ENGI2203 - Engineering Design II

Design Project: Final Report

Prepared for: Jean-François Bousquet

Group 16

Luigi Cortez B00839128
Jiawei Yang B00813077
Curtis Raymond B00829669
Ethan Johnston B00828763

DATE DUE: April 7 / 2021

Table of Contents

Table of Figures

Table of Tables

Chapter 1: Introduction

1.1 Project objectives

For modern university students, the vast majority of them will choose to leave home and rent a house with their peers. Sometimes problems arise between roommates due to different personalities, habits and cultural differences. In these scenarios, people choose to share items like cooking oil, laundry detergent, hand sanitizer, and toilet paper as public goods and split the cost equally. We focused on toilet paper, an underappreciated, usually underestimated but fairly expensive item of daily necessity. Even though each roommate's toilet paper consumption varies, the inevitable cost of toilet paper is often split equally among the roommates. Currently, roommates are unable to detect and count the amount of toilet paper used by others. As a result, those who use less toilet paper must pay the same amount of money as other roommates who might use a lot more. Since no one wants to spend their own money on providing someone else with a luxury amount of toilet paper, we aim to build a device that will finally settle arguments over who uses the most toilet paper.

Through internet research and investigation, we found that current toilet paper dispensers only feature automatic dispensing of toilet paper. Our design project is to develop a smart toilet paper tracker (toilet paper dispenser) to solve a perennial roommate debate. We aim to create a smart toilet paper tracker which has four features that we want to achieve.

- To limit and reduce the amount of toilet paper used by each roommate.
- To track and identify who is using the most toilet paper.
- The old toilet paper roll can simply be replaced by the new one under the bar in the roller system when the old one has been used up.
- The device must be easy to use and sanitize, and safe
- To keep the device cost under $34.99.

A list of functional objectives and specification

Since we are limited to the components in the parts list, the most important task is to program the AVR C code into the microprocessor that will control our different peripherals. We want to create the smartest toilet paper dispenser on the market and our design's superior intelligence will be reflected in several key areas.

- The motion module will be motion activated by waving a hand over the sensor.
- The dispensing module should dispense a preset number of sheets of toilet paper per given cycle.
- The input module will identify and track who is using the dispenser and how much toilet paper each person is using over a period of time as well as manually reset to start a new cycle of counting toilet paper usage.

- The processing module will record the total number of toilet paper cycles used by each individual to compare and identify who is using the most toilet paper.
- The run-out module is triggered when the toilet paper roll is almost fully used.
- The alarm module will sound when the hall effect sensor is triggered.

1.2 Methodology

We are limited to use the components in the parts list. In our project, we use:

- Nano V3 microprocessor as the Processing Module.
- The Serial Interface (Putty) as the Interface Module.
- The 3x4 Matrix Keypad as the Input Module.
- The modified SG90 Servo as the Dispensing Module.
- The PIR Motion Sensor (HC-SR501) as the Motion Module.
- The Speaker (PAM8302) as the Alarm Module.
- Hall Effect Sensor as the Run Out Module.

Chapter 2: System Architecture

2.1 Modules and their Interfacing

- **Interface Module**

The Interface Module represents the IOT and on device display functionality. This would be replaced with a smartphone app or web browser which shows similar data and setting changes. The device would still have a small display on it to display main pin entry and sheet counts at users program exit. The settings would be configured on the app and every month, users would receive an email which shows the amount of toilet paper each user has used.

- **Input Module**

The Input module is the main communication a user has with the TPT. It allows for initial user pin code entry and entry to the master user. The setting changes which can be done though the master user include changing the users pin codes, resetting the total amount of sheets used per user, and changing the number of sheets dispensed everytime the Motion Module is activated. Ideally this module is used before the user touches anything unsanitary in the bathroom. It should be easily sanitized to remove germs.

- **Dispensing Module**

The Dispensing Module allows for automatic toilet paper dispensing. When the Motion Module triggers this module, it dispenses a preset amount of toilet paper sheets. Once the amount of sheets is dispensed, the amount is added to the users total sheet count. The Dispensing Module allows for zero contact between the device and the user. Once the toilet paper is dispensed, the user may rip off the sheets and use them as they see fit.

- **Motion Module**

The Motion Module allows the user to dispense toilet paper without touching the device. Ideally they are able to wave their hand over the sensor component, activating the Dispensing Module. The Motion Module will not activate if there is movement by anything other than something above the sensor.

- **Alarm Module**

The Alarm Module will produce an annoying, audible noise to get the user's attention. It is triggered by the Run Out Module when the current roll of toilet paper is empty and needs replacement.

- **Run Out Module**

The Run Out Module detects when the toilet paper roll is depleted. If the roll is sensed as depleted, it triggers the alarm module to make a noise while also alerting the user via the Interface Module.

- **Processing Module**

The Processing Module is where all of the code is running. It is used to interface all hardware components, keep track of time, and store information. The information that needs to be stored is the total sheet count of each user, each user's pins, and the sheets dispensed per cycle.

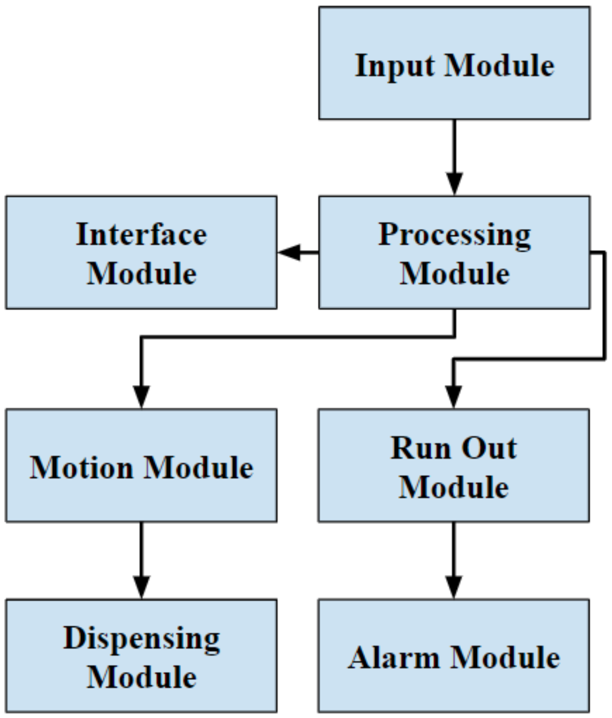


Figure 1: Module Block diagram showing how each module interacts.

2.2 Overview of Electronic System Architecture

- Serial Interface: (Putty) as the Interface Module. The serial interface acts as the placeholder visual interface allowing users to communicate with the Toilet Paper Tracker device and manage important settings and functions.

- 3x4 Matrix Keypad: as the Input Module. This was required to be used for the project, so we used it to identify users. It also was used to change the settings of the different modules.

- Modified SG90 Servo: as the Dispensing Module. Originally could only rotate by 180 degrees. Was modified to rotate continuously to fit the needs of the project.

- PIR Motion Sensor: as the Motion Module. Initially went with the Ultrasonic sensor but decided to go with this simpler option.

- Speaker (PAM8302): as the Alarm Module. This component performs more functions than is needed. Only used to play a simple tune. If available we would have chosen a Piezo Buzzer.

- Hall Effect Sensor: as the Run Out Module. Initially chose the push button but changed due to easier activation. Works by sensing a magnet and then activating when the magnet is sensed.

- Arduino Nano V3: as the Processing Module. It is used to process all of our code. Can potentially be used to store information.

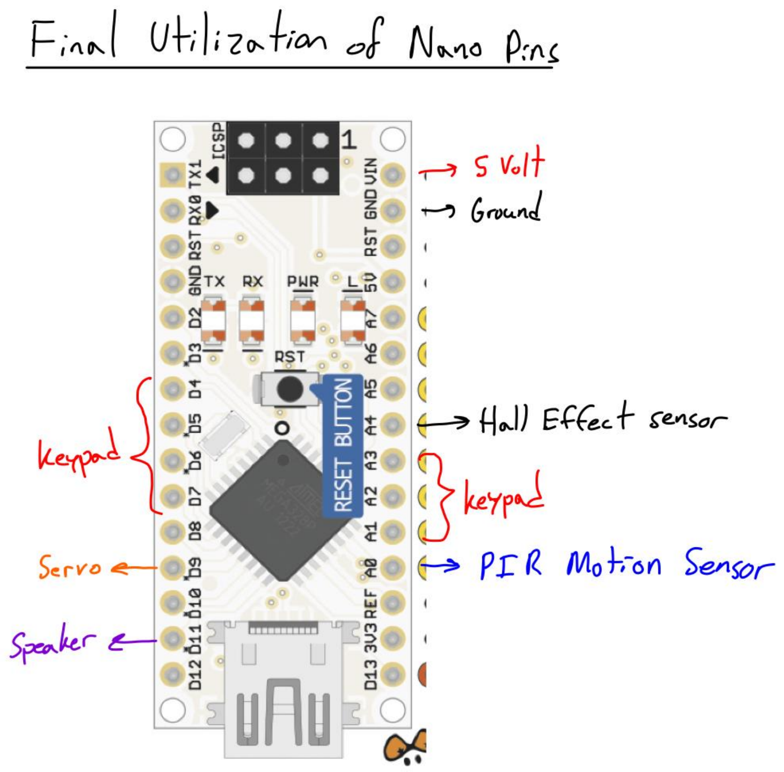


Figure 2: Arduino Pin usage.

2.3 Overview of Program Architecture

The implementation of the coding aspects of the toilet paper tracker were quite substantial as the design demanded usage of a large majority of the available peripherals. The code for each peripheral was divided amongst the group members and programmed in Microchip Studio's AVR-C. While the individual development process was challenging, it was very tedious to convert all of the individual codes into different functions that were controlled by a single main. The eventual integration of the various peripheral functions was achieved through a meticulous process of function calls, infinite while loops and manually assigned return values.

This section of the report will highlight the overall functionality of the combined Toilet Paper Tracker code.

The main flow of the program is governed by a user-friendly arrangement of print statements that communicate a variety of messages between the Arduino Nano and the user through the Putty computer interface. Another essential part of the main program is the use of various infinite while loops that help to control the flow of commands for the microprocessor. A data flow diagram is provided below in Figure **X** to aid in conceptualizing the flow of our code. Upon initiation of the components and putty interface, the software is designed to prompt the user for a 4-digit pin entry from the keypad to identify the given user. At the same time, the code enters a central infinite while loop that constantly checks for a pin entry or for a signal that the hall effect sensor has been triggered. If the hall effect sensor is triggered, then the code will call the speaker code to sound the alarm. Once an entry pin is received and recognized as a user pin, the software is programmed to distinguish whether the given user has entered a code to access what we call the master function or one of four normal user functions that enables the main workings of the device. If the recognized code is a regular user pin, the software enters another infinite while loop that waits for either a signal from the motion sensor function or a keypad 0 to end the given user session. If a signal is received from the motion sensor, the servo motor function is called to dispense a set number of sheets and the number of sheets dispensed are updated in a storage variable assigned to the user. When a user presses 0 to exit the user function, the program prompts for a new entry code and re-enters the central infinite while loop.

The software was also programmed to have a default master function initiated by entering pin “2203”, which happens to conveniently be the design course code. The master function gives access to control all of the Toilet Paper Tracker settings as well as the important function of displaying all users sheet counts to determine who is using the most toilet paper. Other important features of the master function include changing user entry codes, resetting user sheet counts, and changing the number of sheets dispensed per cycle by the servo motor. The master function is featured in the data flow diagram below.

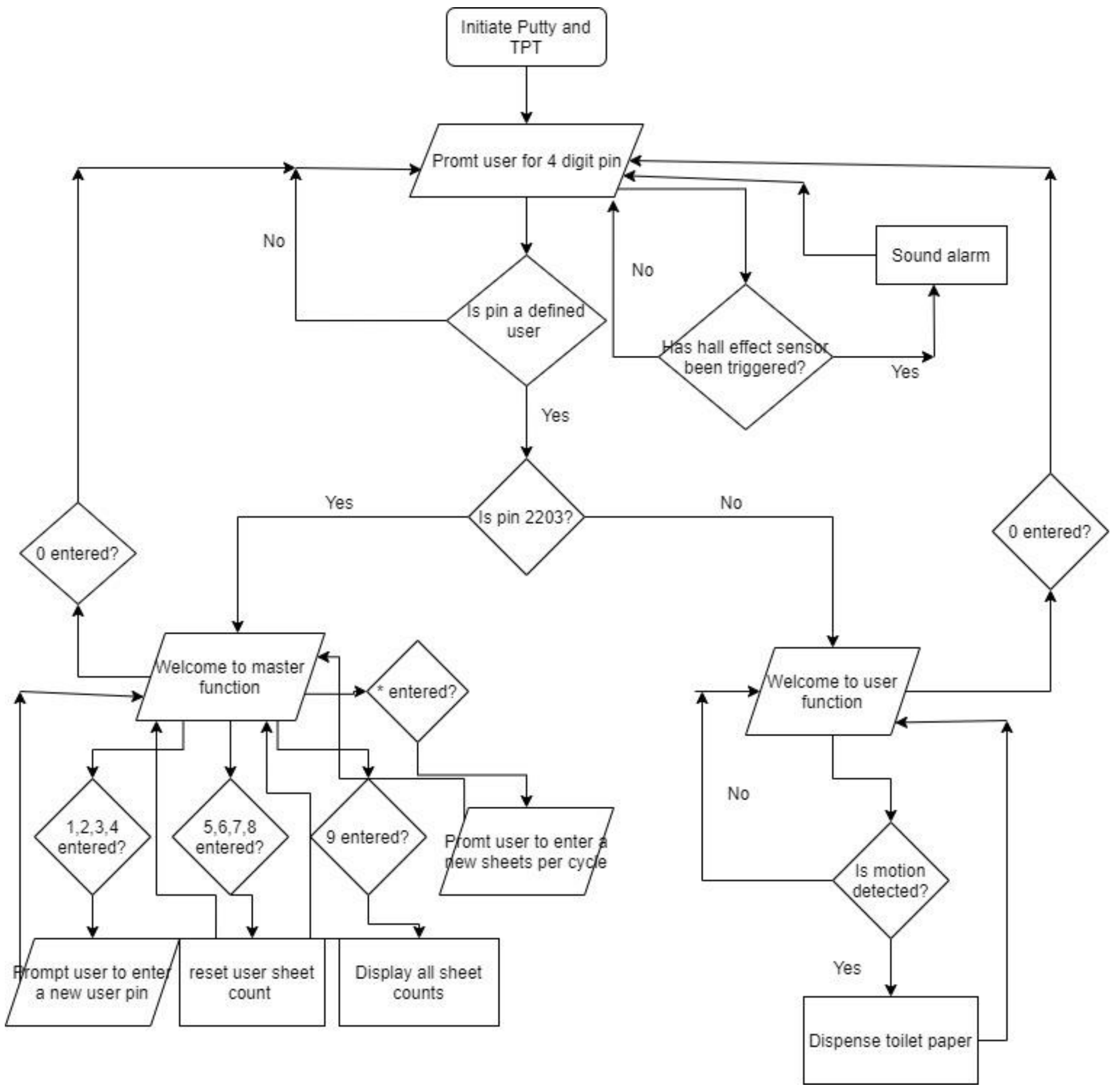


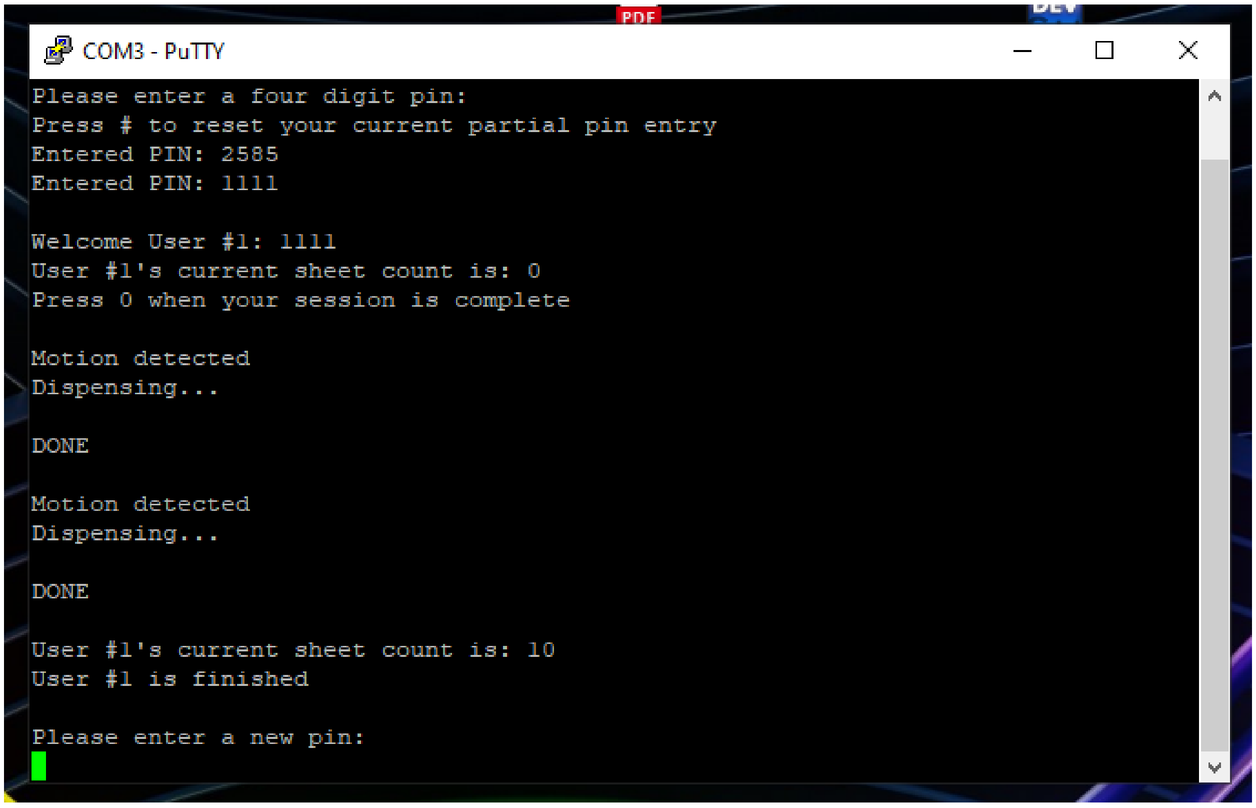
Figure 3: Program flow chart

Chapter 3: Detailed design

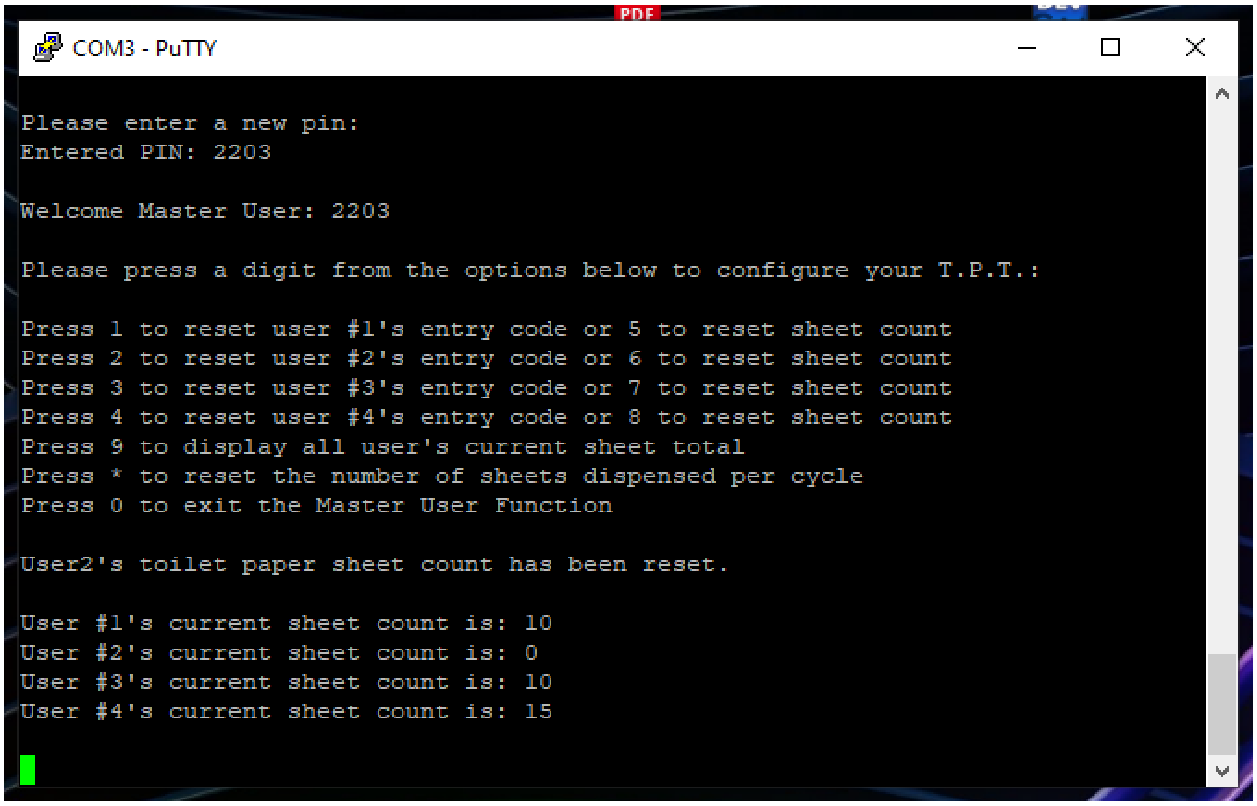
3.1 Electronics Design

3.1.1 Serial Interface: (Putty) as the Interface Module.

PuTTY is a program run on a computer that can be used as a troubleshooting tool or an interface. For our design PuTTY was used as an LED screen that showed the user the entered pin, sheets used, and setting changes.



```
Please enter a four digit pin:
Press # to reset your current partial pin entry
Entered PIN: 2585
Entered PIN: 1111

Welcome User #1: 1111
User #1's current sheet count is: 0
Press 0 when your session is complete

Motion detected
Dispensing...

DONE

Motion detected
Dispensing...

DONE

User #1's current sheet count is: 10
User #1 is finished

Please enter a new pin:
```



```
Please enter a new pin:
Entered PIN: 2203

Welcome Master User: 2203

Please press a digit from the options below to configure your T.P.T.:

Press 1 to reset user #1's entry code or 5 to reset sheet count
Press 2 to reset user #2's entry code or 6 to reset sheet count
Press 3 to reset user #3's entry code or 7 to reset sheet count
Press 4 to reset user #4's entry code or 8 to reset sheet count
Press 9 to display all user's current sheet total
Press * to reset the number of sheets dispensed per cycle
Press 0 to exit the Master User Function

User2's toilet paper sheet count has been reset.

User #1's current sheet count is: 10
User #2's current sheet count is: 0
User #3's current sheet count is: 10
User #4's current sheet count is: 15
```

Figure 4: Putty Terminal Output.

3.1.2 3x4 Matrix Keypad (Input Module)

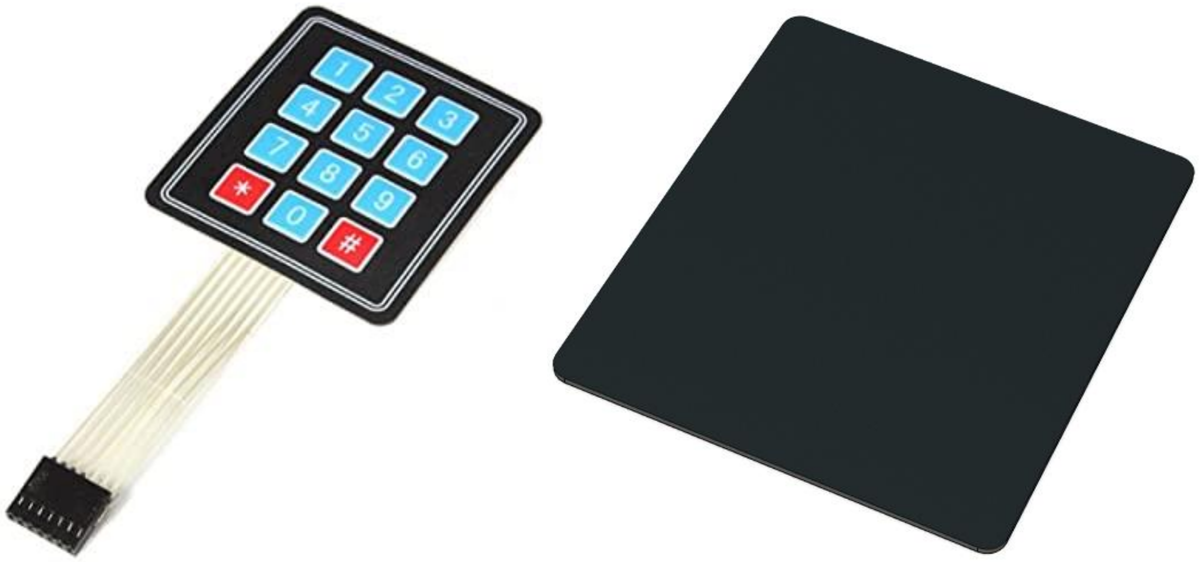


Figure 5: Keypad vs CAD representation.

3.1.3 Modified SG90 Servo (Dispensing Module)

The SG90 servo motor is a low power, compact rotational actuator that allows for precise control of angular position. It rotates to a specified position using a motor and potentiometer. Its rotation is limited to +/- 90°. It has a gear reduction for added torque. We have modified ours to rotate continuously, acting as a high torque motor.

The potentiometer no longer rotates when the motor spins. When our servo is set to move to its center position (Z), it is stopped because that is where the potentiometer is set and glued so it believes it is already at Z. When the servo is commanded to spin to (M90 or P90) it will rotate indefinitely in a direction. This is because the potentiometer will never reach the commanded angular position, so it continues to spin, thinking it still needs to rotate to get to the new position.

Once TP dispensing is required, the servo function is given the current sheets per cycle variable, converts this sheet count to seconds and then sets the servo to P90 (spins direction to dispense) for the calculated time. Once the time is over, the position is set back to Z (stopped).

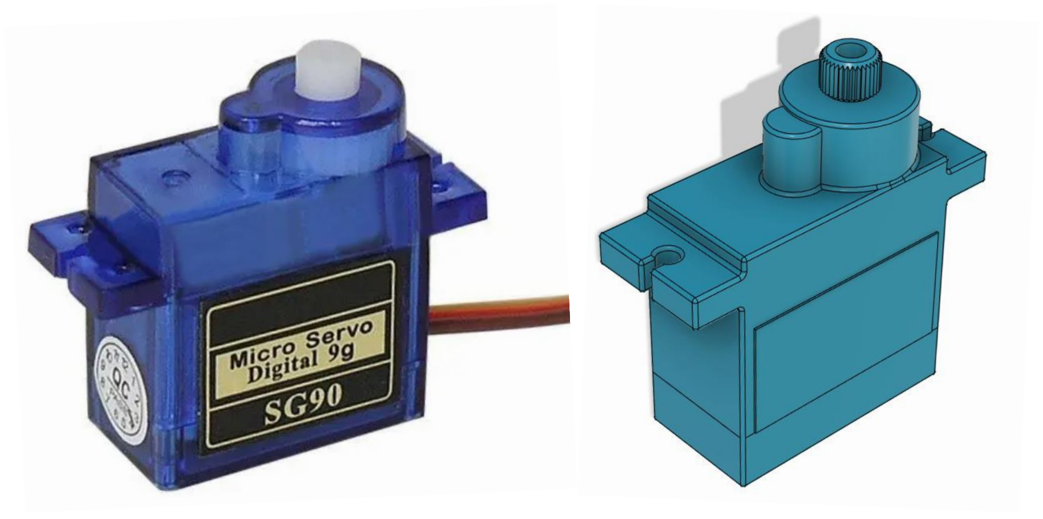


Figure 6: Actual Servo & modeled Servo.

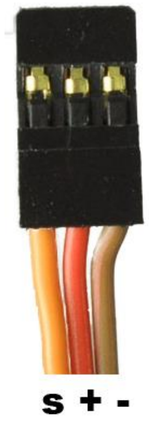


Figure 7: Servo pinout: s to D9 on the board or PORTB1, + to VCC, - to GND

3.1.4 PIR Motion Sensor HC-SR501 (Motion Module)

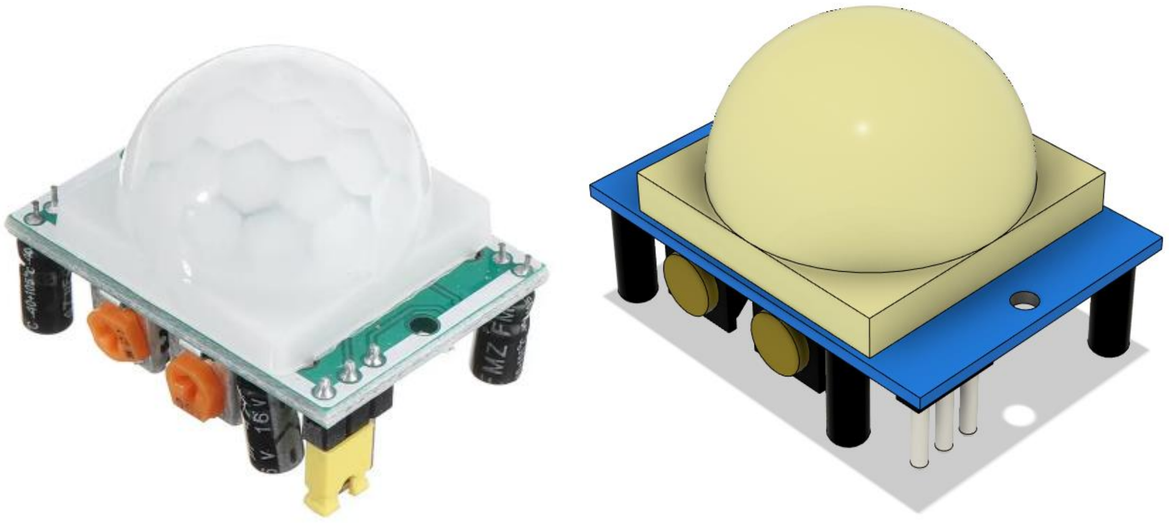


Figure 8: The PIR motion sensor (HC-SR501) vs our CAD representation.

For motion detection, we first tried to use the ultrasonic sensor, although the code was working perfectly, but we found that it crashed easily while testing. We realized that the ultrasonic sensor and servo both used the same clock pin. In the end, we decided to use a PIR motion sensor since it is easier to implement.

The PIR module has a passive infrared sensor that detects the movement from the infrared radiated from the human body. It has a Vcc pin, GND pin, and digital output. This output can be connected directly to one of any microcontroller digital pins. If any motion is detected by the sensor, this pin value will be set to “1”.

There are two potentiometers behind this module. By changing the SENSITIVITY potentiometer, you can reduce or increase the sensitivity of the sensor, and also by changing TIME potentiometer the output delay after movement detection will be changed.

There is a jumper behind this module. If you move the jumper to L position, the sensor will ‘toggle’ (change state) whenever motion is detected. This mode is called non-triggering or single triggering mode; however, this is unlikely to be of much use in our cace. Moving the jumper to the H position will result in the more usual sensor logic. The sensor will turn on when motion is detected and turn off a while after the last motion is detected. This sensor will reset the timer (which would otherwise turn the output off) each time motion is detected. This is called retriggering mode. (or repeatable trigger mode). There is also a ball like a lens on the sensor that improves the viewing angle.

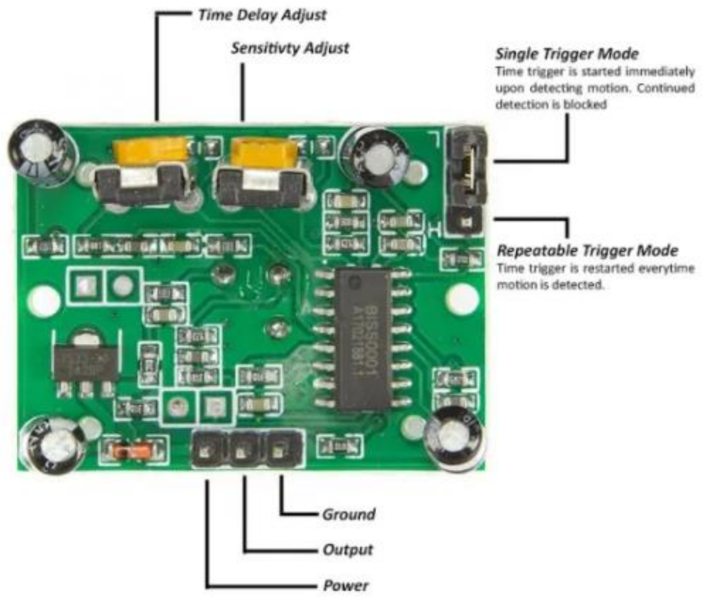

Time Delay Adjust
Sensitivty Adjust
Single Trigger Mode
Time trigger is started immediately upon detecting motion. Continued detection is blocked
Repeatable Trigger Mode
Time trigger is restarted everytime motion is detected.
Ground
Output
Power

Figure 9: Motion sensor pin description

To make the PIR motion sensor work properly, the Vcc pin and the GND pin are connected to VIN and GND respectively, and the digital output is connected to PINC0.

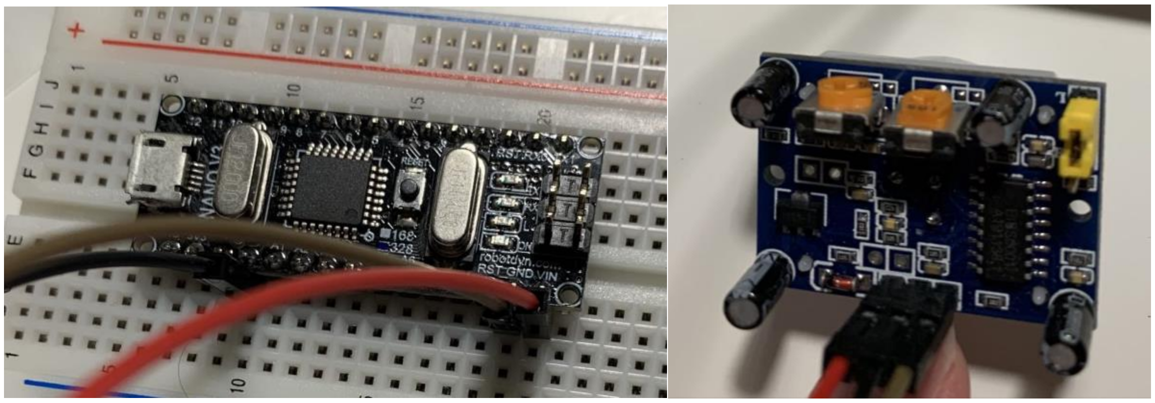


Figure 10: Motion sensor pinout: digital output to A0 on the board or PORTC0, + to VCC, - to GND

3.1.5 Speaker & PAM8302A (Alarm Module)

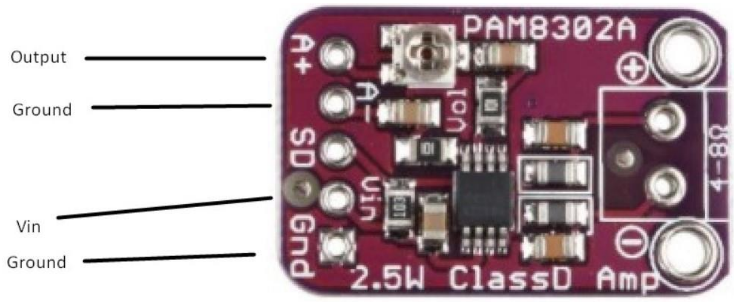


Figure 11: PAM8302A Pin Diagram

The component that we used to act as our alarm module was the PAM8302A mono audio amplifier. This component is a main audio controller, and a separate speaker must be acquired and soldered onto the controller.

This component allows for high quality sound reproduction and is able to output sound at a wide range of frequencies provided that it is supplied a voltage of 2.0V to 5.5V. The volume of the sound can be adjusted by using different octaves, another way to control the sound is by adjusting the amount of supplied voltage that goes into the system.

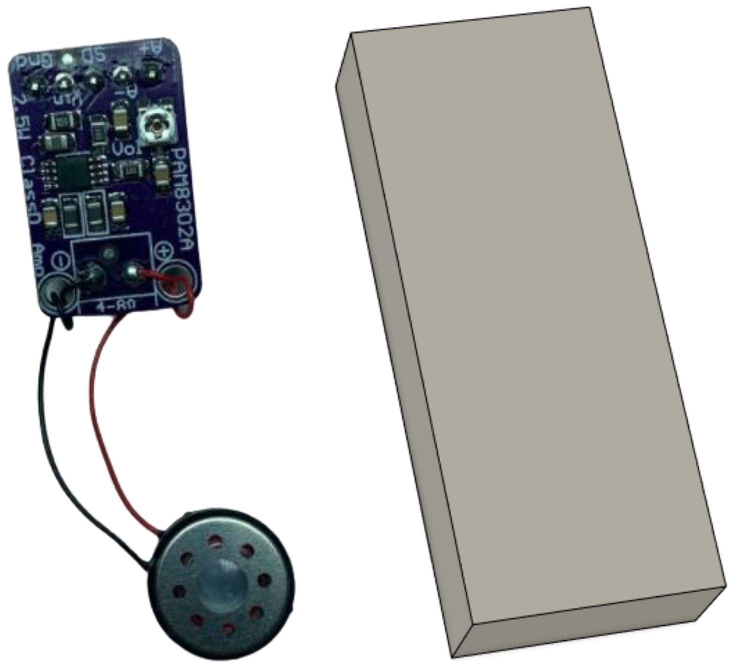


Figure 12: PAM8302A and speaker vs it's CAD mockup.

3.1.6 Hall Effect Sensor (Run Out Module)

- Include schematics, simulations if available, a description of the hardware, and measured module performance, if available.

The Hall Effect sensor is designed to work like a switch or button that is turned on or closed when a strong enough magnetic field is brought near the sensor and turned off or opened when the magnetic field is taken away.

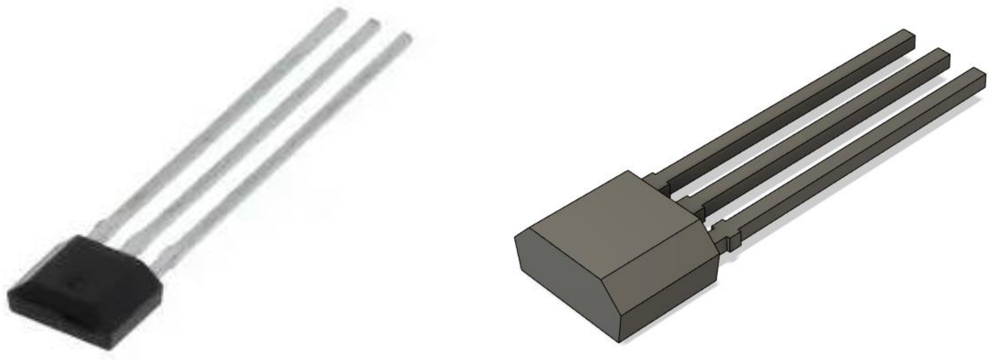


Figure 13: Our hall effect sensor vs our CAD representation.

Using the provided magnet, we were able to recover from previous setbacks using a pushbutton by taking full advantage of the much more sensitive Hall Effect sensor. The magnet and hall effect sensor combination can be seen below in figure **B** that enable us to accurately trigger the PAM8302A mono audio amplifier to sound when the toilet paper roll was fully depleted.

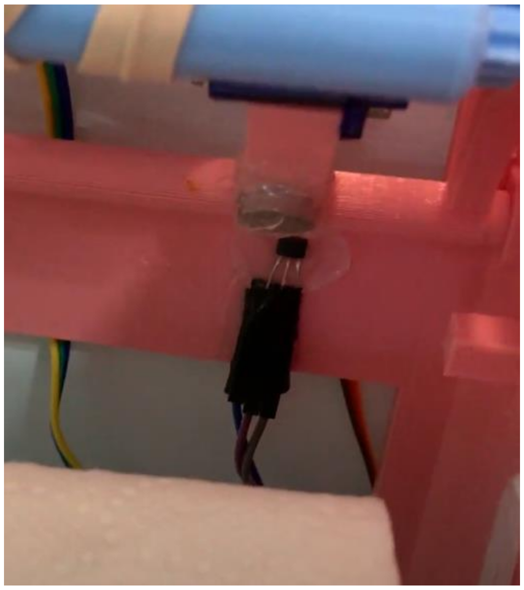


Figure 14: Hall effect sensor and magnet on mounted on TPT.

3.1.7 Arduino Nano V3 (Processing Module)

This component is a small micro processing board that is powered by plugging in a Mini-B USB link. The component holds all the software code and runs it. Depending on the outputs and inputs of the different modules attached to the board. It can use that information, store it, and output voltages.

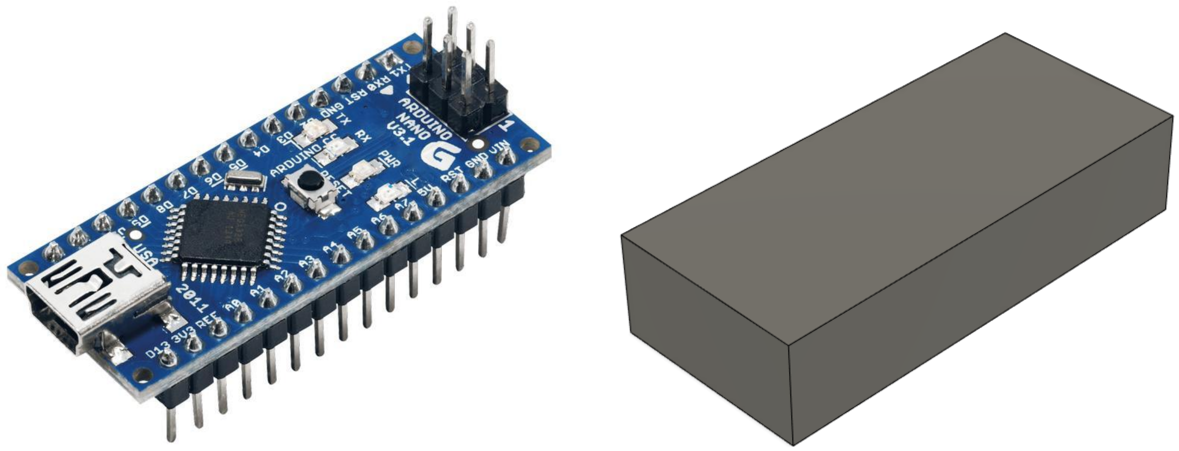


Figure 15: Arduino Nano V3 vs it's CAD representation.

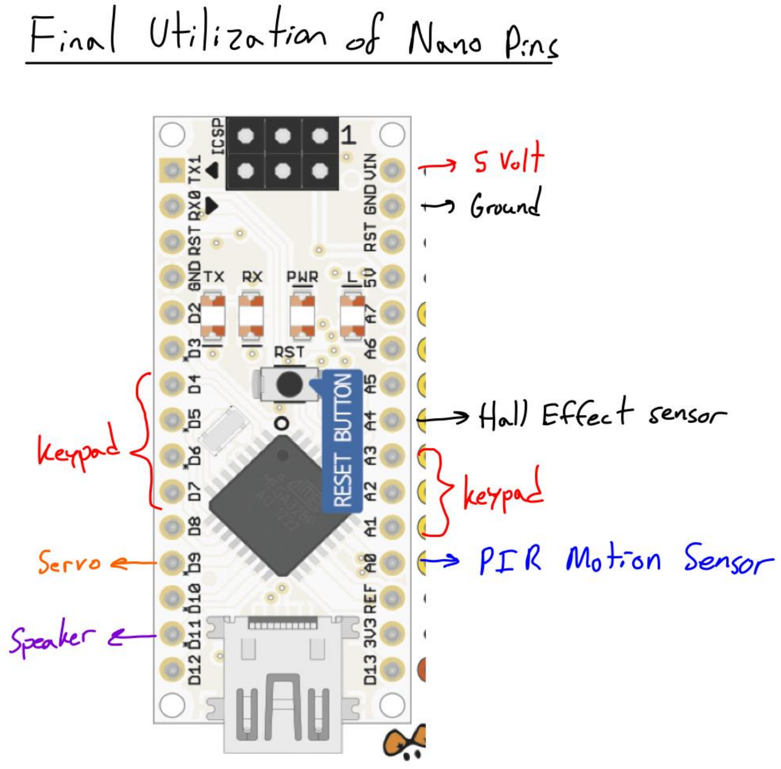


Figure 16: Arduino Nano pin usage.

3.2 Software Modules & Software Verification Strategy.

3.2.1 Speaker (PAM8302A)

Speaker Software

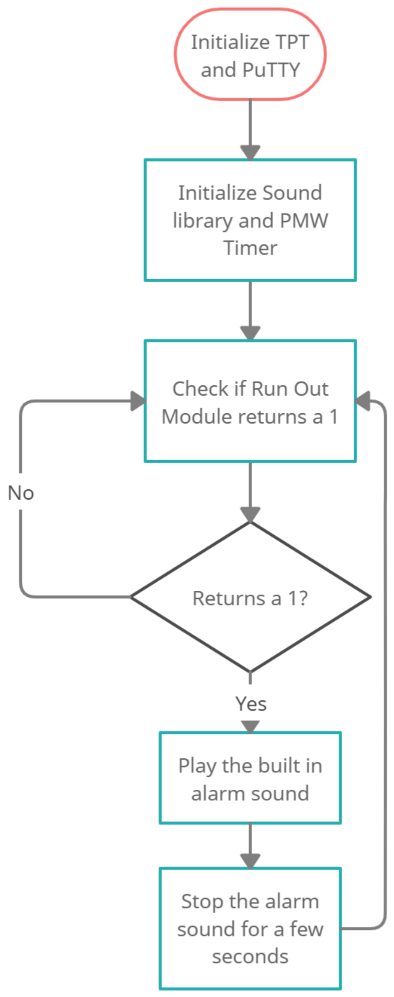

Initialize TPT and PuTTY
Initialize Sound library and PMW Timer
Check if Run Out Module returns a 1
No
Returns a 1?
Yes
Play the built in alarm sound
Stop the alarm sound for a few seconds

Figure 17: Flow diagram of the PAM8302 and its functions.

The following edited code was taken from this site: https://blog.podkalicki.com/attiny13-tone-generator/
PAM8302A Function Code. See appendix for full code with function and library definitions.

```
void alarmActivate(void)
{
    fprintf_P(fio_0, PSTR("Please replace roll\n"));

    /* setup */
    DDRB |= (1 << PORTB3); // set BUZZER pin as OUTPUT
    TCCR2A |= (1<<WGM21); // set timer mode to Fast PWM
    TCCR2A |= (1<<COM2A0); // connect PWM pin to Channel A of Timer0

    tone(1, 9); // A
    _delay_ms(1000);
    tone(1, 0); // C
    _delay_ms(1000);
    tone(1, 7); // G
    _delay_ms(1000);
    tone(1, 5); // F
    _delay_ms(1000);
    stop();
    _delay_ms(1000);
}
```

The software program consists of a library of the different frequencies at different octaves that the speaker can use and output, a tone function which controls what frequency and octave that is played by the speaker, and finally a stop function which turns the speaker off when it is not being used. The stop function was created so that the speaker would not overheat or get too hot. In order to control the speaker, a while function is implemented and if the motion module returns a '1' it runs the while loop which plays a sound using a tone() function and a stop() function to stop.

Test procedure to verify that the speaker and PAM8302A components are working:

1. Rebuild solution of code and implement it on the Arduino board.
2. Ensure that the PAM8302 Is properly wired onto the Arduino board and that the correct pins are inserted.
3. Wait for a few seconds and listen for a sound coming from the speaker connected to the PAM8302A.

After completing all the test procedures, if a sound is not heard from the speaker, it indicates that the module is not working. If a sound is heard from the speaker, it indicates that the module is working properly.

3.2.2 Keypad Software

Code Description:

The keypad device is the main peripheral that allows the user to communicate with the Arduino Nano and control important functions of the Toilet Paper Tracker. The keypad is called a matrix because it is designed with 4 rows and 3 columns that give the microcontroller positional awareness of each button and the associated character. When a given button is pressed, a connection is made between the corresponding row and column, allowing a properly programmed microcontroller to pinpoint the given character assigned to that keypad position with a 2-dimensional array. Figure **W** below shows the matrix connections between row and column:

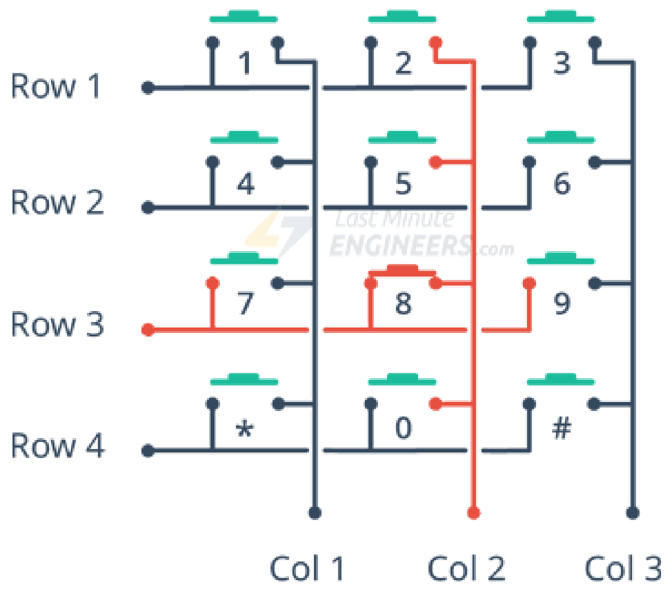


Figure 18: Keypad Wiring

In our implementation of the keypad code design, we enable pull up resistors for all three of the columns, setting each column to a high voltage signal. A function named set_row_low() then sets each row to a zero volt signal and switches from row to row every 0.02 seconds. If a button is pressed, the given row that is set low will connect to the corresponding column forcing a low signal onto the column. The microcontroller then reads this forced low signal and uses the low signal for the corresponding row and column to pinpoint the desired character.

The previous description only contains part of the story for the fundamental features of our keypad program. In order for our design to make any sense, we required a multi-digit entry to provide access security for the different Toilet Paper Tracker users. Luckily for us, once a single digit entry with the keypad is established, it is fairly simple to morph the single entry design into an “n” entry design. Figure **Y** below shows the overall function of our base 4-digit pin entry keypad code and it is important to point out that the first four steps are the exact same as a single button entry. The major change to create a 4-digit pin entry is to add an array with a purpose devoted to storing the four elements of the pin. Every time a button press is received, the button is stored in the pin array at a given index and then the index is incremented by one. An if statement controls when a null character is added to the end of the array of characters to mark the end of the character string. Because an array index starts at zero, the array has four elements when the index reaches three, but our code increments the counter after every button is recorded. Therefore, the if statement enters the utilization part of the code when the index reaches to four, indicating that four button characters are stored in the array. Figure **Y** below showcases the process of code to implement the 4-digit pin entry.

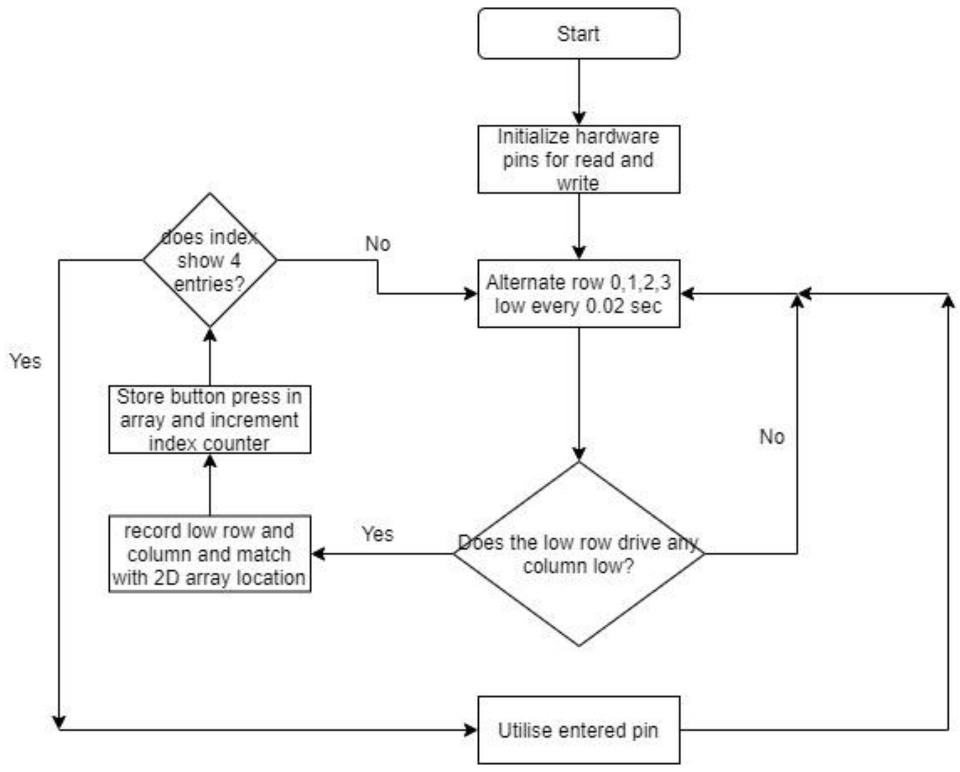
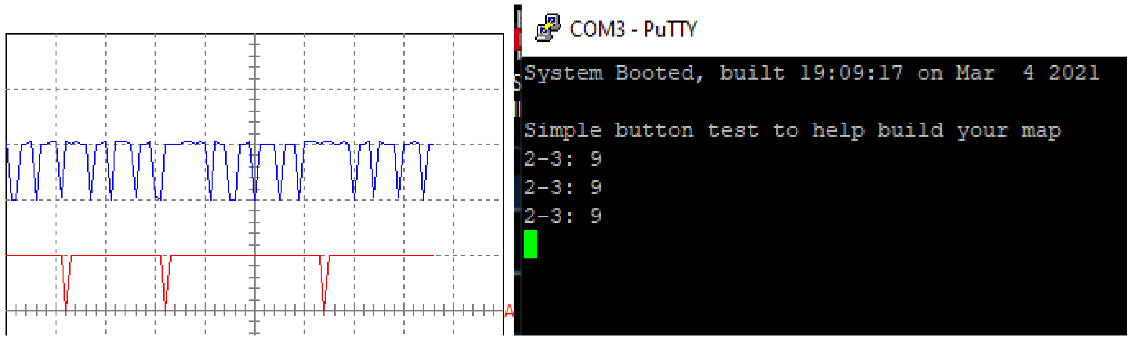

Start
Initialize hardware pins for read and write
Alternate row 0,1,2,3 low every 0.02 sec
Does the low row drive any column low?
No
Yes
record low row and column and match with 2D array location
Store button press in array and increment index counter
does index show 4 entries?
No
Yes
Utilise entered pin

Figure 19: Keypad Program Flow Diagram

Testing Procedure:

The functionality of the keypad code and peripheral is highly time reliant as rows are set low in a cyclic alternating fashion every 0.02 seconds. This heavy reliance on time led to the conclusion that testing should be completed through an oscilloscope, a time based voltage signal measurement device. The kit-included MFO-1 has two signal measurement channels A and B, allowing us to monitor two different signals at the same time. We therefore decided to monitor Row 2 (really Row 3 but the array index starts at 0) on Channel B and Column 3 on Channel A. The results from this testing can be seen below in figure **Z.**



```
System Booted, built 19:09:17 on Mar  4 2021

Simple button test to help build your map
2-3: 9
2-3: 9
2-3: 9
```

Figure 20: Keypad Testing

This test not only demonstrated that our keypad button press code functioned properly, it also shows the incredibly remarkable timing accuracy of the keypad system. Comparing the timing of the two signals when a button is pressed shows that Column 3 was driven low as expected and remained at a high signal when it was supposed to.

3.2.3 Servo Software

Code Description

Servo motors, unlike other motors, do not need a driver. The shaft rotates to a certain angle when a PWM signal is applied to its signal pin, depending on the duty cycle of the pulse. Pulse width modulation (PWM) is an acronym for pulse width modulation. It is a modulation technique in which the width of the carrier pulse is varied by the analog message signal.

Servo Function Code. See Appendix for full code with servo & timer initialization.

```
int servo(int sheets)
{
    float sec = sheets / 2; //Math to convert from sheets to seconds
    double sElapsed = 0;
    double sInitialTime = count;
    fprintf_P(fio_0, PSTR("\nDispensing...\n\r"));

    while(sec > sElapsed)
    {
        if(button_function() == 1)
        {
            OCR1A = Z;
            return 1;
        }
        sElapsed = count - sInitialTime;
        OCR1A = M90; //M90 or P90 depending on final direction
    }
    OCR1A = Z;
    fprintf_P(fio_0, PSTR("DONE\n\r"));
    //_delay_ms(100);
    return 0;
}
```

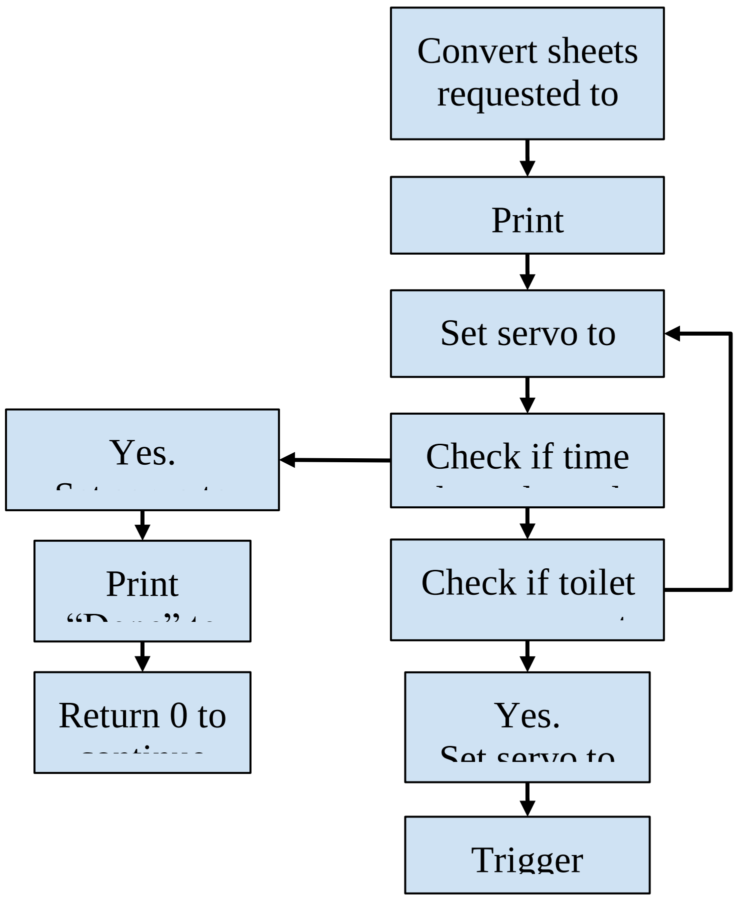


Figure 21: Flow diagram of code structure of the servo function.

Testing Procedure:

Test procedure to verify that the SG90 Servo component is working:

1. Rebuild solution of code and implement it on the microprocessor.
2. Ensure that the servo is properly wired onto the microcontroller.
3. Open serial monitor, enter user pin, trigger motion sensor.
4. Servo should rotate in a direction.

3.2.4 PIR Motion Sensor Software

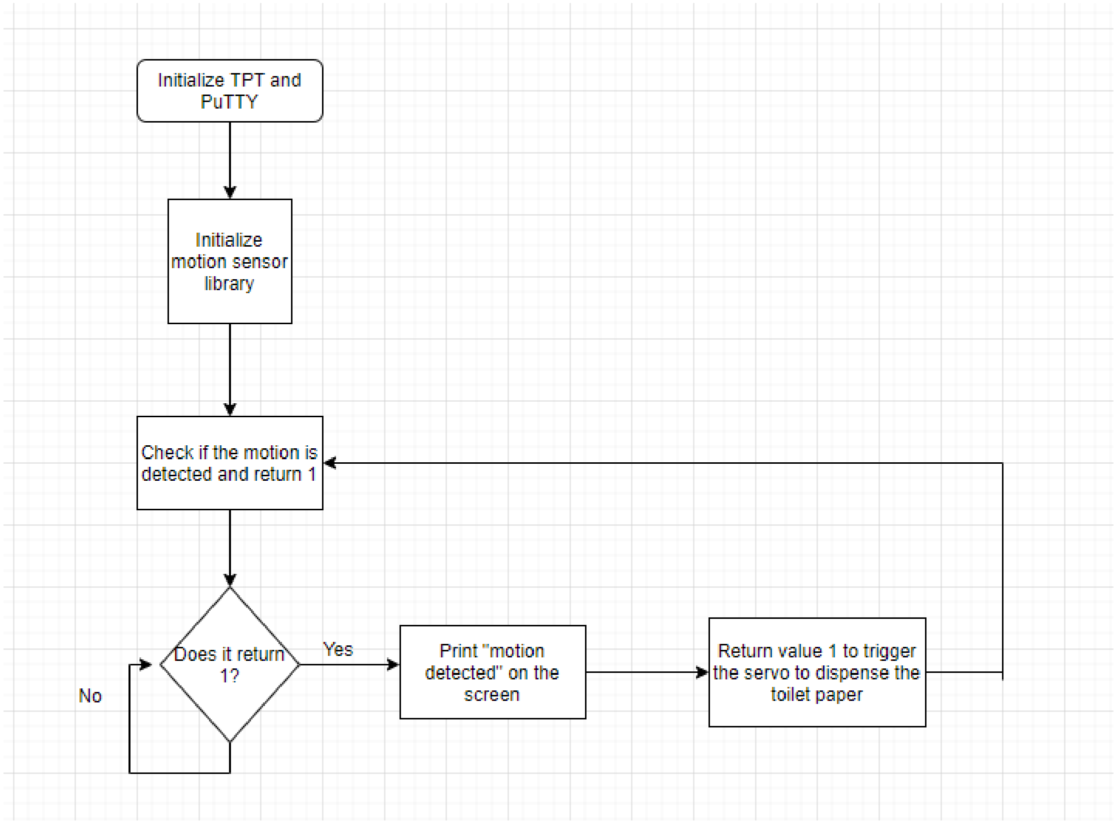

Initialize TPT and PuTTY
Initialize motion sensor library
Check if the motion is detected and return 1
Does it return 1?
Yes
No
Print "motion detected" on the screen
Return value 1 to trigger the servo to dispense the toilet paper

Figure 22: Flow diagram of the HC-SR501

PIR motion sensor Function Code. See Appendix for full code.

```
int Motion_Sensor()
{
    PORTC |=(1<<PINC0);  //configuring PortC pin 0 as input
    double mElapsed = 0;
    int sec = 120;
    double mInitialTime = count;

    while(sec > mElapsed)
    {
        mElapsed = count - mInitialTime;
        if (PINC & (1<<PINC0))
        {
            fprintf_P(fio_0,PSTR("Motion detected"));
            return 1;
        }
        if (get_new_button() == '0')
        {
            break;
        }
    }
    return 0;
}
```

Testing Procedure:

1. Rebuild solution of code and implement it on the Arduino board.
2. Ensure that the motion sensor is properly wired onto the Arduino board and that the correct pins are inserted.
3. A while loop continuously detects the signal caused by the motion. The users just need to wave their hands over the sensor, if any motion is detected by the sensor, this infinite loop will break, print “motion detected” on the screen, and return 1 which triggers the servo to dispense TP.

3.3 Mechanical Design

The expected behavior from our mechanical design:

1. The frame or body part prevents toilet paper roll from falling off after continuous dispensing.
2. The servo spins its gear which spins the rollers gear which spins toilet paper roll. This action is consistent over many hours of use.
3. Arm freely adapts as roll depletes allowing for constant pressure to be applied to roll as it depletes.

Description:

The mechanical design brings all aspects of the design together. We 3D printed most of our parts which allowed for easier iteration of parts. The part that held all the components together is seen in figure below. This is the main body of the mechanical design. It held the roll of toilet paper, the PIR motion sensor, and the hall effect sensor and magnet. The cutouts in the back are to save on plastic and to wire the cables through.

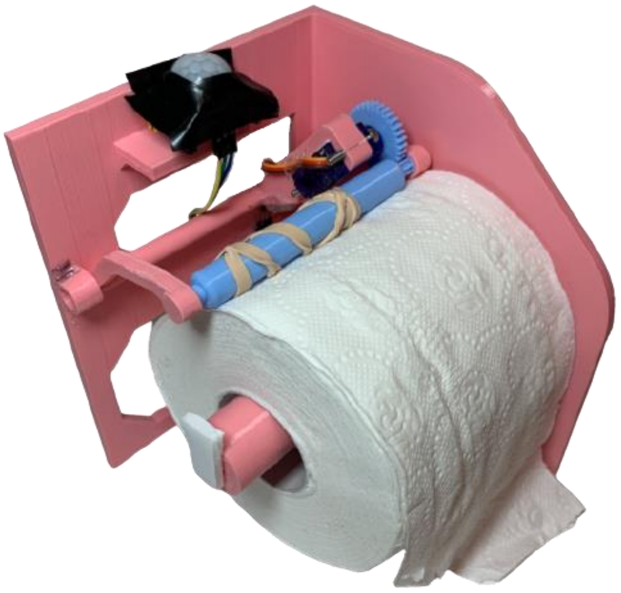
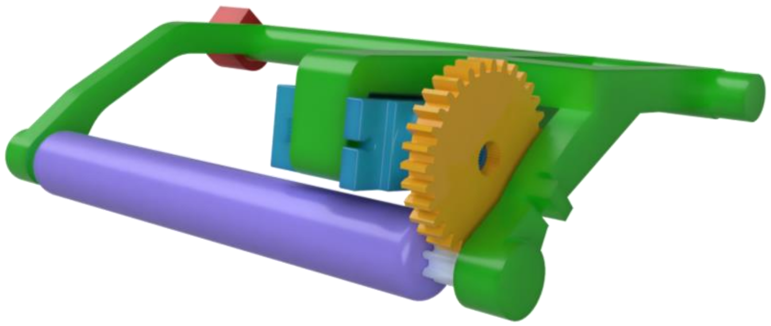


Figure 23: Final Prototype

The main complex mechanical assembly in our design is the arm with the servo which spins a roller to dispense the toilet paper. Seen in the figure below, this assembly has an arm which rotates around a piece which is glued onto the body during assembly and holds the servo. The arm allows for the roller to rotate inside two holes on either side of its ends. On the roller there is a six tooth gear built in which meshes with the 24 tooth gear. This ratio of gears allows the servo to spin the roller approximately four times faster than if the servo was directly spinning the roller. This allows us to meet our unofficial requirement of a quick dispense. Nobody wants to be sitting waiting for the toilet paper to come out.



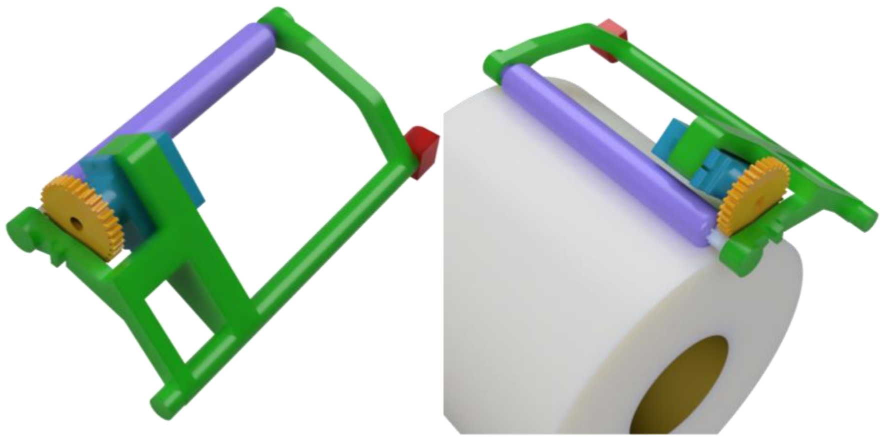


Figure 24: Arm in green, roller in purple, servo in blue, gear in yellow.

Overall Testing Procedure:

1. Test the essential keypad functions and the corresponding Putty interface:
   - Test that after the user exits that the proper total sheet count is displayed.
   - Test that only after a standard user key input (pin 1111, 2222...), the motion sensor can be triggered.
   - Show master function (pin 2203) and its 4 main uses: to change user pins, to reset user sheet counts, to change # of sheets dispensed per cycle, and to display a list of all user's current total sheets used.
2. Test that the toilet paper dispenses when the motion sensor is activated.
3. Test that when the toilet paper roll is depleted it activates the hall effect sensor and activates the alarm

Testing:

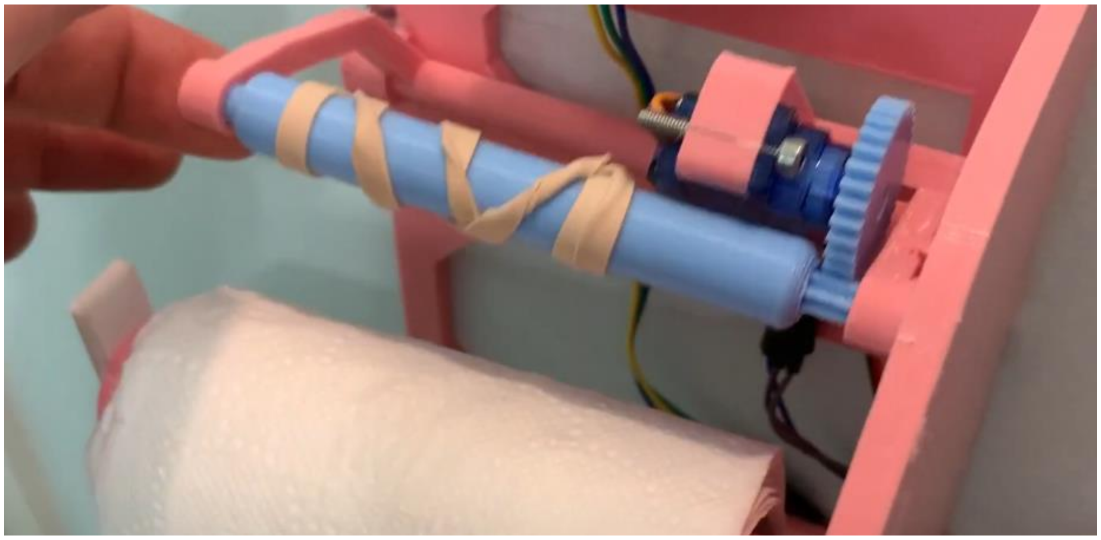


Figure 25: Testing

Our mechanical design came together quite easily, with Ethan's CAD and 3d printing skills, it was not challenging to get the tolerances and meshing of the gear working. From our mechanical design, we expected the body part to hold the toilet paper without falling off the open side while being dispensed. This can be seen in the photo below where we needed to add a white piece of foam core in order to keep the roll from falling off as more toilet paper was dispensed. This is there in place of where the door for the device would be.

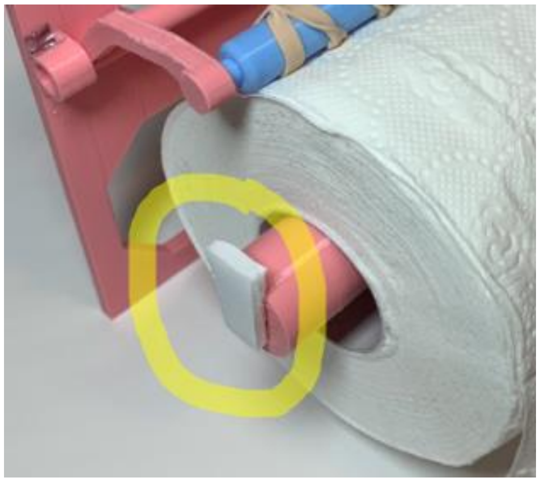


Figure 26: Foam core piece keeping roll from falling off of the holder.

Our arm and gear mechanism work flawlessly, we expected that the arm would be able to freely rotate inside the holes of the body piece. Also, we needed the roller to do the same with the holes on the arm. The gears lined up perfectly which was great from the very start and the servo seated nicely in its place with the servo gear fitting without any modification. In our CAD we never put an elastic in since it would be annoying to model, but we had expected we would have grip issues between the plastic and the paper from the beginning, it was always the plan to add an elastic to allow for perfect grip of the toilet paper.

Chapter 4: System Performance

4.1 Summary of the performance of the design

Performance analysis of key modules:

Keypad:
The keypad reliably detects individual button presses and allows the user to enter a four-digit password that the microcontroller can compare with the set user entry pins.

PIR Motion Sensor:
When enabled, the PIR motion sensor successfully detects movement and motion sensitivity can be easily adjusted with the potentiometer. Combined with the microcontroller, the motion sensor reliably triggers the servo motor to dispense toilet paper.

Modified SG90 Servo:
After being triggered by the PIR motion sensor, the servo rotates the bar at a 4:1 gearing ratio, which successfully rotates the toilet paper roll and dispenses approximately one sheet every 0.5 seconds.

Hall Effect Sensor:
This device successfully triggers when the magnet is brought close enough to the sensor when the toilet paper roll is almost depleted.

PAM8302A Speaker: Consistently produces the programmed tones using the tone function when triggered by the Hall Effect sensor. The stop function is created to make sure that the speaker would not overheat or get too hot.

Integrated tests
The integration of all of the different peripherals posed a significant challenge as many peripherals competed over pins that provided specific functions. After much trial and error we realized that we needed to swap the ultrasonic sensor with the motion sensor to free up the needed timer for the servo motor. After making this change and altering individual module functions slightly to work together, the integrated system worked together perfectly, just like clockwork!

Demo performance
To test our device, the following steps should be completed:

- Inputting the correct user PIN code
- Waving your hand over the motion sensor according to how many sheets are desired
- Pressing ‘0’ on the keypad to exit the user function
- Testing the master functionality when entering the pin 2203

Following a list of steps outlined above to test our device, which leads to the toilet paper being dispensed perfectly.

4.2 Comments on the achievement of engineering characteristics and specifications as defined in Chapter 1.

After the final test of our toilet paper tracker, our device successfully fulfills a list of functional objectives and all engineering characteristics.

1. The AVR C programming code successfully lets the keypad peripheral identify and track each person's usage of toilet paper.
2. The microcontroller successfully records the total number of toilet paper cycles used by everyone to compare and identify who is using the most toilet paper.
3. The system is successfully initiated when people wave their hand by a PIR motion sensor which prompts the servo to dispense the preset amount of toilet paper immediately.

4. The microprocessor enables the servo motor to rotate a preset number of turns to distribute the same number of sheets as the preset value.
5. The device successfully activated the speaker to sound when the toilet paper roll is almost fully used, which triggers the hall effect sensor.
6. The device count of toilet paper successfully be manually reset through the keypad to start a new cycle of counting toilet paper usage.
7. When a toilet paper roll has been used up, the roll is easily removed, and the new roll is simply placed under the purple bar in the roller system.

Furthermore, our toilet paper dispenser successfully meets four objectives.

1. Limit and reduce the amount of toilet paper used by each roommate.
2. Track and identify who is using the most toilet paper.
3. Easy to use and sanitize, and safe.
4. The cost of the device is under $34.99.

4.3 Cost Analysis

Table 1: Cost Analysis of Prototype vs Production

Item	**Cost (Prototype)**	**Cost (Production, PPU 100)**
1 Arduino Nano V3	$5.96	-
1 ATMEGA328P SMD MP	-	**$2.80**
1 Keypad Matrix	$2.19	$0.81
1 Hall Effect & magnet	$1.12	$0.14
1 Piezo Buzzer	-	**$0.22**
1 Speaker (PAM8302)	$13.95	**-**
1 PIR Motion Sensor (HC-SR501)	$2.70	$1.20
1 Servo Motor (SG 90)	$2.67	$2.67
1 Breadboard	$5.00	-
Pack of Jumpers	$3.49	-
1 Printed Circuit Board	-	**$0.30**
3D Printer Plastic	$4.00	$4.00
Total Cost	**$41.08**	**$12.14**

Our last objective was to keep the price under $34.99. This will be easily achieved if we produce at scale and find more affordable suppliers. Replacing the PAM8302A audio amplifier with a piezo buzzer reduces the price while accomplishing the same design functionality set for the Alarm Module. A custom printed circuit board and using the ATMEGA328P SMD chip would eliminate the need for jumpers, a breadboard, and the Arduino development board. The Cost for Production on the chart is based on if we sourced materials to produce 100 units per unit price would drop more if produced in larger quantities. There is also the possibility to injection mold our enclosure allowing for cheaper per part costs but a significant mold cost. We have not researched this enough to make an accurate estimate for this process. It is not included in our pricing. With all cost savings and producing in 100 quantities the approximate price per unit changes from $41.08 to $12.14. This allows for a healthy profit margin and room for labor costs.

Conclusion

When the group first started working on the design, one of the issues that was brought up was the fact that the provided servo motor was not able to rotate continuously, it was only capable of moving in a 180-degree angle which was a problem as we needed to properly dispense toilet paper. So, a major innovation in our design process was modifying the motor servo so that it could rotate continuously. This change was made by Ethan Johnston and the modification worked seamlessly.

During our design process, the coding and implementation of all the modules went smoothly and our team faced little to no problems setting it up. Minor issues such as pin usage and code integration were quickly solved in a few minutes. One key issue, however, was the ultrasonic sensor component which acted as the motion module. The ultrasonic sensor required the usage of a timer on our Arduino that the servo motor was also utilizing. This caused both the dispensing module and motion module to cease functioning. Both of these modules were key functionalities so our group decided that we would either make the ultrasonic sensor utilize another timer or we would replace the component with another sensor. Jiawei, who was responsible for coding the ultrasonic sensor did not find a way to fix this issue by altering the code, so we decided to replace the component. The component used was the PIR motion sensor, this component could fulfill all of the functionalities of the ultrasonic sensor hence why it was chosen. Jiawei was responsible for coding the PIR motion sensor, and when the code was finished and the component was fully functioning on the Arduino, it worked perfectly with all of the modules causing no errors.

The outcomes of the final design process is the full completion and implementation of the Toilet Paper Tracker. This machine fulfilled all of the outlined objectives that the team was assigned to achieve at the start of the design process. The Toilet Paper Tracker is able to dispense toilet paper, track the toilet paper usage of the user, and notify the user when the toilet roll is depleted. The performance of the device performed as expected with respect to the functional requirements and there were minimal issues that were easily fixed.

A summary of our project and its key modules and features is as follows:

- The keypad identifies the user and allows settings to be changed.

- The motion sensor detects the wave of a hand which triggers the servo motor to dispense.
- The servo motor automatically and accurately dispenses a set amount of TP.
- The hall effect sensor activates the speaker when the TP roll is depleted.
- The speaker outputs a beeping noise sound to get users attention.

Multiple tests of the device reveals that all of the functions stated above are repeatable and consistent. No notable errors occur even if multiple usages are conducted back to back. This is ideal since it shows that the device will likely not run into any problems when fulfilling its objectives and only a minimal amount of work is needed to get a fully functional and sellable product into the market.

The potential application of our device is limited to any restroom or bathroom environment that includes a toilet, without a toilet, the usefulness of our product is minimized.

Some of the potential applications include:

- Usage in a home environment with a small group of people.
- Usage in public restrooms as a general toilet paper dispenser.
- Usage in commercial or business restrooms.

Appendix A - Code

```
/*
 * Final_Design_Project_Code_TPT.c
 *
 * Created: 2021-03-12 2:53:38 PM
 * Author : Group 16
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <string.h>
#define MXN 5

#defineN_1     (_BV(CS00))
#defineN_8     (_BV(CS01))
#defineN_64   (_BV(CS01)|_BV(CS00))
#defineN_256 (_BV(CS02))
#defineN_1024          (_BV(CS02)|_BV(CS00))

#define F_CPU 16000000UL
#include <util/delay.h>

#include "USART0.h"
FILE *fio_0 = &usart0_Stream;

/*************************************
Button Functions:

**************************************/



int button_function(void)
{

    if (PINC & (1<<PC4))
    {
        return 0;
    }
```

```
   else
   {
      return 1;
   }
}

/*****************************************
Keypad Functions:
*****************************************/

/****************************************************************************
function init_hardware(void)
called to initialize the pins for the keypad matrix.
recall that the 4 rows are configured as output (write),
and 3 columns are configured as input (read).
TO DO: defined the appropriate pin, depending on where you connect each pint of
the keypad  matrix
****************************************************************************
**/
void init_hardware(void)
{
   //Configure pins you need as OUTPUT. You'll have to look at where you plug
   //the keypad in. For example if you were using PORTD6, PORTD5, and PORTB0 as outputs
   //you would do:

   //DDRD |= (1<<PORTD6) | (1<<PORTD6);
   //DDRB |= (1<<PORTB0);
   //NOTE TO STUDENTS: THE ABOVE ARE NOT CORRECT PINS AND NOT ENOUGH
PINS!!!!

   //Next, you might want to use built-in pull-up resistors. For example if we
   //decide PINDB2 and PINB3 are inputs, you could enable pull-ups with:
   //PORTB = (1<<PORTB2) | (1<<PORTB3);
   //NOTE TO STUDENTS: THE ABOVE ARE NOT CORRECT PINS AND NOT ENOUGH
PINS!!!!

   // configuring the four write pins (Black Labeled Lines on the Keypad):
   char temp = (1<<PORTD7) | (1<<PORTD6) | (1<<PORTD5) | (1<<PORTD4); // 0xF0 =
0b11110000
   DDRD |= temp; // we don't know what is on DDRD previously. We are forcing ones on bits
4...7
```

```
    // 1<<PORTDX places a decimal 1 in the register and shifts it to bit position x while
remaining bits are 0
    // each statement is "or" together therefore only the specified ports will have a 1 for write
mode and the
    // rest will be 0

    // next enable pull up resistors for the 3 pins that we are reading from
    PORTC |= (1<<PORTC1) | (1<<PORTC2) | (1<<PORTC3);

}

/***************************************************************************
***
This function writes to the appropriate pin, depending on the row called.
Note that the row indices are 0, ..., 3.
TO BE COMPLETED BY THE STUDENTS.
****************************************************************************
**/
void set_row_low(unsigned int row)
{
    //This code should set a single pin LOW associated with
    //the passed row. Ensure only a single pin goes low. - we are passed a row index from 0 - 3
    // This function sets the given pin/row to low (0)

    //You could use for example an if statement, case statement, etc.

    // we just want to set a given row to 0 - the rest remain untouched!

    PORTD |= (1<<PORTD4 | 1<<PORTD5 | 1<<PORTD6 | 1<<PORTD7); // set PD4-7 all to
high: PORTD | 0b11110000

    if (row == 0)
    {
        PORTD &= ~(1<<PORTD4); // PD4 corresponds to our row 0 - clears PD4 to 0 but
everything else remains untouched
    }
    else if (row == 1)
    {
        PORTD &= ~(1<<PORTD5);
    }
    else if (row == 2)
```

```
   {
      PORTD &= ~(1<<PORTD6);
   }
   else   // row = 3
   {
      PORTD &= ~(1<<PORTD7);
   }
}

/*************************************************************************
***
This function reads the status of each keypad matrix column.
The return value is the value of the first column that is in a low state.
Note that by definition, the column indices are 1, 2, 3
**************************************************************************
**/
int col_pushed(void)
{
   //This code should return what column was detected.
   //Remember the following:
   // 1) We need to mask off such we only get the bit we care about.
   // 2) We need to detect the LOW state (as we are setting the row to be LOW)

   //For example, if you wanted to check if PINB2 was low and return '1':
   //if ((PINB & (1<<PINB2)) == 0){
   //    return 1;
   //}
   //If no column detected, return 0

   if ((PINC & (1<<PINC1)) == 0)
   {
      return 1;
   }
   if ((PINC & (1<<PINC2)) == 0)     // (1<<PINB1) = 0b0000 0010
   {
      // PINC & (1<<PINC1) will give 0 @ PC1 if PINC has a 0 and 1 if PINC has a 1
      // statement automatically sets all other bits to 0 based off & function
      // A value of 0 means that there is a short between the row and column
      return 2;
   }
   if ((PINC & (1<<PINC3)) == 0)
```

```
    {
        return 3;
    }

    return 0;
}

//An easy way to map the XY location is a lookup table.
//You'll need to fill this in - you might need to figure out
//if mirrored etc or something funky.
// TODO: map the appropriate characters.
// buttons is a global 2 dimensional array - 4 rows by 3 columns
char buttons[4][3] = {{'1', '2', '3'},
    {'4', '5', '6'},
    {'7', '8', '9'},
    {'*', '0', '#'}
};

/*******************************************************************************
***
This function activates each row one by one, and reads the column.
If a button is pressed, returns the character pressed.  Otherwise, returns a zero.
Use main_test_1() to test this function.
This function must be completed by the students. - finished and working properly!
********************************************************************************
**/
char get_button(void)
{
    //Use the scanning example from main_test_1()

    char x;

    for(int row = 0; row < 4; row++)
    {
        set_row_low(row); // sets one row at a time low as the for loop progresses
        _delay_ms(20); // This delay function amounts to the fact that it takes 80 ms to go through
the entire loop to check each row

        // delay gives time to verify if a given column was shorted

        int col = col_pushed(); // determine if a column is pushed (or shorted) - returns 0 if no
buttons pushed or 1,2,3 depending on column

      if(col)  // prints this message as long as 0 is not returned
      {
         return x = buttons[row][col-1]; // returns corresponding button pressed from our
defined array matrix
      }
   }
   return x = 0;
}

/*****************************************************************************
***
This function calls get_button() and makes sure that it is only registered once.
This function is complete.
******************************************************************************
**/
char get_new_button(void)
{
   static char last_button;
   char b = get_button();

   //Check if we held button down
   if(b == last_button) return 0;

   last_button = b;

   return b;
}

void master_function(char * pin, char * user1, char * user2, char * user3, char * user4, int
*ptr_sheetspercyc, char * ptr_n1, char * ptr_n2, char * ptr_n3, char * ptr_n4)
{

   fprintf_P(fio_0, PSTR("\nWelcome Master User: %s\n\n"), pin);
   fprintf_P(fio_0, PSTR("Please press a digit from the options below to configure your T.P.T.:
\n\n"));
   fprintf_P(fio_0, PSTR("Press 1 to reset user #1's entry code or 5 to reset sheet count \n"));
   fprintf_P(fio_0, PSTR("Press 2 to reset user #2's entry code or 6 to reset sheet count \n"));
   fprintf_P(fio_0, PSTR("Press 3 to reset user #3's entry code or 7 to reset sheet count \n"));
   fprintf_P(fio_0, PSTR("Press 4 to reset user #4's entry code or 8 to reset sheet count \n"));
fprintf_P(fio_0, PSTR("Press 9 to display all user's current sheet total \n"));
fprintf_P(fio_0, PSTR("Press * to reset the number of sheets dispensed per cycle \n"));
fprintf_P(fio_0, PSTR("Press 0 to exit the Master User Function \n\n"));

while (1)
{
    char control = get_new_button();
    if (control == '9')   // display all sheet count totals
    {
        fprintf_P(fio_0, PSTR("User #1's current sheet count is: %d\n"), *ptr_n1);
        fprintf_P(fio_0, PSTR("User #2's current sheet count is: %d\n"), *ptr_n2);
        fprintf_P(fio_0, PSTR("User #3's current sheet count is: %d\n"), *ptr_n3);
        fprintf_P(fio_0, PSTR("User #4's current sheet count is: %d\n\n"), *ptr_n4);
    }

    if (control == '1')   // reset user1 code
    {
        int w = 0; // counter
        fprintf_P(fio_0, PSTR("Please Enter User1's New Pin:\n"));
        fprintf_P(fio_0, PSTR("Press # to reset your current partial pin entry \n"));
        while(1)
        {
            char a = get_new_button();
            if(a == '#')
            {
                w = 0;
                continue;
            }
            if(a)
            {
                user1[w] = a;
                w++;
            }
            if(w >= 4)
            {
                user1[4] = 0;
                fprintf_P(fio_0, PSTR("User1's New Pin: %s\n\n"), user1);
                break;
            }
        }
    }
```

```
if (control == '5')
{
    *ptr_n1 = 0;
    fprintf_P(fio_0, PSTR("User1's toilet paper sheet count has been reset.\n\n"));
}
if (control == '2')   // Reset user2 pin
{
    int x = 0; // counter
    fprintf_P(fio_0, PSTR("Please Enter User2's New Pin:\n"));
    fprintf_P(fio_0, PSTR("Press # to reset your current partial pin entry \n"));
    while(1)
    {
        char c = get_new_button();
        if(c == '#')
        {
            x = 0;
            continue;
        }
        if(c)
        {
            user2[x] = c;
            x++;
        }
        if(x >= 4)
        {
            user2[4] = 0;
            fprintf_P(fio_0, PSTR("User2's New Pin: %s\n\n"), user2);
            break;
        }
    }
}
if (control == '6')
{
    *ptr_n2 = 0;
    fprintf_P(fio_0, PSTR("User2's toilet paper sheet count has been reset.\n\n"));
}
if (control == '3')   // User 3 Pin reset
{
    int y = 0; // counter
    fprintf_P(fio_0, PSTR("Please Enter User3's New Pin:\n"));
    fprintf_P(fio_0, PSTR("Press # to reset your current partial pin entry \n"));
```

```
        while(1)
        {
            char d = get_new_button();
            if(d == '#')
            {
                y = 0;
                continue;
            }
            if(d)
            {
                user3[y] = d;
                y++;
            }
            if(y >= 4)
            {
                user3[4] = 0;
                fprintf_P(fio_0, PSTR("User3's New Pin: %s\n\n"), user3);
                break;
            }
        }
    }
    if (control == '7')
    {
        *ptr_n3 = 0;
        fprintf_P(fio_0, PSTR("User3's toilet paper sheet count has been reset.\n\n"));
    }
    if (control == '4')   // Reset user4 pin
    {
        int z = 0;
        fprintf_P(fio_0, PSTR("Please Enter User4's New Pin:\n"));
        fprintf_P(fio_0, PSTR("Press # to reset your current partial pin entry \n"));
        while(1)
        {
            char e = get_new_button();
            if(e == '#')
            {
                z = 0;
                continue;
            }
            if(e)
            {
                    user4[z] = e;
                    z++;
                }
                if(z >= 4)
                {
                    user4[4] = 0;
                    fprintf_P(fio_0, PSTR("User4's New Pin: %s\n\n"), user4);
                    break;
                }
            }
        }
        if (control == '8')
        {
            *ptr_n4 = 0;
            fprintf_P(fio_0, PSTR("User4's toilet paper sheet count has been reset.\n\n"));
        }
        if (control == '*')
        {
            fprintf_P(fio_0, PSTR("Please enter the new amount of sheets per cycle: \n"));
            while (1)
            {
                char tempButton = get_new_button();
                if (tempButton)
                {
                    *ptr_sheetspercyc = tempButton - 48;
                    fprintf_P(fio_0, PSTR("The new toilet paper cycle amount is: %d \n\n"),
*ptr_sheetspercyc);
                    break;
                }
            }
        }
        if (control == '0')   // Break statement to exit the master while loop
        {
            fprintf_P(fio_0, PSTR("\nYou have exited the master function!\n"));
            break;
        }
    }
}

/**************************************************************
 Speaker Function
```

```
****************************************************************/

typedef struct s_note
{
    uint8_t OCRxn; // 0..255
    uint8_t N;
} note_t;

typedef struct s_octave
{
    note_t note_C;
    note_t note_CS;
    note_t note_D;
    note_t note_DS;
    note_t note_E;
    note_t note_F;
    note_t note_FS;
    note_t note_G;
    note_t note_GS;
    note_t note_A;
    note_t note_AS;
    note_t note_B;
} octave_t;

PROGMEM const octave_t octaves[8] =
{
    {
        // octave 0
        .note_C = {142, N_256}, // 16.35 Hz
        .note_CS = {134, N_256}, // 17.32 Hz
        .note_D = {127, N_256}, // 18.35 Hz
        .note_DS = {120, N_256}, // 19.45 Hz
        .note_E = {113, N_256}, // 20.60 Hz
        .note_F = {106, N_256}, // 21.83 Hz
        .note_FS = {100, N_256}, // 23.12 Hz
        .note_G = {95, N_256}, // 24.50 Hz
        .note_GS = {89, N_256}, // 25.96 Hz
        .note_A = {84, N_256}, // 27.50 Hz
        .note_AS = {79, N_256}, // 29.14 Hz
        .note_B = {75, N_256} // 30.87 Hz
    },
    {
        // octave 1
        .note_C = {71, N_256}, // 32.70 Hz
        .note_CS = {67, N_256}, // 34.65 Hz
        .note_D = {63, N_256}, // 36.71 Hz
        .note_DS = {59, N_256}, // 38.89 Hz
        .note_E = {56, N_256}, // 41.20 Hz
        .note_F = {53, N_256}, // 43.65 Hz
        .note_FS = {50, N_256}, // 46.25 Hz
        .note_G = {47, N_256}, // 49.00 Hz
        .note_GS = {44, N_256}, // 51.91 Hz
        .note_A = {42, N_256}, // 55.00 Hz
        .note_AS = {39, N_256}, // 58.27 Hz
        .note_B = {37, N_256} // 61.74 Hz
    },
    {
        // octave 2
        .note_C = {142, N_64}, // 65.41 Hz
        .note_CS = {134, N_64}, // 69.30 Hz
        .note_D = {127, N_64}, // 73.42 Hz
        .note_DS = {120, N_64}, // 77.78 Hz
        .note_E = {113, N_64}, // 82.41 Hz
        .note_F = {106, N_64}, // 87.31 Hz
        .note_FS = {100, N_64}, // 92.50 Hz
        .note_G = {95, N_64}, // 98.00 Hz
        .note_GS = {89, N_64}, // 103.83 Hz
        .note_A = {84, N_64}, // 110.00 Hz
        .note_AS = {79, N_64}, // 116.54 Hz
        .note_B = {75, N_64} // 123.47 Hz
    },
    {
        // octave 3
        .note_C = {71, N_64}, // 130.81 Hz
        .note_CS = {67, N_64}, // 138.59 Hz
        .note_D = {63, N_64}, // 146.83 Hz
        .note_DS = {59, N_64}, // 155.56 Hz
        .note_E = {56, N_64}, // 164.81 Hz
        .note_F = {53, N_64}, // 174.61 Hz
        .note_FS = {50, N_64}, // 185.00 Hz
        .note_G = {47, N_64}, // 196.00 Hz
        .note_GS = {44, N_64}, // 207.65 Hz
```

```
        .note_A = {42, N_64}, // 220.00 Hz
        .note_AS = {39, N_64}, // 233.08 Hz
        .note_B = {37, N_64} // 246.94 Hz
    },
    {
        // octave 4
        .note_C = {35, N_64}, // 261.63 Hz
        .note_CS = {33, N_64}, // 277.18 Hz
        .note_D = {31, N_64}, // 293.66 Hz
        .note_DS = {29, N_64}, // 311.13 Hz
        .note_E = {27, N_64}, // 329.63 Hz
        .note_F = {26, N_64}, // 349.23 Hz
        .note_FS = {24, N_64}, // 369.99 Hz
        .note_G = {23, N_64}, // 392.00 Hz
        .note_GS = {22, N_64}, // 415.30 Hz
        .note_A = {20, N_64}, // 440.00 Hz
        .note_AS = {19, N_64}, // 466.16 Hz
        .note_B = {18, N_64} // 493.88 Hz
    },
    {
        // octave 5
        .note_C = {142, N_8}, // 523.25 Hz
        .note_CS = {134, N_8}, // 554.37 Hz
        .note_D = {127, N_8}, // 587.33 Hz
        .note_DS = {120, N_8}, // 622.25 Hz
        .note_E = {113, N_8}, // 659.25 Hz
        .note_F = {106, N_8}, // 349.23 Hz
        .note_FS = {100, N_8}, // 369.99 Hz
        .note_G = {95, N_8}, // 392.00 Hz
        .note_GS = {89, N_8}, // 415.30 Hz
        .note_A = {84, N_8}, // 440.00 Hz
        .note_AS = {79, N_8}, // 466.16 Hz
        .note_B = {75, N_8} // 493.88 Hz
    },
    {
        // octave 6
        .note_C = {71, N_8}, // 1046.50 Hz
        .note_CS = {67, N_8}, // 1108.73 Hz
        .note_D = {63, N_8}, // 1174.66 Hz
        .note_DS = {59, N_8}, // 1244.51 Hz
        .note_E = {56, N_8}, // 1318.51 Hz
        .note_F = {53, N_8}, // 1396.91 Hz
        .note_FS = {50, N_8}, // 1479.98 Hz
        .note_G = {47, N_8}, // 1567.98 Hz
        .note_GS = {44, N_8}, // 1661.22 Hz
        .note_A = {42, N_8}, // 1760.00 Hz
        .note_AS = {39, N_8}, // 1864.66 Hz
        .note_B = {37, N_8} // 1975.53 Hz
    },
    {
        // octave 7
        .note_C = {35, N_8}, // 2093.00 Hz
        .note_CS = {33, N_8}, // 2217.46 Hz
        .note_D = {31, N_8}, // 2349.32 Hz
        .note_DS = {29, N_8}, // 2489.02 Hz
        .note_E = {27, N_8}, // 2637.02 Hz
        .note_F = {26, N_8}, // 2793.83 Hz
        .note_FS = {24, N_8}, // 2959.96 Hz
        .note_G = {23, N_8}, // 3135.96 Hz
        .note_GS = {22, N_8}, // 3322.44 Hz
        .note_A = {20, N_8}, // 3520.00 Hz
        .note_AS = {19, N_8}, // 3729.31 Hz
        .note_B = {18, N_8} // 3951.07 Hz
    }
};

static void
tone(uint8_t octave, uint8_t note)
{
    uint32_t ret;
    note_t *val;
    ret = pgm_read_word_near((uint8_t *)&octaves + sizeof(octave_t) * octave + sizeof(note_t)
* note);
    val = (note_t *)&ret;
    TCCR2A |= (1<<COM2A0);
    TCCR2B = (TCCR2B & ~((1<<CS02)|(1<<CS01)|(1<<CS00))) | val->N;
    OCR2A = val->OCRxn - 1; // set the OCRnx
}

static void
stop(void)
{
    TCCR2A &= ~(1<<COM2A0);
    TCCR2B &= ~((1<<CS02)|(1<<CS01)|(1<<CS00)); // stop the timer
}

/*****************************************************************
Servo Functions
*****************************************************************/

//SERVO POSITION
#define TOP 20000 //Width of a interation
#define M90 2400 //spin one dir
#define P90 400 //spin other dir
#define Z 1400 //Middle (stopped)

unsigned long miliTime=0, count=0;

//TIMER Interrupt
ISR(TIMER0_COMPA_vect)  //The interrupt occurring
{
    miliTime++;
    if(miliTime > 1000)
    {
        count++;
        miliTime = 0;
        //fprintf_P(fio0, PSTR("%d\n\r"), count);
    }
    return;
}

//INITALIZE PWM FOR SERVO & TIMER
void Timer1Init(void)
{
    DDRB = (1<<PORTB1);    // Assign OC1A pin as output.

    // PWM Mode with Phase and Frequency Correction and pre-scale of 1/1.
    TCCR1A = (1<<COM1A1) ; //
    TCCR1B = (1<<CS11) | (1<<WGM13); //8 prescaler //Mode 8: PWM, phase and frequency
correct
    TCNT1 = 0;                          // Zero Timer.
    ICR1 = TOP;                         // Set TOP resolution.
    OCR1A = Z;                          // Set initial Pulse Width.
```

```
   //TIMER Initialize
   TCCR0A = (1<<WGM01); //Set to CTC mode
   OCR0A  = 250; //# of of ticks to match
   TIMSK0 = (1<<OCIE0A); //Sets interrupt
   sei(); //set external interrupt (i bit)

   TCCR0B = (1<<CS01) | (1<<CS00); //start at 64 prescaler clk/64 bitD = 011
}

//Servo dispensing function
int servo(int sheets)
{
   float sec = sheets / 2; //Math to convert from sheets to seconds
   double sElapsed = 0;
   double sInitialTime = count;
   fprintf_P(fio_0, PSTR("\nDispensing...\n\r"));

   while(sec > sElapsed)
   {
      if(button_function() == 1)
      {
         OCR1A = Z;
         return 1;
      }
      sElapsed = count - sInitialTime;
      OCR1A = M90; //M90 or P90 depending on final direction///////////////
   }
   OCR1A = Z;
   fprintf_P(fio_0, PSTR("DONE\n\r"));
   //_delay_ms(100);
   return 0;
}

/*******************************************************************
PIR Motion Sensor Functions:
*******************************************************************/

int Motion_Sensor()
{
    PORTC |=(1<<PINC0);  //configuring PortC pin 0 as input
    double mElapsed = 0;
    int sec = 120;
    double mInitialTime = count;

    while(sec > mElapsed)
    {
        mElapsed = count - mInitialTime;
        if (PINC & (1<<PINC0))
        {
            fprintf_P(fio_0,PSTR("Motion detected"));
            return 1;
        }
        if (get_new_button() == '0')
        {
            break;
        }
    }
    return 0;
}

/**************************************************************************
***
Activate run out alarm
***************************************************************************
**/

void alarmActivate(void)
{
    fprintf_P(fio_0, PSTR("Please replace roll\n"));

    /* setup */
    DDRB |= (1 << PORTB3); // set BUZZER pin as OUTPUT
    TCCR2A |= (1<<WGM21); // set timer mode to Fast PWM
    TCCR2A |= (1<<COM2A0); // connect PWM pin to Channel A of Timer0

    tone(1, 9); // A
    _delay_ms(1000);
    tone(1, 0); // C
    _delay_ms(1000);
    tone(1, 7); // G
    _delay_ms(1000);
    tone(1, 5); // F
    _delay_ms(1000);
    stop();
    _delay_ms(1000);
}


/******************************************************************************
***
Main Function
******************************************************************************
**/
int main(void)
{
    Timer1Init();
    init_hardware();
    init_uart0(103); // initialization
    // booting message
    fprintf_P(fio_0, PSTR("System Booted, built %s on %s\n\r"), __TIME__, __DATE__);
    fprintf_P(fio_0, PSTR("Press # to restart your current pin entry \n"));
    fprintf_P(fio_0, PSTR("Please enter your four digit pin: \n"));

    int sheetspercyc = 5; // number of sheets defined for a given dispense cycle

    char pin[MXN]; // define a string as a vector of characters

    // Set up strings to store the pins for the following users!

    char user1[MXN] = "1111";
    char n1 = 0; // sheet count for user 1

    char user2[MXN] = "2222";
    char n2 = 0; // sheet count for user 2

    char user3[MXN] = "3333";
    char n3 = 0; // sheet count for user 3

    char user4[MXN] = "4444";
    char n4 = 0; // sheet count for user 4
```

```
   char masteruser[MXN] = "2203";

   int i = 0;

   while(1)
   {
      //RUN THIS AT TOP OF WHILE LOOP
      if(count > 100000)
      {
         count = 0;
      }

      char b = get_new_button();

      //Do something special with "#", for example clear partial entry
      if(b == '#')
      {
         i = 0;
         continue; // the continue statement can be used to ignore the remainder of the loop and
return to the top of the loop!
      }

      if(b)
      {
         pin[i] = b;
         i++;
      }

      if (button_function() == 1)
      {
         alarmActivate();
      }

      if(i >= 4)  // if code enters this if statement, a four digit pin has pin entered!
      {
         pin[4] = 0; // terminate the string with a 0. Always end a string with the null character!

         fprintf_P(fio_0, PSTR("Entered PIN: %s\n"), pin);

         if (strcmp(pin, masteruser) == 0)   // strcmp returns 0 if both strings are identical -
THIS IS THE MASTER USER FUNCTION
```

```
          {

             master_function(pin, user1, user2, user3, user4, &sheetspercyc, &n1, &n2, &n3,
&n4);

          }

          if (strcmp(pin, user1) == 0)   // strcmp returns 0 if both strings are identical - user1
          {

             fprintf_P(fio_0, PSTR("\nWelcome User #1: %s\n"), pin);
             fprintf_P(fio_0, PSTR("User #1's current sheet count is: %d\n"), n1);
             fprintf_P(fio_0, PSTR("Press 0 when your session is complete\n\n"));
             // separate function with a while loop that lasts a certain amount of time, if 1 is
returned on the ultrasonic function, calls servo function to dispense cycle
             // increments nx for given user.

             while(1)
             {
                int val = Motion_Sensor();

                if (val == 1)   // ultrasonic function returns 1 if hand is sensed
                {
                   int alarmTemp = servo(sheetspercyc);
                   if(alarmTemp == 1)
                   {
                      break;
                   }
                   n1 = n1 + sheetspercyc; // updates number of sheets;
                }
                else if (val == 0)
                {

                   fprintf_P(fio_0, PSTR("User #1's current sheet count is: %d\n"), n1);
                   fprintf_P(fio_0, PSTR("User #1 is finished\n\n"));

                   //fprintf_P(fio_0, PSTR("Please enter a new pin: \n"));
                   break;
                }
             }
          }
```

```
if (strcmp(pin, user2) == 0)   // strcmp returns 0 if both strings are identical - user2
{

    fprintf_P(fio_0, PSTR("\nWelcome User #2: %s\n"), pin);
    fprintf_P(fio_0, PSTR("User #2's current sheet count is: %d\n"), n2);
    fprintf_P(fio_0, PSTR("Press 0 when your session is complete\n\n"));

    while(1)
    {
        int val = Motion_Sensor();

        if (val == 1)   // ultrasonic function returns 1 if hand is sensed
        {
            servo(sheetspercyc);
            n2 = n2 + sheetspercyc; // updates number of sheets;
        }
        else if (val == 0)
        {

            fprintf_P(fio_0, PSTR("User #2's current sheet count is: %d\n"), n2);
            fprintf_P(fio_0, PSTR("User #2 is finished\n\n"));

            //fprintf_P(fio_0, PSTR("Please enter a new pin: \n"));
            break;
        }
    }
}

if (strcmp(pin, user3) == 0)   // strcmp returns 0 if both strings are identical - user3
{

    fprintf_P(fio_0, PSTR("\nWelcome User #3: %s\n"), pin);
    fprintf_P(fio_0, PSTR("User #3's current sheet count is: %d\n"), n3);
    fprintf_P(fio_0, PSTR("Press 0 when your session is complete\n\n"));

    while(1)
    {
        int val = Motion_Sensor();

        if (val == 1)   // ultrasonic function returns 1 if hand is sensed
```

```
            {
                servo(sheetspercyc);
                n3 = n3 + sheetspercyc; // updates number of sheets;
            }
            else if (val == 0)
            {

                fprintf_P(fio_0, PSTR("User #3's current sheet count is: %d\n"), n3);
                fprintf_P(fio_0, PSTR("User #3 is finished\n\n"));

                //fprintf_P(fio_0, PSTR("Please enter a new pin: \n"));
                break;
            }
        }
    }

    if (strcmp(pin, user4) == 0)   // strcmp returns 0 if both strings are identical - user4
    {

        fprintf_P(fio_0, PSTR("\nWelcome User #4: %s\n"), pin);
        fprintf_P(fio_0, PSTR("User #4's current sheet count is: %d\n"), n4);
        fprintf_P(fio_0, PSTR("Press 0 when your session is complete\n\n"));

        while(1)
        {
            int val = Motion_Sensor();

            if (val == 1)   // ultrasonic function returns 1 if hand is sensed
            {
                servo(sheetspercyc);
                n4 = n4 + sheetspercyc; // updates number of sheets;
            }
            else if (val == 0)
            {
                fprintf_P(fio_0, PSTR("User #4's current sheet count is: %d\n"), n4);
                fprintf_P(fio_0, PSTR("User #4 is finished\n\n"));

                //fprintf_P(fio_0, PSTR("Please enter a new pin: \n"));
                break;
            }
        }
```

```
            }

            i = 0; // reset the index of the pin array back to the first element (ie pin[0])
        }
    }
    return 0;
}
```