The TP Tracker

By

William Rick
Kevin Wang

Final Report for ECE 445, Senior Design, Spring 2018
TA: Kexin Hui

May 2, 2018

Team No. 7

Abstract

The TP Tracker is an Internet of Things device that allows the environmentally conscious user to monitor their toilet paper consumption. It currently has enough memory to store the statistics of up to four users for a timeframe of one month. All data is stored in non-volatile memory, so it will not be erased in the case of power outage. The device identifies users with assigned RFID cards, and automatically logs the user out when the lights are turned off. The device enters a low power mode at this time, consuming only 330uA. When a user is logged in, the device will dispense an adjustable “serving” of toilet paper when signaled by a hand wave. All data is displayed on a high resolution OLED display with a clean user interface. Graphs and other statistics can be viewed through various menu options.

Contents

1. Introduction

The design of the TP Tracker is motivated by the environmental impact of toilet paper overuse in North America. The United States uses over 50% more toilet paper per capita than Japan, the UK, and other western European countries [1]. The Environmental Protection Agency has advocated for the use of recycled toilet paper for its numerous benefits to the environment. If every person in the US swapped one roll for a recycled roll, 470,000 trees, 1.2 million feet of cubic landfill space, and 169 million gallons of water could be saved [2]. Unfortunately, most people still use paper made from cut trees due to the high cost of manufacturing recycled paper. However, if everyone simply used one less roll, the benefits could be even greater.

The TP Tracker solves this problem in two ways. First, the device makes users aware of their usage of toilet paper. The philosophy is similar to that of wearing a step counter: by giving the user a tool to monitor themselves, they become more inclined to set goals to improve. Second, the device can reinforce portion control. The design of the TP Tracker is inspired by hand wave paper towel dispensers. These dispensers set a portion size for the user, encouraging them to only use one or two portions. Installing portioned paper towel dispensers in restrooms can reduce consumption by up to 30%, so it is worthwhile to implement the same concept for toilet paper dispensers.

The TP Tracker meets three major high level requirements:

1. It dispenses a serving of toilet paper when prompted by the user, and records this to the users data log.
2. It warns the user when the roll needs to be replaced.
3. It conserves energy by entering low power mode when not in use.

The overall design of the dispenser and description of individual components are described in section 2. The verification of all blocks is shown in section 3. The bill of materials for the prototype, as well as the cost of labor to develop the product, is covered in section 4. Finally, section 5 explores ethical considerations as well as further improvements to be made to the product.

2. Design

Our design is broken into five components, shown in Figure 1. The components work together to recognize a user's ID, dispense and record toilet paper when prompted by the user, and consume as little power as possible when not actively in use. The Control Unit is the "brain" of the product, controlling many core processes: reading and interpreting data from sensors, processing and displaying the User Interface, and sending the order to dispense. The Mechanical Unit dispenses the toilet paper using a continuous rotation servo, coupled with two rollers. The User Interface displays statistics and settings screens, while also handling user input from buttons. The Power Unit keeps the system powered from a wall outlet. It also enables and disables low power mode, which will cut all but the necessary power for the microcontroller when entering sleep mode. Finally, The Light Sensing Circuit will determine whether the lights in the restroom are on or off based on an adjustable brightness threshold. This light sensing circuit signals the microcontroller to enter low power mode and wake up.

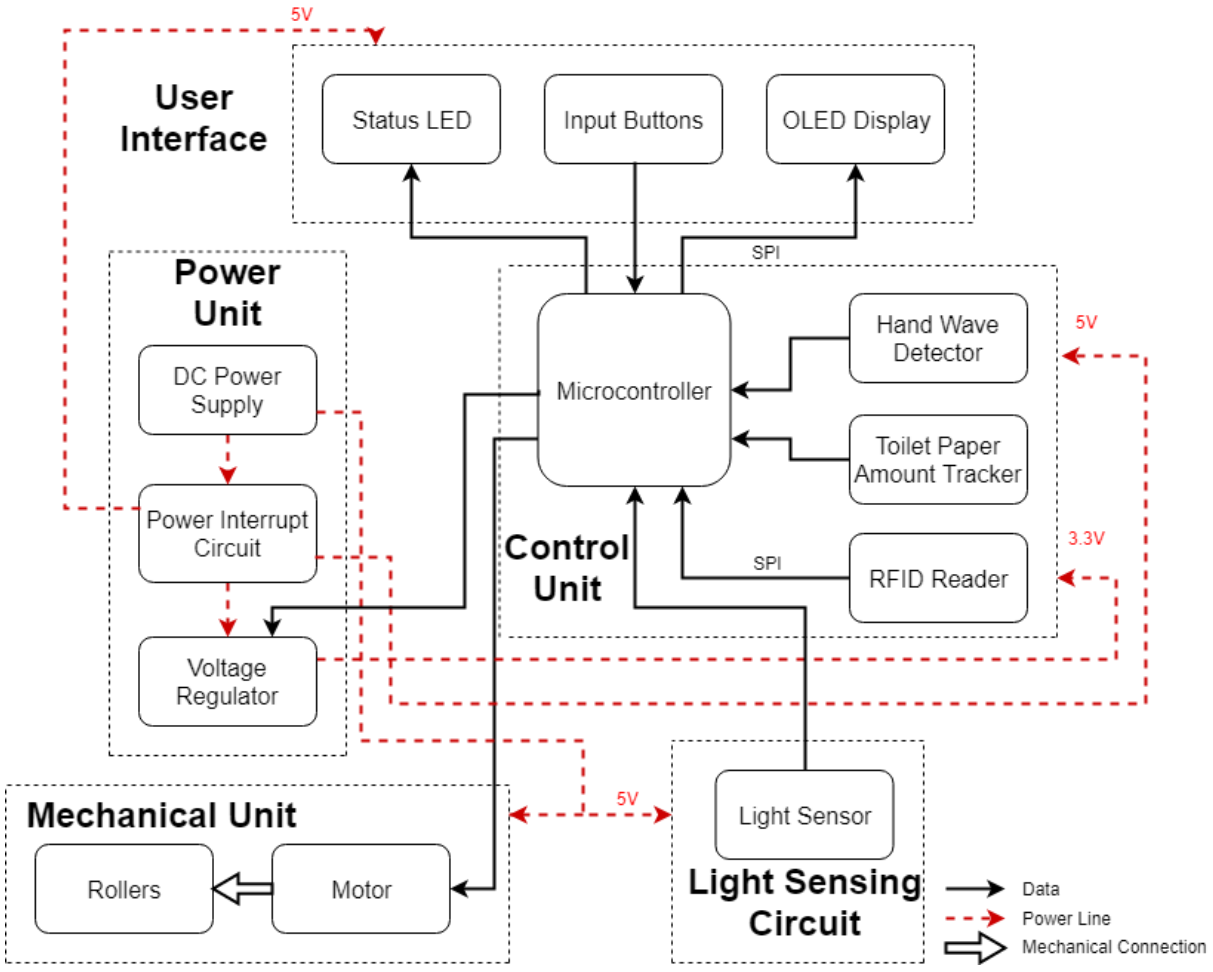

Figure 1: High Level Block Diagram

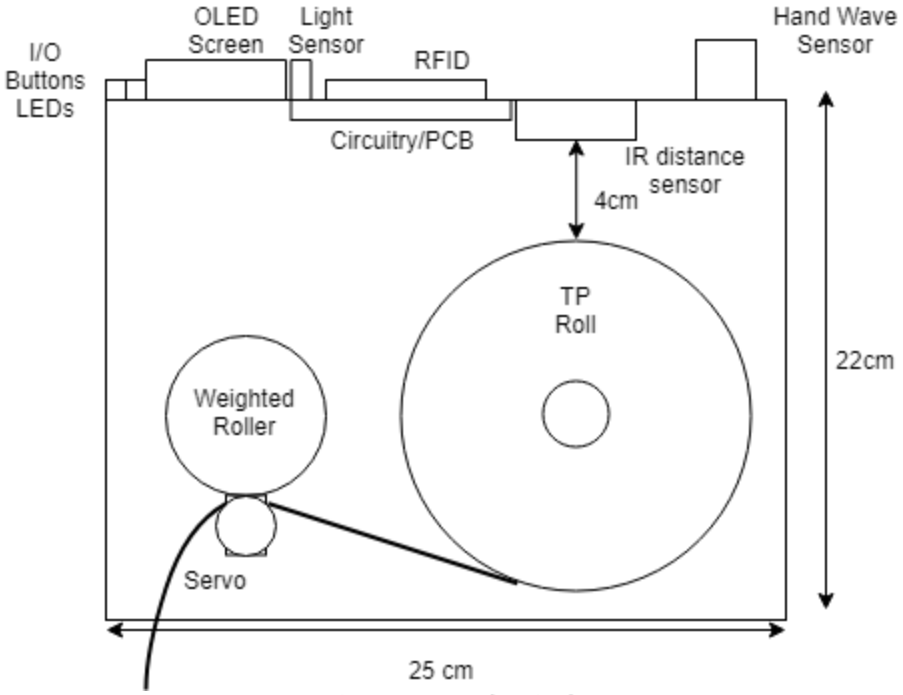

Figure 2: Mechanical Layout

The overall shape of the device is inspired from commercial paper towel dispensers and is shown in Figure 2. This generally includes a mechanical dispenser, and easily triggered wave detector. Our design has the addition of a perfboard accessible to the user containing the screen and buttons as well as an inaccessible PCB containing the control unit and sensors.

2.1 Control Unit

The Control Unit consists of the ATmega328P microcontroller, the Hand Wave Detector IR proximity sensor, the Toilet Paper Amount Tracker IR distance sensor, and the RFID reader. The microcontroller connects to the IR sensors, buttons, and LED through its digital pins. It connects to the OLED and RFID reader over SPI bus. The Light Sensing Circuit is connected to the Control Unit through digital pin as well. The Control Unit detects which user is signed in, prompts the dispensing of toilet paper, and tracks consumption. The operation of the unit is described by the flow chart in Figure 3. The circuit schematic is given in Appendix A.

2.1.1 Microcontroller

The microcontroller is the ATmega328P. This was chosen due to its large number of support libraries and ease of rapid prototyping. The microcontroller processes data from the sensors on the device, controls the dispensing motor in the Mechanical Unit, and processes the display of the User Interface. It also stores information on each user in its runtime variables and logs data to the onboard EEPROM, which stores and retains values for when the device is shut off.

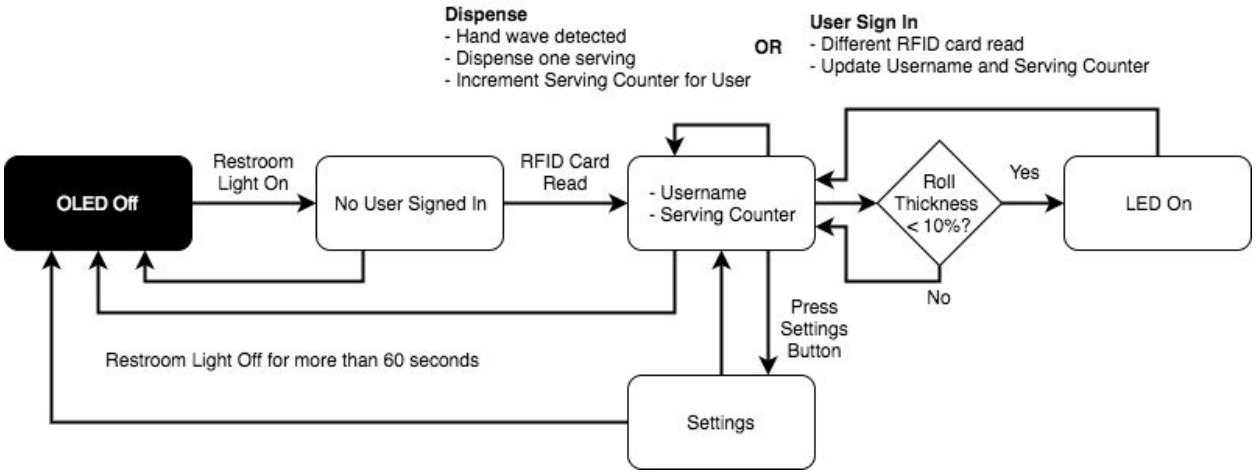

Figure 3: High Level Flow Chart

2.1.2 Hand Wave Sensor

This sensor is a Sharp GP2Y0D810Z0F Digital Distance Sensor, which has a binary output of HIGH when an object is present in the 2 to 10 centimeter range, and an output of LOW when no object is present in this range. The sensor will be mounted in a way such that it is pointing upwards towards the ceiling to prevent false detections of hands and other objects. It is assumed that the product will be mounted in a position such that there is no obstruction of view between the sensor and the ceiling of the room.

2.1.3 Toilet Paper Amount Tracker

This sensor is the Sharp GP2Y0A41SK0F Infrared Proximity Sensor. This short range distance sensor is used to track the size of the toilet paper roll to determine when the roll is running low (10% remaining). It is mounted pointing downward from the top of the unit. It is placed 4 cm away from the edge of the roll, pointing towards the center of the roll. When a new toilet paper roll is inserted into the device, the microcontroller records the radius of the fresh roll and uses this metric to calculate the percentage remaining.

The equation used to calculate the percentage of toilet paper remaining is shown as equation 1. When a new roll is installed, the *Original Radius* is updated. The distance to the center of the roll is fixed and known by mechanical design. The radius of the cardboard roll is generally fixed between 2 and 2.5cm. This variance changes the slope of the percentage calculation, but is ignored as the percentage still converges to zero on an empty roll and 100% on a full roll.

$$TP\% = \frac{(Outer\ Radius)^2 - (2.25\ cm)^2}{(Original\ Radius)^2 - (2.25\ cm)^2}$$

Equation 1: Toilet Paper Remaining Percentage

2.1.4 RFID Reader

The Mifare RC522 RFID reader is used in our design. The reader has a low cost and communicates over the SPI bus for reliable, fast data transfer to the microcontroller. The RFID reader is used to identify different users by recognizing the unique ID string of their login card.

2.2 User Interface

The User Interface displays information such as consumption of each resident, low paper warnings, and settings. It consists of an OLED screen, three buttons, and a status LED.

2.2.1 OLED Screen

The Solomon Systech SSD1306 OLED screen is used to display a text menu to users. It has a resolution of 128x64 pixels and communicates with the microcontroller over the SPI bus. The menus are displayed in the following logical order: The device first prompts the user to tap an RFID card to sign in. This brings the user to their statistics page, showing current usage, average calculations, and a histogram of this month's usage. When prompted by the user, the display shows the settings menu, allowing the user to adjust serving size and other adjustable values.

2.2.2 Input Buttons

There are three input buttons which provide Up, Down, and Enter functionality for the menu. They are debounced, as shown in the circuit in Appendix A. They are run as active high to properly interface with the libraries used with the OLED.

2.2.3 Status LED

The status LED illuminates when the toilet paper level is below the 10% mark. It LED is red because it is used as a warning color, and is easily visible from a distance.

2.3 Mechanical Unit

This unit handles the dispensing of the toilet paper. It contains rollers which are driven by a motor.

2.3.1 Motor

Our design uses a small, continuous rotation servo to drive the rollers. It has adjustable speed, and runs on 5V. At max speed it has a peak draw of approximately 500mA, but under normal conditions it only draws 100mA.

2.3.2 Mechanical Rollers

We use a two roller design to feed toilet paper. The bottom roller is driven by the motor, and the top is a passive roller. The passive roller allows the paper to grip the rubber threads on the bottom roller.

2.4 Power Unit

The design is powered from a wall outlet. The power supply must consist of an AC to DC converter and a voltage regulator to ensure the microcontroller and sensors receive the correct voltage. The AC/DC converter is a packaged 5V converter. This is used instead of batteries for convenience to the user. The Power Interrupt Circuit will receive a binary signal from the microcontroller to enter or leave low power mode. It will also cut the power to all sensors and the motor to eliminate leaked power. In the case that the microcontroller is in low power mode. Finally, there is a 3.3V regulator which provides power to the RFID reader.

In normal operation, the unit is expected to draw approximately 250 mA, but may peak at 800 mA. This is shown in Table 1. In sleep mode the unit is expected to draw 275 uA. The actual power consumption will be discussed in section 3, but it is close to these estimates.

2.4.1 AC/DC Converter

This product uses a simple 5V wall adapter. We have chosen one with a max current of 2A, to ensure plenty of overhead and low ripple.

2.4.2 Voltage Regulator

The voltage regulator chosen is a Texas Instruments TLV62568DBVR. This is a buck regulator which is adjustable from 1V to Vin, with a maximum input of 5.5V. This switching regulator was chosen for its high efficiency and low output ripple. It also has a maximum output of .5A which is much higher than the peak current of the RFID reader. The circuit is shown in Appendix A.

2.4.3 Power Interrupt

A pnp MOSFET is used to interrupt power to the sensors and motor. The particular the FQP27P06 is used because of its low switching voltage and high maximum current. An inverter is used at the gate due to the sleep mode requirements of the ATmega. The ATmega must drive pins low on sleep mode, and the pnp requires a high input to cut power. The circuit is shown in Appendix A.

	Peak Current	Nominal Current	Sleep Current
Servo	500mA	100mA	**0A**
OLED	100mA	20mA	**0A**
Prox Sensor	10.5mA	5mA	**0A**
RFID	150mA	100mA	**0A**
ATMEGA	20mA	20mA	**100uA**
Distance Sensor	22mA	12mA	**0A**
Light Sensor	-	115uA	**115uA**
3.3V Regulator	-	35uA	**0A**
Inverter	-	60uA	**60uA**
TOTAL	**822mA**	**262mA**	**275uA**

Table 1: Current Draw Estimates

2.5 Light Sensing Circuit

The light sensing circuit outputs High when the light in the room is above an adjustable threshold, and Low when the light level is below the threshold. This is achieved by comparing the voltage across a biased photoresistor and a potentiometer. Setting the potentiometer allows the light threshold to be set. As shown in the schematic in Appendix A, both the photoresistor and potentiometer are in series with a 100KΩ resistor to limit current. A 20KΩ potentiometer is used to cover the entire range of the photoresistor. The photoresistor has a resistance of about 2KΩ when light, and 18KΩ when dark. The voltages from the photoresistor and potentiometer are read by a comparator which converts this to logic high or low. The comparator chosen is a high efficiency model. Because of this, it has an open drain configuration. It cannot drive the output high, only ground it, and as such must be biased between 5V and ground.

3. Design Verification

Individual unit tests have been completed as per the requirements and verifications table in Appendix B. This section will highlight some of the more significant tests (functional tests) and their respective results.

3.1 Control Unit and User Interface

The results of this section are best shown in usage of the full working model. For the test case, we have populated User 4 with a month's worth of usage data. See figures 4 and 5 for the reference screens. Upon turning the device on, we see the *Power Up* screen prompting the user to tap an RFID card to login. It also shows the percentage of toilet paper remaining. When User 4's RFID card is tapped to the reader, the *Stats1* page is shown. This shows the Daily, Weekly, and Monthly servings for this User to day. While on Stats1, a serving will be dispensed if the user triggers the handwave sensor. This serving will update the stats. Pressing up brings up the *Stats2* page, showing average daily usage over the week, month, and previous month. *Stats2* also shows the current day of the month in the top left corner. Pressing down brings up the *Stats3* page, showing a histogram of the months usage to day. This is dynamically scaled. If enter is pressed on this screen, the histogram becomes interactive as on the Histogram screen. Up and Down now cycle through days of the month showing the usage per day. Through this screen we have verified the populated data is correct.

If Enter is pressed on the *Stats1* page, the Settings menu is brought up. These screens are highlighted in Figure 5. The settings menu is scrollable with the Up/Down buttons and all of the options are shown on

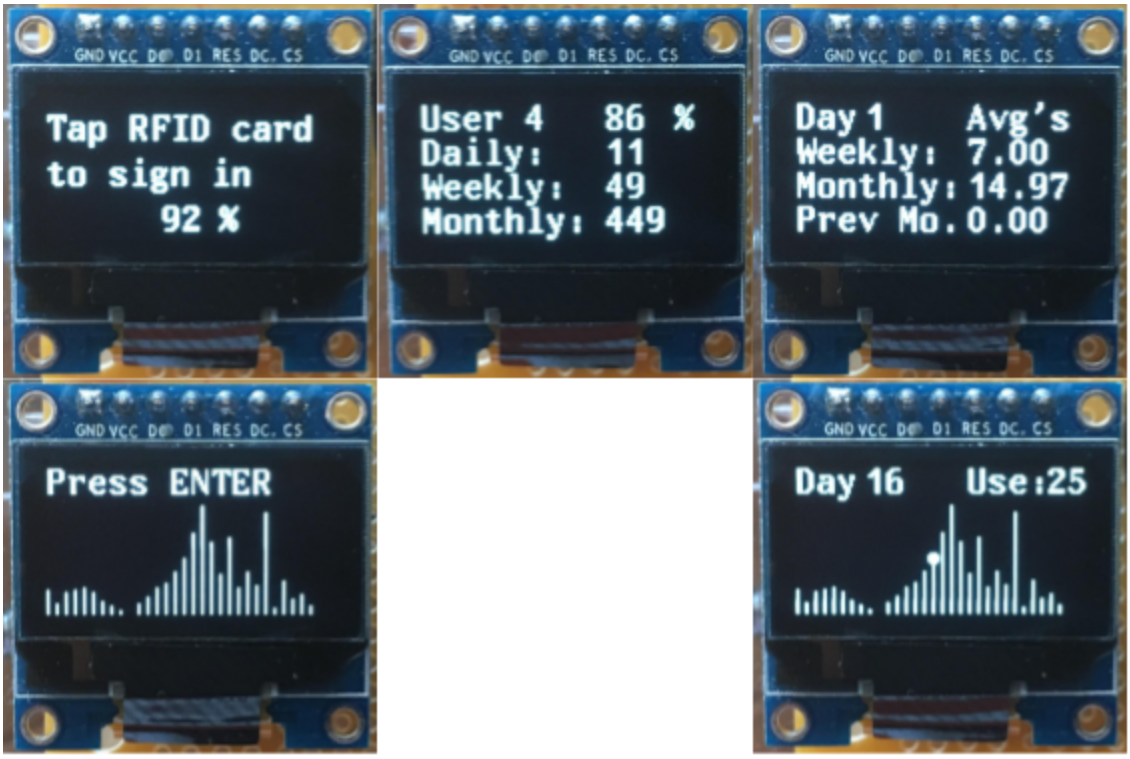

Figure 4: Power Up(top-left), Stats1(top-center), Stats2(top-right), Stats3(bottom-left), Histogram(bottom-right)

Figure 5: Settings1(top-left), Settings2(top-center), Confirmation(top-right), Change Roll(bottom-left), Serving Size(bottom-right)

Settings1 and *Settings2*. Upon selection of an option, a *Confirmation* screen is shown where the user can continue or cancel. The Change Roll option brings up the *Change Roll* screen where the thickness of the new roll is displayed. This value is then used to calculate the percentage remaining. Verification of this feature is shown in section 3.2. If Serving Size is selected, the *Serving Size* screen is shown. This allows the user to select a serving multiplier. The default is 10. For example, choosing 5 halves the original serving size. The maximum multiplier is 15 and the minimum is 1. This changes the duration of motor rotation rather than the speed, and the operation has been verified. The other menu options are Change Day and RESET DATA. Change day allows the user to manually increment the day of the month. RESET DATA does exactly what it says. It resets all user data, essentially equivalent to a factory reset.

At any point, if the lights are turned off for more than eight seconds, the unit will log the current user out, update the user's data, store it to non-volatile memory, and enter low power mode. This has been verified by unplugging the unit while in sleep mode after some dispensing of servings. Upon reboot and login, the correct data was displayed.

3.2 Toilet Paper Amount Tracker

To accurately display the percentage of toilet paper remaining, the radius of the roll must be precisely measured. To do this, we first measured the voltage output of the distance sensor. These measurements are shown in Figure 6. The output of the sensor is nearly linear with inverse distance. Using this fit line, we could calculate percentage of TP remaining from Equation 1.

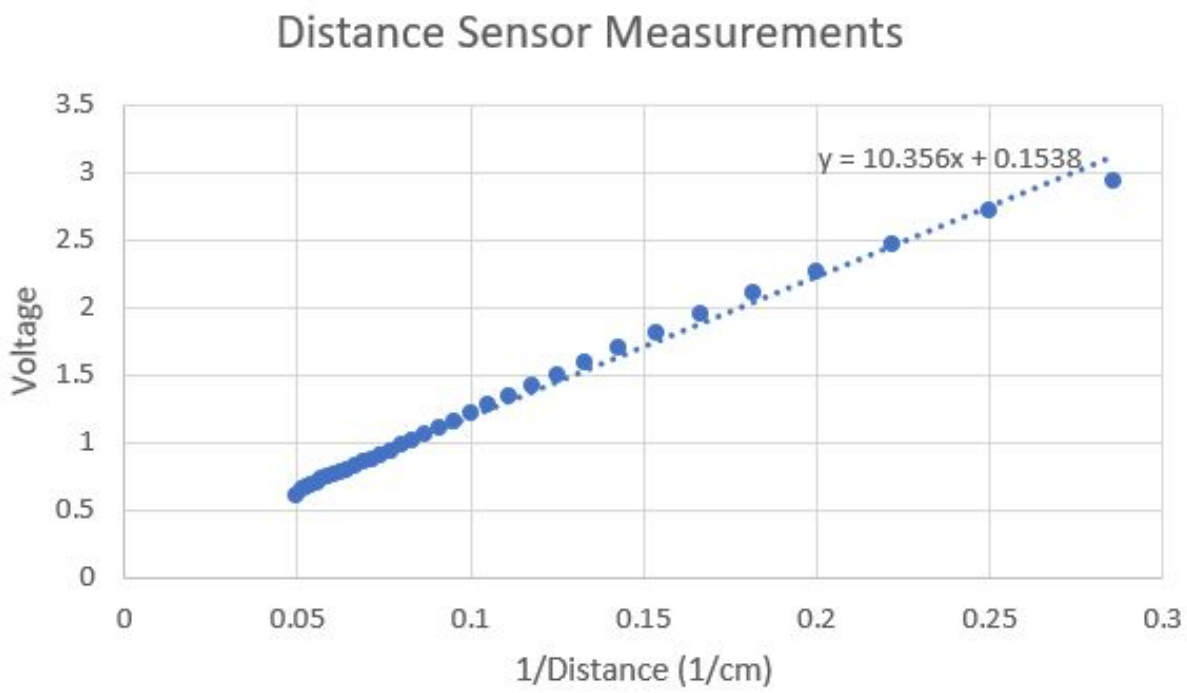

Figure 6: Distance Sensor Measurements

The percentage measurement is noisy, varying +/- 5%. With more time we would add a filter to smooth this output. Other solutions might be a 5V regulator on the ATmega input, or better insulation of the sensor wires.

3.3 Power and Sleep Mode

To test the current draw of the circuit, we placed an ammeter between the power supply and the pcb. During normal operation with the servo running, the circuit draws 220mA on average. When the servo is not running the circuit only draws about 160mA. This is lower than anticipated. The servo seems to be drawing less than the nominal value, but it still has enough torque to dispense paper so it is not a problem.

When sleep mode is activated, the current draw drops to 330uA. This is slightly higher than the 275uA expected value. This is caused by the microcontroller waking up every 4 seconds to check if the lights are on. When this functionality is removed, the current draw drops to 265uA.

4. Cost Analysis

4.1 Labor

We assume a labor cost of $35 hourly. It is estimated that we have worked approximately 10 hours each per week, for the project timeframe of 10 weeks. This incurs a labor cost of $17,500 shown in Equation 2. This is just the labor for production of the prototype.

$$2 * \frac{\$35}{hr} * 2.5 * \frac{10hrs}{week} * 10\ weeks\ =\ \$\,17{,}500$$

Equation 2: Labor Cost Calculation

4.2 Components

This is just a cost calculation of components used in the prototype. Bulk ordering of components would significantly reduce the costs, possibly up to 30% or more.

Component	**Manufacturer**	**Serial**	**Price($)**	**Quantity**
IR Prox Sensor	Sharp	GP2Y0D810Z0F	$5.95	1
IR Distance Sensor	Sharp	GP2Y0A41SK0F	$14.00	1
Microcontroller	Atmel	Atmega328p	$3.00	1
OLED		SSD1306	$9.99	1
Servo	Adafruit	LLC 169	$5.95	1
RFID reader	Mifare	RC522	$6.00	1
RFID card	QIAOYUAN		$0.80	5
Photoresistor	API	PDV-P8001	$0.95	1
Potentiometer	Bourns	3362P-1-104LF	$1.02	2
Comparator	Microchip	MCP6548-I/P	$0.58	1
Inverter	Texas Instruments	SN74AH	$0.52	1
Pushbutton			$0.50	4
Voltage Regulator	Texas Instruments	TLV62568DBVR	$1.10	1
MOSFET interrupt	Fairchild	FQP27P06	$0.95	2
Assorted RLC			$5	1
Enclosure Wood			$5	1
TOTAL			**$67.98**	

Table 2: Prototype Component Costs

5. Conclusion and Ethics

5.1 Accomplishments

The TP Tracker has reached a functional milestone. It contains functionality for dispensing toilet paper in adjustable servings, provides useful data on a visually appealing user interface, and tracks toilet paper usage on a daily, weekly, and monthly basis.

5.2 Uncertainties

There are a few points where performance is not yet known. We cannot make estimates as to how waterproof the components are, or how they will perform under the stresses of a humid bathroom. There is also the issue of noise on the toilet paper remaining measurement. The source of this has not yet been located, but ideas have been discussed in the results section which may solve it. The noise may be coming from external sources, or from ripple in the pcb supply current. These are only two minor problems which don't interfere excessively with the operation of the prototype.

5.3 Future Work

Possible additions to this project could include the ability to transmit data over wifi to a database for better storage and analytics. Currently, all usage data is stored on the machine. This becomes cumbersome to manage in a commercial setting where multiple devices are installed. Two serial pins on the microcontroller have been left open for the ability to add a wifi module in the future. In addition, RFID can be deprecated for the use of mobile phone login using NFC. By shifting data storage to a cloud solution, users could also access their data more easily, such as through a website or mobile application. In addition, all intensive data analytics can be computed on a server, rather than the embedded device. This would decrease the development cost of the unit as a whole.

5.4 Ethical Considerations

There is the issue of potential harm to others through disclosure of information [5] [6]. This is touched upon in IEEE Codes 2 and 9, which state that an invention should avoid conflict and avoid injuring users. We believe this includes not only physical safety, but the security of information and personal privacy of our users. Personal usage data is only displayed to the current logged in user. However, we cannot control external peer pressure from roommates to share the said information; we can only provide the option of restricting visibility. This product is designed such that everyone can become more aware of their own usage, not for purposes of spying or regulation. Participation should be voluntary. Another ethical issue brought up is that female users may generally have higher toilet paper usage than males. We do not want to violate IEEE Code 8 and discriminate against any gender. This product is designed for use as a personal tracker. If one would like to share their usage voluntarily, this is an option, but it is designed primarily for personal tracking and improvement. Therefore, there should be no inherent gender-based discrimination. It is also known RFID is not the most secure sign in method. Other login methods may be explored if this is ever marketed in public areas such as hotels and airports.

References

[1] New Scientist, “Toilet Paper Use is a Growing Problem,” New Scientist, p. 5-5, April 24, 2010. [Online]

[2] Noelle Robbins, “Not a Square to Spare,” Earth Island Journal, vol. 25, no. 3, p. 57, 2010. [Accessed: 3-Feb-2018]

[3] Javier C. Hernandez, “China’s High Tech Tool to Fight Toilet Paper Bandits,” The New York Times, March 20, 2017. [Online] Available: https://www.nytimes.com/2017/03/20/world/asia/china-toilet-paper-theft.html. [Accessed: 5-Feb-2018]

[4] Adafruit, “Measuring Light,” 2018. [Online] Available: https://learn.adafruit.com/photocells/measuring-light. [Accessed 7-Feb-2018]

[5] ACM.org, “ACM Code of Ethics and Professional Conduct,” 2018. [Online] Available: https://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct. [Accessed: 5-Feb-2018]

[6] IEEE.org, "IEEE IEEE Code of Ethics", 2018. [Online]. Available: https://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 6-Feb-2018]

Appendix A: Schematics

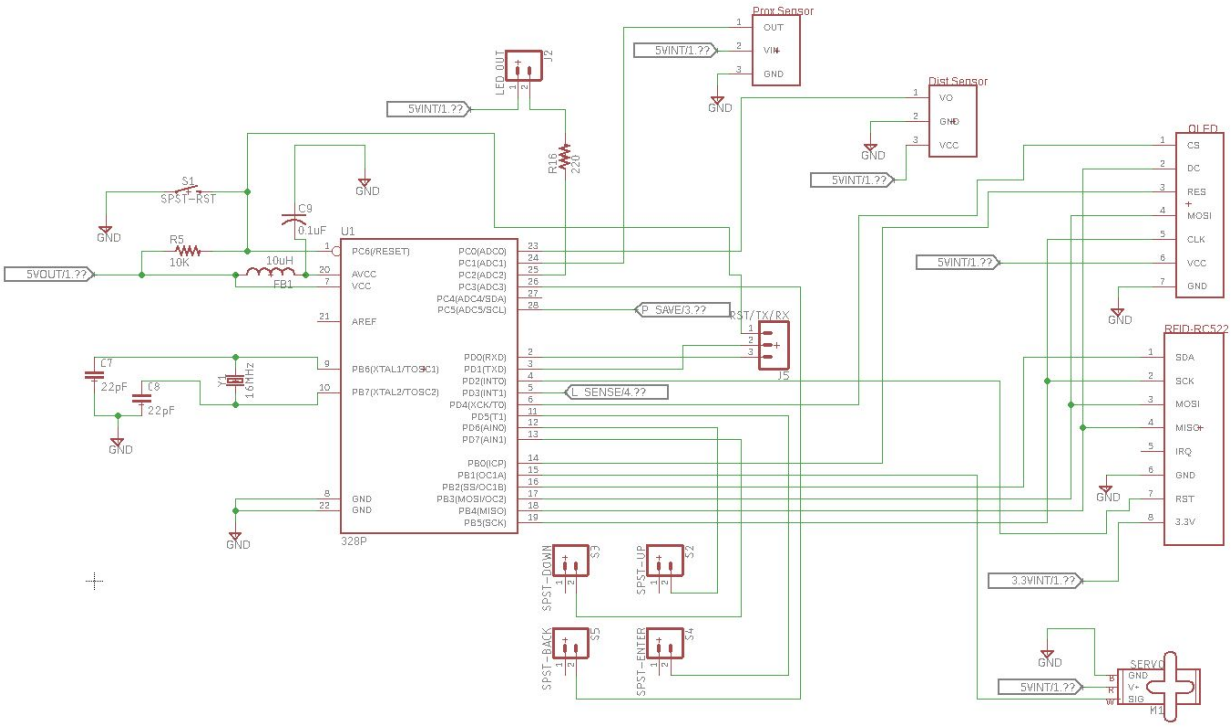

Figure 7: Control Unit Schematic

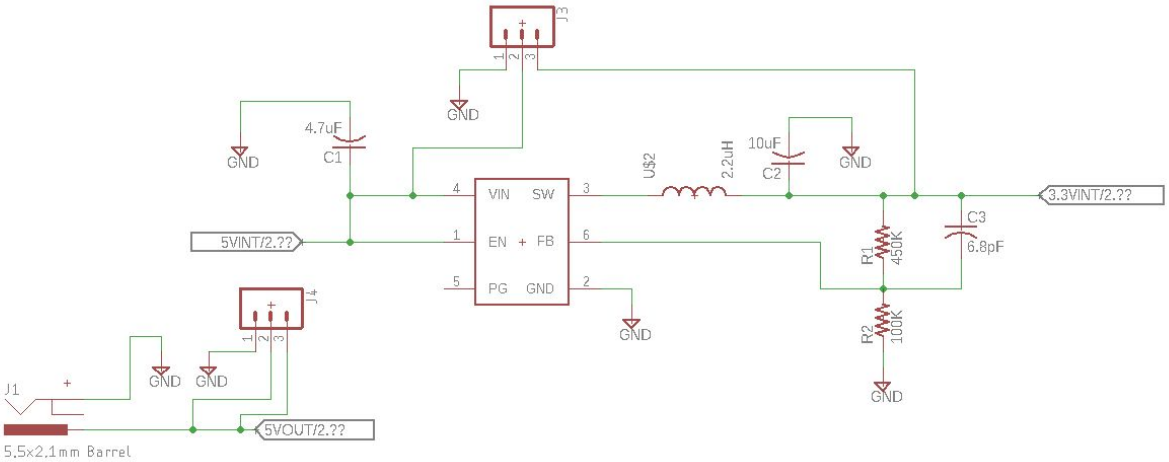

Figure 8: 3.3V Regulator Schematic

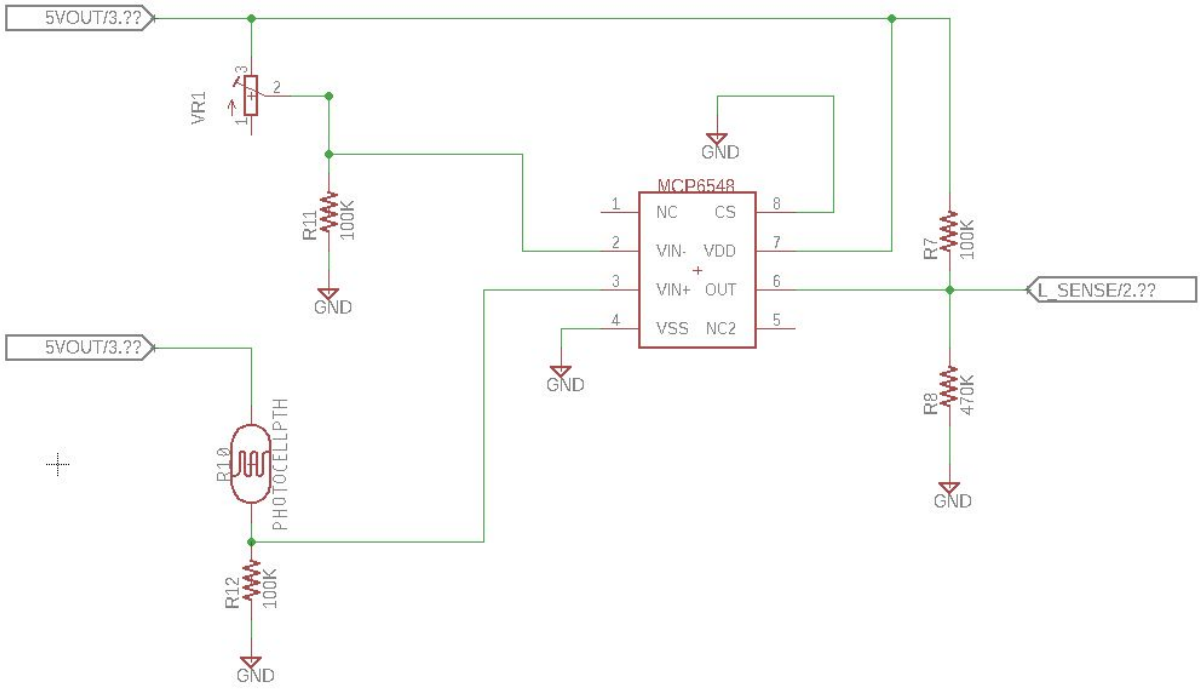

Figure 9: Light Sensor Schematic

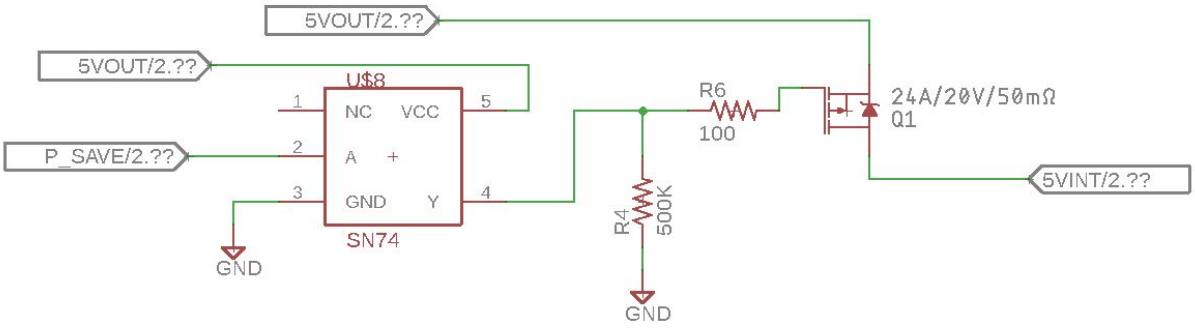

Figure 10: MOSFET Interrupt Schematic

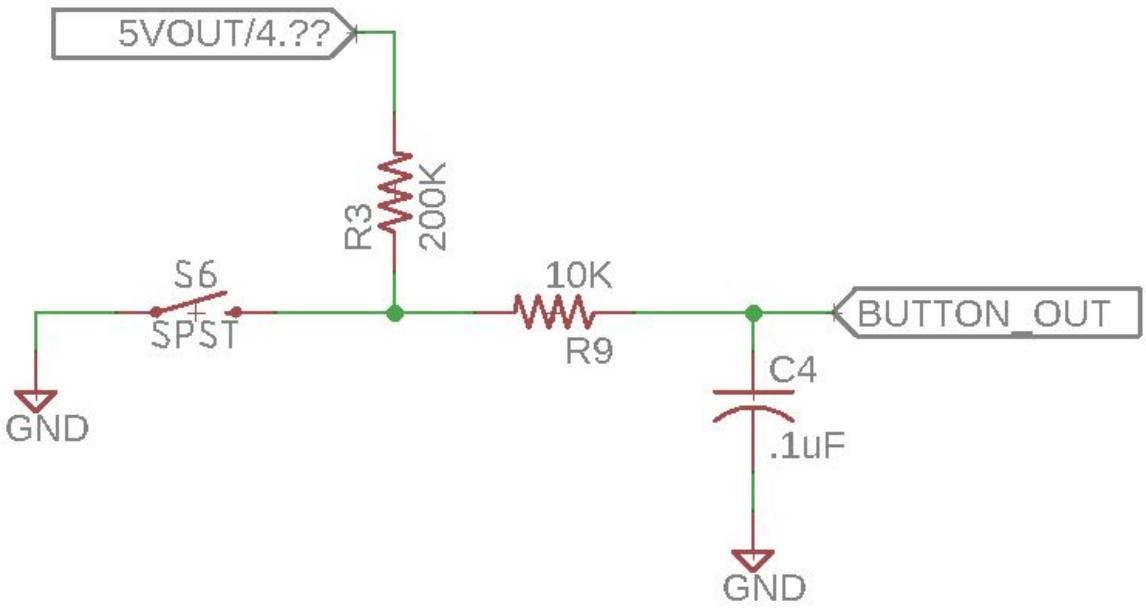

Figure 11: Input Button Active High Debounce Schematic

Appendix B: Requirements and Verifications Table

Control Unit		
Microcontroller		
Requirements	**Verification**	**Verified?(Y/N)**
1. Can store usage data for 4 average users for up to one month in EEPROM (based on calculations described in prior).	1. A. Create a program to populate EEPROM with a simulated month's worth of data for four users (based on the calculations described prior). B. Check if the program succeeds with populating the memory by reading the data at each address written to the EEPROM in step A.	Y Y
Hand Wave Sensor		
Requirements	**Verification**	
1. Outputs HIGH when a hand sized (flat, square 7 cm) object is directly above the sensor at a distance of 2-10 cm, with 98 percent accuracy. 2. Outputs LOW when hand sized object is at a distance outside the distance range of 2-10 cm, with 98 percent accuracy. *May output HIGH or LOW when hand sized object is between 10 and 15 cm. **There is a 1 cm tolerance on all measurements in this section.	1. A. Connect sensor to microcontroller and connect microcontroller to computer through USB. Open a serial connection to computer. Create a software program to display the output value (HIGH/LOW) of the sensor every .25 seconds. B. Record 100 samples each for hand sized objects at distances of 3, 5, 7, and 9 cm. Ensure false negative rate is less than 2 percent. 2. A. Repeat step A in part 1 above. B. Record 100 samples each for hand sized objects at distances of 50, 25, and 16 cm. Ensure false positive rate is less than 2 percent.	Y Y Y Y
Toilet Paper Amount Tracker		
Requirements	**Verification**	

1. At a distance of 10 cm from the center of the roll, the tracker will determine percentage of roll left relative to measured radius when new roll is loaded. 2. User will prompt microcontroller to record new initial radius of roll when replacing roll.	1. A. Connect sensor to microcontroller and connect microcontroller to computer through USB. Open a serial connection to computer. Create a software program to display the output value (distance) of the sensor every .25 seconds. B. Measure toilet paper roll radius with ruler. Install roll and record output distance shown on serial monitor. Verify the output distance with ruler. C. Repeat for rolls with approximately 10%, 25%, 50%, 75%, and 100% left. 2. A. Use menu to record radius of new roll. B. Verify roll reads completely full.	Y Y Y Y Y
RFID Reader		
Requirements	**Verification**	
1. Must identify the 16B unique ID of multiple unique 13.65MHz RFID cards within a range of 1 to 4 cm, one at a time, each within 5 seconds.	1. A. Connect sensor to microcontroller and connect microcontroller to computer through USB. Open a serial connection to computer. Create a software program to print the 16B ID value read by the RFID reader to the serial monitor. Hold an individual RFID card parallel to the reader within a distance of 1 to 4 cm. Confirm that a value is displayed on the serial monitor within 5 seconds and record the value displayed on the serial monitor. B. Repeat step A for another RFID card with a different 16B ID value. If the values differ, the reader is functioning properly.	Y Y
User Interface		
OLED Screen		
Requirements	**Verification**	

1. Refresh screen at a rate of at least 2 frames per second. This ensures a fluid user interface experience.	1. A. Connect the screen to the microcontroller and connect the microcontroller to a computer via USB connection. Create a software program to move a single 1 x 1 dot in a straight horizontal line from the top left of the screen to the top right of the screen. The screen has a horizontal width of 128 pixels and the program will move the dot one pixel to the right every 0.5 second (500 milliseconds). Therefore the pixel should move two pixels to the right each second, and thus will require (128 pixels) / (2 pixels per second) = 64 seconds to move from the left side of the screen to the right side of the screen. B. At the start of the software program, start a software timer in the code. When the pixel reaches the last (rightmost) pixel in the horizontal line, stop the timer. Print the finished timer value to the serial monitor and verify that it is within 64 +/- 1 seconds.	Y Y
Input Buttons		
Requirements	**Verification**	
1. Button is pressable and debounced.	1. Connect sensor to microcontroller and connect microcontroller to computer through USB. Open a serial connection to computer. Create a software program display the output (HIGH/LOW) of the button to a serial monitor at a 9600 baud rate (9600 bits per second is more than enough speed for a 10 millisecond segment detection of a 1 bit 'button press' message). Ensure when the button is pressed there is a smooth transition with no bounce. Check if the serial monitor displays the button press.	Y
Status LED		
Requirements	**Verification**	

1. Must be visible from 1 meter away with a drive current of 15 +/- 5 mA.	1. A. Measure current through LED with multimeter. Ensure current is within correct range. B. Ensure easy visibility of illuminated LED from 1 meter.	Y Y
Mechanical Unit		
Motor		
Requirements	Verification	
1. Servo will be adjustable from 0 to 1 rotations per second.	1. Attach a marked wheel to the motor. Connect the motor to the microcontroller. Create a software program to rotate the motor at a predetermined speed, and be able to adjust the speed up or down from hardcoded values in the program. Verify rotation speed with stopwatch by tracking how long it takes for the marking to complete one rotation.	Y
Roller		
Requirements	**Verification**	
1. Roller must be able to grip the toilet paper when paired with the motor on the opposite side of the paper.	1. A. Insert roll of toilet paper on to roller. Clamp the beginning sheet of toilet between the motor and the passive roller. B. Attempt to dispense 5 servings. If the paper dispenses consistently, the roller is working.	Y Y
Power Unit		
AC/DC Converter		
Requirements	**Verification**	
1. The converter must output a voltage of 5+/- 1 volts DC.	1. Connect the AC/DC converter to a standard United States household wall outlet (120V 60Hz).	Y

	Connect a voltmeter to the plug of the AC/DC converter and measure the output voltage.	
Voltage Regulator		
Requirements	**Verification**	
1. The regulator circuits must output 3.3 +/- 0.3 V, respectively.	1. Measure voltage output with multimeter when connected to final circuit during both low power and normal modes. Ensure voltage is within specified ranges.	Y
Power Interrupt Circuit		
Requirement	Verification	
1. Less than 100 uA will flow to all connected sensors when HIGH signal is given from microcontroller. Conversely, power will be virtually unrestricted when output is LOW. 2. Less than 20 mA will flow in total while in sleep mode.	1. A. Set input to interrupt circuit HIGH from microcontroller. Read current flow to sensors with a multimeter and ensure it is limited. B. Set input to interrupt circuit LOW from microcontroller. Ensure all sensors run as expected. 2. A. Connect an ammeter between the power supply and circuit. B. Ensure the unit is in sleep mode. C. Measure Current.	Y Y Y Y Y
Light Sensing Circuit		
Requirements	Verification	
1. Photoresistor circuit can distinguish between an average lit room and average dark room. 2. Light threshold is adjustable such that different brightness thresholds can be set.	1. Connect circuit output to a multimeter. Turn lights on and off, ensure HIGH and LOW values are output, respectively. Attempt this test in a variety of brightnesses. Tune if necessary. 2. Adjust tuning potentiometer. Ensure both HIGH and LOW can be output from the circuit in dark and light environments of varying brightness/lux.	Y Y

*What constitutes as a “lit” and “dark” room are up to the interpretation of the user. Lux is used to determine the resistance of different light levels. [4] The user is able to set whatever threshold they want, and whenever the lux is above or below that threshold, the circuit will output a binary (HIGH/LOW) value.		

Table 3: Requirements and Verifications for All Blocks

Appendix C: Code

```
/* final_demo
 * - Full Working Prototype
 * - 4-22-2018
 */
#include <Arduino.h>
#include <avr/sleep.h>
#include <EEPROM.h>
#include <MFRC522.h>
#include <Servo.h>
#include <SPI.h>
#include <U8g2lib.h>
#include <Wire.h>

//////////////////
//PIN ASSIGNMENTS//
//////////////////
#define RFID_RST_PIN 2
#define LIGHT_SENSOR_PIN 3
#define OLED_SS_PIN 4
#define ENTER_PIN 5
#define UP_PIN 6
#define DOWN_PIN 7
#define OLED_RST_PIN 8
#define SERVO_PIN 9
#define RFID_SS_PIN 10
#define DIST_SENSOR_PIN A0
#define PROX_SENSOR_PIN A1
#define LED_PIN A2
#define OLED_DC_PIN A4
#define MOSFET_PIN A5

/* SCREEN STATES */
#define WELCOME        0
#define STATS          1
#define SETTINGS       2
#define CHANGE_ROLL    3
#define CHANGE_SERVING 4
#define LOGOUT         5
#define CHANGE_DAY     6
#define STATS2         7
#define STATS3         8
#define GRAPH          9

//////////////////
//USER RFID KEYS//
//////////////////
#define USER1 3152062345
#define USER2 1923064761
#define USER3 2130017437
#define USER4 4083157929

/* RFID  Setup */
MFRC522 rfid(RFID_SS_PIN, RFID_RST_PIN); // Instance of
the class
MFRC522::MIFARE_Key key;
// Init array that will store new NUID
uint32_t cardID = 0;
byte nuidPICC[4];
boolean first_time = true;
boolean was_asleep = false;

/* OLED Setup */
//
U8G2_SSD1306_128X64_NONAME_F_4W_HW_SPI(rotatio
n, cs, dc [, reset])
U8G2_SSD1306_128X64_NONAME_2_4W_HW_SPI
u8g2(U8G2_R0, 4, A4, 8);

/* Servo */
Servo myservo;
unsigned long lights_off_time;

volatile long sleep_counter = 0;
uint8_t day = 0;
uint8_t days_added = 0;

void setup() {
  // Define OUTPUTS
  pinMode(RFID_RST_PIN, OUTPUT);
  pinMode(OLED_SS_PIN, OUTPUT);
  pinMode(SERVO_PIN, OUTPUT);
  pinMode(RFID_SS_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(OLED_DC_PIN, OUTPUT);
  pinMode(MOSFET_PIN, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(A3, OUTPUT);
  // Write Unused Pins LOW
  digitalWrite(0, LOW);
  digitalWrite(1, LOW);
  digitalWrite(A3, LOW);
  // Enable Power to MOSFET
  digitalWrite(MOSFET_PIN, HIGH);
  digitalWrite(LED_PIN, HIGH);
```

```
  // Define Inputs
  pinMode(LIGHT_SENSOR_PIN, INPUT);
  pinMode(ENTER_PIN, INPUT);
  pinMode(UP_PIN, INPUT);
  pinMode(DOWN_PIN, INPUT);
  pinMode(DIST_SENSOR_PIN, INPUT);
  pinMode(PROX_SENSOR_PIN, INPUT);

  // OLED Initialization
  u8g2.begin(ENTER_PIN, DOWN_PIN, UP_PIN,
U8X8_PIN_NONE, U8X8_PIN_NONE, U8X8_PIN_NONE);
  u8g2.setFont(u8g2_font_t0_17b_tr);

  // RFID MFRC522 Initialization
  rfid.PCD_Init();

  // Servo Initialization
  myservo.attach(SERVO_PIN);
  myservo.write(90);

  Serial.begin(9600);
}

void loop() {
  what_day();

  if (digitalRead(LIGHT_SENSOR_PIN) == HIGH) {
    first_time = true;
  }

  // Look for new cards and verify if card has been read
  if (rfid.PICC_IsNewCardPresent() &&
rfid.PICC_ReadCardSerial()){
    if (rfid.uid.uidByte[0] != nuidPICC[0] ||
        rfid.uid.uidByte[1] != nuidPICC[1] ||
        rfid.uid.uidByte[2] != nuidPICC[2] ||
        rfid.uid.uidByte[3] != nuidPICC[3] ) {
          cardID = saveHex(rfid.uid.uidByte, rfid.uid.size);
          Serial.println(cardID);
    }

    else {
      Serial.println(F("Card read previously."));
    }

    rfid.PICC_HaltA();
  }

  // OLED Write to Screen
  u8g2.firstPage();
  do {
    draw();
  } while(u8g2.nextPage());

  if (digitalRead(LIGHT_SENSOR_PIN) == LOW) {
    if (first_time) {
      lights_off_time = millis();
      first_time = false;
    }

    if (millis() - lights_off_time >= 2000) {
      if (cardID != 0)
        logout();

      while (digitalRead(LIGHT_SENSOR_PIN) == LOW) {
        was_asleep = true;
        SPI.end();
        digitalWrite(MOSFET_PIN, LOW);
        digitalWrite(RFID_RST_PIN, LOW);
        digitalWrite(OLED_RST_PIN, LOW);
        digitalWrite(RFID_SS_PIN, LOW);
        digitalWrite(OLED_SS_PIN, LOW);
        digitalWrite(OLED_DC_PIN, LOW);
        digitalWrite(11, LOW);
        digitalWrite(12, LOW);
        digitalWrite(13, LOW);
        sleep_now();
      }
    }
  }

  if (was_asleep) {
    was_asleep = false;
    first_time = true;
    digitalWrite(MOSFET_PIN, HIGH);
    digitalWrite(LED_PIN, HIGH);
    SPI.begin();
    u8g2.begin(ENTER_PIN, UP_PIN, DOWN_PIN, UP_PIN,
DOWN_PIN, U8X8_PIN_NONE);
    rfid.PCD_Init();
  }
}

/*********************
 * DISPLAY VARIABLES *
 *********************/
// User Object
struct User {
  uint8_t daily_servings[31];
  int previous_month;
```

```
};
User CurrentUser;

// Shared Variables
uint8_t button = 0;
uint8_t response = 0;
uint8_t draw_state = 0;

// Settings Variables
const char *settings_list =
  "Logout\n"
  "Change Roll\n"
  "Serving Size\n"
  "Change Day\n"
  "RESET DATA\n"
  "Exit Settings";

// Stats Variables
String username;
uint8_t weekly;
int monthly;

// Roll Thickness Variables (change_roll())
const float dist_slope = .0004715;
const float dist_offset = 0.01485;
const float inner_radius = 2.25;
const float dist_to_holder = 9.5;
float full_roll_radius = 4.25; //Radius of New Roll
float curr_roll_radius = 4.25; //Radius of Roll Currently
float percentage = 100;
float previous_percentage = 100;



/*********************
 * DISPLAY FUNCTIONS *
 *********************/
/*
 * Draw Function
 * - Called in each loop to determine which page to display
 */
void draw(void) {
  switch(draw_state) {
    case WELCOME: welcome_page(); break;
    case STATS: stats_page(); break;
    case SETTINGS: settings_page(); break;
    case CHANGE_ROLL: change_roll(); break;
    case CHANGE_SERVING: change_serving(); break;
    case STATS2: stats_page2(); break;
    case STATS3: stats_page3(); break;
    case GRAPH: graph_page(); break;
  }
}

/*
 * Welcome Page
 * - Displays welcome text
 * - Only page where user variables are set, when RFID
detected
 */
void welcome_page() {
  u8g2.setCursor(0, 20);
  u8g2.print("Tap RFID card");
  u8g2.setCursor(0, 40);
  u8g2.print("to sign in");
  u8g2.setCursor(50, 60);
  u8g2.print(percent_left());
  u8g2.setCursor(75, 60);
  u8g2.print("%");

  if (cardID) {
    read_eeprom();
    draw_state = STATS;
  }
}

/* User Statistics Page
 * - Displays Username and Daily/Weekly/Monthly
statistics
 * - Only page where dispense() is allowed
 * - Press SELECT to go to Settings Page
 */
void stats_page() {
  u8g2.setCursor(0, 15);
  u8g2.print(username);
  u8g2.setCursor(80, 15);
  u8g2.print(percent_left());
  u8g2.setCursor(110, 15);
  u8g2.print("%");

  u8g2.setCursor(0, 30);
  u8g2.print("Daily: ");
  u8g2.setCursor(80, 30);
  u8g2.print(CurrentUser.daily_servings[day]);

  u8g2.setCursor(0, 45);
  u8g2.print("Weekly: ");
  u8g2.setCursor(80, 45);
  u8g2.print(weekly);
```

```
  u8g2.setCursor(0, 60);
  u8g2.print("Monthly: ");
  u8g2.setCursor(80, 60);
  u8g2.print(monthly);

  // Check if handwave detected
  dispense();

  switch(u8g2.getMenuEvent()) {
    case U8X8_MSG_GPIO_MENU_SELECT: draw_state =
SETTINGS; break;
    case U8X8_MSG_GPIO_MENU_NEXT: draw_state =
STATS2; break;
    case U8X8_MSG_GPIO_MENU_PREV: draw_state =
STATS3; break;
    case U8X8_MSG_GPIO_MENU_UP: draw_state =
STATS2; break;
    case U8X8_MSG_GPIO_MENU_DOWN: draw_state =
STATS3; break;
    default: break;
  }
}



void stats_page2() {
  // u8g2.drawVLine(x, y, height)
  // top left screen = 0,0
  int maximum = 1;
  for(int i = 0; i<30; i++){
    if(CurrentUser.daily_servings[i] > maximum){
      maximum = CurrentUser.daily_servings[i];
    }
  }

  for (int i = 0; i < 31; i++) {
    u8g2.drawVLine(i*4, 64 -
(48*CurrentUser.daily_servings[i])/maximum, 64);
  }
  u8g2.setCursor(0, 11);
  u8g2.print("Press ENTER");

  switch(u8g2.getMenuEvent()) {
    case U8X8_MSG_GPIO_MENU_SELECT: draw_state =
GRAPH; break;
    case U8X8_MSG_GPIO_MENU_NEXT: draw_state =
STATS3; break;
    case U8X8_MSG_GPIO_MENU_PREV: draw_state =
STATS; break;
    case U8X8_MSG_GPIO_MENU_UP: draw_state =
STATS3; break;
    case U8X8_MSG_GPIO_MENU_DOWN: draw_state =
STATS; break;
    default: break;
  }
}

int delta_x = 0;
void graph_page() {
  int maximum = 1;
  for(int i = 0; i<30; i++){
    if(CurrentUser.daily_servings[i] > maximum){
      maximum = CurrentUser.daily_servings[i];
    }
  }

  for (int i = 0; i < 30; i++) {
    u8g2.drawVLine(i*4, 64 -
(48*CurrentUser.daily_servings[i])/maximum, 64);
  }

  switch(u8g2.getMenuEvent()) {
    case U8X8_MSG_GPIO_MENU_SELECT: draw_state =
STATS2; delta_x = 0; break;
    case U8X8_MSG_GPIO_MENU_NEXT:
      if(day+delta_x < 29)
        delta_x++;
      break;
    case U8X8_MSG_GPIO_MENU_PREV:
      if(day+delta_x > 0)
        delta_x--;
      break;
    case U8X8_MSG_GPIO_MENU_UP:
      if(day+delta_x < 29)
        delta_x++;
      break;
    case U8X8_MSG_GPIO_MENU_DOWN:
      if(day+delta_x > 0)
        delta_x--;
      break;
    default: break;
  }

  u8g2.drawDisc((day+delta_x)*4, 64 -
(48*CurrentUser.daily_servings[day+delta_x])/maximum,
2);

  u8g2.setCursor(0, 11);
  u8g2.print("Day");
```

```
  u8g2.setCursor(30, 11);
  u8g2.print(day+1+delta_x);

  u8g2.setCursor(75, 11);
  u8g2.print("Use:");
  u8g2.setCursor(110, 11);
  u8g2.print(CurrentUser.daily_servings[day+delta_x]);
}

void stats_page3() {
  u8g2.setCursor(0, 15);
  u8g2.print("Day");
  u8g2.setCursor(30, 15);
  u8g2.print(day+1);
  u8g2.setCursor(75, 15);
  u8g2.print("Avg's");

  u8g2.setCursor(0, 30);
  u8g2.print("Weekly: ");
  u8g2.setCursor(75, 30);
  u8g2.print(weekly/7.0);

  u8g2.setCursor(0, 45);
  u8g2.print("Monthly: ");
  u8g2.setCursor(75, 45);
  u8g2.print(monthly/30.0);

  u8g2.setCursor(0, 60);
  u8g2.print("Prev Mo.: ");
  u8g2.setCursor(75, 60);
  u8g2.print(CurrentUser.previous_month/30.0);

  switch(u8g2.getMenuEvent()) {
    case U8X8_MSG_GPIO_MENU_SELECT: draw_state =
SETTINGS; break;
    case U8X8_MSG_GPIO_MENU_NEXT: draw_state =
STATS; break;
    case U8X8_MSG_GPIO_MENU_PREV: draw_state =
STATS2; break;
    case U8X8_MSG_GPIO_MENU_UP: draw_state = STATS;
break;
    case U8X8_MSG_GPIO_MENU_DOWN: draw_state =
STATS2; break;
    default: break;
  }
}

/* Settings Page
 * - Select from list of settings
 * - Each item has a confirmation screen
 *  - "ok" returns 1
 *  - "cancel" returns 2
 */
void settings_page() {
  // button contains which item you selected. Begins
indexing at 1
  button = u8g2.userInterfaceSelectionList(
    "Settings",
    button,
    settings_list);

  // response contains 1 for "ok" and 2 for "cancel"
  response = u8g2.userInterfaceMessage(
    "Selection:",
    u8x8_GetStringLineStart(button-1, settings_list),
    "",
    " ok \n cancel ");

  if (response == 1) {
    switch(button) {
      case 1: logout(); draw_state = WELCOME; break;
      case 2: draw_state = CHANGE_ROLL; break;
      case 3: draw_state = CHANGE_SERVING; break;
      case 4: draw_state = STATS; change_day(); break;
      case 5: reset_eeprom(); draw_state = STATS; break;
      case 6: draw_state = STATS; break;
    }
  }
}

/* Page for Changing Rolls
 * - Used when a new roll is inserted, detect and set initial
thickness of full roll
 * - Updates screen with thickness of roll
 */
void change_roll() {
  full_roll_radius = measure_roll(); // Measure roll
thickness

  u8g2.setCursor(0, 15);
  u8g2.print("Thickness: ");
  u8g2.setCursor(0, 30);
  u8g2.print(full_roll_radius);

  if (u8g2.getMenuEvent() ==
U8X8_MSG_GPIO_MENU_SELECT) {
    draw_state = STATS;
  }
}
```

```
uint8_t serving_size = 10;
/* Page for Adjusting Serving Size (multiplier)
 * - Sets the multiplier for the dispense delay
 * - Stores multiplier in serving_size
 */
void change_serving() {
  // response contains 1 if value is set
  response = u8g2.userInterfaceInputValue("Serving Size",
"Size = ", &serving_size, 0, 15, 2, "x");
  if (response)
    draw_state = STATS;
}

/* Page for Changing Day
 * - Increments the Day by 1
 */
void change_day() {
  if (day == 29) {
    CurrentUser.previous_month = monthly;
    for (int i = 0; i < 31; i++) {
      CurrentUser.daily_servings[i] = 0;
    }
  }

  days_added++;
  what_day(); // updates 'day' with correct value
  day = day % 30;

  CurrentUser.daily_servings[day] = 0;
  weekly = 0;
  monthly = 0;

  // Calculate number of servings this month
  for (int i = 0; i < 31; i++) {
    monthly += CurrentUser.daily_servings[i];
  }

  // Calculate number of servings this week
  for (int j = 0; j < 7; j++) {
    weekly += CurrentUser.daily_servings[(day-j+31) % 31];
  }

  draw_state = STATS;
}

/* Manual Logout
 * - Logs user out when pressed
 * - Resets cardID
 * - FUTURE: store CurrentUser object to EEPROM
 */
void logout() {
  write_eeprom();
  cardID = 0;
  draw_state = WELCOME;
}

//////////////////////
// HELPER FUNCTIONS //
//////////////////////

/* RFID HEX Helper Function
 * - Converts 4 byte-buffered RFID value into one 32 bit
value
 */
uint32_t saveHex(byte *buffer, byte bufferSize) {
  uint32_t longhex = 0;
  for (byte i = 0; i < bufferSize; i++) {
    longhex += (uint32_t)buffer[i] << (8*(bufferSize-i-1));
  }
  return longhex;
}

/* Dispense Function
 * - Rotates servo motor to dispense paper
 */
void dispense() {
  if (!digitalRead(PROX_SENSOR_PIN)) {
    CurrentUser.daily_servings[day]++;
    weekly++;
    monthly++;
    myservo.write(105);
    delay(100*serving_size);
    myservo.write(90);
  }
}

/* Roll Measuring Function
 * - Function to measure distance of roll from sensor
 */
float measure_roll(){
  float dist_to_roll =
1.0/(dist_slope*float(analogRead(DIST_SENSOR_PIN))-dist
_offset);
  return dist_to_holder - dist_to_roll;
}

/* Roll Percentage Left Function
 * - Function to calculate percentage remaining of roll
 */
int percent_left(){
```

```
  curr_roll_radius = measure_roll();
  percentage = 100*(curr_roll_radius*curr_roll_radius -
inner_radius*inner_radius)/
  (full_roll_radius*full_roll_radius -
inner_radius*inner_radius);

  // Discard values with delta > +/- 4 of the last read
percentage
  if (percentage > previous_percentage + 10 || percentage
< previous_percentage - 10) {
    return previous_percentage;
  }

  // Clean the percentage value of noise
  if (percentage > 100) {
    percentage = 100;
  }
  else if (percentage < 0) {
    percentage = 0;
  }

  // Turn on LED if below 10%
  if (percentage <= 10) {
    digitalWrite(LED_PIN, LOW);
  }
  else {
    digitalWrite(LED_PIN, HIGH);
  }

  previous_percentage = percentage;
  return percentage;
}

/* Read from EEPROM
 * - Sets username, daily, monthly, and weekly from
recently read CurrentUser object
 */
void read_eeprom() {
  if (cardID == USER1) {
    EEPROM.get(0, CurrentUser);
    username = "User 1";
  }

  else if (cardID == USER2) {
    EEPROM.get(100, CurrentUser);
    username = "User 2";
  }

  else if (cardID == USER3) {
    EEPROM.get(200, CurrentUser);
    username = "User 3";
  }

  else if (cardID == USER4) {
    EEPROM.get(300, CurrentUser);
    username = "User 4";
  }

  else
    return;

  monthly = 0;
  weekly = 0;
  for (int i = 0; i < 31; i++) {
    monthly += CurrentUser.daily_servings[i];
  }

  for (int j = 0; j < 7; j++) {
    weekly += CurrentUser.daily_servings[(day-j+31) % 31];
  }
}

/* Write to EEPROM Function
 * - Writes new data to EEPROM before low power mode
 * - Only called from logout() function
 */
void write_eeprom() {
  EEPROM.put(500, day); // store current day

  if (cardID == USER1) {
    EEPROM.put(0, CurrentUser);
  }

  else if (cardID == USER2) {
    EEPROM.put(100, CurrentUser);
  }

  else if (cardID == USER3) {
    EEPROM.put(200, CurrentUser);
  }

  else if (cardID == USER4) {
    EEPROM.put(300, CurrentUser);
  }

  else
    return;
}

/* Updates what day it is
```

```
 * - Calculates how many days passed since poweron
 */
void what_day() {
  day = (((millis() + (sleep_counter*4000)) / 86400000) +
days_added) % 30;
}

/* Reset the EEPROM
 * - clears all EEPROM values with 0
 */
void reset_eeprom() {
  for (int i = 0; i < 31; i++) {
    CurrentUser.daily_servings[i] = 0;
  }
  CurrentUser.previous_month = 0;

  EEPROM.put(0,CurrentUser);
  EEPROM.put(100,CurrentUser);
  EEPROM.put(200,CurrentUser);
  EEPROM.put(300,CurrentUser);

  weekly = 0;
  monthly = 0;
}




////////////////////////////////
// Low Power/Sleep FUNCTIONS //
////////////////////////////////
void sleep_now() {

  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  watchdogOn();
  sleep_enable();
  sleep_mode();

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

  sleep_disable();
  watchdogOff();
  //Serial.println("Woke Up");

  //Serial.print("Sleep Count: ");
  //Serial.println(sleep_counter);

}


//SEE PAGE 78 of atmega328p manual for more detail
void watchdogOn(){
  //MCUSR = MCU Status Register
  //Clear WDRF(Reset Flag)(bit3) so watchdog doesn't
cause system reset
  MCUSR = MCUSR & B11110111;
  //WDTCSR = Watchdog Timer Control Register
  //set WDCE and WDE to 1
  //WDCE must be set to allow changes to WDE
  //Allows changes to prescalers for 4 clock cycles
  WDTCSR = WDTCSR | B00011000;

  //Setting Prescaler value to 512K or ~4seconds
  //Other options:
  //.25s = B00000100
  //.5s  = B00000101
  // 1s  = B00000110
  // 2s  = B00000111
  // 4s  = B00100000
  // 8s  = B00100001
  WDTCSR = B00100000;

  //Enable the watchdog timer interrupt
  WDTCSR = WDTCSR | B01000000;
  MCUSR = MCUSR & B11110111;
}

void watchdogOff(){
  //Clear Reset Flag
  MCUSR = MCUSR & B11110111;
  //Set WDCE to 1, set WDE and WDIE to 0 to stop
watchdog timer
  WDTCSR = (WDTCSR | B00010000) & B10110111;
}

ISR(WDT_vect){
  sleep_counter++;
}
```