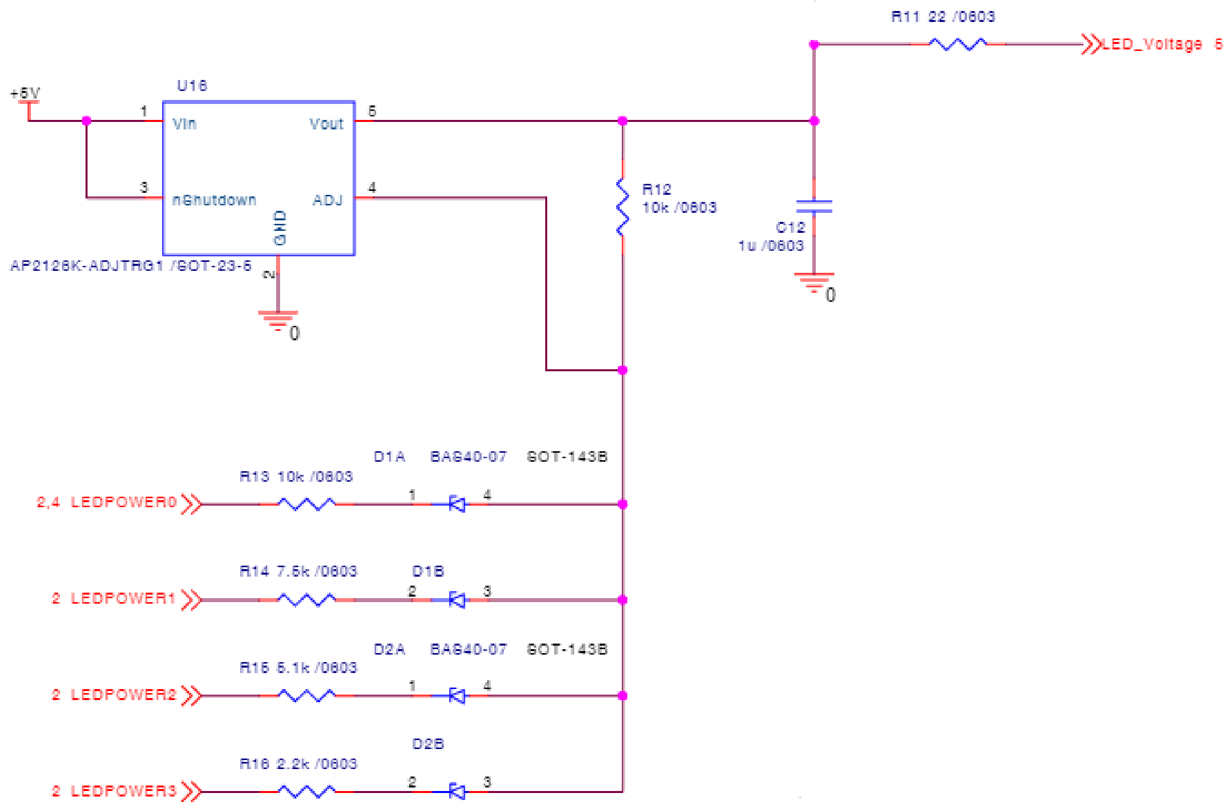

Figure 21 Schematic of Calibration Circuit to be used in our system

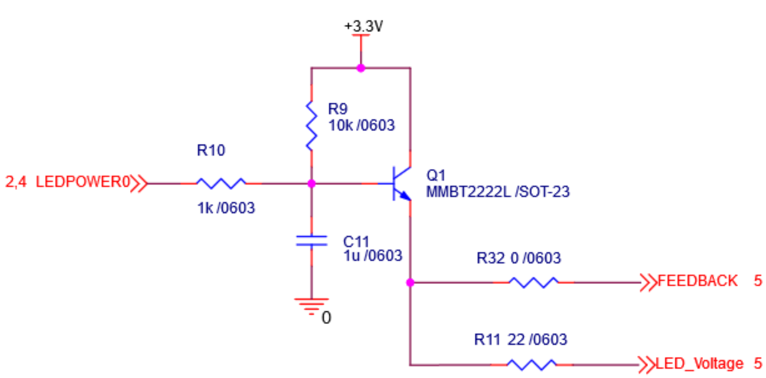

Figure 22 Schematic of Proposed Alternate Calibration Circuit

One idea of the design was to use all detectors all the time which as specified earlier gave us a 95.83% detection rate. Our current design we are planning to use increases our detection rate to 97.86% by only using 5 detectors at a time. The system works by utilizing the detector directly across from the currently pulsing LED and the 2 detectors on either side. If an outside LED is on, then we only use the detector directly across from it and the 2 detectors inside of it. This allows us to reduce the power through each LED by not requiring all detectors to see the current emitter that is on, thereby minimizing possible reflections.

Appendix III: Other Considerations:

While we were pleased with our final testing results, that showed an improvement in detection rate to 95.83%, we feel as though this is actually lower than what the system can obtain for several reasons. The final boards that were provided to us had a manufacturer defect in which one of the detectors was misplaced. This can be seen in our test results where we have a hole in the first few columns of row C. Once this detector is placed in the correct location, we believe the hole will be eliminated, raising our detection rate up to 96.55%. For the system in which we only view 5 detectors at a time, we believe our detection rate will get raised to 100% with proper part placement.

We also feel the detection rate could be increased if a larger object was used. The test results we obtained were completed with a screw with a diameter of roughly 0.25”. If we were to use an object that was the same size as the minimum requirements, 0.338”, we feel as though the detection rate would also increase. An object with a larger diameter was not tested because the testing board would not fit an object that large through its holes, so an additional testing system would have had to be created.

This project was unique in that it was a revision to a current design, rather than a project that asked us to develop a system from scratch based upon their needs. The importance of this was mainly in the fact that the new system would replace the current system in their machines that are already in used. Because of this, we needed to improve upon their current design, rather than having the option to go in a completely different direction.

Having contacts at Fawn Engineering that were willing to help us in any way made a huge difference in the project for us. Right from the start they provided us with plenty of resources. Within in the first two weeks they sent us a full size vending machine (with some candy), a testing tray set up, and three sets of the current boards. They were also very helpful in the process of creating the new boards. We provided Fawn with our schematics and they took the reins from there as far as getting the boards built and shipped to us in roughly two weeks. Their staff was also very helpful in discussing new ideas with the calibration circuit.

The main lessons we learned from this project dealt with the design processes itself rather than our specific system. We learned that it is very important to get all of the constraints and objectives upfront, and then go over them together to make sure everyone is on the same page. There were a few instances late in our design where we had to consider new constraints to the system that we were not initially aware of. Another lesson was in documenting progress. We found it was easy to get wrapped up in testing and in the technical side of the project, but then when the documentation is needed we hadn’t kept up with it as well as we should of. Although in the end we were able to fully document the project, it would have made it easier on us to spend a little time at the end of each meeting to put some notes together for future use.

Appendix IV: Schematics

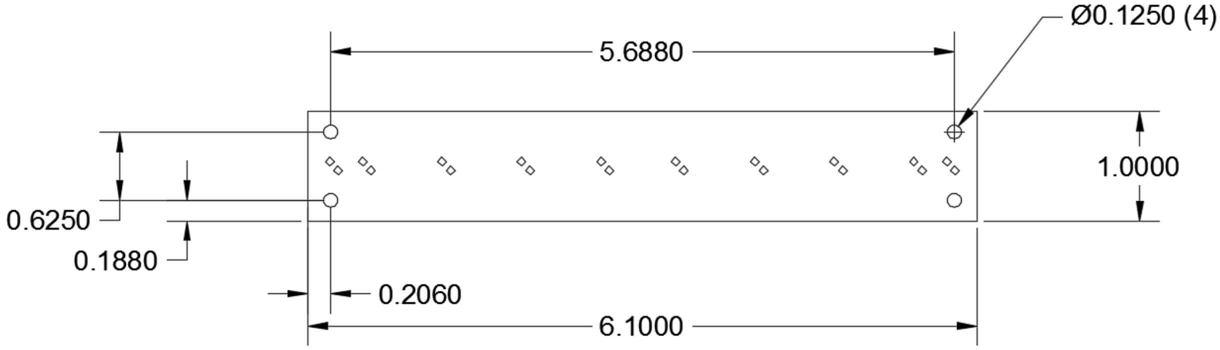

Figure 23 Board dimensions 1

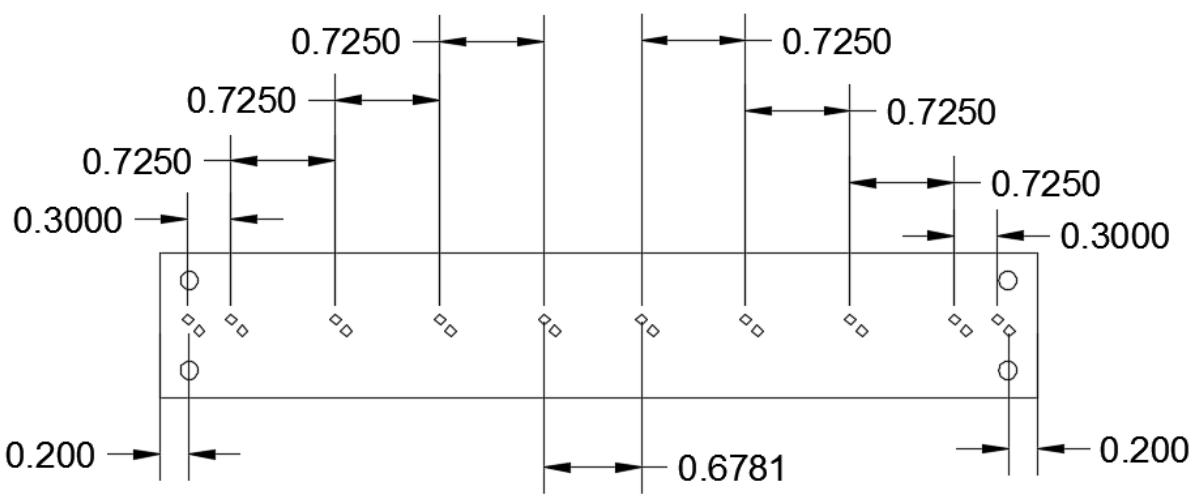

Figure 24 Board dimensions 2

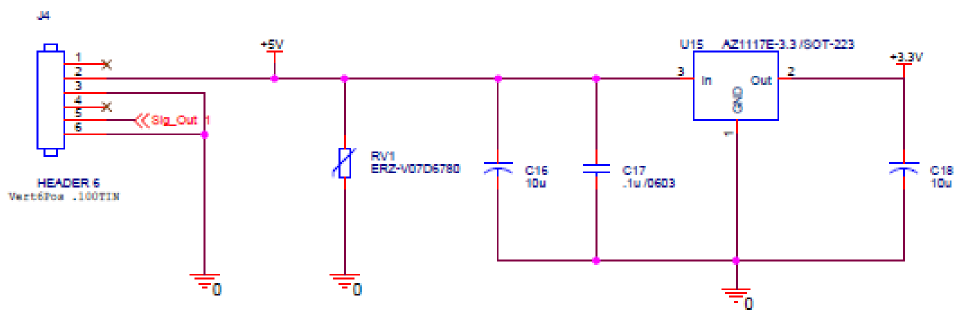

Figure 25 Power conditioning circuit

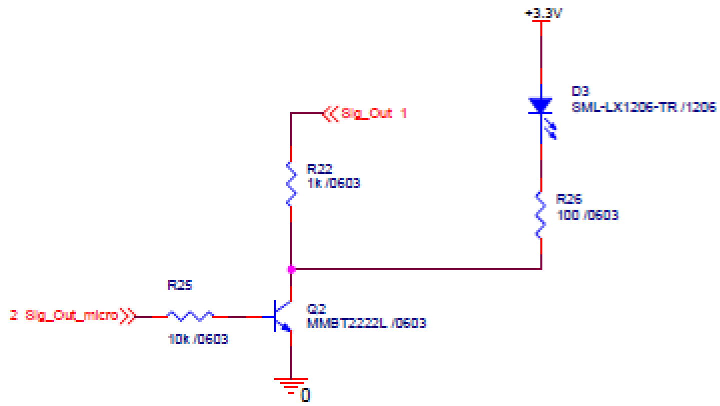

Figure 26 Open collector output

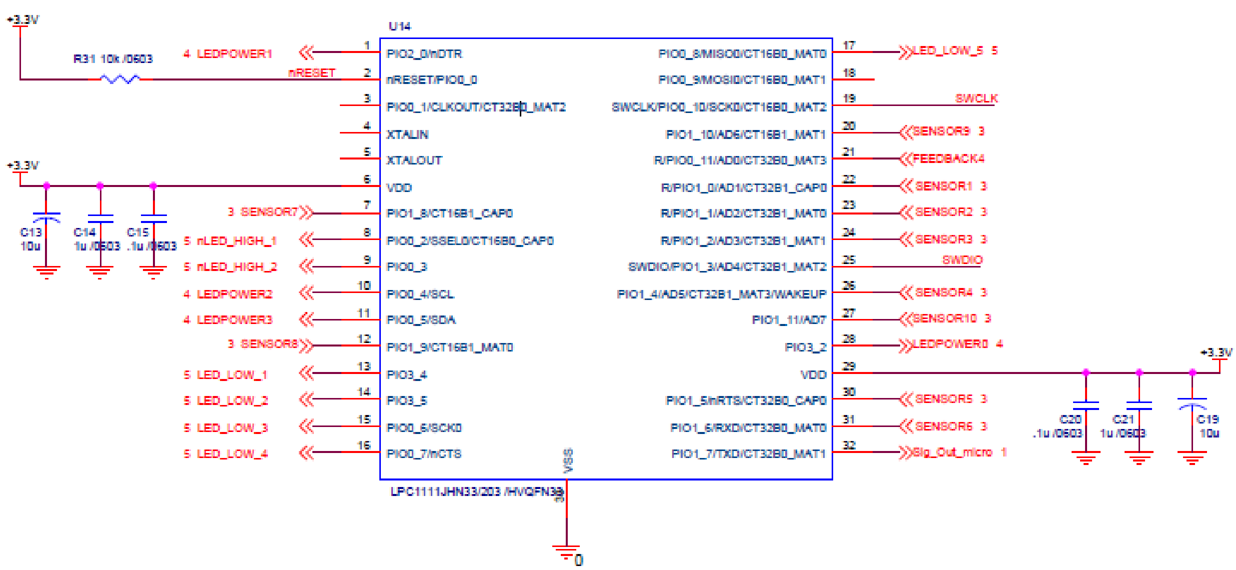

Figure 27 Microcontroller

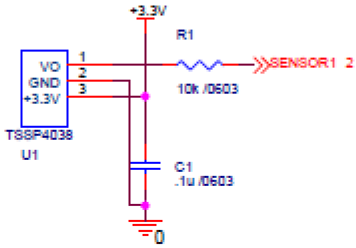

Figure 28 Detector

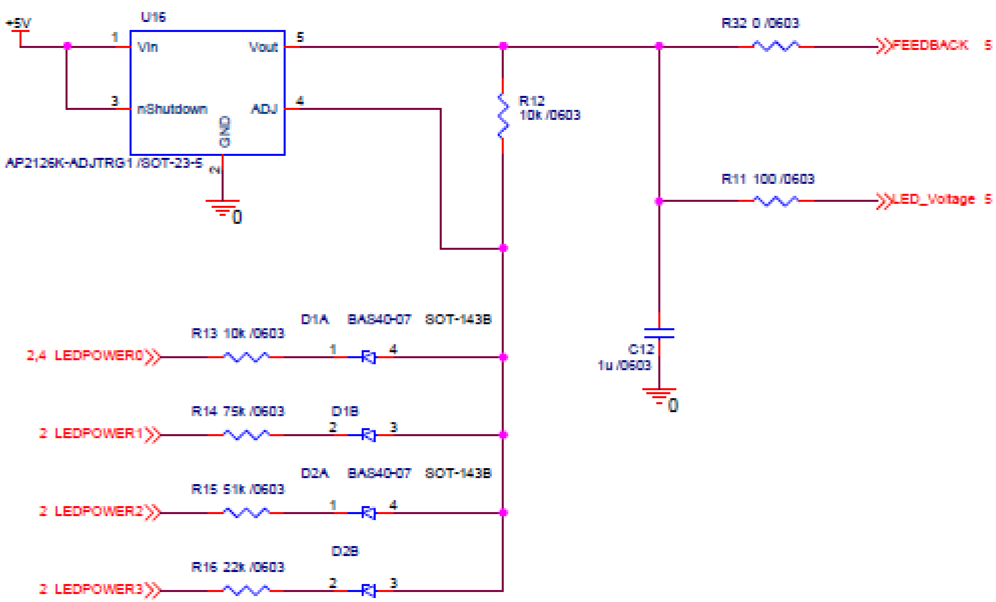

Figure 29 LED Power

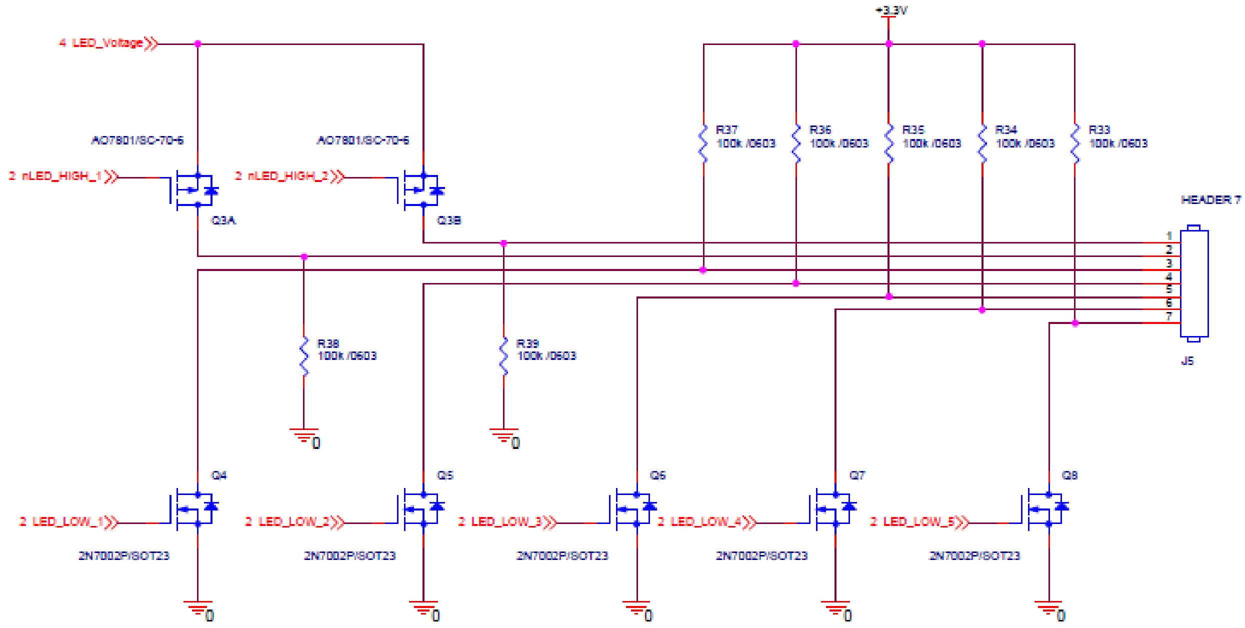

Figure 30 LED Select circuit

Appendix V: Code

```
/* Definitions.h
 *
 *  Created on: Nov 30, 2015
 *      Author: Jake
 */

#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_

#define BIT0 0x01
#define BIT1 0x02
#define BIT2 0x04
#define BIT3 0x08
#define BIT4 0x10
#define BIT5 0x20
#define BIT6 0x40
#define BIT7 0x80
#define BIT8 0x100
#define BIT9 0x200
#define BIT10 0x400
#define BIT11 0x800

// define LED high
#define nLED_High_1 BIT2
#define nLED_High_2 BIT3

#define LED_Low_1 BIT4
#define LED_Low_2 BIT5
#define LED_Low_3 BIT6
#define LED_Low_4 BIT7
#define LED_Low_5 BIT8

#define LED12_Port 3
#define LED345_Port 0
#define LEDHigh_Port 0

#define LEDPOWER0_PORT 3
#define LEDPOWER1_PORT 2
#define LEDPOWER23_PORT 0

#define LEDPOWER0 BIT2 // P3.2
#define LEDPOWER1 BIT0 // P2.0
#define LEDPOWER2 BIT4 // P0.4
#define LEDPOWER3 BIT5 // P0.5


#define DEC1  BIT0   // P0.0
#define DEC2  BIT1   // P1.1
#define DEC3  BIT2   // P1.2
#define DEC4  BIT4   // P1.4
#define DEC10 BIT11  // P1.11
#define DEC5  BIT5   // P1.5
#define DEC6  BIT6   // P1.6
#define DEC8  BIT9   // P1.9
#define DEC9  BIT10  // P1.10
```

```
#define DEC7  BIT8   // P1.8

#define LED_SIG BIT7  //P1.7
#define LED_SIG_PORT 1


#endif /* DEFINITIONS_H_ */

/*
 * Globals.h
 *
 *  Created on: Dec 1, 2015
 *      Author: Jake
 */

 #include "Definitions.h"

#ifndef GLOBALS_H_

#define GLOBALS_H_

// char to index through to get the led we want
extern unsigned int led_index; // we only have leds on pins 4-8

// char to tell if the led should be active or not
extern unsigned char led_active;
extern unsigned char pulse_count;
extern unsigned char led_high_index; // only high switch on pins 2/3
extern unsigned int delay; // time to leave led on for.
extern unsigned char mask_i;
extern unsigned int dec_mask[10];

#endif /* GLOBALS_H_ */

/*
 * init.h
 *
 *  Created on: Dec 2, 2015
 *      Author: Jake
 */

#ifndef INIT_H_
#define INIT_H_

    void _init(); // function will call all startup stuff

    // set up function
    void _led_dir_set(); // function to set pin directions
    void _led_pin_set(); // function to set pin high/low
    void _power_dir_set(); // function to set led power direction
    //void _power_pin_set(); // function to set led power high/low
    void _timer_std(); // function to set timer up

#endif /* INIT_H_ */

/*
 * LED_Functions.h
```

```
 *
 *  Created on: Nov 30, 2015
 *      Author: Jake
 */

#ifndef LED_FUNCTIONS_H_
#define LED_FUNCTIONS_H_

#include "Definitions.h"
#include "LPC11xx.h"
#include "Globals.h"

// running function
void _leds_off(); // function to turn all leds off
void _led_index(); // logic for led indexing
void _dec_check();
unsigned char _dec_check_cal();
void _led_index_normal();

#endif /* LED_FUNCTIONS_H_ */

/*
 * Calibration.h
 *
 *  Created on: Nov 30, 2015
 *      Author: Jake
 */

#ifndef CALIBRATION_H_
#define CALIBRATION_H_

#include "LED_Functions.h"
#include "Definitions.h"
#include "LPC11xx.h"
#include "Globals.h"

void _power_pin_set(); // function to set led power high/low
void _power_up(unsigned char power_lvl); // function will increment the power
setting.

#endif /* CALIBRATION_H_ */

/*
=============================================================================
==
=============================================================================
==
 Name        : iVend_11.c
 Author      : $(author)
 Version     :
 Copyright   : $(copyright)
 Description : main definition
 // set SYSPLLCLKSEL_Val to 0 in system_lpcxxx
=============================================================================
==
*/
```

```
#include "LPC11xx.h"
#include "Calibration.h"   // header for calibration function
#include "LED_Functions.h" // header for functions that control led logic
#include "Definitions.h"   // define statements (BIT defs, LED defs, etc)
#include "Globals.h"       // Global Values for everyone
#include "Init.h"          // initialization



// set global values
unsigned int  led_index = LED_Low_1; // we only have leds on pins 4-8
unsigned char led_active = 0;
unsigned char pulse_count = 0;
unsigned char led_high_index = nLED_High_2; // only high switch on pins 2/3
unsigned int  delay = 10000; // time to leave led on for.
unsigned char mask_i = 0;
unsigned int dec_mask[10] =
    {
    (DEC1 + DEC2 + DEC3 + DEC4),
    (DEC1 + DEC2 + DEC3 + DEC4), (DEC1 + DEC2 + DEC3 + DEC4),
    (DEC2 + DEC3 + DEC4 + DEC5 + DEC6),  (DEC3 + DEC4 + DEC5 + DEC6 + DEC7),
    (DEC4 + DEC5 + DEC6 + DEC7 + DEC8), (DEC5 + DEC6 + DEC7 + DEC8 + DEC9),
  (DEC6 + DEC7 + DEC8 + DEC9 + DEC10), (DEC8 + DEC9 + DEC10), (DEC8 + DEC9 +
DEC10)
    };


int main(void) {

    // Force the counter to be placed into memory
  volatile static int i = 0 ;


    // initialize everything
    _init();

  // power calibration
  //power_lvl_end = _power_pin_set_end();
    _power_pin_set();


  // reset leds after calibration to kown state
  _leds_off();
    LPC_GPIO0->DATA &= ~nLED_High_1; // turn high on
    led_index = BIT4;
    led_high_index = nLED_High_2;
    mask_i = 4;
    // reset what led we are on




  // enable clock interrupt
  NVIC_EnableIRQ(TIMER_16_0_IRQn);
```

```
////////////////////////////////////////////////////////////////////////////////
    // start normal operation
    while(1) {
        i++ ;

        if(pulse_count > 20)
        {
            pulse_count = 0;
            led_active = 1;

        }


            // if to increment leds
        if(pulse_count > 14 && led_active == 1)
        {

            led_active = 0;      // toggle led active
            _leds_off();         // turn the high/low off
            _led_index_normal(); // index the led location

        }

        if(pulse_count > 14)
        {

                if(LPC_GPIO1->DATA & dec_mask[mask_i])
                {
                    LPC_GPIO1->DATA |= LED_SIG; // turn signal on

                    for(i = 0; i < delay; i++)
                    {
                    }

                    LPC_GPIO1->DATA &= ~LED_SIG; // turn signal off
                }
            }

    } // end of while(1)
    // end of normal operation

////////////////////////////////////////////////////////////////////////////////
    return 0 ; // should never reach this
}



////////////////////////////////////////////////////////////////////////////////
// interrupts
void TIMER16_0_IRQHandler(void)
{

     pulse_count++; // increment pulse count
```

```
    // if leds are active toggle on off
     if(led_active == 1)
     {
        if(led_index == BIT6 || led_index == BIT7 || led_index == BIT8)
        {
            LPC_GPIO0->DATA ^= led_index;
        }
        else
        {
            LPC_GPIO3->DATA ^= led_index;
        }
     }

     LPC_TMR16B0->IR |= BIT0;

}
/////////////////////////////////////////////////////////////////////////////

#include "LPC11xx.h"
#include "Definitions.h"



/////////////////////////////////////////////////////////////////////////////
// functions
void _led_dir_set()
{

    // start with led outputs
    LPC_GPIO3->DIR   |= LED_Low_1 + LED_Low_2 ;
    LPC_GPIO0->DIR   |= LED_Low_3 + LED_Low_4 + LED_Low_5;
    LPC_GPIO0->DIR   |= nLED_High_1 + nLED_High_2;
    LPC_GPIO1->DIR   |= LED_SIG;

}

void _led_pin_set()
{

    // set all leds as off to start
    LPC_GPIO3->DATA  &= ~(LED_Low_1 + LED_Low_2);
    LPC_GPIO0->DATA  &= ~(LED_Low_3 + LED_Low_4 + LED_Low_5);
    LPC_GPIO0->DATA  |=  (nLED_High_1 + nLED_High_2);
    LPC_GPIO1->DATA  &= ~LED_SIG;

}

void _power_dir_set()
{

    // set up power out for now use Linear power circuit
    LPC_GPIO3->DIR |= LEDPOWER0;
    LPC_GPIO2->DIR |= LEDPOWER1;
    LPC_GPIO0->DIR |= LEDPOWER2 + LEDPOWER3;

}
```

```
/*
void _power_pin_set()
{

    // only turn on LEDPOWER0
    LPC_GPIO3->DATA  |= LEDPOWER0;
    LPC_GPIO2->DATA  &= ~LEDPOWER1;
    LPC_GPIO0->DATA  &= ~LEDPOWER2;
    LPC_GPIO0->DATA  |= LEDPOWER3;

}
*/ //this is to hard code


void _timer_std()
{

    // set up the timer
    LPC_TMR16B0->PR = 1; // divide down to 12 = 1MHz
    LPC_TMR16B0->MCR |= BIT0 + BIT1; //interrupt and reset on MR0
    LPC_TMR16B0->MR0 = 350; // value for register to count to
    LPC_TMR16B0->TCR |= BIT0; // enable counter


}

void _init()
{


  LPC_SYSCON->SYSAHBCLKCTRL |= (1<<6); // enable clock to the GPIO so that we
can make use of them.
  LPC_SYSCON->SYSAHBCLKCTRL |= (1<<7); // enable clock to 16bit timer 0

    // set up pins as outputs and inputs
  _led_dir_set();

  // set pin high/low
  _led_pin_set();

  // set led power direction
  _power_dir_set();

  // set up the clock
  _timer_std();


}

/////////////////////////////////////////////////////////////////////////////////

/*
 * LED_Functions.c
 *
 *  Created on: Nov 30, 2015
 *      Author: Jake
```

```
 */

#include "LED_Functions.h"
#include "Calibration.h"

void _leds_off()
{

    // turn low off
    LPC_GPIO3->DATA  &= ~(LED_Low_1 + LED_Low_2);
    LPC_GPIO0->DATA  &= ~(LED_Low_3 + LED_Low_4 + LED_Low_5);

}

void _led_index_normal()
{
    led_index = led_index << 1;
    mask_i ++;


    if(led_index > BIT8)
    {
        led_index = BIT4;

        if(led_high_index == nLED_High_1)
        {
            LPC_GPIO0->DATA &= ~nLED_High_1; // turn high on
            LPC_GPIO0->DATA |= nLED_High_2;
            led_high_index = nLED_High_2;

            mask_i = 5; // resest mask
        }
        else
        {
            LPC_GPIO0->DATA &= ~nLED_High_2; // turn high on
            LPC_GPIO0->DATA |= nLED_High_1;
            led_high_index = nLED_High_1;

            mask_i = 0; // reset mask
        }



    }

}

#include "Calibration.h"

#define LED_PERIOD 100   // number of pulses each led gets
#define PULSE_CHECK 50  // number of pulses till led is checked if its high
(50 gives plenty of time for input to stabilize before checking)


unsigned char power_set_complete = 0;
// calibration function for all pins
void _power_pin_set()
```

```
{
    unsigned char power_select = 1;

    unsigned char led_not_seen = 0;
    power_set_complete = 0;

    // set the light high so we know we're in calibration
    LPC_GPIO1->DATA |= LED_SIG;

    LPC_GPIO0->DATA &= ~nLED_High_1;

    _power_up(power_select);

    // enable the counter interrupt
    NVIC_EnableIRQ(TIMER_16_0_IRQn);

    // while loop that performs the calibration
    while(power_set_complete == 0)
    {

        // if to increment leds
        if(pulse_count > LED_PERIOD)
        {

            led_active ^= 1; // toggle led active
            _leds_off(); // turn the high/low off
            pulse_count = 0; // reset pulse count

            // if leds are not active and it's not on the last LED then
increment index

            if( led_active == 0 )
            {
                _led_index_normal();
            }

            // im going to test by only using the first led to make things a
little easier for troubleshooting
            // if the leds are off, it's on the last one, and the leds were
seen by the detectors; mark the power set function as complete
            if( (led_active == 0) && (led_index >= LED_Low_5) &&
(led_high_index >= nLED_High_2) && (led_not_seen == 0) )
            {
                power_set_complete=1;
                LPC_GPIO1->DATA &= ~LED_SIG;
                break;
            }

        }

        if(led_active == 1 && pulse_count > PULSE_CHECK && power_set_complete
== 0)
        {
            int i = 0;

            if(LPC_GPIO1->DATA & dec_mask[mask_i])
                {
```

```
                    LPC_GPIO1->DATA |= LED_SIG;

                    for(i = 0; i < delay; i++)
                    {
                    }

                    LPC_GPIO1->DATA &= ~LED_SIG;
                }
        }

        // if the led was not seen by one or more detector ADD MORE POWER TIM
"THE TOOLMAN" TAYLOR STYLE
        if( (led_not_seen == 1) && (led_active == 0) )
        {
            power_select++;           // increment power level
            if(power_select > 15)
            {
                power_select = 1;
            }
            _power_up(power_select); // set power pins to new level
            led_not_seen = 0;         // reset the led not seen flag
        }



    } // end calibration while loop

    // turn led off when done


} // end _power_pin_set function




//////////////////////////////////////////////////////////////////////////////
///////////////////////////
void _power_up(unsigned char power_lvl)
{
    // disable counter interrupt so we dont get out while doing this stuff.
    //NVIC_DisableIRQ(TIMER_16_0_IRQn);

    // need to reset led index each time to restart look
    if(power_set_complete == 0)
    {
    led_index = LED_Low_1;
    led_high_index = nLED_High_1;
    pulse_count = 0; // reset pulse count too
    }

    switch(power_lvl)
        {

    //set the outputs for LED Power based on the current power_select value
    case 1:
        LPC_GPIO3->DATA  |= LEDPOWER0;
        LPC_GPIO2->DATA  |= LEDPOWER1;
```

```
        LPC_GPIO0->DATA   |= LEDPOWER2;
        LPC_GPIO0->DATA   |= LEDPOWER3;
        break;

    case 2:
        LPC_GPIO3->DATA   |= LEDPOWER0;
        LPC_GPIO2->DATA   |= LEDPOWER1;       // 1.495098 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   |= LEDPOWER3;
        //power_lvl = 2;
        break;

    case 3:
        LPC_GPIO3->DATA   |= LEDPOWER0;
        LPC_GPIO2->DATA   &= ~LEDPOWER1;      // 1.661765 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   |= LEDPOWER3;
        //power_lvl = 3;
        break;

    case 4:
        LPC_GPIO3->DATA   |= LEDPOWER0;
        LPC_GPIO2->DATA   |= LEDPOWER1;       // 1.818182 V
        LPC_GPIO0->DATA   |= LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        //power_lvl = 4;
        break;

    case 5:
        LPC_GPIO3->DATA   |= LEDPOWER0;
        LPC_GPIO2->DATA   &= ~LEDPOWER1;      // 1.984848 V
        LPC_GPIO0->DATA   |= LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        break;

    case 6:
        LPC_GPIO3->DATA   |= LEDPOWER0;
        LPC_GPIO2->DATA   |= LEDPOWER1;       // 2.06328 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        break;

    case 7:
        LPC_GPIO3->DATA   |= LEDPOWER0;
        LPC_GPIO2->DATA   &= ~LEDPOWER1;      // 2.229947 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        break;

    case 8:
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   |= LEDPOWER1;       // 2.5 V
        LPC_GPIO0->DATA   |= LEDPOWER2;
        LPC_GPIO0->DATA   |= LEDPOWER3;
        break;

    case 9:
```

```
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   &= ~LEDPOWER1;      // 2.66667 V
        LPC_GPIO0->DATA   |= LEDPOWER2;
        LPC_GPIO0->DATA   |= LEDPOWER3;
        break;

    case 10:
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   |= LEDPOWER1;       // 2.745098 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   |= LEDPOWER3;

        break;

    case 11:
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   &= ~LEDPOWER1;      // 2.911765 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   |= LEDPOWER3;
        break;

    case 12:
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   |= LEDPOWER1;       // 3.068182 V
        LPC_GPIO0->DATA   |= LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        break;

    case 13:
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   &= ~LEDPOWER1;      // 3.234848 V
        LPC_GPIO0->DATA   |= LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        break;

    case 14:
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   |= LEDPOWER1;       // 3.31328 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        break;

    case 15:
        LPC_GPIO3->DATA   &= ~LEDPOWER0;
        LPC_GPIO2->DATA   &= ~LEDPOWER1;      // 3.479947 V
        LPC_GPIO0->DATA   &= ~LEDPOWER2;
        LPC_GPIO0->DATA   &= ~LEDPOWER3;
        break;

        } // end case statements

} // end _power_up
///////////////////////////////////////////////////////////////////////////////
///////////////////////////
```