2019

Secure Parcel Deposit Station

USER GUIDE AND TECHNICAL MANUAL

BY WILLIAM DOREY

Table of Contents

1 Introduction 3
2 Setup 4
2.1 Wireless Configuration 4
2.2 Initial Setup 7
3 Specifications 9
3.1 Physical Specifications 9
3.2 Operating Specifications 10
4 System Diagram 11
5 System Description 12
5.1 5V Power Supply 12
5.2 Raspberry Pi Zero W 12
5.3 Pi Camera Module 12
5.4 Motion Sensor 12
5.5 12V/5V Power Supply with Battery Backup 12
5.6 Raspberry Pi 3B + ... 13
5.7 Web Interface 13
5.8 3.3V to 5V Logic Level Converter 13
5.9 ATmega8515 13
5.10 Solenoid 14
5.11 Weight Sensor, Amplifier and ADC 14
5.12 Keypad and LCD 14
5.13 Address Decoding Unit 14
6 Circuit Description 15
6.1 Camera Unit 16
6.2 12V/5V Power Supply with Battery Backup 17
6.3 3.3V to 5V Logic Level Converter 19
6.4 ATmega8515 19
6.5 Solenoid 21
6.6 Weight Sensor, Amplifier and ADC 22
6.7 Keypad and LCD 24
6.8 Address Decoding Unit 25

7 Software Description 26
7.1 Camera Unit 26
7.2 Microcontroller Assembly Code 29
7.2.1 Main.asm 29
7.2.2 General.inc 32
7.2.3 Keypad.inc 32
7.2.4 LCD.inc 33
7.2.5 Serial.inc 34
7.2.6 Delays.inc 34
7.3 Raspberry Pi 3B+ 35
7.4 Web Interface 37
8 Testing and Calibration 38
9 Troubleshooting 39
10 Appendixes 41
10.1 Appendix A: Illustrations 41
10.2 Appendix B: Code/Script Listings 57
10.2.1 Microcontroller Code 57
10.2.2 Microcontroller Test Functions 72
10.2.3 Python Scripts 82
10.2.4 Bash Scripts 85
10.2.5 Webpage files 86
10.3 Appendix C: Bill of Materials 94
10.4 Appendix D: Cost Analysis 96

1 Introduction

With more and more purchases being made online, the need to have a secure location for these packages to be delivered becomes increasingly important. There are times when people are unable to receive a package in person. These packages are usually left on doorsteps, out in the open, for long periods of time. This makes them vulnerable to theft. It is at times, such as these, that my project is a vital part in maintaining a watchful eye over these unattended packages that could be worth significant amounts of value to the recipients, either monetary or emotionally. The project container can only be opened after a user enters in the passcode set by the owner keeping any unwanted users out.

Figure 1 - The SPDS Main Unit

All scripts, circuit schematics and codes related to this project are available at the following link:

https://www.github.com/WilliamDorey/SPDS

2 Setup

This section will primarily be focusing on the initial setup of the two devices included in the project. The main points of setup will focus on the Wi-Fi functionality of the unit. The following steps assume that the unit has an adequate source of power already connected to it.

2.1 Wireless Configuration

1. (Optional) Connect an ethernet cable from a local/home router to the Raspberry Pi 3B+ located on the side wall of the main enclosure. This will allow for a reliable connection to the device while altering the Wi-Fi configuration files.
2. Connect to the Raspberry Pi 3B+ using an SSH client software such as PuTTY. Alternatively, a monitor and keyboard could be connected directly to the device in order alter the configuration without wireless or wired network availability. The default SSID, passphrase and IP of the Raspberry Pi’s wireless network interface are ‘**WILLIAM-PROJECT**’ , ‘**DORW20079804**’ and **192.168.42.1** respectively.
3. Login to the Pi using the username **pi** and pre-configured password of the unit **DORW20079804**. If the unit has not been preconfigured for use in the SPDS, the password will be **raspberry**.
   NOTE: there will be no graphical interface, only a command line.

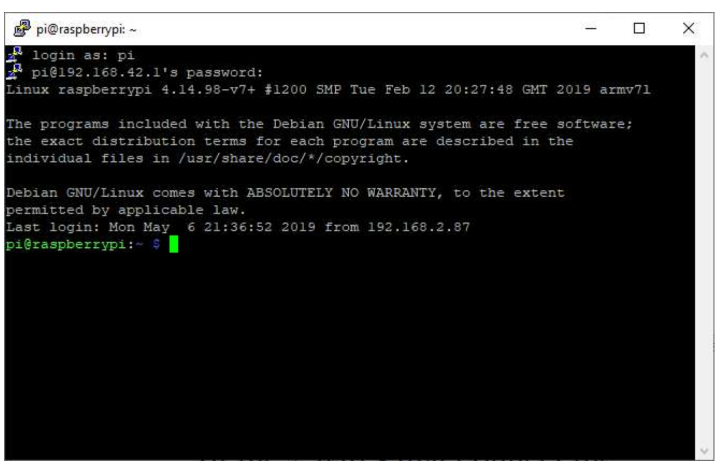


Figure 2 - PuTTY Login Screen

4. Before altering any configurations on the Pi, the configurations of the Pi Zero unit must be altered to accommodate any planned changes. To do this, SSH into the Pi Zero from the Pi using the following command: **ssh pi@pizero** . The device will not ask for a password.
5. Open the wireless configuration file, '**/etc/wpa_supplicant/wpa_supplicant.conf** ', using the text editor of your choosing. If you are not experienced with command line text editors, it is suggested that you use 'nano', through the command:
   nano /etc/wpa_supplicant/wpa_supplicant.conf
6. Using the arrow keys on the keyboard, navigate to and change the values of the **ssid** (name of the wireless network) and **psk** (passphrase that will be used by the wireless network) variables to your desired values.

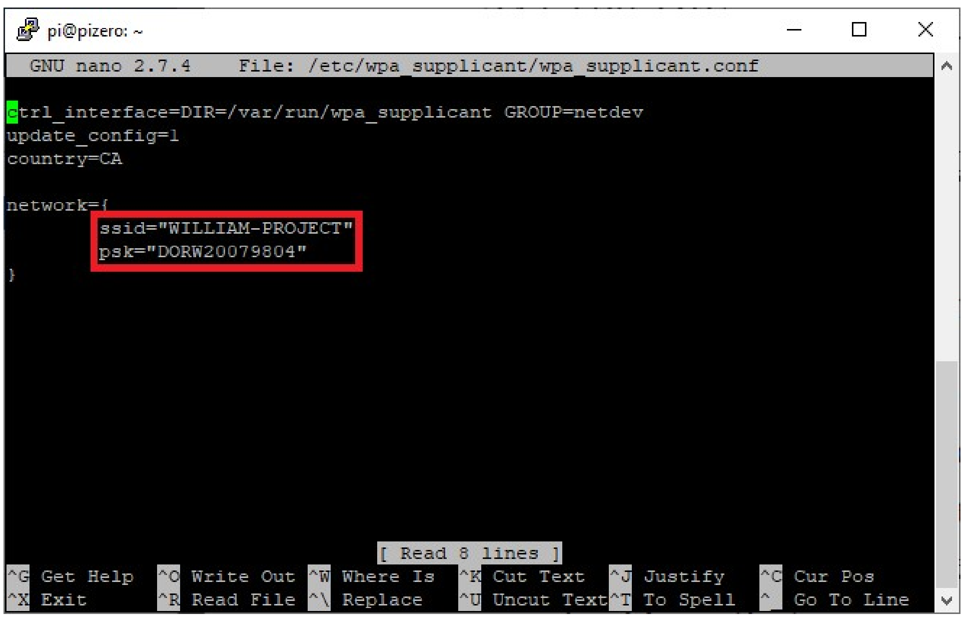


Figure 3 - '/etc/wpa_supplicant/wpa_supplicant.conf' File of the Raspberry Pi Zero W

7. Save the changes (in nano, press "ctrl+x" and then press enter). Then reboot the device by using the following command: **sudo reboot**
   NOTE: The connection to pizero will close itself due to the connection being lost.
8. Next, we will configure the configuration of the wireless network itself. This configuration is altered in the ' **/etc/hostapd/hostapd.conf** 'file. Once again using the text editor used in step 5.

9. Use the arrow keys to navigate to the **ssid** and **wpa_passphrase** values and change them to the values used in step 6.

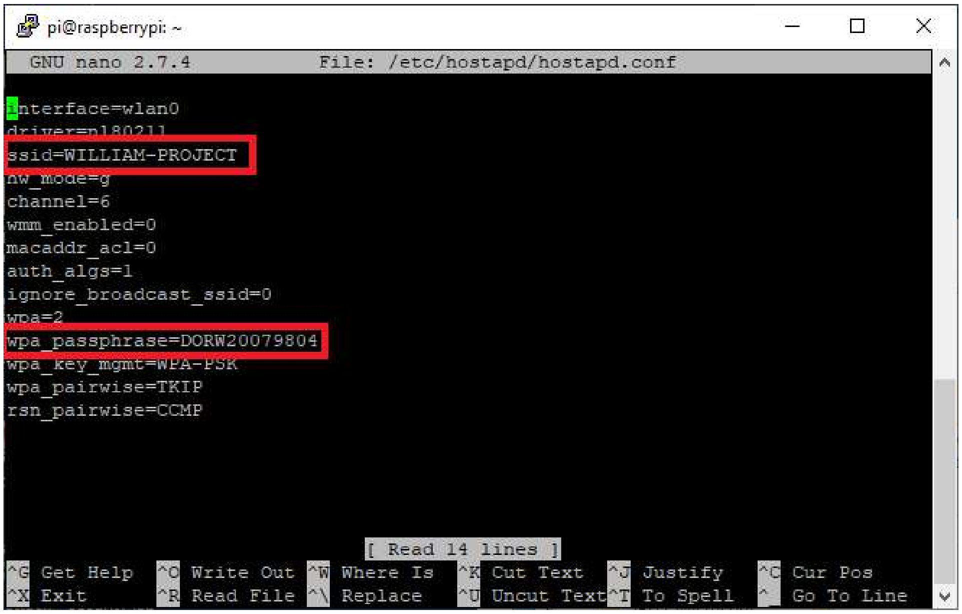


Figure 4 - '/etc/hostapd/hostapd.conf' File of the Raspberry Pi 3B+

10. Save the changes to the file. Reboot the device following the same instructions in step 7.

2.2 Initial Setup

1. Before connecting any power sources (either the internal battery or the external wall plug), with the unit open, connect an ethernet cable with an internet connection to the Raspberry Pi inside the unit in order to update the date and time.
2. Connect a power source to power-on the unit.
3. Using a wireless device (smartphone, tablet, or PC) open a web browser and in the address bar type **192.168.42.1** to open the unit's web interface.
4. If the date displayed under the 'Archive' section is incorrect, wait approximately 30 seconds and then refresh the web page. Repeat this step until the date and time are correct.

   NOTE: the ethernet connection is no longer needed and can be disconnected.
5. Click on the link at the bottom of the webpage to access the advanced monitoring page.

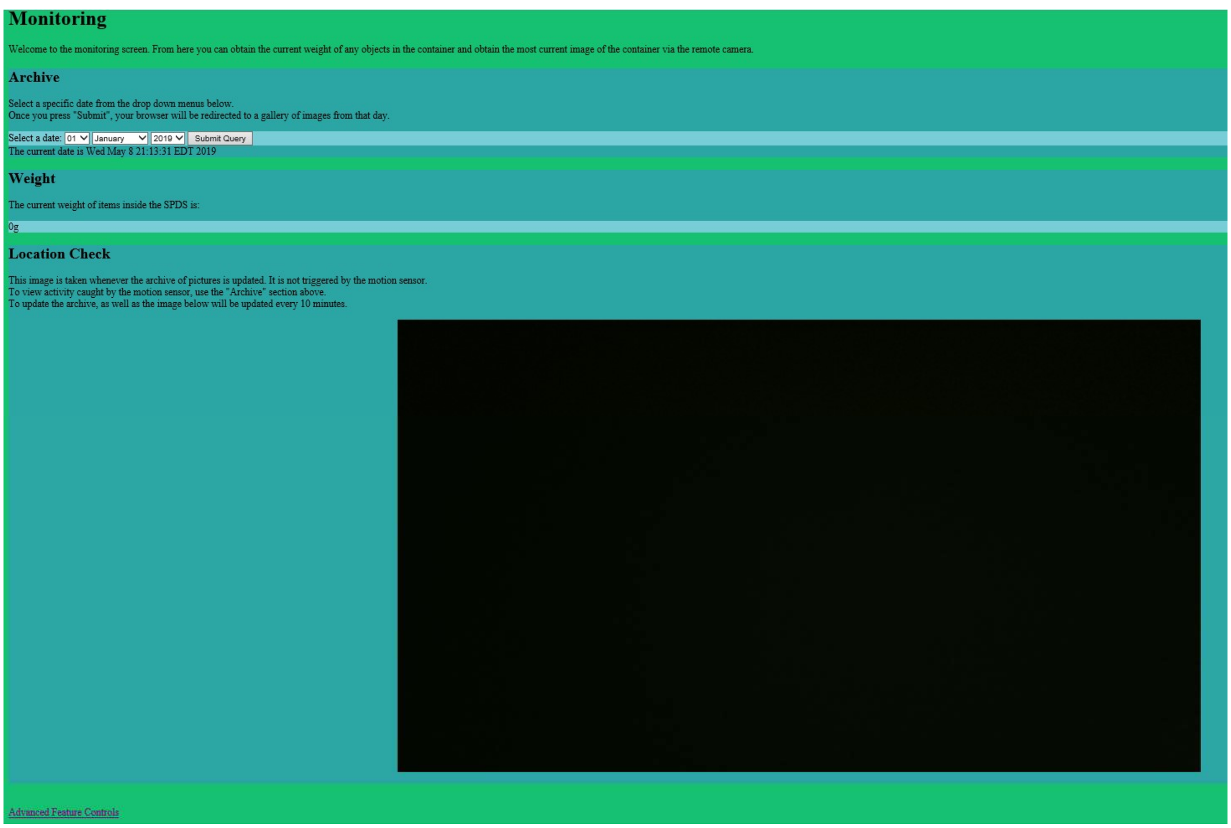


Figure 5 - Web Interface Home Page

6. From this page, you may set a new password for the unit, zero the weight scale, and clear the archive.

7. After filling out the required fields, click the submit button and wait for the web page to reload.

8. The unit is now ready for use. Make sure that the internal battery has been connected before disconnecting the external power to move the unit.

9. After powering on the main unit, the LCD will display a checksum value that should match the following image. If it does not, the Microcontroller will need to be re-flashed with the proper files.

Figure 6 - Checksum Value Displayed at Power-On

3 Specifications

3.1 Physical Specifications

Table 1 - Physical Specifications of Main Unit

Part	Measurements (Inches)		
	Length	Width	Height
Outer Case	23	11	12
Weight Sensor Platform	11	9	N/A
Circuitry Area	8	10	9
Main Circuit Board	4½	3½	N/A
Power Supply Board	3	2	N/A
Keypad & LCD Board	3	2½	N/A
Amplifier and ADC Board	2	3 ¼	N/A
Keypad and LCD Interface	N/A	3	4.5

Table 2 - Physical Specifications of Camera Unit

Part	Measurements (Inches)		
	Length	Width	Height
Outer Case	4	3 ¾	3 ¾
Inner Case	3 ½	3	3
Circuit Board	2	2 ½	N/A
Motion Sensor	1 ½	1	N/A
Camera	1	1	N/A
Power Cord Opening	N/A	¾	N/A

3.2 Operating Specifications

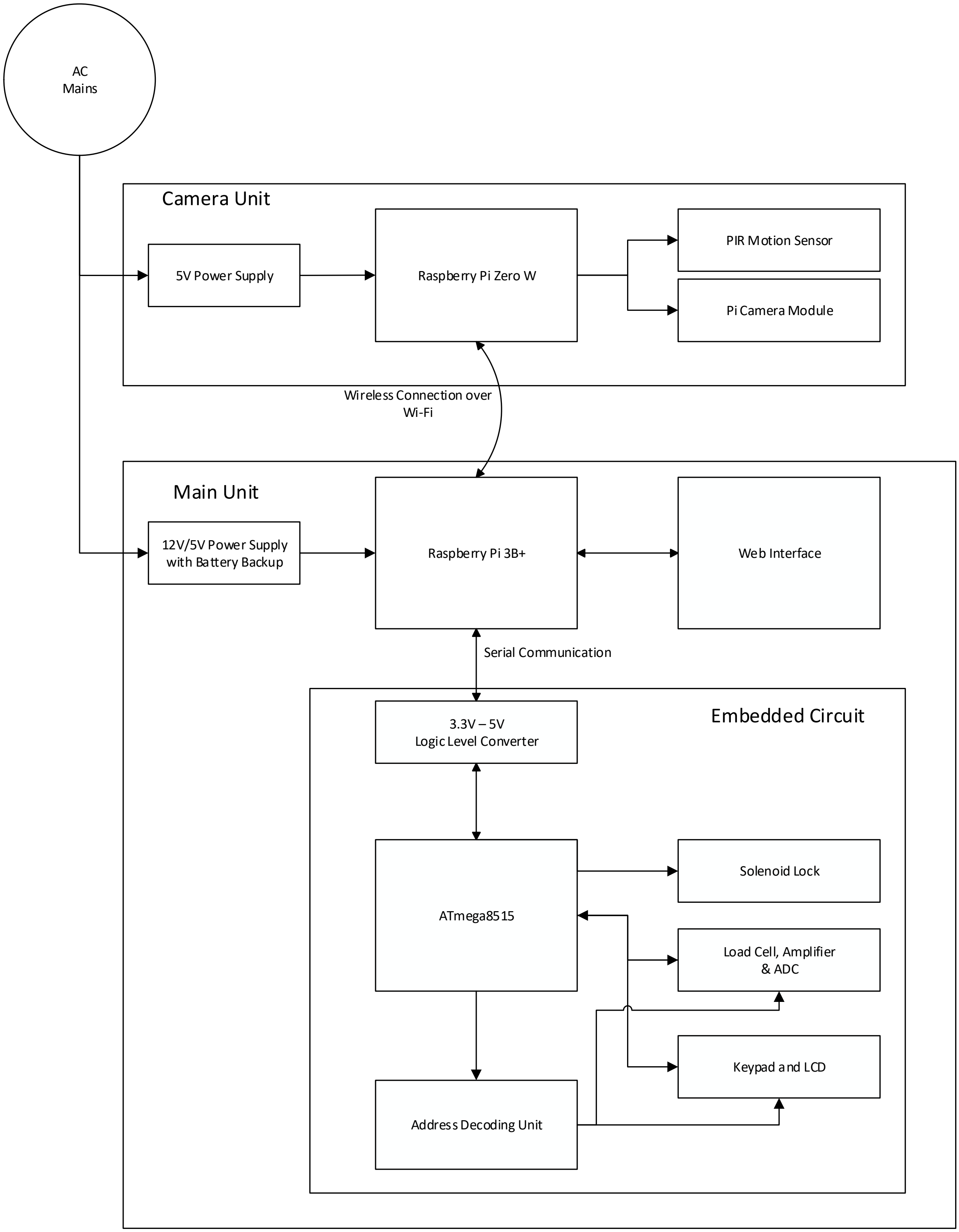
Table 3 - Operational Specifications of Main Unit

Parameter	Units	Values		
		Minimum	Nominal	Maximum
AC Voltage input	V AC	100	115	120
Battery Voltage	V	12	12.6	12.7
Operating Temperature	C	-15	20	45
Solenoid Voltage	V	10	12	12
Solenoid Current	mA	500	600	800
Embedded System Voltage	V	4.5	5	5.4
Raspberry Pi Voltage	V	4.85	5	5.15
Raspberry Pi Current	A	0.4	0.6	3

Table 4 - Operational Specifications of Camera Unit

Parameter	Units	Values		
		Minimum	Nominal	Maximum
DC Voltage Input	V	6	9	12
Current Draw	A	0.3	0.4	1.5
Operating Temperature	C	-30	20	40
Camera Operating Temperature	C	-30	20	110

4 System Diagram



5 System Description

5.1 5V Power Supply

The power supply for the Raspberry Pi Zero W and its attached modules must supply 5V of DC voltage, and so a ‘ wall wart’ is used as a compact solution in order to convert the AC voltage that is supplied by the mains to a more suitable 9V DC.

5.2 Raspberry Pi Zero W

This discrete Raspberry Pi Zero W controls the motion sensor and camera module of the camera unit. Whenever a signal is received from the motion sensor, the Pi Zero triggers the camera to capture a picture. The pictures that are taken this way are transmitted via a wireless connection to the Raspberry Pi 3B+ in the main unit every 10 minutes.

5.3 Pi Camera Module

This is the official Pi Camera v2 module designed specifically for a Raspberry Pi. It is capable of full HD imagery, however the resolution of photos taken are reduced in order to preserve storage space.

5.4 Motion Sensor

This component of the camera unit will trigger anytime it detects horizontal motion inside of it’s seven-meter range.

5.5 12V/5V Power Supply with Battery Backup

The power supply of the main unit is a linear power supply using a single stepdown transformer connected to an AC main (wall outlet). The output of the transformer is first reduced to 12V for use by the solenoid and to accommodate the battery-backup, allowing for trickle-charging to increase the amount of time between depletions. The output of the battery in addition to the 12V from the transformer are then converted to 5V for the operation of the Raspberry Pi and embedded circuit.

5.6 Raspberry Pi 3B +

The Raspberry Pi 3B+ serves as a webserver for data from the embedded circuit, and images from the camera unit. The server also acts as a wireless access point.

5.7 Web Interface

The web interface displays important information from the embedded circuit and images from the camera unit. A home page displays the most recent image from the camera unit and the current weight of the main unit's contents. Two additional pages are available. One for viewing images from a certain date specified on the homepage, and another for handling advanced controls (setting a new passcode, zero the weight scale and clearing the archive of images).

5.8 3.3V to 5V Logic Level Converter

The serial communication ports of the embedded circuit's microcontroller and the Raspberry Pi operate at 5V and 3.3V respectively, and so the signals that they send and receive need to be converted to a level suitable for each device.

5.9 ATmega8515

The ATmega8515 serves as the microcontroller of the embedded circuit in the main unit. It manages an interface between the embedded circuit and the Raspberry Pi 3, the inputs from the keypad and weight sensor and the outputs that are sent to the LCD and solenoid. The connection to the Raspberry Pi is managed by serial communication using Tx and Rx signals. The hardware connected to the embedded circuit are primarily controlled with the use of data bus communications, apart from the solenoid that is connected to a parallel port.

5.10 Solenoid

The solenoid is used as a lock for the lid of the main unit. The microcontroller operates it using a single transistor. It is the only component outside of the power supply that uses 12V for its input.

5.11 Weight Sensor, Amplifier and ADC

The weight sensor uses a resistive load cell to alter its resistance based on the strain placed across of the cell. The output of the load cell is two voltages with a difference of sometimes less than a milli-volt, and so the difference must be amplified for use in the analog to digital converter, which is needed to translate the analog value of the voltage to an eight-bit digital value more suitable for use in the embedded circuit.

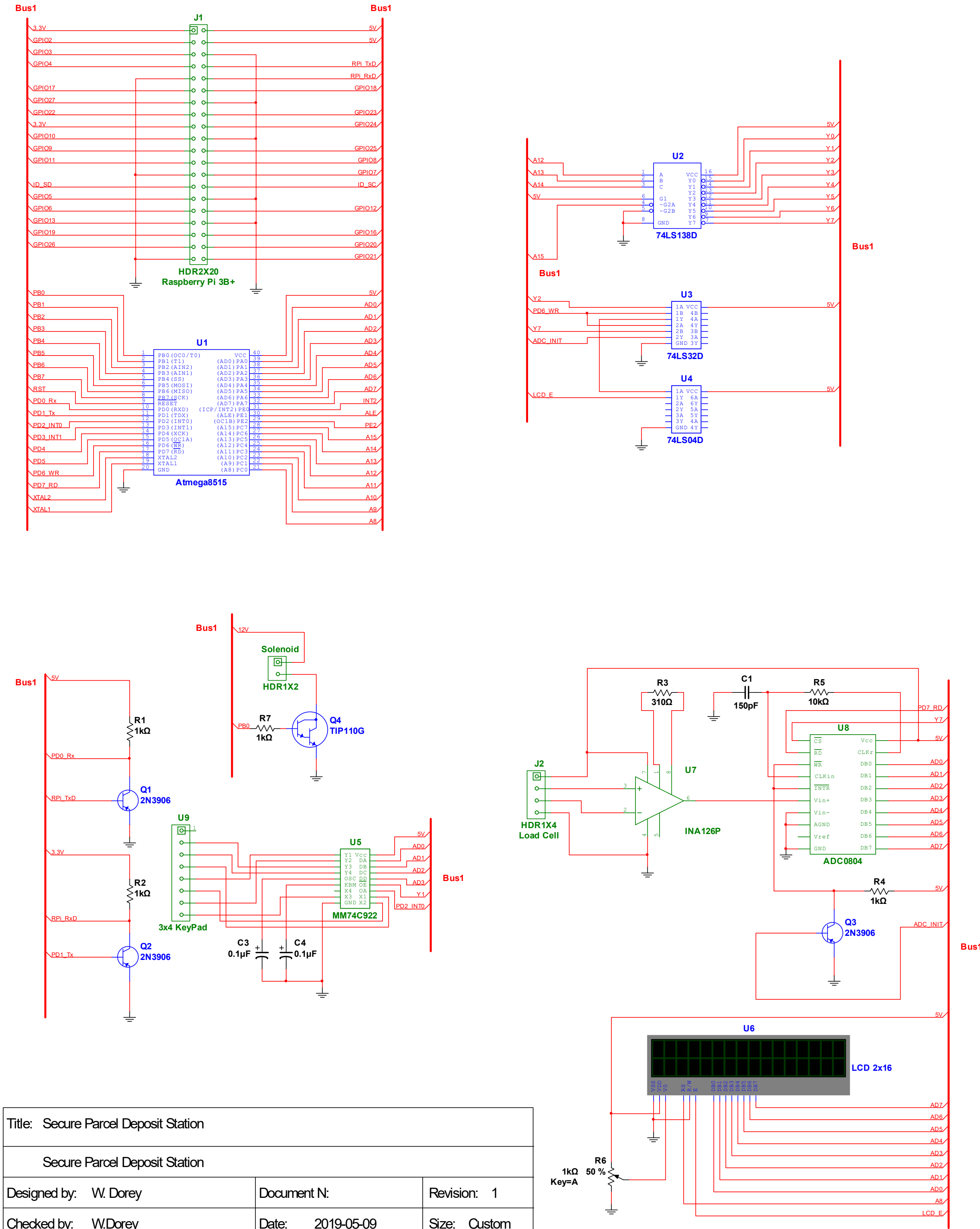
5.12 Keypad and LCD

The keypad and LCD are the primary interface for users. They are used to input a 6-digit passcode to unlock the main unit. The keypad consists of a three column by four row matrix that connects pins together depending on the button that is pressed. The LCD unit is also used for troubleshooting purposes as it is the only module used to display information from the test programs.

5.13 Address Decoding Unit

Because the Atmega8515 is using data buses to control the embedded circuit hardware, an address decoder is necessary to activate the destination devices at the correct times. This circuit consists of a decoder as well as simple combinational logic gates.

6 Circuit Description



Title: Secure Parcel Deposit Station			
Secure Parcel Deposit Station			
Designed by: W. Dorey	Document N:		Revision: 1
Checked by: W.Dorey	Date: 2019-05-09		Size: Custom
Approved by:	Sheet 1 of 1		

6.1 Camera Unit

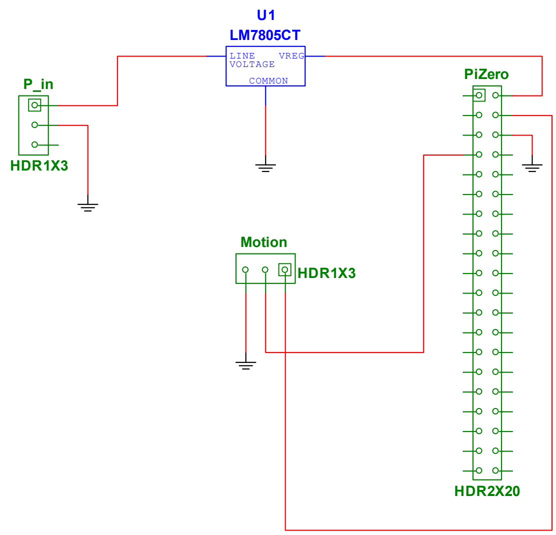


Figure 7 - Camera Unit Circuit/Wiring Diagram

The 5V power supply for this unit consists only of an LM7805 linear regulator in series between one of the Pi Zero and a barrel connector. The barrel connector is used for connecting a 'wall wart', acting as a compact and efficient way of drawing power from the AC mains, to the unit.

The motion sensor is connected to the GPIO pins of the Raspberry Pi Zero W. Whenever motion is detected, a short pulse is generated by the motion sensor, triggering an action in a python script that is running. This python script will trigger the camera, saving the image.

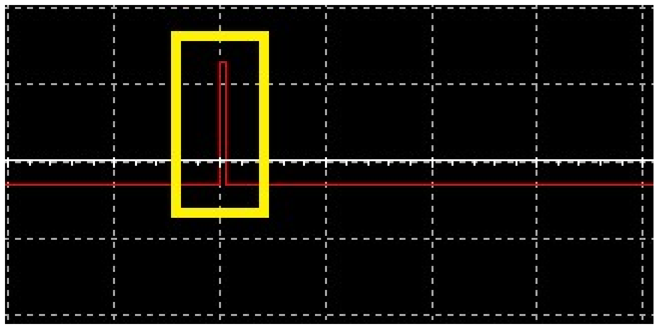


Figure 8 - Simulated Output Pulse from Motion Sensor

6.2 12V/5V Power Supply with Battery Backup

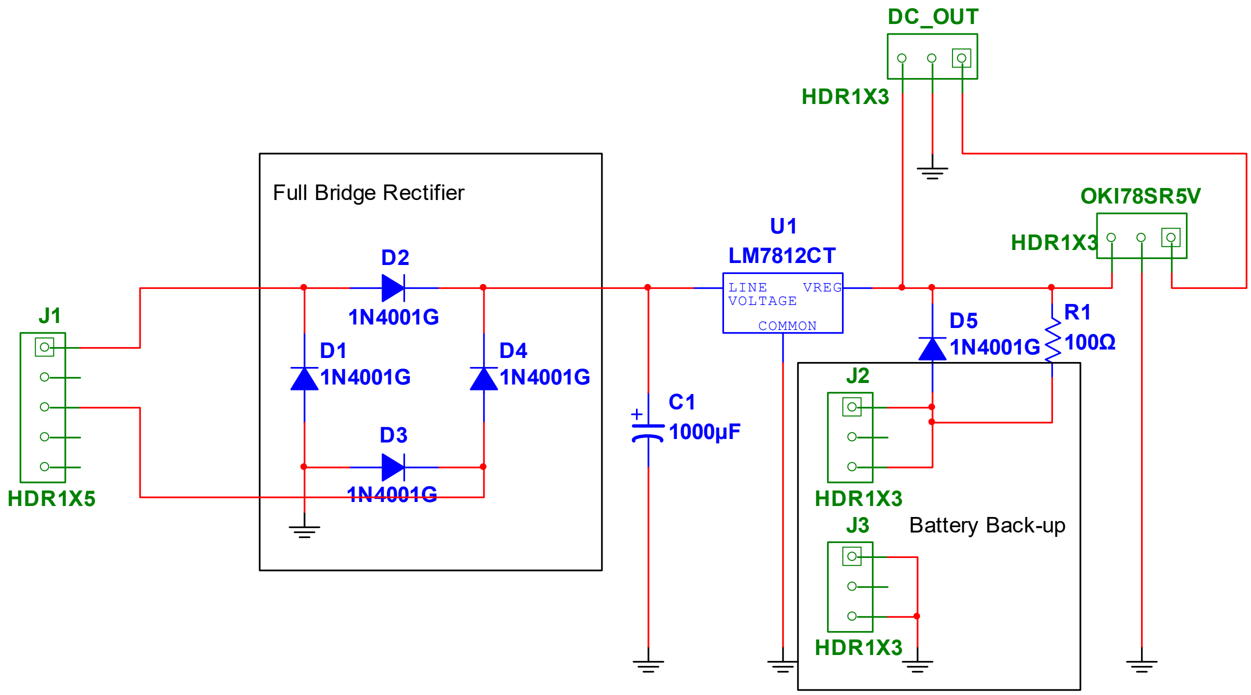


Figure 9 - 12V/5V Power Supply Schematic

The power supply begins by drawing power from the AC mains and down stepping the m through a 48VA transformer. This reduces the 115V_{AC} of power to a more reasonable signal of 18.4$V_{Pk\text{-}Pk}$ at the center-tap, that is then sent threw a full bridge rectifier. The full bridge rectifier is used to separate the negative and positive sets of voltage that are taken from the center-tap of the transformer.

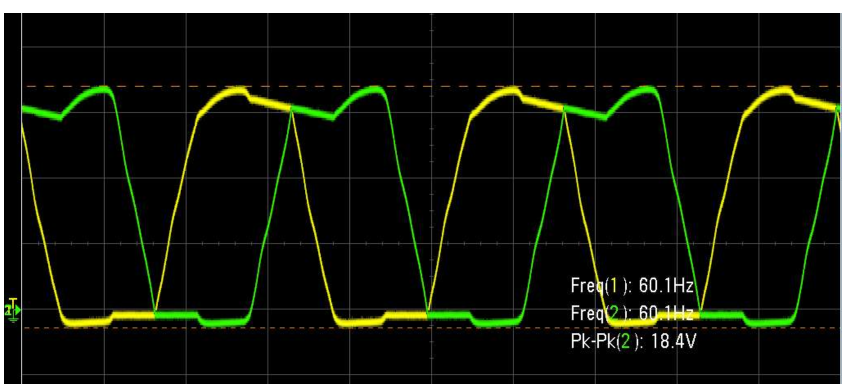


Figure 10 - Output Signal from Across the Transformer's Center-Tap

The output of the full bridge rectifier is connected to a smoothing capacitor that will hold the voltage when both outputs from the transformer are low. Creating a more consistent source of Voltage for the 12V linear regulator to use in order to make a DC voltage.

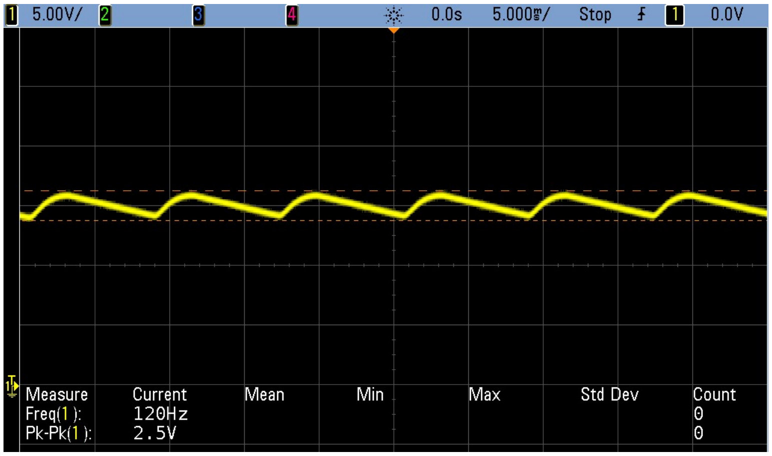


Figure 11 - Output Signal of the Full Bridge Rectifier

The output of the linear regulator is a stable $12V_{DC}$ signal suitable for use to trickle charge the battery back-up and power the solenoid. An additional voltage level of $5V_{DC}$ is needed to power the Raspberry Pi and the embedded circuit of the main unit. This voltage is converted through an OKI-78SR-5 DC-to-DC converter.

6.3 3.3V to 5V Logic Level Converter

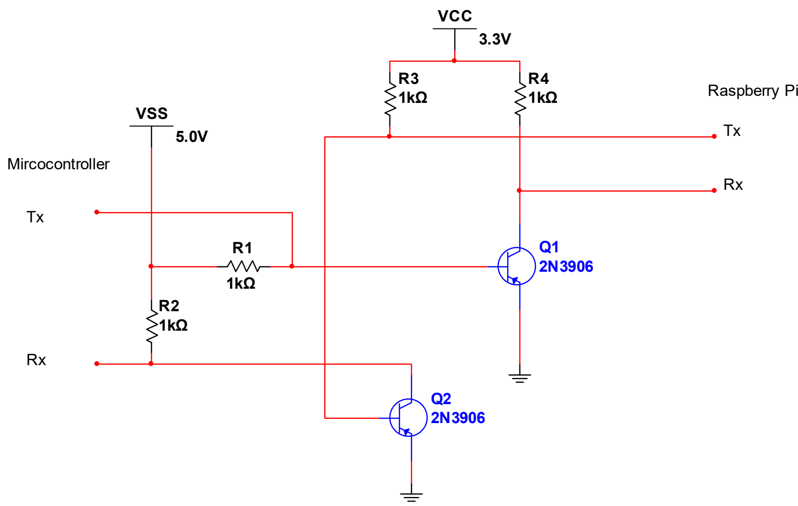


Figure 12 - 3.3V to 5V Logic Level Converter Circuit

This sub-circuit translates both the 5V signals from the serial pins of the microcontroller and the 3.3V signals of Raspberry Pi to allow for their communication. The circuit consists of two 2N3906 BJT transistors and two 1kΩ resistors. It is important that these two transistors are PNP. When a PNP transistor's base connection receives a voltage greater than the one present at the emitter, the connection between the emitter and the collector short. This sis important because it is how the different voltage level outputs will control the inputs of the opposite level. When a one of the serial outputs (Tx) are inactive, they remain at a low level, sinking the transistor and connecting the serial receiving input (Rx) of the other device. The opposite is true when the output signal is at a high level. The delay that is generated by this process is minimal enough that it does not affect the operation of the two devices.

6.4 ATmega8515

The ATmega8515 is responsible for controlling the data that is sent to, and from each of the modules connected to the embedded circuit. These transmissions of data are done primarily using data buses. Below is an example of the signal that is created by these data buses as displayed by an oscilloscope

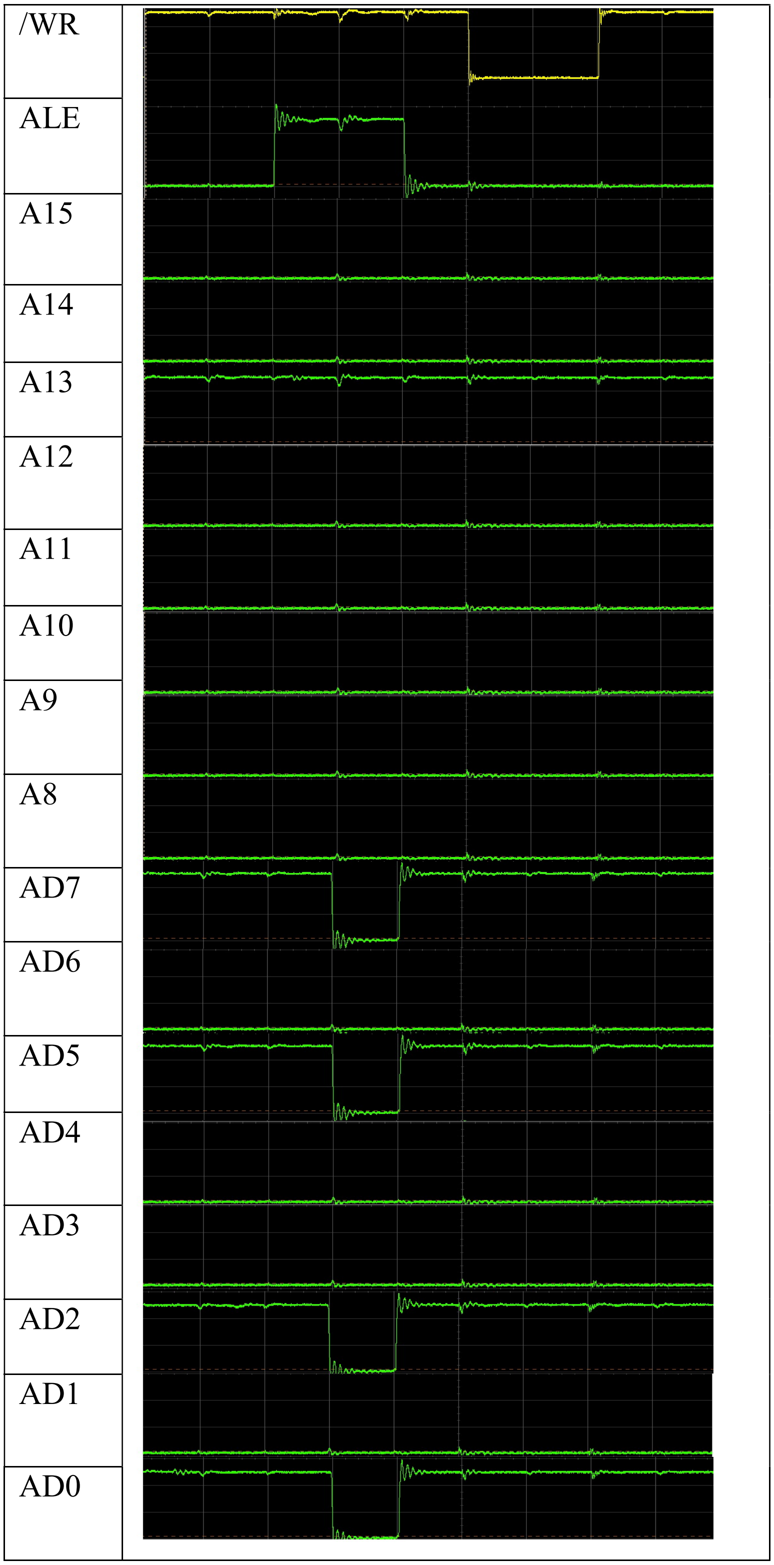


Figure 13 - Data Bus Output for Address $2000 Carrying $65 as Data

The eight-bits that are labelled as ‘Ax’ are known as address bits and are used only to output the address of the target device that is being either read from or written to, depending on whether the /RD or /WR signal is pulsed low respectively. The eight-bits that are labeled as ‘ADx’ additionally carry the data that is being written following a short period after the address is sent. The data is carried through the bus during the period during which the /WR (or /RD) signal is low. The ALE signal indicates when the address is being sent through these buses. The information that is sent using this method can be simplified into a composite wave form for easier understanding of the information.

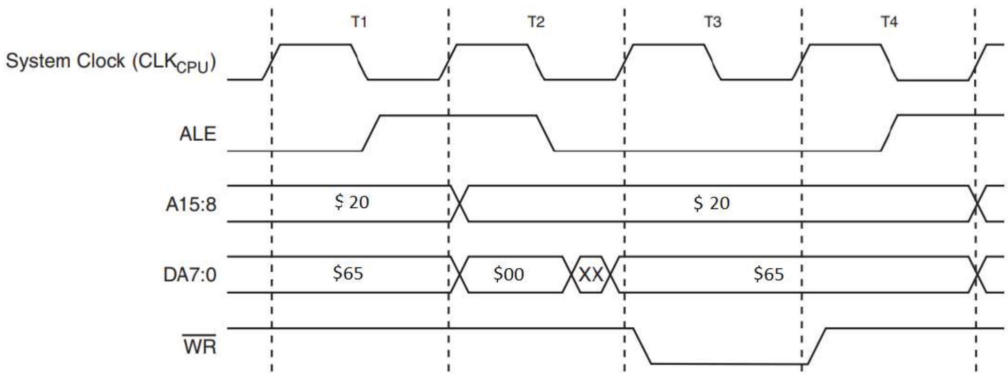


Figure 14 - Composite Waveform of Data Buses from Figure 11

There is an additional output that is configured as a parallel port, meaning that the value that is sent out remains static until the port is altered using a command to change its value specifically. This port is connected to the solenoid circuit to control it.

An additional input pin is used to listen for an interrupt signal from the keypad to notify the controller that there is a value ready to be read. When the signal on this pin transitions from being high to low, a new process is begun inside the microcontroller’s code.

6.5 Solenoid

The solenoid’s circuit is used to power the solenoid whenever the correct pass code is given. This is done by applying a 5V signal through a 1kΩ resistor to a TIP110 transistor’s base connection. This resistance value allows for a limited amount of current to flow through the transistor connecting the solenoid to ground, activating it. The current that is required to activate the solenoid is very large compared to the rest of the main unit, and so it is advised to limit the time that it spends active to prolong the battery life of the unit.

6.6 Weight Sensor, Amplifier and ADC

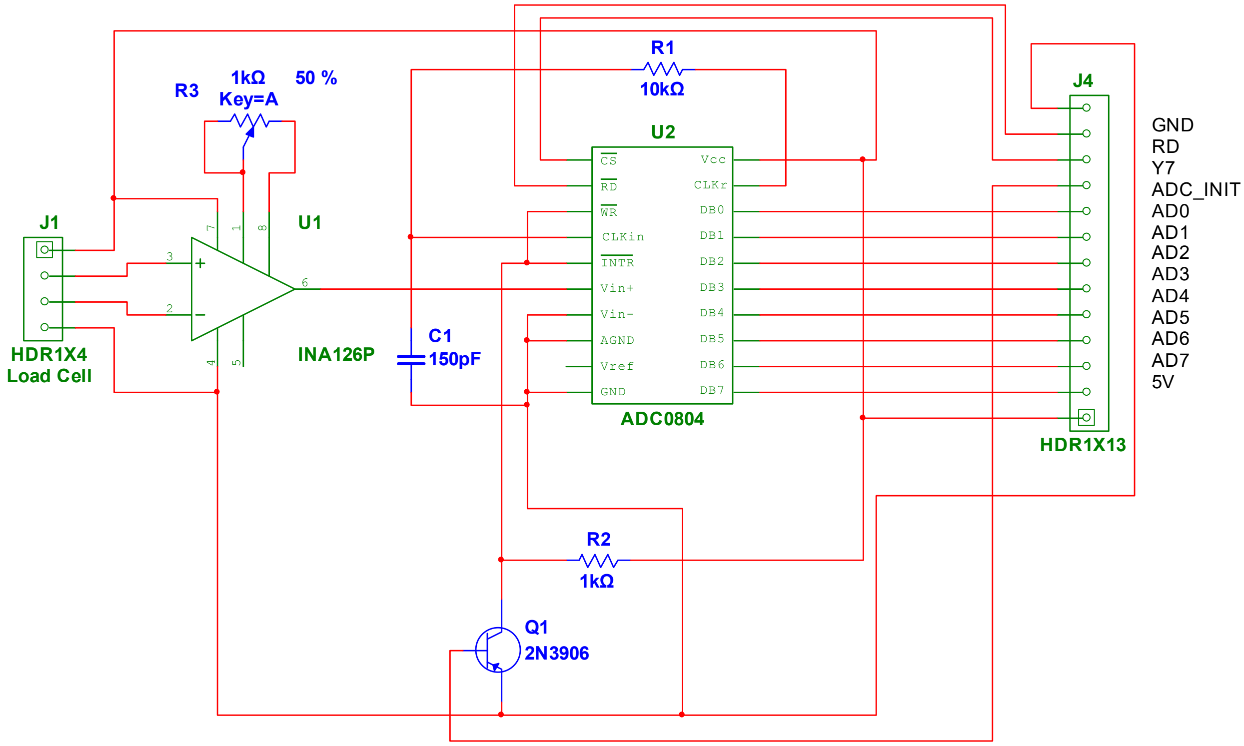


Figure 15 - Weight Sensor, Amplifier and ADC Wiring Diagram

The weight sensor is a resistive load cell that requires an input of 5V in order to properly operate. The output consists of two similar signals that are separated by a voltage between 0.2mV and 10 mV. The small changes in voltage that correspond to the weight that is applied to the load cell must be amplified in order to be detected by the ADC. The weight limit of the load cell being used is approximately 5kg, and the base that rests on top is weighed at 480g, meaning that the limit for its measurements is 4.5kg. The accuracy of the scale must also be considered and so the recommended maximum weight that can be applied without maxing out the load cell should be approximately 4kg.

The amplifier circuit uses the INA126 Instrumentation amplifier. This is a set of 2 operational amplifiers interconnected to detect minute differences between two input signals, essential in amplifying a resistive load cell's output. The gain generated by the amplifier is calculated using the following equation where Rg is the external resistor:

$$Gain = 5 + \frac{80k\Omega}{Rg}$$

In this instance, a potentiometer is used in place of a resistor in order to more accurately calibrate the weight sensor unit. The value that is calculated to give the most efficient amount of gain for the load cell that is used is a value of 310Ω.

The ADC compares the input signal that is given to it against the Voltage that is supplied to it through Vcc. The equation to find the value that is output by the unit is as follows:

$$Output = \frac{Vin * 255}{Vcc}$$

The output form the ADC is calculated and sent the microcontroller any time both the /RD and the /CS pins are set to low by the addressing circuit.

6.7 Keypad and LCD

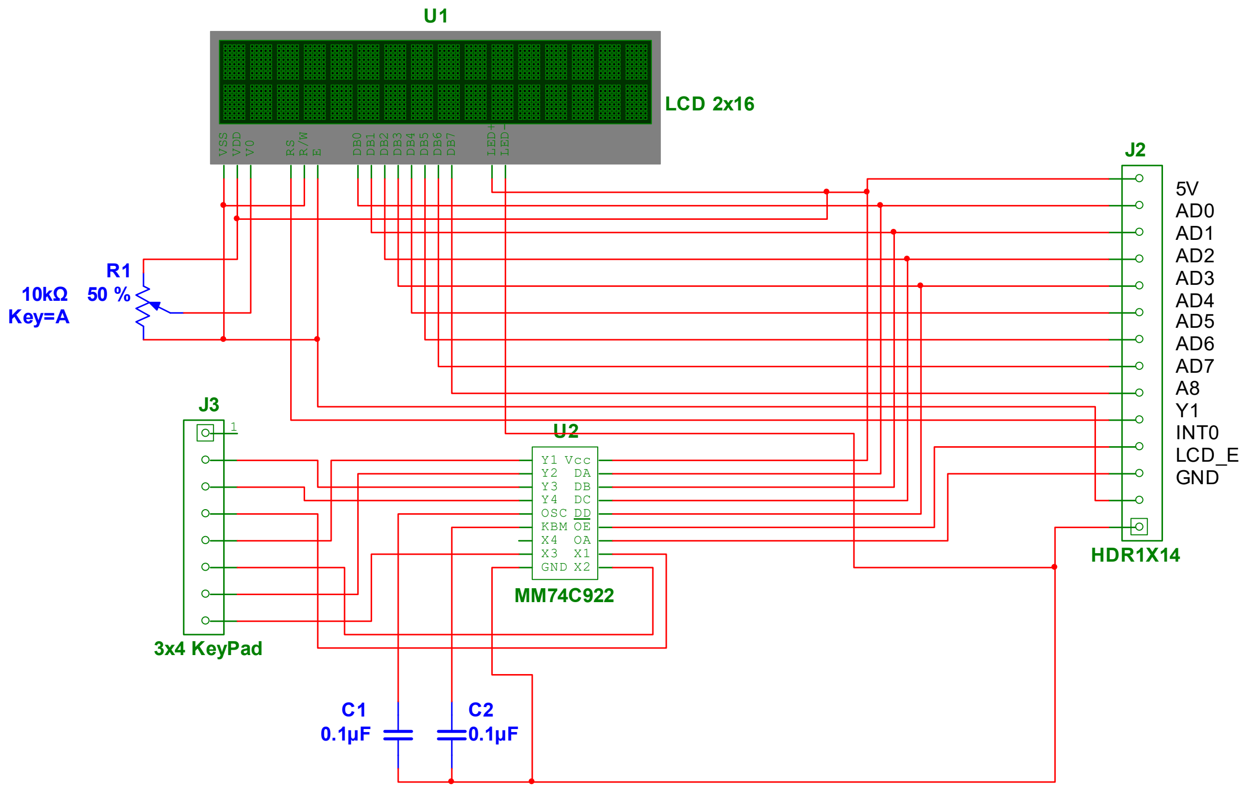


Figure 16 - Keypad and LCD Wiring Diagram

The two components of this module both operate using the data buses of the microcontroller. The LCD receives eight-bits of data to determine either which ASCII character is meant to be displayed or which part of its control unit needs to be altered and how. A potentiometer is placed between Vcc and ground with the middle pin connected to the LCD's

The keypad operates through an additional IC, the 74C922, that translates the connections made by each button into a corresponding four-bit binary value. The IC determines which button has been pressed on the keypad by monitoring the pins that represent the three columns and four rows. Whenever a connection is made between a column and a row, a signal is sent out to the microcontroller letting it know that an output is currently available. When the button is released, the signal to notify the microcontroller goes low, activating the interrupt, and making the microcontroller read from the keypad's location. A table of the Keypad's pins correspond to rows and columns is available in *Illustrations*.

6.8 Address Decoding Unit

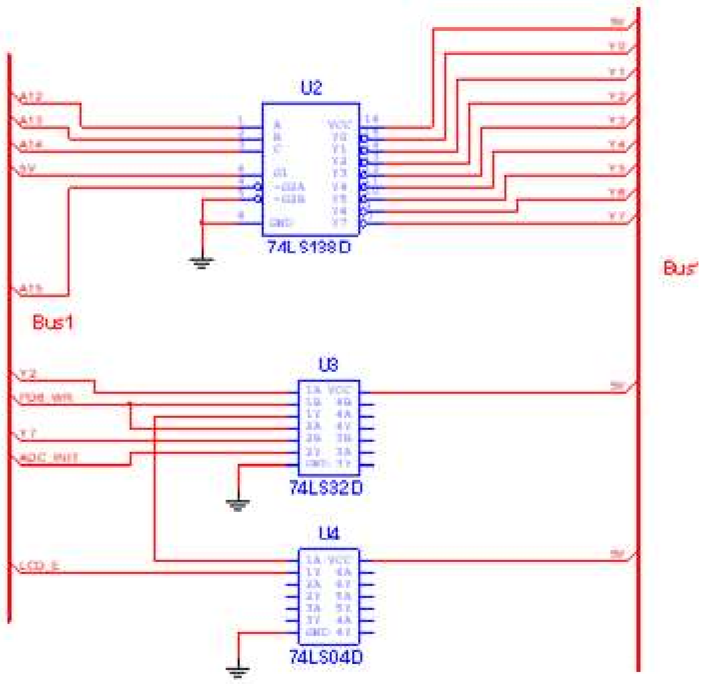


Figure 17 - Address Decoding Circuit

This circuit plays a crucial role in the operation of the embedded circuit. Using combinational logic, it reads the address values of all the outgoing data bus transmissions and sets the appropriate output to a low output in order to enable the destination device. The main IC used for this is the 74LS138, a 1-8 decoder. Depending on the four outputs from the microcontroller that are connected to this IC, a specific output pin will go low, otherwise the outputs will all remain as a logical high, preventing any devices from becoming enabled.

Table 5 - Output of 74LS138

A15	A14	A13	A12	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
0	0	0	0	1	1	1	1	1	1	1	0
0	0	0	1	1	1	1	1	1	1	0	1
0	0	1	0	1	1	1	1	1	0	1	1
0	0	1	1	1	1	1	1	0	1	1	1
0	1	0	0	1	1	1	0	1	1	1	1
0	1	0	1	1	1	0	1	1	1	1	1
0	1	1	0	1	0	1	1	1	1	1	1
0	1	1	1	0	1	1	1	1	1	1	1
1	N/A	N/A	N/A	1	1	1	1	1	1	1	1

7 Software Description

7.1 Camera Unit

The camera unit's software consists of Raspbian Lite, a version of Raspbian for which all graphical interface components have been removed along with any unnecessary packages to reduce CPU tasks and storage space used by the OS itself, optimizing the system for autonomous operations.

In order to operate the Pi Camera module, the required python libraries must be installed on the system. Both the python-picamera and python3-picamera libraries are suitable for this use, and seeming as how python3 is the latest version, that is the one that has been installed.

Using python to operate the camera is as simple as importing the object from its library, defining the object as the attached camera, and then sending instructions to take a picture/video. This is demonstrated in the three-line python script *live.py* That is used to quickly take a snapshot with the camera.

```
# Basic python script to take a picture
#
# Located on Pi Zero
from picamera import PiCamera

camera = PiCamera()

camera.capture('/home/pi/live.jpg')
```

Figure 18 - live.py

The motion sensor attached to the GPIO pins of the Pi Zero can be defined as its own object type in python, simplifying the task of triggering a script when motion has been detected. As seen below, all that needs to be done to have the sensor trigger an event to take a picture is to define the sensor by the pin it is connected to and then have it call a function that takes the picture when it detects motion.

```
[…]
from gpiozero import MotionSensor
[…]
sensor = MotionSensor(18)
[…]
sensor.when_motion = take_pic
[…]
```

Figure 19 - Snippet of motion_camer.py

A basch script is used alongside the crontab function of Linux to automate the operation of the camera unit, namely the process it takes to transfer the pictures to the webserver. The script used to do this, *update.sh*, begins by first killing the previous instance to the *motion_camera.py* script being run, allowing the camera to be called in another script. After ending the script, the pictures that have been taken are sent to the webserver using 'scp' and 'ssh'. The images are then removed from the device to make room in the storage for new ones. A single picture is then taken, in case no motion was detected, and sent to the server. The script ends by once again starting the *motion_camera.py*.

The crontab file of the Pi Zero is given a single entry. This entry is used to run the *update.sh* bash script every ten minutes. The list of crontab operations is accessed through the command line using the following command: 'crontab -e'.

```
*/10  *  *  *  *  bash /home/pi/update.sh
```

Figure 20 - Line Appended to Crontab File to run Bash Script every 10 Minutes

The functionality of the bash script depends on the fact that the pi user on the Pi Zero can login to the Raspberry Pi server without entering a password. This is done by generating a new rsa key with the following command: '**ssh-keygen -t rsa**'. The key that is generated then needs to be sent using scp to the '/home/pi/.ssh/' directory of the Raspberry pi server. Once the file is in place, it becomes possible to use ssh and scp without the need for a password.
NOTE: this should also be performed from the Raspberry Pi server in order to simplify connectivity.

7.2 Microcontroller Assembly Code

The various sections of the embedded circuit have been somewhat divided into include files that allow for one over-arching file, main.asm, to call upon their functions in order to keep the code organized.

7.2.1 Main.asm

The main.asm section of code contains the initialization sequence for the microcontroller, the interrupt signal subroutine that is responsible for tracking passcode attempts, and the waiting loop where the microcontroller waits for a signal from the Raspberry Pi requesting an action.

The initialization sequence begins with the stack pointers to allow the code to call on subroutines that are defined outside of the main sequence.

```
[…]
; Main initialization sequence start
init:
              ; Stack pointer to allow for subroutines
              LDI R16, LOW(RAMEND)
              OUT SPL, R16
              LDI R16, HIGH(RAMEND)
              OUT SPH, R16
 […]
```

Figure 21 - Stack Pointer Initialization

After the stack pointers, the data bus style of data transmission is enabled along side the interrupt, int0, immediately followed by a kickstart signal for the ADC and defining of the solenoid's control pin on the microcontroller.

```
[…]
            ; Selecting interrupts to enable
            LDI R16, (1<<INT0)
            OUT GICR, R16

            ; Bus and Interrupt initialization
            LDI R16, $82
            OUT MCUCR, R16

            ; Sets Solenoid and kickstarts ADC
            LDI R16, $FF
            OUT DDRB, R16
            STS $1000, R16
            LDI R16, $00
            OUT PORTB, R16
[…]
```

Figure 22 - Interrupt, Bus, ADC and Solenoid Initialization

The last pieces before completing the initialization sequence are the serial initialization and LCD initialization, both of which are handled by subroutines in their respective include files.

```
[…]
            ; LCD and Serial communications
            RCALL Serial_init
            RCALL LCD_init
[…]
```

Figure 23 - Serial Communication and LCD Initialization

The sequence ends by displaying a banner message on the LCD and generating a checksum value. Then moving into a section that will continuously loop until power is lost, apart from the interrupt subroutine.

This main loop section is constantly waiting to receive a value via the serial connection to the Raspberry Pi in order to either set a new passcode or return the value of the ADC's output.

```
[…]
wait:  ; Wait for a signal from server
              ; to either set a new passcode
              ; or to obtain a value from the ADC
              RCALL Serial_get

              CPI R18, 'W'
              BRNE skip0
              RCALL Gen_ADC

skip0:
              CPI R18, 'S'
              BRNE fini
              RCALL Gen_set_pass

fini:  ; Return to waiting for a signal
       ; from the Raspberry Pi server
              RJMP wait
[…]
```

Figure 24 - Main Loop of Microcontroller's Operation

The Interrupt subroutine defined at the end of the main.asm file is triggered by the falling edge of the signal connected to the corresponding pin on the microcontroller. The routine begins by translating the data received from the keypad into the actual value that is shown on the button that was pressed, using an external subroutine. The value that is returned is checked against 3 main rules:

1. If there is no active attempt and the value is '*', start a new attempt
2. If there is an active attempt and the value is not '#', add the value to the current pass code attempt.
3. If the there is an active passcode attempt and the value is '#' end the attempt and check it against the saved passcode.

The end of this file contains the pattern's displayed on the LCD and the files that are included when generating the HEX code to flash the microcontroller.

7.2.2 General.inc

The General.asm file contains subroutines that either do not fall under a hardware component or do not justify their own include file. These subroutines include the ones for generating a checksum, converting a register's contents to the ASCII equivalent, setting a pass code, checking a pass code attempt, unlocking the solenoid, and obtaining a value from the ADC.

The subroutine to generate a checksum value, Gen_Check, begins by loading the last address of the cseg from the microcontroller, and then proceeds to add the values from each previous address location until all the locations have been added together. This value is then converted to the ASCII characters it uses and sent to be displayed on the LCD.

The subroutine to obtain a value from the ADC additionally converts the value to ASCII and returns the value to the Raspberry Pi using serial communications.

7.2.3 Keypad.inc

The only subroutine in this include file is used to translate the incoming values from the keypad into the values that had been selected by the user. This is done by checking key bits in the values and the n proceeding to correct the values.

```
[…]
Key_map:
              MOV  R15, R16
              SBRS R15, 3
              RJMP Key_map0
              SBRS R15, 2
              RJMP Key_map2

              SBRC R15, 0
              LDI  R16, '0'
              SBRS R15, 0
              LDI  R16, '*'
              SBRC R15, 1
              LDI  R16, '#'
              RJMP Key_map_end
[…]
```

Figure 25 - Snippet of Keypad Translation Process

7.2.4 LCD.inc

This file includes subroutines used to simplify the use of the LCD display using a reset option and a subroutine to print a pre-loaded pattern from cseg, for a pre-defined length of bytes.

```
[…]
;
; Nested subroutine to reset the LCD in order to
; refresh the display
;
LCD_rst:
              RCALL delay_37us

              ; Clear the display
              LDI R16, $01
              STS LCD_CON, R16
              RCALL delay_1_52ms

              ; Setting the entry mode
              LDI R16, $06
              STS LCD_CON, R16
              RCALL delay_37us
              RET
[…]
```

Figure 26 - Reset Subroutine for LCD

```
[…]
;
; Subroutine to print a message on the LCD
; using values obtained from a preloaded
; pattern
;
LCD_print:
              RCALL delay_1ms
              LPM R16, Z+
              STS LCD_OUT, R16
              DEC R25
              BRNE LCD_print
              RCALL delay_1ms
              RET
[…]
```

Figure 27 - Subroutine to Quickly Print a Message on the LCD

7.2.5 Serial.inc

This file contains the initialization subroutine alongside subroutine to send and receive messages with another device.

Unfortunately, due to an error that commonly occurs when trying to communicate serially, with a Raspberry Pi, a form of TCP messaging is required to ensure 100% completion in messaging.

```
[…]
;
; Subroutine used to reliably send and receive data
; between the Atmega8515 microcontroller and a
; Raspberry Pi using a TCP-like style of communication
;
simple_msg:
              LDI R18, $31
              RCALL Serial_send
              RCALL Serial_get
ready:
              LDI R18, $31
              RCALL Serial_send
              RCALL Serial_get
              RET
[…]
```

Figure 28 - Subroutine to Serially Communicate with a Raspberry Pi with TCP-like Confirmations

7.2.6 Delays.inc

This file simply contains the various loops that are used to delay the code.

These delays are created by loading registers with large values and then entering a looping function that will continue to repeat itself until the registers are empty. The number of loops that are needed for certain times can be determined using the following equation:

$$Loops = (time/5usec)$$

7.3 Raspberry Pi 3B+

The Raspberry Pi 3B+ is the cornerstone of this project. It operates as a web interface for the user as well as the controller for the microcontroller in the embedded circuit.

The web interface is handled by a partial LAMP stack of Linux, Apache2 and PHP. The MySQL component of a LAMP stack was left out because it was not needed.

In order to make the web interface more accessible, the Pi is also configured as a wireless access point using the dnsmasq and hostapd services. These services need to be installed, as they are not included in Raspbian Lite, using the following command: 'sudo apt install hostapd dnsmasq -y'.

Before configuring the access point, a network must be made for the wireless clients to obtain an IP address. Before configuring a network, a static address must be given to the wireless interface of the Pi by adding these lines to the bottom of '/etc/dhcpcd.conf':

```
interface wlan0
   static ip_address=192.168.4.1/24
   nohook wpa_supplicant
```

Figure 29 - Information added to /etc/dhcpcd.conf

Next, a new network must be defined for usage. This is done by replacing the '/etc/dnsmasq.conf' file with one that reads:

```
interface=wlan0
dhcp-range=192.168.42.10,192.168.4.20,255.255.255.0,24h
denyinterfaces eth0
dhcp-host=pizero,192.168.42.2   # assign a static IP to Pi Zero
```

Figure 30 - /etc/dnsmasq.conf on Raspberry Pi Server

Now that there is an address pool set up for the network, the access point may now be configured, the network needs to be configured in '/etc/hostapd/hostapd.conf'.

```
interface=wlan0
driver=nl80211
ssid=WILLIAM-PROJECT
hw_mode=g
channel=6
wmm_enabled=0
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=DORW20079804
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Figure 31 - /etc/hostapd/hostapd.conf on Raspberry Pi Server

Before the access point is now available for use, however, no traffic will be forwarded from eth0 to the wireless connections.

To complete the process of adding internet connectivity for devices connecting these devices, follow the instructions available at:

https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md

7.4 Web Interface

The necessary services to run the web server are installed via command line using the command: 'sudo apt install apache2 php libapache2-mod-php -y'.

The web interface consists mostly of html mark-ups for basic, static, information that is displayed such as explanatory text and forms. Information that is provided through the forms on the webpage are then processed using PHP using the "POST" method of passing the data, which will run the corresponding command line operations or data manipulations to complete the required task.

The most important part of the PHP sections on the webpage are those which handle the serial communication of the server to the microcontroller using the *serial_control.py* script. The script requires a minimum of one argument to determine what it will be communicating with the embedded circuit. The 'W' argument will grab the current value that is available from the weight sensor, where as the 'S' argument will require a second argument to set a new pass code for the device.

```
[…]
<?php
  $code = "";
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $response = $_POST["pass-yes-no"];
   if ($response == "yes"){
    $code = str_replace('.','',$_POST["passcode"]);
    if( (strlen($code)==6)&&(is_numeric($code)) ){
     `sudo python /var/www/html/scripts/serial_control.py S $code`;
    }else{
     echo "Invalid Passcode";
    }
   }
  }
?>
[…]
```

Figure 32 - Snippety from Advanced.php

8 Testing and Calibration

It is advised that use place something the between the lid of the unit and the solenoid lock before testing the unit so that the unit will not close and lock itself with no way of activating the solenoid to unlock.

For testing the embedded circuit modules, use the ‘test_routines’ set of files available from the GitHub link in the introduction or by copying the files from the corresponding appendix section.

If the issue that is being tested for originated from the LCD or the keypad entries begin with the keypad option, if the issue is from the webpage not working correctly or at all, but no changes were made to the webserver, use the TxRx_ option to test the serial connection. Finally, the ADC option can be used to check the operation of the amplifier for the load cell’s output signal.

The TxRx_ option of the testing_functions require additional user input on the side of the Raspberry Pi through the command line using the serial_test.py python script found in the ‘/home/pi/’ directory. Run the script using python 3 and by supplying an argument of a single character following the script. The output on the command line should correspond to those displayed on the LCD of the unit.

The sensitivity of the weight sensor is adjust using the potentiometer on the circuit’s PCB. This was done intentionally in order to permit the changing of load cells should a more sensitive one, or even a more durable one, be needed in place of the current one.

9 Troubleshooting

Why is there no wireless even though the Raspberry Pi's red LED is on?

When there is a lack of current for the Raspberry Pi to use, certain services are disabled to minimize the power usage of the device. Either connect the external power cord or replace the battery with a fully charged one.

Why doesn't the webpage respond when I try to access the advanced control page?

The advanced control page requires a response from the microcontroller in order to process certain data for the page. First try restarting the device in order to check if the microcontroller or Raspberry Pi missed a serial transmission confirmation bit, if the problem persists, remove the microcontroller and re-program it using the files available from the GitHub repository, found in the introduction, or using the code found in the appendix.

Why does the web interface say that the weight is a negative value?

Unfortunately, due to the nature of the load cell's output signal, it is impossible to obtain a completely stable value. If the value is significant, access the advanced control page, choose to zero the scale, and click the submit button.

Why does the unit work when a battery is connected, but not when only using external power?

There is a chance that the fuse in-line with the external power may have blown. Disconnect the power cord and open the fuse holder to inspect the fuse. If it has blown, replace the fuse and it should resume normal operation. If the problem persists, use a digital mustimeter and two thin needles to check for continuity across the transformer wires on either side. Replace wire segments as needed.

Why are there no new pictures being added to the archive section, but the live image is updating?

There are 2 possibilities as to why this is happening:

1. The camera unit's motion detector's wires have disconnected, in which case the unit must be opened, and the cables reconnected to either the board or the sensor itself.
2. The date of the camera unit is incorrect and as such is mislabelling the pictures. Disconnect the power from the camera unit and then reconnect it to updates its date and time.

Why are there black rectangles blocking the LCD display?

If there are only black rectangles on the LCD, the contrast potentiometer is turned to an extreme on the module's PCB board. Use a Philips head screwdriver to turn the potentiometer until the blocks disappear to reveal the characters.

The characters that are showing up on the LCD aren't corresponding to the ones I press on the keypad.

This means that the wires connected to the keypad internally are not in the correct order. Refer to the illustration in the appendix to reconnect the wires in the correct order.

10 Appendixes

10.1 Appendix A: Illustrations

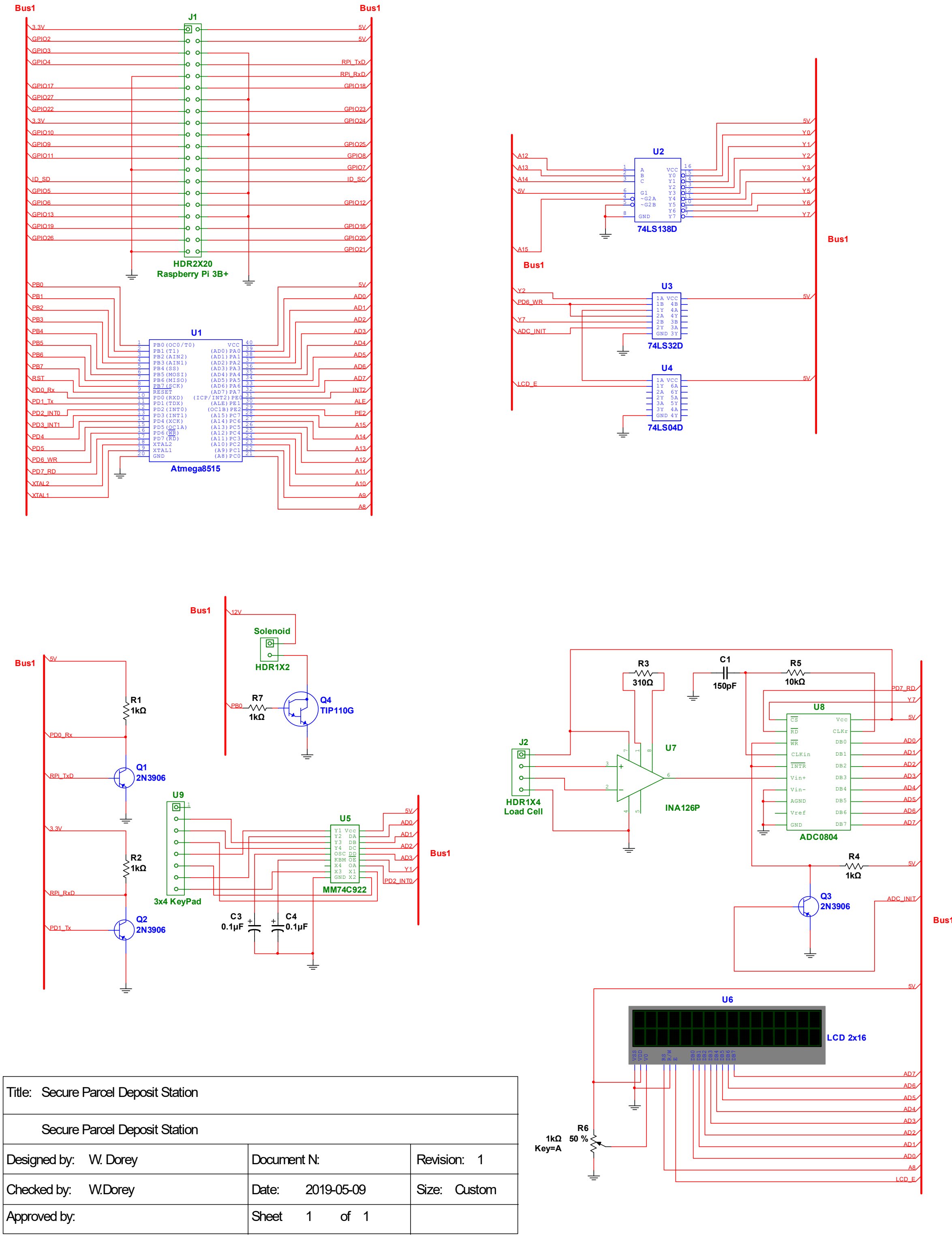
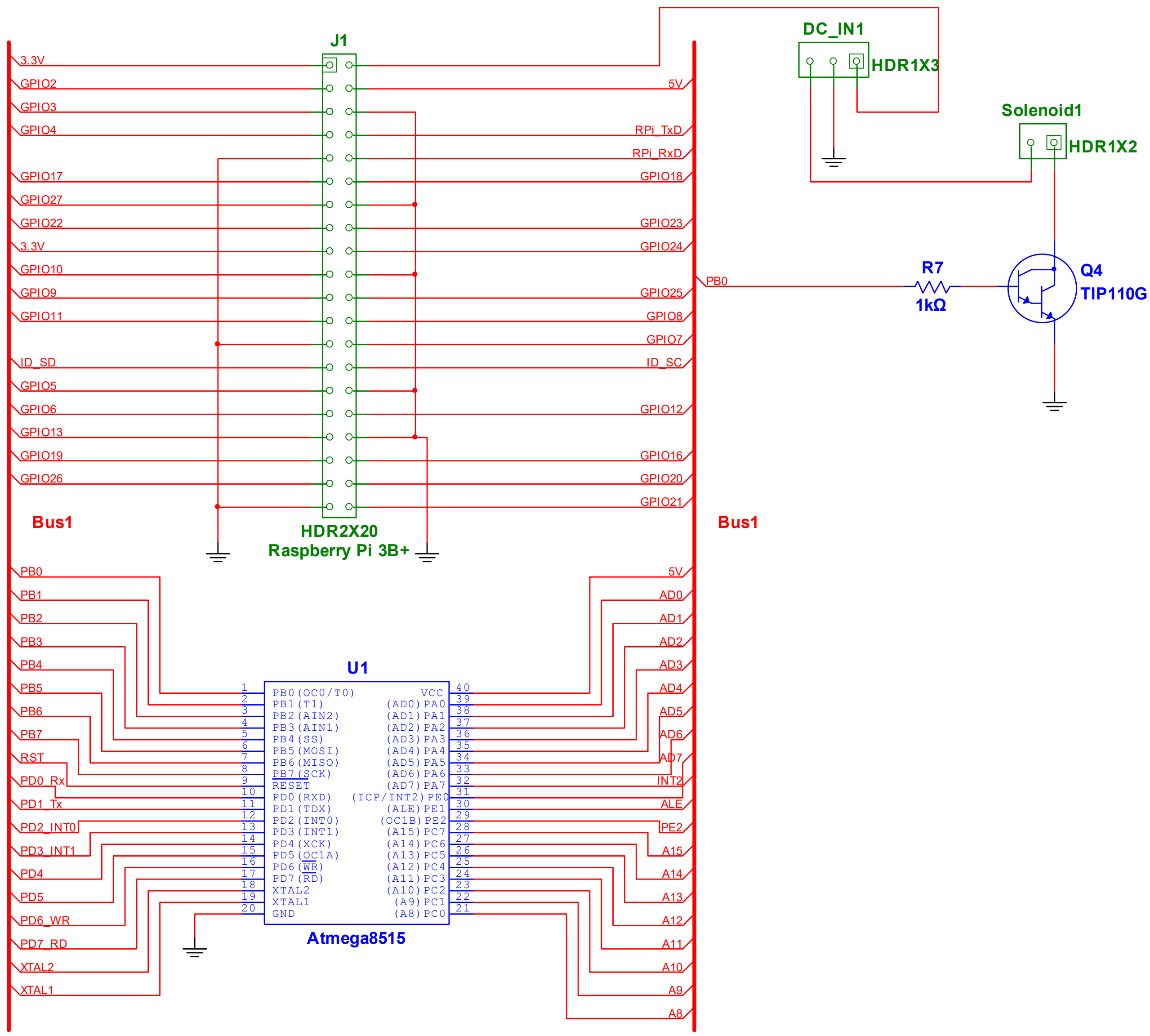
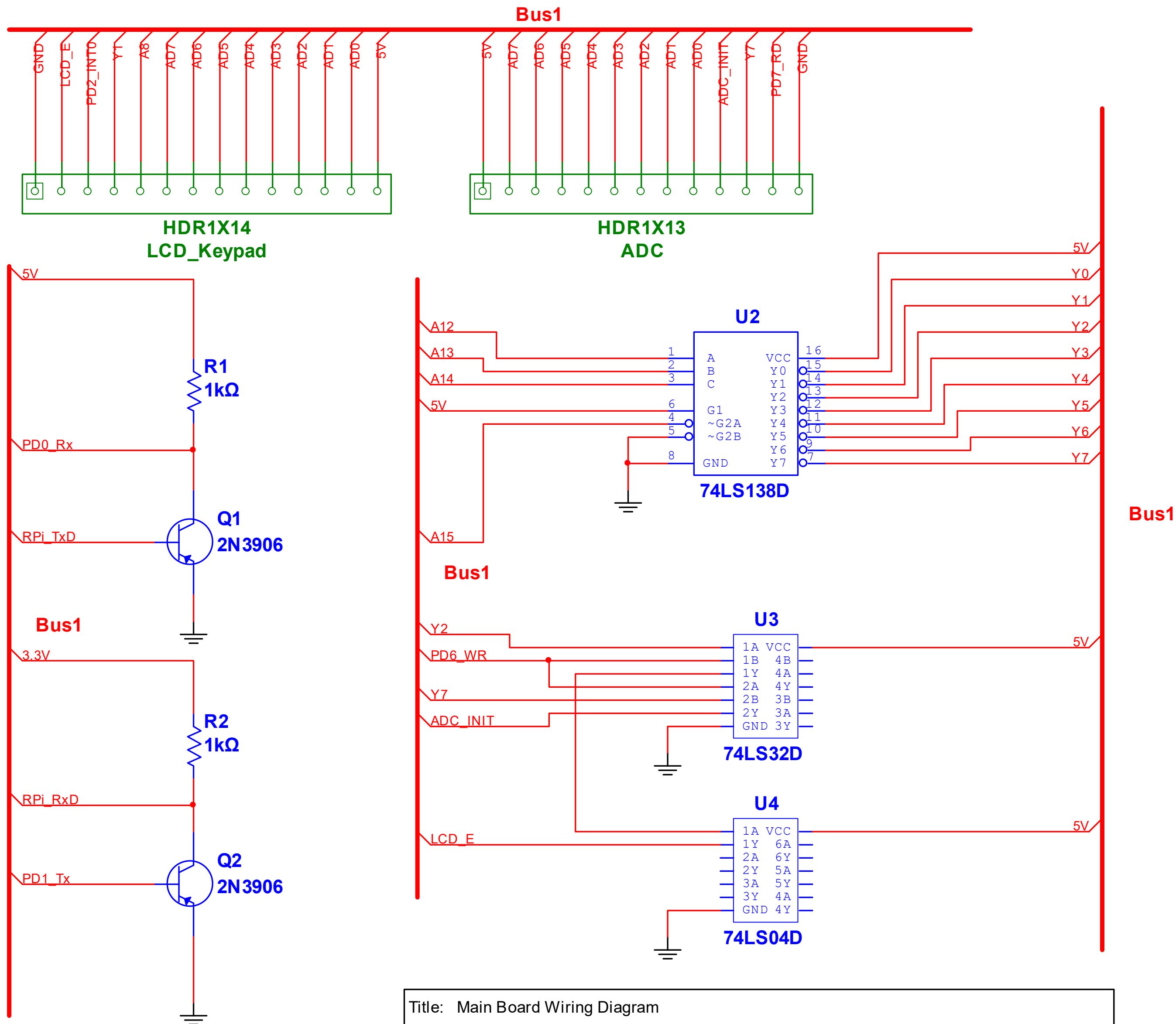


Illustration 1 - Full Circuit Schematic



Title: Main Board Wiring Diagram		
The wiring diagram for the main board of the SPDS		
Designed by: William Dorey	Document N:	Revision:
Checked by: William Dorey	Date: 5/8/2019	Size: C
Approved by: William Dorey	Sheet 1 of 2	

Illustration 2 - Main Board Wiring Diagram Part 1 of 2



Title: Main Board Wiring Diagram		
The wiring diagram for the main board of the SPDS		
Designed by: William Dorey	Document N:	Revision:
Checked by: William Dorey	Date: 5/8/2019	Size: C
Approved by: William Dorey	Sheet 2 of 2	

Illustration 3 - Main Board Wiring Diagram Part 2 of 2

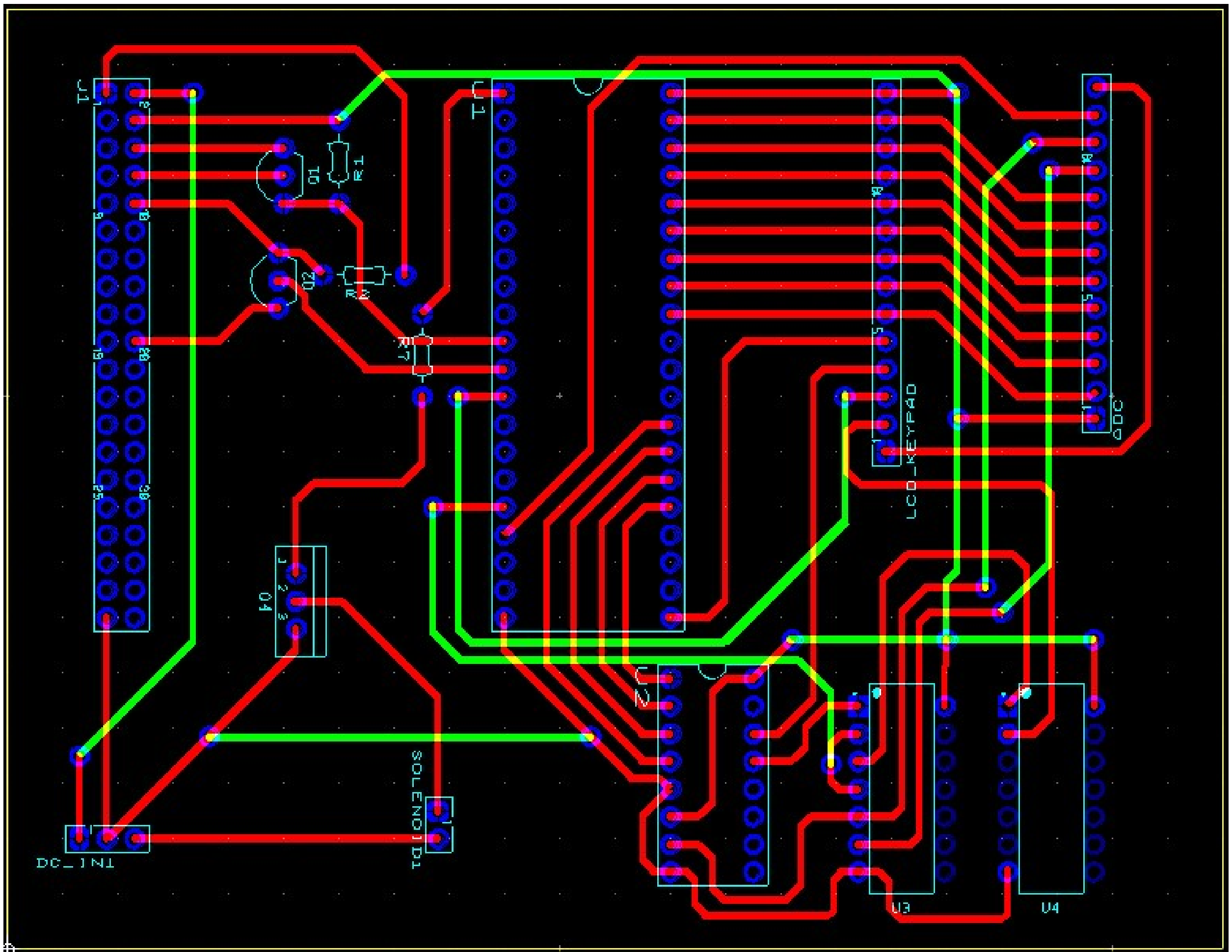
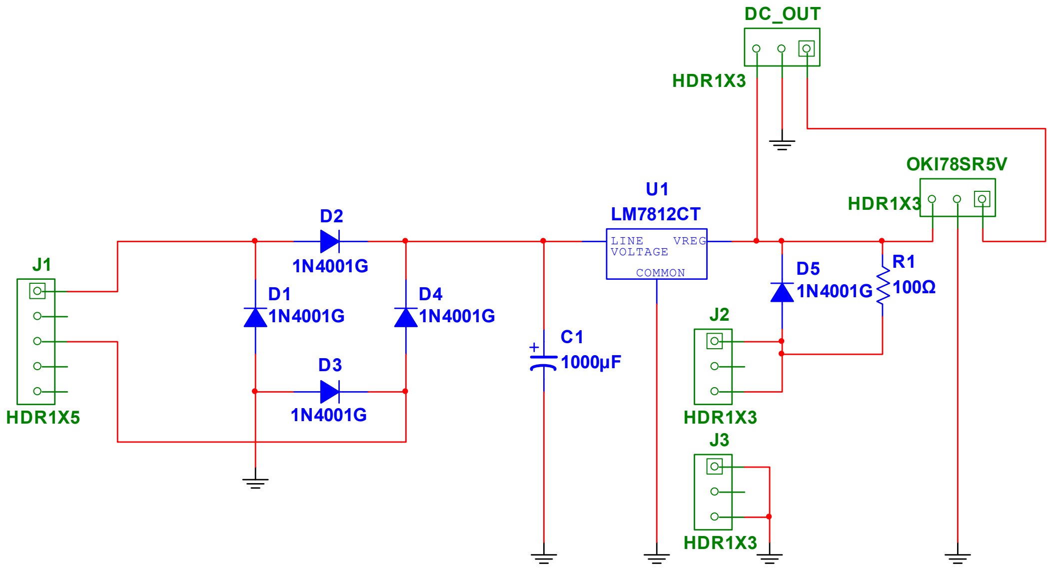


Illustration 4 - Main Board PCB Layout



Title: Power Supply Wiring Diagram		
Wiring Diagram of the power supply for the SPDS		
Designed by: William Dorey	Document N: 1	Revision: 2
Checked by: William Dorey	Date: 5/8/2019	Size: A
Approved by: William Dorey	Sheet 1 of 1	

Illustration 5 - Power Supply Wiring Diagram

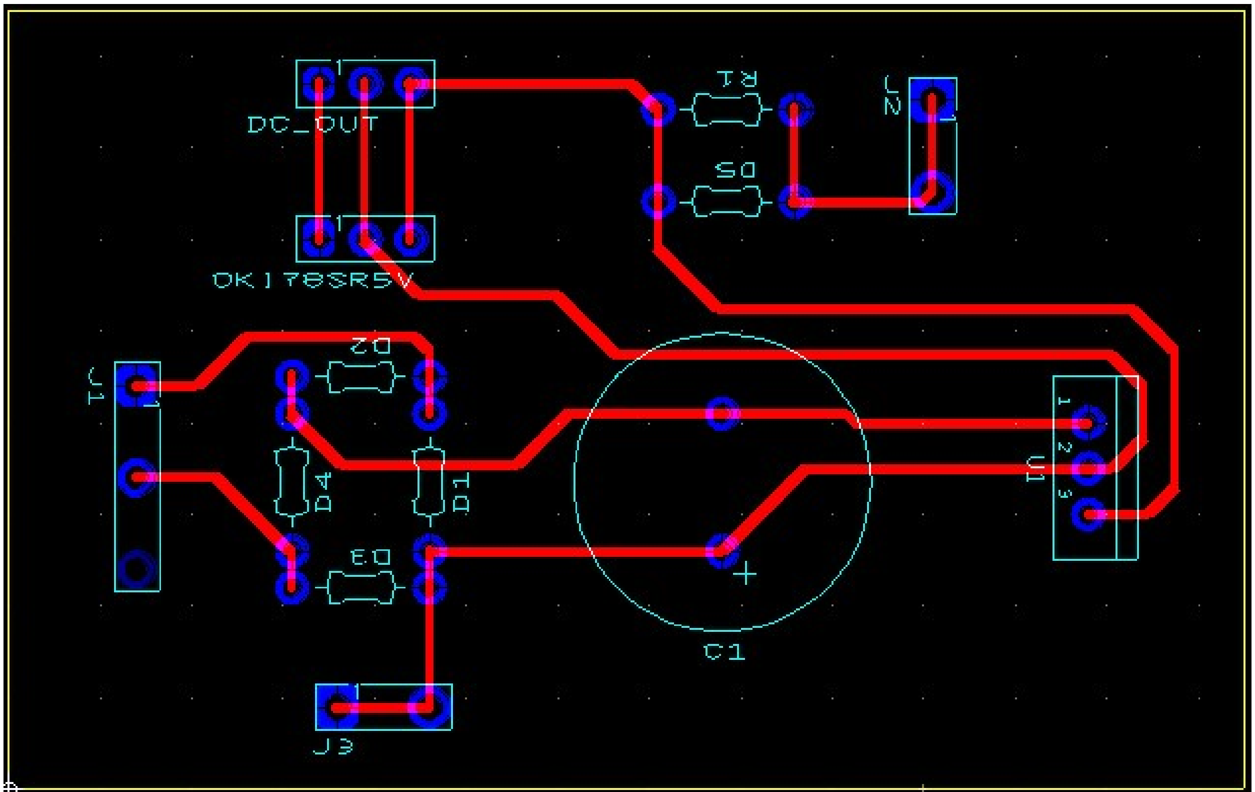
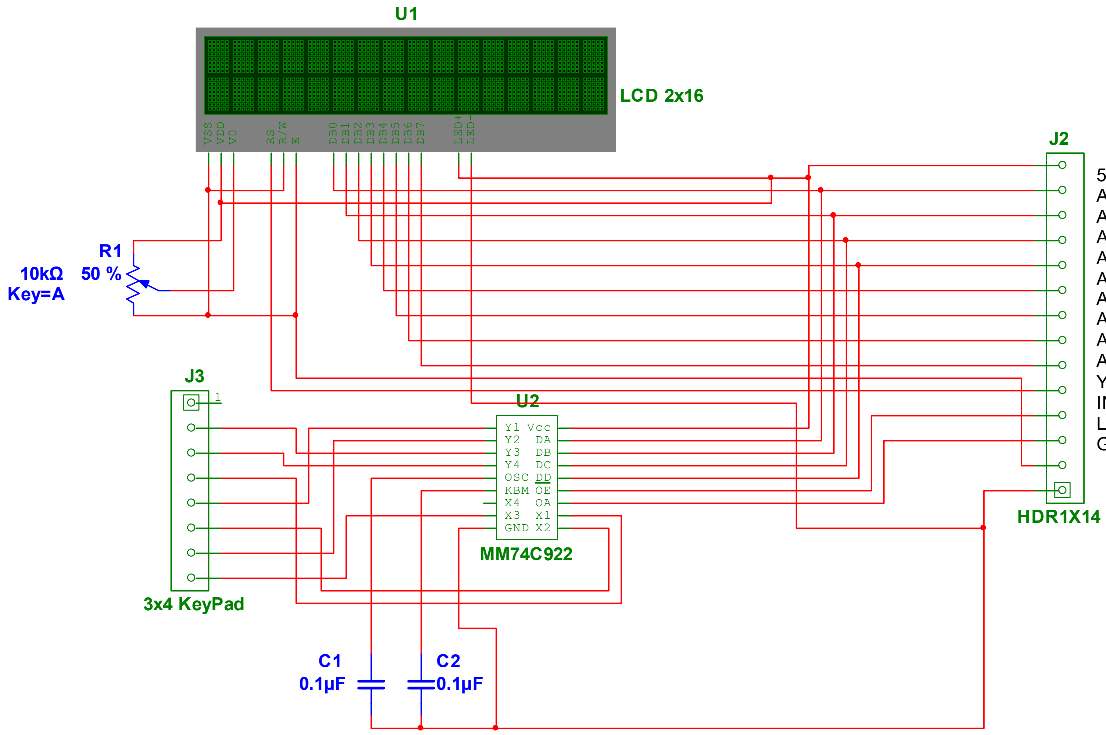


Illustration 6 - Power Supply PCB Layout



Title: LCD and Keypad Wiring Diagram		
Wiring diagram for the keypad/LCD break-out board for the SPDS		
Designed by: William Dorey	Document N:	Revision: 2
Checked by: William Dorey	Date: 2019-05-09	Size: A
Approved by: William Dorey	Sheet 1 of 1	

Illustration 7 - Keypad and LCD Wiring Diagram

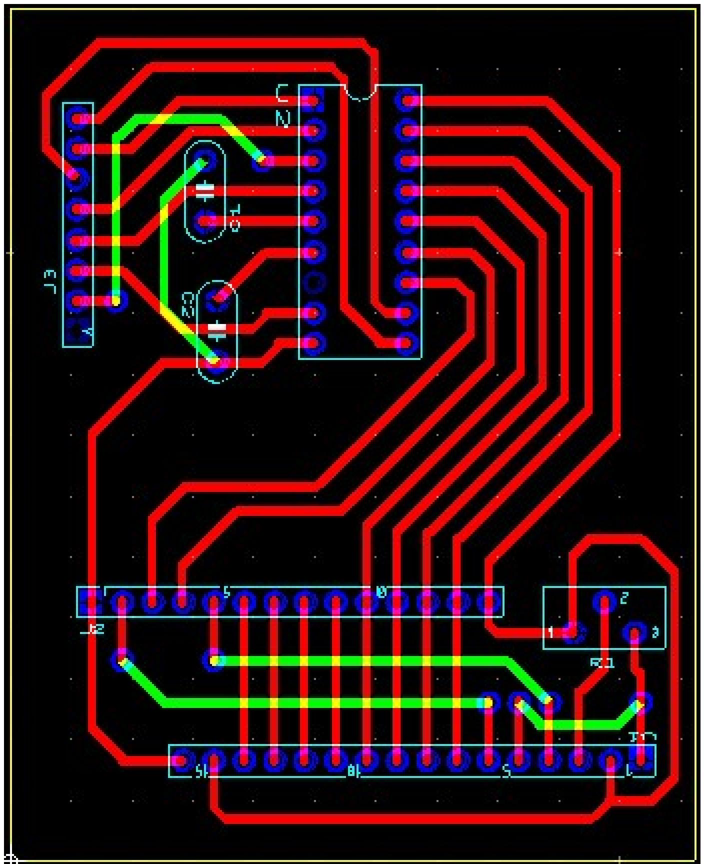
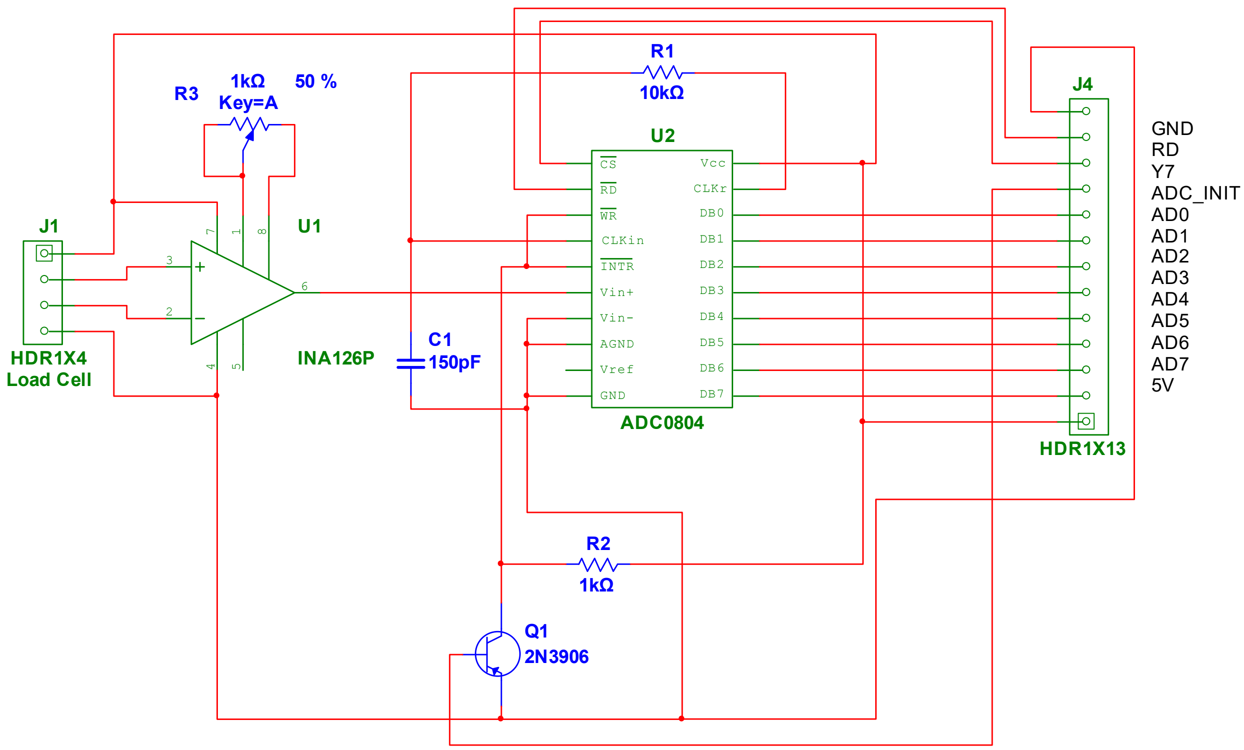


Illustration 8 - Keypad and LCD PCB Layout



Title: ADC and Amplifier Wiring Diagram		
Wiring Diagram for the amplifier and ADC break-out board of the SPDS		
Designed by: Wiliam Dorey	Document N:	Revision: 1
Checked by: William Dorey	Date: 2019-05-09	Size: A
Approved by: William Dorey	Sheet 1 of 1	

Illustration 9 - Weight Sensor, Amplifier and ADC wiring Diagram

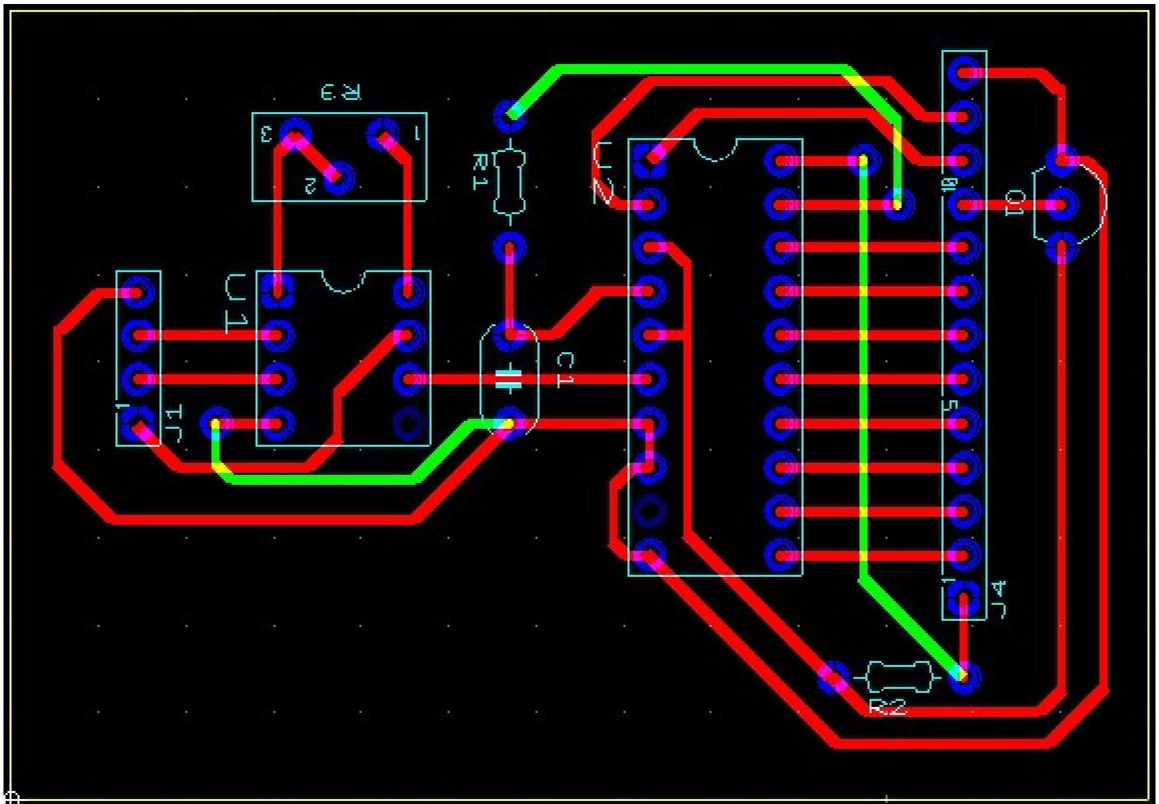


Illustration 10 - Weight Sensor, Amplifier and ADC Wiring Diagram

Pins of Keypad	4	6	8
5	1	2	3
7	4	5	6
2	7	8	9
3	*	0	#

Illustration 11 - Keypad Pin Map

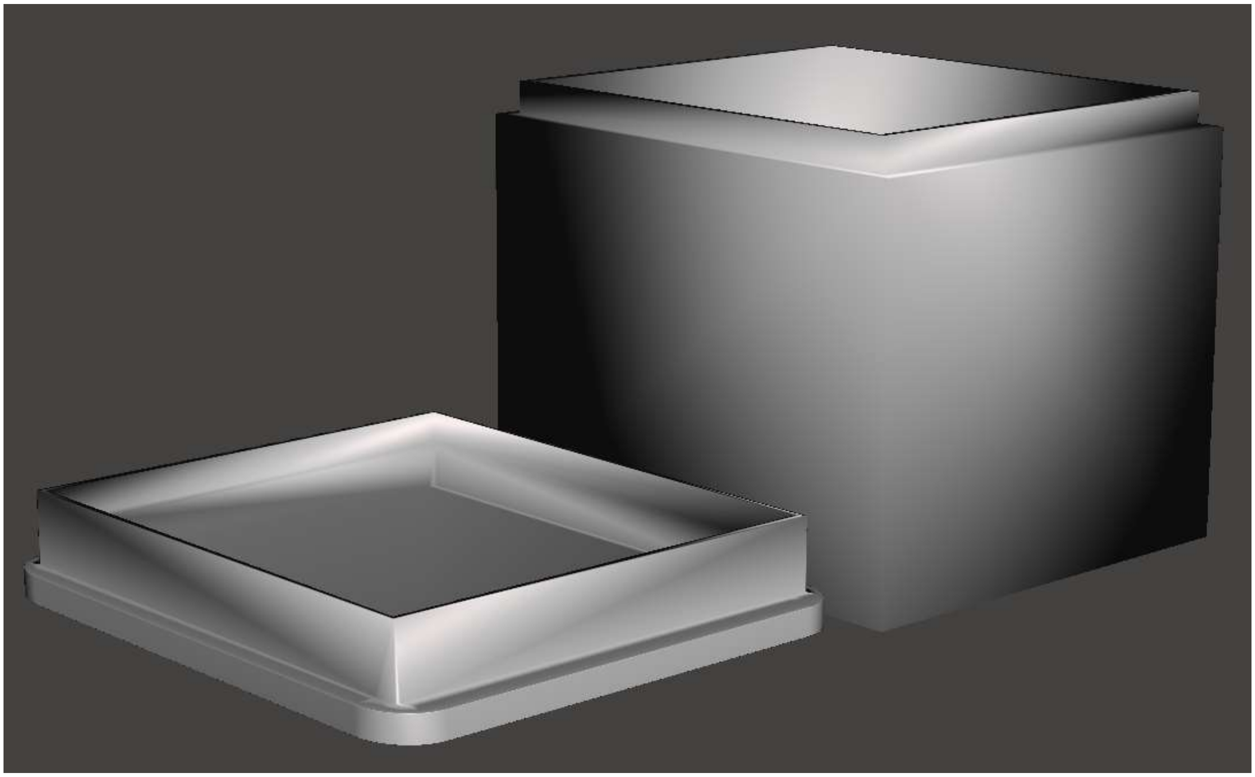


Illustration 12 - 3D model used for Camera Unit Enclosure

Illustration 13 - Solenoid Lock

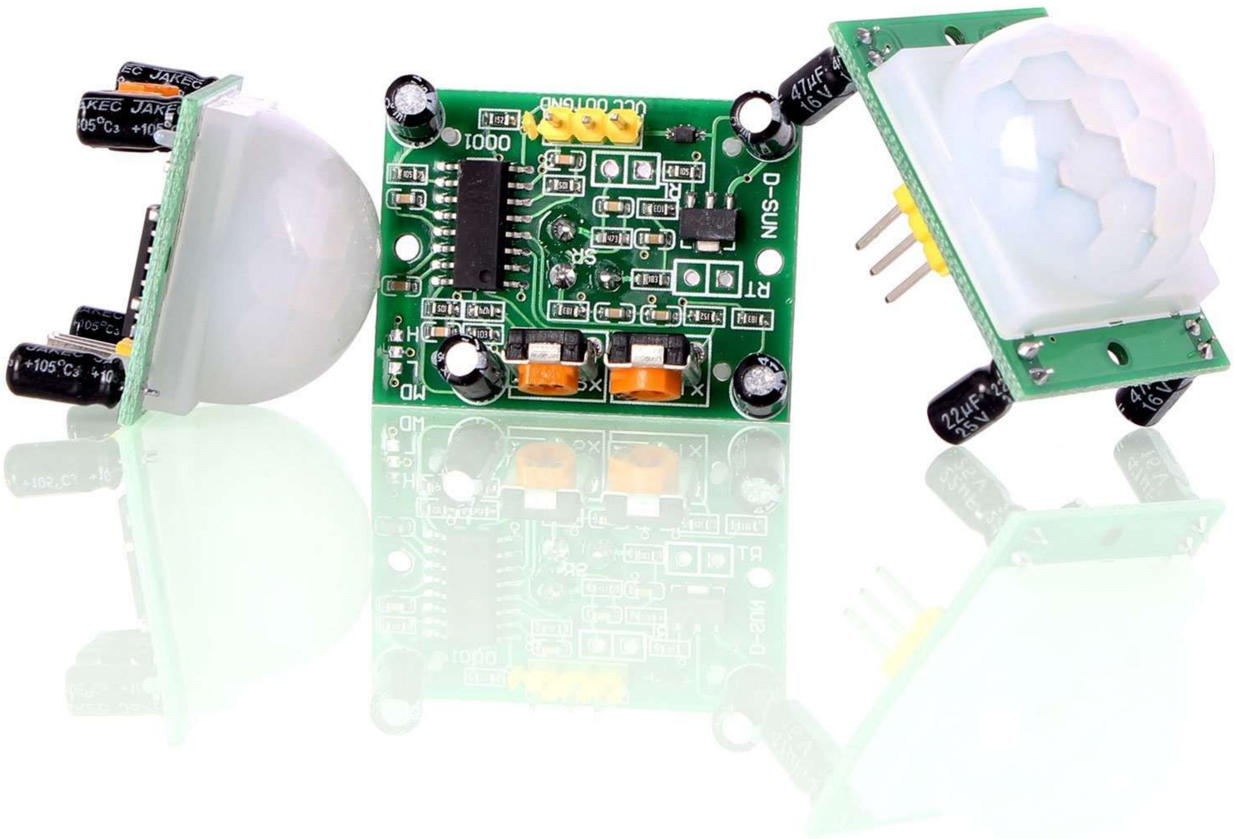


Illustration 14 - PIR Motion Sensor

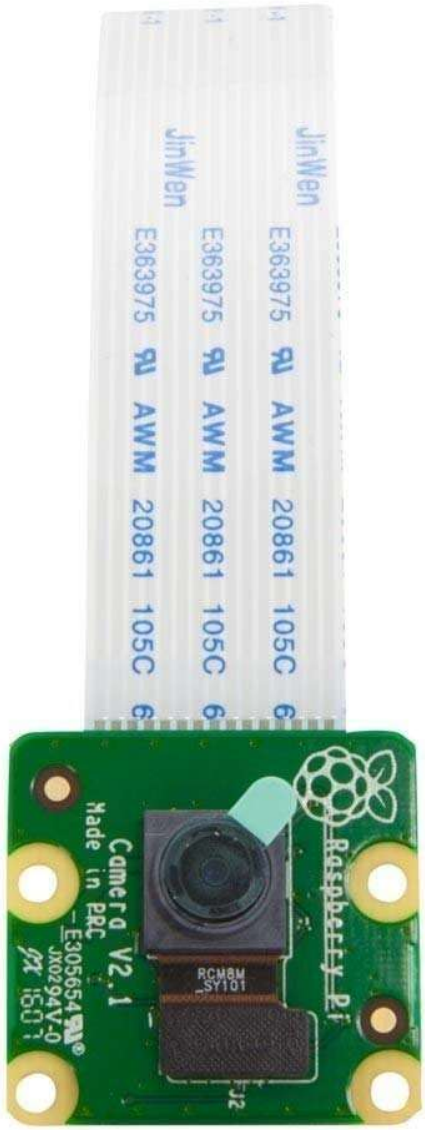


Illustration 15 - Pi Camera v2

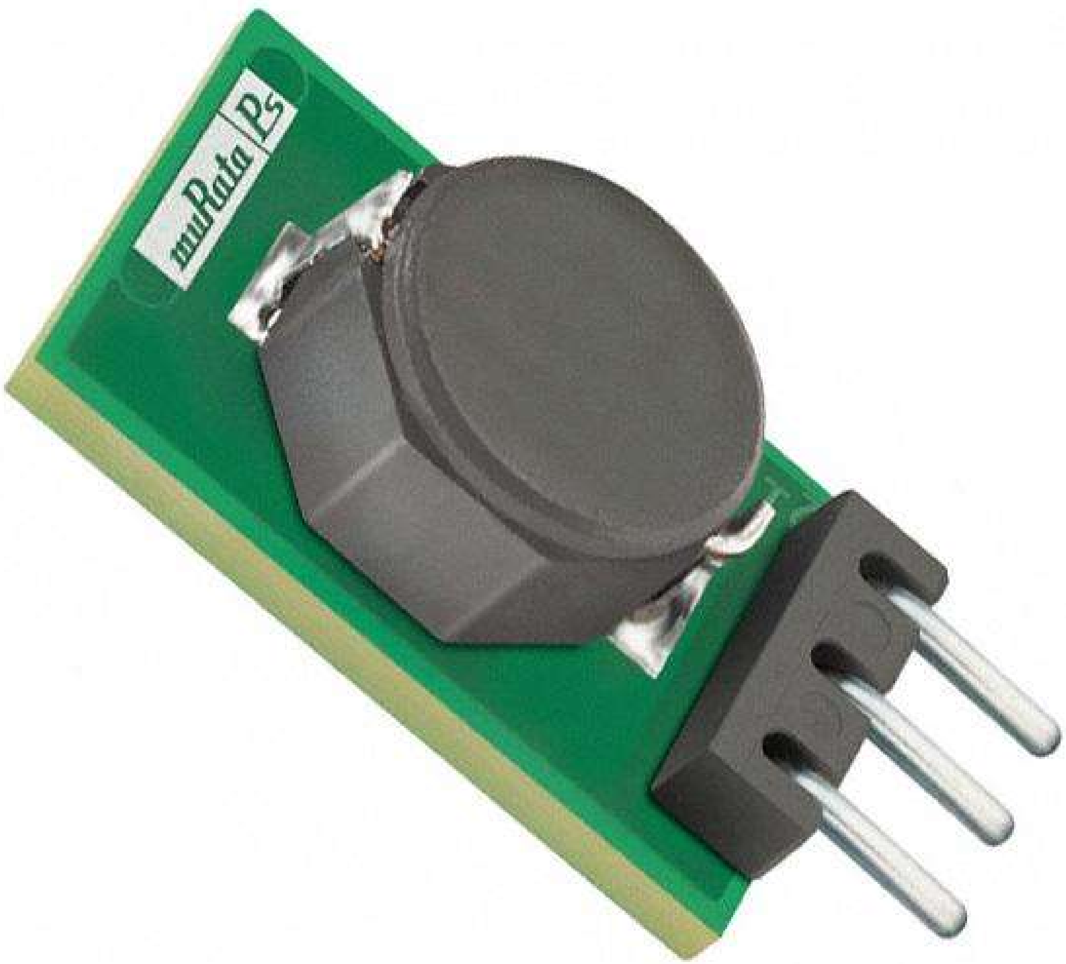


Illustration 16 - OKI-78SR-5

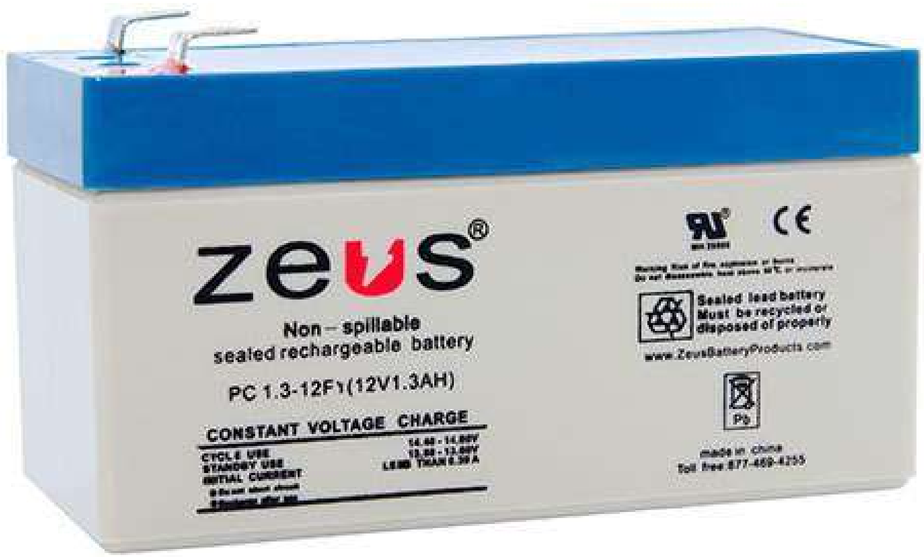


Illustration 17 - 12V Lead Acid Battery

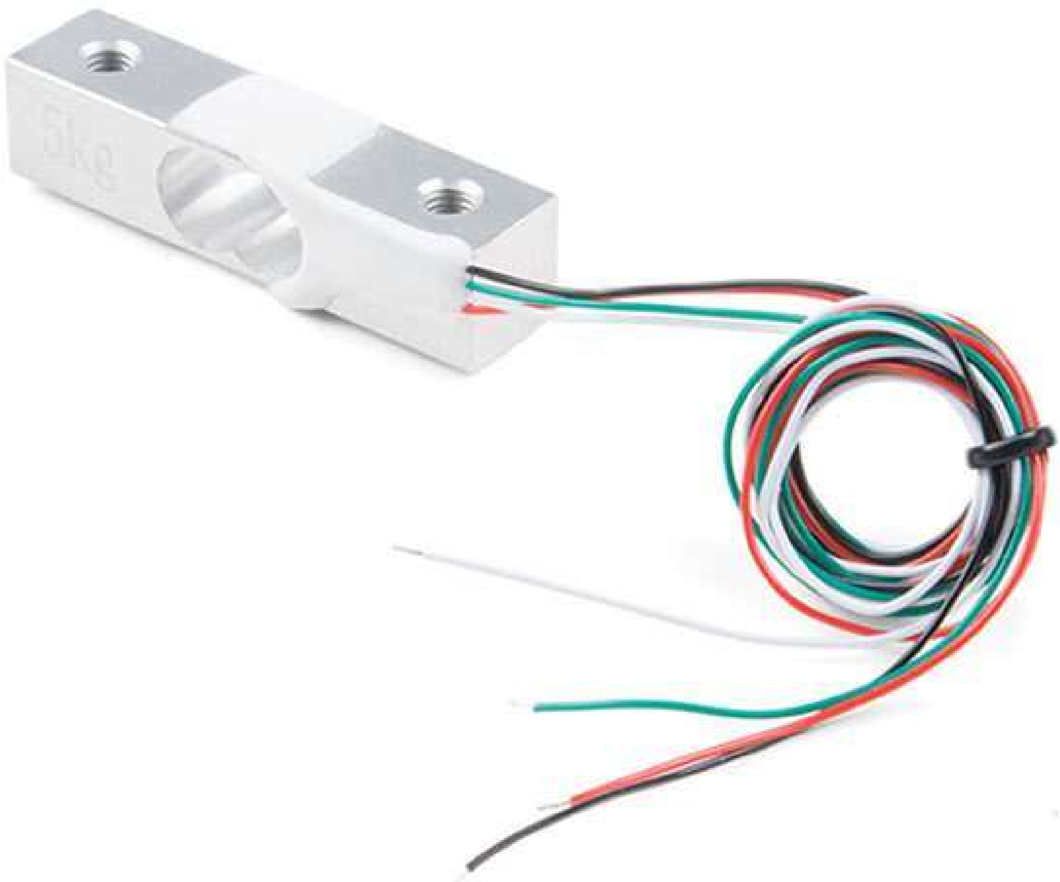


Illustration 18 - Resistive Load Cell used for Weight Sensor

10.2 Appendix B: Code/Script Listings

10.2.1 Microcontroller Code

10.2.1.1 Main.asm

```
;
; main.asm
;
; Author : William Dorey
;

;
; This is the main Assembly code file for
; the SPDS. This unit controls the locking
; mechanism of the unit as well as the
; monitoring of the weight for contents
; placed in the unit. Information is
; communicated to a Raspberry Pi server
; using UART serial transmission.
;

; Data
.dseg
pass_code:    .byte 6
.ORG $100
pass_atmp:    .byte 6

; Code
.cseg
.INCLUDE "m8515def.inc"

reset: RJMP init
              RJMP isr0
              .org $1E

; Main initialization sequence start
init:
              ; Stack pointer to allow for subroutines
              LDI R16, LOW(RAMEND)
              OUT SPL, R16
              LDI R16, HIGH(RAMEND)
              OUT SPH, R16

              ; Selecting interrupts to enable
              LDI R16, (1<<INT0)
              OUT GICR, R16

              ; Bus and Interrupt initialization
              LDI R16, $82
              OUT MCUCR, R16

              ; Sets Solenoid and kickstarts ADC
              LDI R16, $FF
              OUT DDRB, R16
              STS $1000, R16
              LDI R16, $00
              OUT PORTB, R16
```

```
              ; LCD and Serial communications
              RCALL Serial_init
              RCALL LCD_init


              ; Display banner and Checksum
              LDI R25, 9
              LDI R30, LOW(BANR<<1)
              LDI R31, HIGH(BANR<<1)
              RCALL LCD_print
              LDI R16, $C0
              STS LCD_CON, R16
              LDI R25, 9
              RCALL LCD_print

              RCALL default_pass

              RCALL Gen_Check
              RCALL delay_2s
              RCALL LCD_rst
; End of main initialization sequence

              ; Enable interrupt signal
              SEI

              LDI R25, 16
              LDI R30, LOW(PROMPT<<1)
              LDI R31, HIGH(PROMPT<<1)
              RCALL LCD_print

wait:   ; Wait for a signal from server
              ; to either set a new passcode
              ; or to obtain a value from the ADC
              RCALL Serial_get

              CPI R18, 'W'
              BRNE skip0
              RCALL Gen_ADC

skip0:
              CPI R18, 'S'
              BRNE fini
              RCALL Gen_set_pass

fini:   ; Return to waiting for a signal
        ; from the Raspberry Pi server
              RJMP wait

;
; Interrupt subroutine that begins when a value
; is made available by the keypad module
;
isr0:
              PUSH R16
              LDS R16, $1000
              RCALL Key_map

              ;Check if there is an active attempt
```

```
              CPI R22, $01
              BREQ attempt

              ; The start of a new Passcode Attempt
              CPI  R16, '*'
              BRNE isr0_end
              RCALL LCD_rst
              LDI R25, 15
              LDI R30, LOW(PASS<<1)
              LDI R31, HIGH(PASS<<1)
              RCALL LCD_print
              LDI R16, $C0
              STS LCD_CON, R16

              RCALL delay_1_52ms
              LPM R16, Z
              STS LCD_OUT, R16

              ; Load the first location of the
              ; passcode attempt data
              LDI R26, LOW(pass_atmp)
              LDI R27, HIGH(pass_atmp)
              LDI R22, $01
              RJMP isr0_end

attempt:      ; continue an active passcode
              ; attempt unless the character
              ; is a '#', in which case, end
              ; the attempt
              CPI R16, '#'
              BREQ stop
              ST X+, R16
              STS LCD_OUT, R16
              INC R23

isr0_end:
              POP R16
              RETI


;End a Passcode Attempt
stop:
              CLR R22
              RCALL Gen_code_check
              RJMP isr0_end


; Patterns
BANR:   .db    "SPDS V2.0Checksum "
PROMPT:.db     "Press * to Begin"
PASS:   .db    "Enter Passcode:>"
GOOD:   .db    "GOOD Unlock!"
BAD:    .db    "BAD Attempt!"

; List of include files
.include       "General.inc"
.include       "Keypad.inc"
.include       "LCD.inc"
```

```
.include      "Delays.inc"
.include      "Serial.inc"

; Marker for end of code
CHKSUM:       .db $00,$00

; End of main.asm
```

10.2.1.2 General.inc

```
; General.inc

;
; Various routines for miscellaneous uses
; that are not for any specific module of
; the embedded unit
;

;
; Subroutine to generate a Checksum of the
; microcontroller's cseg and then display
; it on the LCD
;
Gen_Check:
              LDI R30, LOW(CHKSUM<<1)
              LDI R31, HIGH(CHKSUM<<1)
              MOV R14, R30
              MOV R15, R31
              CLR R30
              CLR R31
              CLR R18
              CLR R19
              CLR R20
              CLR R21
addvalues:
              LPM R16, Z+
              ADD R18, R16
              ADC R19, R21
              ADC R20, R21
              CP  R30, R14
              BRNE addvalues
              CP  R31, R15
              BRNE addvalues
ChkSum_disp:
              MOV R16, R20
              RCALL Gen_hex2asc
              RCALL delay_1ms
              STS LCD_OUT, R16
              RCALL delay_1ms
              STS LCD_OUT, R17
              MOV R16, R19
              RCALL Gen_hex2asc
              RCALL delay_1ms
              STS LCD_OUT, R16
              RCALL delay_1ms
              STS LCD_OUT, R17
              MOV R16, R18
              RCALL Gen_hex2asc
              RCALL delay_1ms
              STS LCD_OUT, R16
              RCALL delay_1ms
              STS LCD_OUT, R17
              RET

;
; This routine converts the value of R16
```

```
; into ASCII text for either transmission
; using USART or display on the LCD
;
Gen_hex2asc:
            CLR R14
            MOV R15, R16
            LSR R16
            LSR R16
            LSR R16
            LSR R16
asc_chk:
            CPI R16, $0A
            BRLO number
letter:
            SUBI R16, $09
            LDI R18, $40
            ADD R16, R18
            RJMP asc_done
number:
            LDI R18, $30
            ADD R16, R18
            RJMP asc_done
next_byte:
            MOV R17, R16
            MOV R16, R15
            ANDI R16, $0F
            RJMP asc_chk
asc_done:
            INC R14
            LDI R18, 2
            CP R14, R18
            BRNE next_byte
            RET

;
; Subroutine to pull a value from the ADC
; and then send the value through USART to
; the Raspberry Pi server
;
Gen_ADC:
            RCALL simple_msg
            LDS R16, $7000
            RCALL Gen_hex2asc
            RCALL Serial_get
            MOV R18, R17
            RCALL Serial_send
            NOP
            RCALL Serial_get
            MOV R18, R16
            RCALL Serial_send
            RET

;
; Subroutine to activate the solenoid, unlocking
; the SPDS for 5 seconds
;
Gen_unlock:
            PUSH R16
```

```
                PUSH R17
                LDI R16, $01
                OUT PORTB, R16
                RCALL delay_5s
                CLR R16
                OUT PORTB, R16
                POP R17
                POP R16
                RET

;
; Subroutine to set the default passcode of
; '******' upon start up after power failure
;
default_pass:
                LDI R16, '*'
                LDI R29, HIGH(pass_code)
                LDI R28, LOW(pass_code)
                ST  Y+, R16
                ST  Y+, R16
                ST  Y+, R16
                ST  Y+, R16
                ST  Y+, R16
                ST  Y+, R16
                RET
;
; Subroutine to set a new passcode by retrieving
; the values from the Raspberry Pi server and
; storing them in the data segment
;
Gen_set_pass:
                RCALL simple_msg
                LDI R29, HIGH(pass_code)
                LDI R28, LOW(pass_code)
                LDI R16, 6
Gen_set_pass_loop:
                RCALL Serial_get
                ST Y+, R18
                RCALL Serial_send
                DEC R16
                BRNE Gen_set_pass_loop
                RET

;
; Subroutine to check an attempted password
; against the one stored in the data segment
;
Gen_code_check:
                CPI R23, 6
                BRNE bad_attempt

                LDI R29, HIGH(pass_code)
                LDI R28, LOW(pass_code)
                LDI R27, HIGH(pass_atmp)
                LDI R26, LOW(pass_atmp)

Gen_code_check_loop:
                LD R16, X+
```

```
                LD R17, Y+
                CP  R17, R16
                BRNE bad_attempt

                DEC R23
                BRNE Gen_code_check_loop

good_attmept:
                RCALL LCD_rst
                LDI R25, 12
                LDI R30, LOW(GOOD<<1)
                LDI R31, HIGH(GOOD<<1)
                RCALL LCD_print
                RCALL Gen_unlock
                RJMP Gen_code_check_end

bad_attempt:
                RCALL LCD_rst
                LDI R25, 12
                LDI R30, LOW(BAD<<1)
                LDI R31, HIGH(BAD<<1)
                RCALL LCD_print
                RCALL delay_2s

Gen_code_check_end:
                RCALL LCD_rst
                LDI R25, 16
                LDI R30, LOW(PROMPT<<1)
                LDI R31, HIGH(PROMPT<<1)
                RCALL LCD_print
                CLR R23
                RET

; End of General.inc
```

10.2.1.3 Serial.inc

```
; Serial.inc
;
; Subroutines used for serial communication using
; the Atmega8515's USART functionality
;

; Important configuration values
.EQU BAUD = 25
.EQU UTCLB = $18
.EQU FRAME = $86

;
; Subroutine used to initialize the UART
; functionality of the Atmega8515 microcontroller
;
Serial_init:
              ; Sets the baud rate
              LDI R16, $00
              OUT UBRRH, R16
              LDI R16, BAUD
              OUT UBRRL, R16
              ; Sets the control and status
              LDI R16, UTCLB
              OUT UCSRB, R16
              ; Sets the frame properties
              LDI R16, FRAME
              OUT UCSRC, R16
              RET

;
; Subroutine used to receive a single byte of data
; through USART on the Atmega8515 microcontroller
;
Serial_get:
              IN R18, UCSRA
              ANDI R18, $80
              BREQ Serial_get
              IN R18, UDR
              RET

;
; Subroutine used to transmit a single byte of data
; through USART on the Atmega8515 microcontroller
;
Serial_send:
              OUT UDR, R18
Serial_wait:         ; wait for a bit confirming reception
              IN R18, UCSRA
              CPI R18, $20
              BREQ Serial_wait
              RET

;
; Subroutine used to reliably send and receive data
; between the Atmega8515 microcontroller and a
; Raspberry Pi using a TCP-like style of communication
```

```
;
simple_msg:
             LDI R18, $31
             RCALL Serial_send
             RCALL Serial_get
ready:
             LDI R18, $31
             RCALL Serial_send
             RCALL Serial_get
             RET

; End of Serial.inc
```

10.2.1.4 LCD.inc

```
; LCD.inc
;
; Subroutines for controlling an LCD unit
; using the Atmega8515's data bus signals
;

; LCD addresses for defining control and
; and output values

.EQU LCD_CON = $2000
.EQU LCD_OUT = $2100


;
; Subroutine to initialize the LCD for basic
; display purposes
;
LCD_init:
              RCALL delay_40ms

              ; Function set
              LDI R16, $38
              STS LCD_CON, R16
              RCALL delay_37us
              STS LCD_CON, R16
              RCALL delay_37us

              ; Turn on the display
              LDI R16, $0C
              STS LCD_CON, R16

;
; Nested subroutine to reset the LCD in order
; refresh the display
;
LCD_rst:
              RCALL delay_37us

              ; Clear the display
              LDI R16, $01
              STS LCD_CON, R16
              RCALL delay_1_52ms

              ; Setting the entry mode
              LDI R16, $06
              STS LCD_CON, R16
              RCALL delay_37us
              RET


;
; Subroutine to print a message on the LCD
; using values obtained from a preloaded
; pattern
;
LCD_print:
```

```
            RCALL delay_1ms
            LPM R16, Z+
            STS LCD_OUT, R16
            DEC R25
            BRNE LCD_print
            RCALL delay_1ms
            RET

; End of LCD.inc
```

10.2.1.5 Keypad.inc

```
; Keypad.inc

;
; This subroutine translates the signal from
; the keypad controller IC into the ASCII
; value that was selected on the keypad
;
Key_map:
              MOV  R15, R16
              SBRS R15, 3
              RJMP Key_map0
              SBRS R15, 2
              RJMP Key_map2

              SBRC R15, 0
              LDI  R16, '0'
              SBRS R15, 0
              LDI  R16, '*'
              SBRC R15, 1
              LDI  R16, '#'
              RJMP Key_map_end

Key_map0:
              SBRC R15, 2
              RJMP Key_map1

              INC  R16
              LDI  R17, $40
              ADD  R16, R17
              RJMP Key_map_end

Key_map1:
              LDI R17, $40
              ADD R16, R17
              RJMP Key_map_end

Key_map2:
              DEC  R16
              LDI  R17, $40
              ADD  R16, R17
Key_map_end:
              RET

; End of Keypad.inc
```

10.2.1.6 Delays.inc

```
; Delays.inc

;
; The subroutines each load their required
; loop counts and then jump to the loop
;
delay_37us:
              PUSH R27
              PUSH R26
              LDI R26, $06
              LDI R27, $00
              RJMP delay_loop

delay_1ms:
              PUSH R27
              PUSH R26
              LDI R26, $C8
              LDI R27, $00
              RJMP delay_loop

delay_1_52ms:
              PUSH R27
              PUSH R26
              LDI R26, $30
              LDI R27, $01
              RJMP delay_loop

delay_5ms:
              PUSH R27
              PUSH R26
              LDI R26, $6F
              LDI R27, $02
              RJMP delay_loop

delay_40ms:
              PUSH R27
              PUSH R26
              LDI R26, $40
              LDI R27, $1F
              RJMP delay_loop

delay_loop:
              SBIW X, 1
              NOP
              BRNE delay_loop
              POP R26
              POP R27
              RET

; Longer delays that require an additional register
; for loop counts
delay_5s:
              PUSH R28
              PUSH R27
              PUSH R26
              LDI  R26, $40
```

```
              LDI  R27, $42
              LDI  R28, $10
              RJMP delay_extended_loop

delay_2s:
              PUSH R28
              PUSH R27
              PUSH R26
              LDI  R26, $80
              LDI  R27, $1A
              LDI  R28, $07
              RJMP delay_extended_loop

delay_extended_loop:
              SBIW X, 1
              NOP
              BRNE delay_extended_loop
              DEC  R28
              BRNE delay_extended_loop
              POP R26
              POP R27
              POP R28
              RET

; End of Delays.inc
```

10.2.2 Microcontroller Test Functions

10.2.2.1 Main.asm

```
;
;File: Project_test_routines.asm
;Author: William Dorey
;
;Description: A compilation of different subroutines
;                           used to test the individual modules
;                           connected to the microcontroller
;
; Bus Map:
;       Keypad=       $1XXX
;       LCD=    $2XXX
;       ADC=    $7XXX

; Data Segment
.dseg
.org $0100
code_saved: .byte 6
code_attempt: .byte 6

;Code segment
.cseg
.INCLUDE "m8515def.inc"

.EQU BAUD = 25   ; 2400bps
.EQU UCTLB = $18 ; Tx Rx enabled
.EQU FRAME = $86 ; Asynchronous, No Parity, 1 Stop bit, 8 Data bits

reset: RJMP init
              RJMP isr0
              .ORG $1E
init:
              LDI R16, LOW(RAMEND) ; Initialize Stack Pointer
              OUT SPL, R16
              LDI R16, HIGH(RAMEND)
              OUT SPH, R16

              LDI R16, (1<<INT0)          ; Enable INT0
              OUT GICR, R16

              LDI R16, $82                ; Initialize Data Buses and Interrupt for
falling edge of int0
              OUT MCUCR, R16

              RCALL init_uart                    ; Initialize Serial Communication
              RCALL init_LCD

              LDI R19, $41

banner:
              LDI R25, 14
              LDI R30, LOW(LCD_TEST<<1)
              LDI R31, HIGH(LCD_TEST<<1)
              RCALL LCD_text

              rcall delay_1_52ms
```

```
              LDI R16, $C0
              STS LCD_CONTROL, R16

main:  ; Uncomment the module that is being tested.
;             RCALL adc_
;             RCALL key_
;             RCALL TxRx_

fini:
              RJMP fini

; Sets the banner to notify the user of what is being
; tested and begins listening for interrupt signals
key_:
              LDI R25, 10
              LDI R30, LOW(KEY_TEST<<1)
              LDI R31, HIGH(KEY_TEST<<1)
              RCALL LCD_text
              SEI
              RET

; Sets the banner to notify the user of what is being
; tested and initializes the ADC
adc_:
              LDI R16, $FF          ; ADC Kickstart
              STS $1000, R16

              LDI R25, 5
              LDI R30, LOW(ADC_TEST<<1)
              LDI R31, HIGH(ADC_TEST<<1)
              RCALL LCD_text
start_adc:    ; Reads the input from the ADC and then
                     ; converts the value to ascii for the LCD
                     ; to display
              LDS R16, $7000
              RCALL hex2asc
              RCALL delay_1ms
              STS LCD_OUT, R17
              RCALL delay_1ms
              STS LCD_OUT, R16
              LDI R16, $10
              RCALL delay_1_52ms
              STS LCD_CONTROL, R16
              RCALL delay_1_52ms
              STS LCD_CONTROL, R16
              RJMP start_adc
              RET

;
; Interrupt subroutine to
; read incoming keypad entries and display them on the LCD
;
isr0:
              LDS R16, $1000
              RCALL key_map
              STS LCD_OUT, R16

              LDI R16, $10
```

```
              RCALL delay_1_52ms
              STS LCD_CONTROL, R16
              RETI

; Patterns
LCD_TEST:     .db    "This is a test"
ADC_TEST:     .db    "ADC: ",$00
KEY_TEST:     .db "Last key: "
TXR_TEST:     .db    "Tx: * Rx: *", $00

; External Files
.include "General.inc"
.include "Delays.inc"
.include "LCD.inc"
.include "Keypad.inc"
.include "Serial.inc"
```

10.2.2.2 General.inc

```
;
; File name: General.inc
; Description:       Subroutines that are used in main.asm
;             but don't fall under a specific module
;


;
; Purpose: To convert the hex value in a register to the ascii value
; split across two registers
; Registers Used: R16(Input), R15, R17:R16(Output)     msb:lsb
;
hex2asc:
              CLR R14
              MOV R15, R16
              LSR R16
              LSR R16
              LSR R16
              LSR R16
asc_chk:
              CPI R16, $0A  ; Check if the HEX value is a letter
              BRLO number

letter:
              SUBI R16, $09
              LDI R18, $40
              ADD R16, R18
              RJMP asc_done
number:
              LDI R18, $30
              ADD R16, R18
              RJMP asc_done

next_byte:
              MOV R17, R16
              MOV R16, R15
              ANDI R16, $0F
              RJMP asc_chk

asc_done:
              INC R14
              LDI R18, 2
              CP R14, R18
              BRNE next_byte
              RET
```

10.2.2.3 Serial.inc

```
;
; File name: Serial.inc
; Description:      Subroutine to aide in serial communication
;                          with a raspberry pi
;


;
; Purpose: To initialize the serial communication registers
; according to the values chosen by changing the values of
; BAUD, UCTLB and FRAME at the top of the code.
; Registers Used: R16
;
init_uart:
              LDI r16, 0
              OUT UBRRH, r16
              LDI r16, BAUD
              OUT UBRRL, R16              ; configures the bit rate
              LDI R16, UCTLB
              OUT    UCSRB, R16           ; configures the Tx/Rx channel
              LDI R16, FRAME
              OUT UCSRC, r16              ; configures frame elements
              RET

simple_msg:
              LDI R16, 1
              RCALL putch
              RCALL getch
ready:
              MOV R16, R19
              RCALL putch
              RCALL getch
              MOV R16, R19
              RET
;
; Purpose: To transmit messages consisting of many bytes
; to the Raspberry Pi through a serial connection
; Registers Used: R16, R17(Input), R30(Input), R31(Input)
;
msg_send:
              MOV R16, R17
              RCALL putch
              RCALL getch
rdy:
              LPM R16, Z+
              RCALL putch
              RCALL getch
              DEC R17
              BRNE rdy
              RET

;
; Purpose: To the read a single Byte of data from the serial
; connection
; Registers Used: R16(Output)
;
getch:
```

```
              IN R16, UCSRA
              ANDI R16, $80
              BREQ getch
              IN R16, UDR
              RET
;
; Purpose to transmit a single byte of data through serial
; and then wait for the process to complete
; Registers Used: R16(Input)
;
putch:
              OUT UDR, R16
wait0:
              IN R16, UCSRA
              CPI R16, $20
              BREQ wait0
              RET

;
; Purpose:    A Subroutine to test communication between
;                    microcontroller and the Raspberry Pi
;
TxRx_:
              LDI R25, 11
              LDI R30, LOW(TXR_TEST<<1)
              LDI R31, HIGH(TXR_TEST<<1)
              RCALL LCD_text

Rx:
              LDI R16, $10
              RCALL delay_1_52ms
              STS LCD_CONTROL, R16
              RCALL getch
              STS LCD_OUT, R16
              LDI R20, 7

Tx:
              LDI R16, $10
              RCALL delay_1_52ms
              STS LCD_CONTROL, R16
              DEC R20
              BRNE Tx

              RCALL simple_msg
              STS LCD_OUT, R16

              INC R19
              LDI R20, 6
              LDI R16, $14
move_back:
              RCALL delay_1_52ms
              STS LCD_CONTROL, R16
              DEC R20
              BRNE move_back
              RJMP Rx
              RET
```

10.2.2.4 LCD.inc

```
;
; File name: LCD.inc
; Description: Subroutines used to control the LCD display
;

.EQU LCD_CONTROL = $2000
.EQU LCD_OUT = $2100

;
; Subroutine to init the LCD display for use
;
init_LCD:
       rcall delay_40ms

       ldi R16, $38
       STS LCD_CONTROL, R16 ; Function Set
       rcall delay_37us

       STS LCD_CONTROL, R16 ; Function Set
       rcall delay_37us

       ldi R16, $0C
       STS LCD_CONTROL, R16 ; Turn on the Display
LCD_reset:                  ; Label used to reset the display when needed
                                   ; without fully reinitializing
       rcall delay_37us

       ldi R16, $01
       STS LCD_CONTROL, R16 ; Clear the Display
       rcall delay_1_52ms

       ldi R16, $06
       STS LCD_CONTROL, R16 ; Entry mode set
       rcall delay_37us
       ret

;
; Subroutine to output ASCII characters to the
; most recent spot available on the LCD
;
LCD_text:
       rcall delay_1ms
       lpm   R16, Z+      ;get char from memory
       STS    LCD_OUT, R16
       DEC R25
       brne  LCD_text
       ret
```

10.2.2.5 Keypad.inc

```
;
; File name: Keypad.inc
; Description: Subroutines to aide in using a keypad
;

;
; Purpose: To translate the input values from a keypad into
; the ASCII values they represent
; Registers used: R16(Input/Output), R15
;
key_map:
              MOV  R15, R16
              SBRS R15, 3
              RJMP key_map_0
              SBRS R15, 2
              RJMP key_map_2

              SBRC R15, 0
              LDI  R16, '0'
              SBRS R15, 0
              LDI  R16, '*'
              SBRC R15, 1
              LDI  R16, '#'
              RJMP keymap_end

key_map_0:
              SBRC R15, 2
              RJMP key_map_1

              INC  R16
              LDI  R17, $40
              ADD  R16, R17
              RJMP keymap_end

key_map_1:
              LDI R17, $40
              ADD R16, R17
              RJMP keymap_end

key_map_2:
              DEC  R16
              LDI  R17, $40
              ADD  R16, R17
keymap_end:
              RET
```

10.2.2.6 Delays.inc

```
;
;Delay Subroutines
;
delay_5ms:
PUSH R26
PUSH R27
	LDI R26, $E8
	LDI R27, $04
timer5:
	NOP
	SBIW X, 1
	BRNE timer5
POP R27
POP R26
	RET

delay_1ms:
	push r24
	LDI R24, $C8
timer4:
	NOP
	NOP
	DEC R24
	BRNE timer4
	pop r24
	RET

delay_2s:
PUSH R26
PUSH R27
PUSH R24
	ldi R26, $80
	ldi R27, $1A
	ldi R24, $06
timer3:
	NOP
	SBIW X, 1
	brne timer3
	dec R24
	brne timer3
POP R24
POP R27
POP R26
	ret

delay_1_52ms:
PUSH R26
PUSH R27
	ldi r26,$30
	ldi r27,$01
timer2:
	NOP
	SBIW X, 1
	brne timer2
POP R27
POP R26
```

```
        ret

delay_37us:
push r24
        ldi R24, 9
timer1:
        NOP
        dec R24
        brne timer1
        pop r24
        ret

delay_40ms:
PUSH R26
PUSH R27
        ldi r26,$40
        ldi r27,$1F
timer0:
        NOP
        SBIW X, 1
        brne timer0
POP R27
POP R26
        ret
```

10.2.3 Python Scripts

10.2.3.1 Motion_camera.py

```
# This script was made using information from the following web address:
# https://randomnerdtutorials.com/raspberry-pi-motion-detector-photo-capture/

# Python script to passively monitor a motion sensor that
# will take a picture using the Pi Camera and label the image
# with the date and time

# Located on Raspberry Pi Zero W

# Importing the required library elements
from time     import sleep
from gpiozero import MotionSensor
from picamera import PiCamera
from signal   import pause
import os
import datetime

# Defining the different resources that are used
camera = PiCamera()
sensor = MotionSensor(18)
count = 0

# Initializing the camera
camera.resolution = (1024, 768)
camera.rotation = 90
camera.start_preview()
sleep(2)

# Wait for the motion sensor to finish powering on
sleep(1)
print("ready")

# A function to take a picture and appropriately store it
# with the date and time
def take_pic():
  print("MOTION!")
  date = datetime.datetime.now()
  day  = date.strftime("%d-%m-%Y")
  time = date.strftime("%H:%M:%S")
  try:
    os.mkdir("/home/pi/archive/{}".format(day))
  except FileExistsError :
    pass
  file_location = "/home/pi/archive/{}/{}.jpg".format(day, time)
```

```
  camera.capture(file_location)

# When the sensor detects motion, it will call the function
# to take a picture
sensor.when_motion = take_pic

pause()
```

10.2.3.2 live.py

```
# Basic python script to take a picture
#
# Located on Pi Zero
from picamera import PiCamera

camera = PiCamera()

camera.capture('/home/pi/live.jpg')
```

10.2.3.3 serial_control.py

```
#
# Python script to send and receive serial communications
# to control the embedded circuit
#
# Located in the scripts section of the webserver

import serial
import sys
from    time    import sleep

port = serial.Serial("/dev/ttyS0", baudrate=2400)
argument = sys.argv[1]

port.write(argument)
port.read()
port.write("1")
port.read()
port.write("1")

# If the argument is an 'S' send 6 numbers out
if argument == 'S' :
  code = list(sys.argv[2])
  i = 0
  while i < 6 :
    port.write(code[i])
    port.read()
    i = i + 1
  print("Passcode Updated Successful!")

# If the arguement is 'W' recieve 2 Hexadecimal characters
elif argument == 'W' :
  val = ''
  i = 0
  while i < 2 :
    port.write("1")
    val += port.read()
    i += 1
  print(val)
```

10.2.4 Bash Scripts

10.2.4.1 update.sh

```
#!/bin/bash
# This is a script to stop the motion_camera.py script,
# send the pictures to the webserver, send a current
# image, clean up the sent pictures to make space and then
# restart the motion_camera.py script again

# Located on Raspberry Pi Zero W

# This script is triggered every 10 minutes by crontabs

# stop the previous motion_camera.py
sudo kill $(pgrep 'python')

# Transfer images to server
scp -r /home/pi/archive/* pi@server:/home/pi/archive_temp/
ssh pi@server sudo rsync -av /home/pi/archive_temp/* /var/www/html/archive/
ssh pi@server sudo rm -r /home/pi/archive_temp/*

# grad an image of the location and send it to the webserver
sudo python3 live.py
scp /home/pi/live.jpg pi@server:/home/pi/archive_temp/
ssh pi@server sudo cp /home/pi/archive_temp/live.jpg /var/www/html/archive/live.jpg
ssh pi@server sudo rm /home/pi/archive_temp/live.jpg

# Remove images and restart motion_camera.py
sudo rm -r /home/pi/archive/*
sudo python3 /home/pi/motion_camera.py&
```

10.2.5 Webpage files

10.2.5.1 index.php

```
<!DOCTYPE html>
<title>SPDS</title>

<head>
<link href="./resources/colours.css" rel="stylesheet" type="text/css">
<style>
.center {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

</style>
</head>

<body>
<h1>Monitoring</h1>
<p>Welcome to the monitoring screen. From here you can obtain the current weight of any
objects in the container and obtain the most current image of the container via the remote
camera. </p>
<div>
<h2>Archive</h2>
<p>Select a specific date from the drop-down menus below. <br>Once you press "Submit", your
browser will be redirected to a gallery of images from that day. </p>
<form class="sub" action="./archive.php" method="post">
  Select a date:
  <select name="day">
    <option value="01">01</option>
    <option value="02">02</option>
    <option value="03">03</option>
    <option value="04">04</option>
    <option value="05">05</option>
    <option value="06">06</option>
    <option value="07">07</option>
    <option value="08">08</option>
    <option value="09">09</option>
    <option value="10">10</option>
    <option value="11">11</option>
    <option value="12">12</option>
    <option value="13">13</option>
    <option value="14">14</option>
    <option value="15">15</option>
    <option value="16">16</option>
    <option value="17">17</option>
```

```
    <option value="18">18</option>
    <option value="19">19</option>
    <option value="20">20</option>
    <option value="21">21</option>
    <option value="22">22</option>
    <option value="23">23</option>
    <option value="24">24</option>
    <option value="25">25</option>
    <option value="26">26</option>
    <option value="27">27</option>
    <option value="28">28</option>
    <option value="29">29</option>
    <option value="30">30</option>
    <option value="31">31</option>
  </select>
  <select name="month">
    <option value="01">January</option>
    <option value="02">February</option>
    <option value="03">March</option>
    <option value="04">April</option>
    <option value="05">May</option>
    <option value="06">June</option>
    <option value="07">July</option>
    <option value="08">August</option>
    <option value="09">September</option>
    <option value="10">October</option>
    <option value="11">November</option>
    <option value="12">December</option>
  </select>
  <select name="year">
    <option value="2019">2019</option>
  </select>
  <input type="submit">
</form>
<?php
  $date = `date`;
  echo "The current date is ", $date;
?>
</div>

<div>
<h2>Weight</h2>
<p>The current weight of items inside the SPDS is:<p>
<div class="sub">
<?php
  $weight = hexdec(`sudo python /var/www/html/scripts/serial_control.py W`);
```

```
  $var = `cat /var/www/html/weight.txt`;
  $displayed = $weight - $var;
  $true_weight = $displayed * 65;
  echo $true_weight,"g";
?>
</div>
</div>

<div>
  <h2>Location Check</h2>
  <p>This image is taken whenever the archive of pictures is updated. It is not triggered by the
motion sensor.<br>To view activity caught by the motion sensor, use the "Archive" section
above.<br>To update the archive, as well as the image below will be updated every 10
minutes.</p>
  <img src="archive/live.jpg" width="1280" height="720" class="center">
<br>
</div>

<br><br>
<a href="./Advanced.php">Advanced Feature Controls</a>
</body>
```

10.2.5.2 archive.php

```
<!DOCTYPE html>

<title>Secure Parcel Deposit Station</title>
<head>
<link href="./resources/colours.css" rel="stylesheet" type="text/css">

<style>
div.gallery {
 border: 1px solid #ccc;
}

div.gallery:hover {
 border: 1px solid #777;
}

div.gallery img {
 width: 100%;
 height: auto;
}

div.desc {
 padding: 15px;
 text-align: center;
}

* {
 box-sizing: border-box;
}

.responsive {
 padding: 0 6px;
 float: left;
 width: 600px;
}

@media only screen and (max-width: 700px) {
 .responsive {
   width: 49.99999%;
   margin: 6px 0;
 }
}

@media only screen and (max-width: 500px) {
 .responsive {
  width: 100%;
```

```
 }
}
</style>

</head>

<h1>Archive</h1>
<div>
 <h2>
 <?php
  $day   = htmlspecialchars($_POST["day"]);
  $month = htmlspecialchars($_POST["month"]);
  $year  = htmlspecialchars($_POST["year"]);
  $date  = `echo $day"-"$month"-"$year`;
  echo "Activity from ", $date,":";
 ?>
 </h2>
 <body>

 <?php
  $filepath = `echo -n "./archive/"$date"/"`;
  $files = scandir($filepath);
  foreach($files as $file) {
   if($file !== "." && $file !== "..") {
    echo '<div class="responsive">';
    echo '<div class="gallery>';
    echo '<a target="_blank" href="',$filepath,'/',$file,'">';
    echo '<img src="',$filepath,'/',$file,'" alt="',$file,'" width="600" height="400" >';
    echo '</a>';
    echo '<div class="desc">',$file,'</div>';
    echo '</div>';
    echo '</div>';
   }
  }
 ?>
 </body>
</div>
```

10.2.5.3 Advanced.php

```
<!DOCTYPE html>
<title>SPDS</title>

<head>
<link href="./resources/colours.css" rel="stylesheet" type="text/css">
<style>
.reveal-if-active {
  opacity: 0;
  max-height: 0;
  overflow: hidden;
  font-size: 16px;
  -webkit-transform: scale(0.8);
          transform: scale(0.8);
  transition: 0.5s;
}
.reveal-if-active label {
  display: block;
  margin: 0 0 3px 0;
}
input[type="radio"]:checked ~ .reveal-if-active, input[type="checkbox"]:checked ~ .reveal-if-
active {
  opacity: 1;
  max-height: 100px;
  padding: 10px 20px;
  -webkit-transform: scale(1);
          transform: scale(1);
  overflow: visible;
}
</style>

</head>

<H1>Advance Monitoring and Control</H1>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="post">
<div>
  <p>Do you want to change the passcode of the SPDS?</p>

  <input type="radio" id="pass-no" name="pass-yes-no" value="no" checked required>
  <label>No</label>

  <div>
  <input type="radio" id="pass-yes" name="pass-yes-no" value="yes" required>
  <label>Yes</label>
  <div class="reveal-if-active">
    <label for="newpass">Enter the desired passscode (must be 6 numbers):</label>
```

```
   <input type="text" id="newpass" name="passcode" class="require-if-active" data-require-
pair="#pass-yes" >
  </div>
  </div>

<?php
  $code = "";
  if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $response = $_POST["pass-yes-no"];
   if ($response == "yes"){
    $code = str_replace('.','',$_POST["passcode"]);
    if( (strlen($code)==6)&&(is_numeric($code)) ){
     `sudo python /var/www/html/scripts/serial_control.py S $code`;
    }else{
     echo "Invalid Passcode";
    }
   }
  }
?>
</div>

<br><br>
<div>
<p>The weight inside the SPDS is currently:</p>
<?php
  `sudo python /var/www/html/scripts/serial_control.py W`;
  $weight = hexdec(`sudo python /var/www/html/scripts/serial_control.py W`);
  echo "The value from the weight sensor is: ",$weight;
?>
<br>
<?php
  if($_SERVER["REQUEST_METHOD"] == "POST"){
   $zero = $_POST["zero"];
   if($zero == "yes"){
    `sudo echo $weight > /var/www/html/weight.txt`;
   }
  }
  $var = `cat /var/www/html/weight.txt`;
  $displayed = $weight - $var;
  echo "The value that is used to calculate is: ",$displayed,"<br>";
  $true_weight = $displayed * 65;
  echo "The weight is:",$true_weight,"g";
?>
 <p>Do you want to zero the scale?</p>
 <input type="radio" name="zero" value="no" id="zero-no" checked required>
 <label>No</label>
```

```
 <input type="radio" name="zero" value="yes" id="zero-yes" required>
 <label>Yes</label>
</div>
<br>

<div>
 <p>Do you want to clear the archives?</p>
 <input type="radio" name="clear-archive" value="no" id="clear-archive-no" checked required>
 <label>NO</label>
 <input type="radio" name="clear-archive" value="yes" id="clear-archive-yes" required>
 <label>YES</label>
 <?php
  if($_SERVER["REQUEST_METHOD"] == "POST"){
   $clear-archive == $_POST["clear-archive"];
   if($clear-archive == "yes"){
    `sudo rm -r /var/www/html/archive/*`;
   }
  }
 ?>
</div>
<br><br>
<input type="submit">
</form>
<br>

<a href="./index.php">Return to Home Page</a>
```

10.3 Appendix C: Bill of Materials

Component	Quantity	Cost (CAD$)
Raspberry Pi 3B+	1	60.00
Raspberry Pi Zero W	1	35.00
ATmega8515	1	4.09
48VA Transformer	1	20.00
Load Cell	1	10.00
ADC0808	1	7.62
INA126	1	6.28
MM74C922	1	5.89
74LS138	1	1.30
74LS04	1	1.13
74LS32	1	0.94
LM78012	1	2.38
LM7805	1	2.38
OKI-78SR-5	1	6.24
2 x 16 LCD	1	12.00
12V, Ah Lead acid Battery	1	15.67
2N3906 transistor	3	0.90
TIP110 transistor	1	1.25
1k resistor	4	1.00
10k resistor	1	
100 resistors	1	
1k potentiometer	1	
10k potentiometer	1	
1000uF capacitor	1	1.00
0.1uF capacitor	2	
150pF capacitor	1	
13 pin headers	2	15.46 (kit)
4 pin headers	1	

14 pin headers	2	
40 pin headers	1	
16 pin headers	1	
8 pin headers	1	
40 pin DIP sockets	1	2.08
20 pin DIP sockets	2	2.00
18 pin DIP sockets	1	1.00
14 pin DIP sockets	2	2.00
8 pin DIP sockets	1	1.00
Single sided PCB	9” x 6”	6.00
Double sided PCB	9” x 6”	8.00
Mounting hardware	Various components	12.00
	Total	244.41

10.4 Appendix D: Cost Analysis

Product/Service	Hours	Cost (CAD$)
Components	N/A	234.41
Web Design	15	600.00
Testing/Research	24	960.00
Embedded Coding	10	400.00
Circuit Design	10	400.00
Circuit Production	14	560.00
System Configuration (Raspberry Pi)	3	120.00
Documentation	20	800.00
Enclosure/Mounting	2	80.00
Total	98	4,164.41