Smart Garage

CS122A: Fall 2017

Xhoni Filo

Table of Contents

Introduction

I have decided to do an Smart Garage because as technology progresses, outdated systems within households will begin to come obsolete and home owners will turn tide to newer, more advanced technologies. The Smart Garage will proposes a solution to existing problems by offering unique ways to automate and advanced technologies within garages.

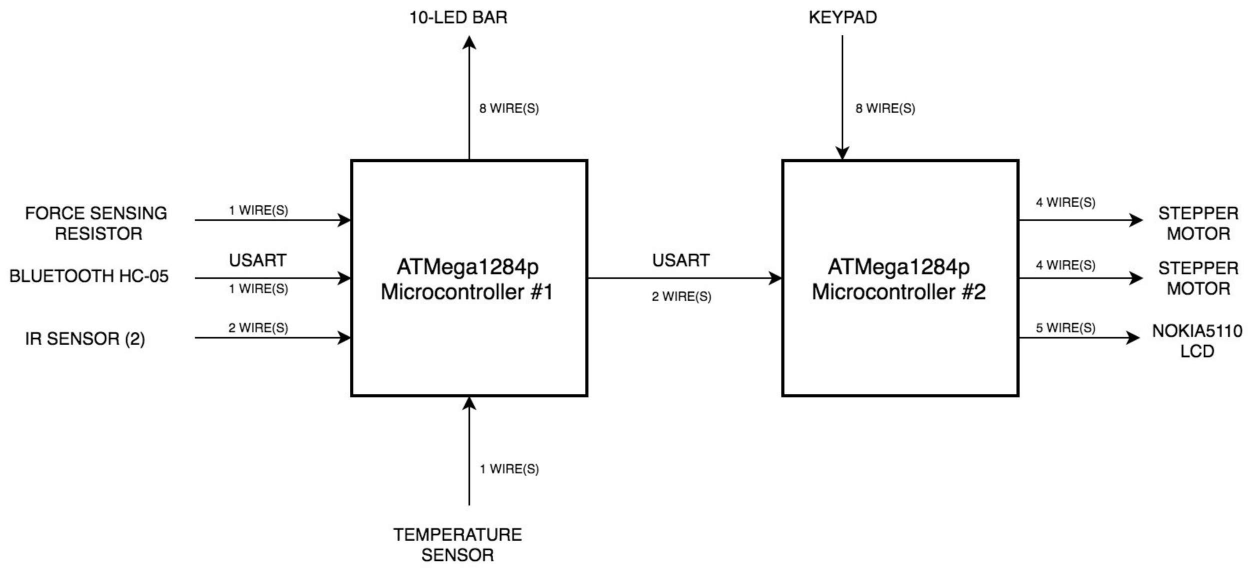
The Smart Garage that I have created can have its logic and components implemented in the majority of households. The reason for this is because it incorporates many different forms of communication between its multiple components. These components include sensors, displays, motors, and communication between microcontrollers and a bluetooth module. An example of intercommunication between components would be a weight sensor transmitting its status to a central hub (Nokia 5110 LCD). The status would show if there is a car sitting in the garage. An infrared sensor can shift the garage movement from closing to opening if the sensor is tripped while closing. Lockdown mode initiated by either the central hub or phone app via bluetooth will not allow the garage to open until lockdown mode is deactivated. The implementation of the logic and components shows how something as simple as a garage could be automated to result in homeowner satisfaction.

Hardware

Parts List

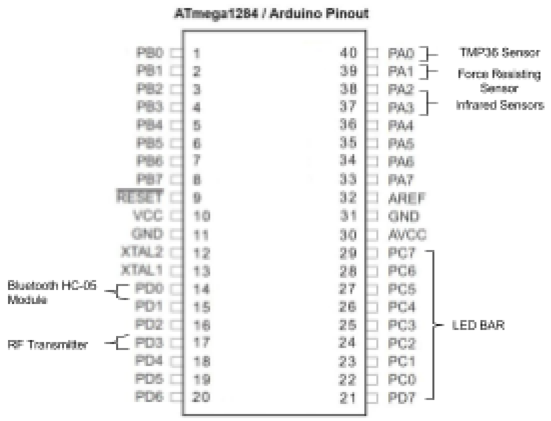
- 2: ATMega1284p microcontroller
- 1: **TMP36 Temperature Sensor**
- 2: **OSOYOO Infrared Sensor (LYSB01I57HIJ0-ELECTRNCS)**
- 1: **Nokia 5110 84x48 Graphic LCD**
- 2: Stepper Motor
- 1: **RF Transmitter**
- 1: **RF Receiver**
- 1: **DSD TECH HC-05 Bluetooth Module**
- 1: Keypad
- 1: 10-LED Bar

Block Diagram

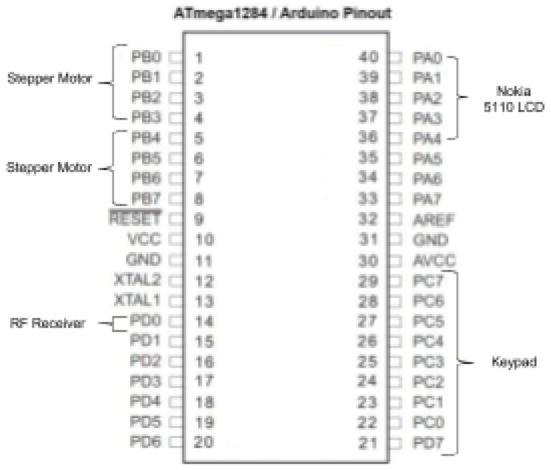


Pinout

<u>ATMega1284p Microcontroller #1:</u>

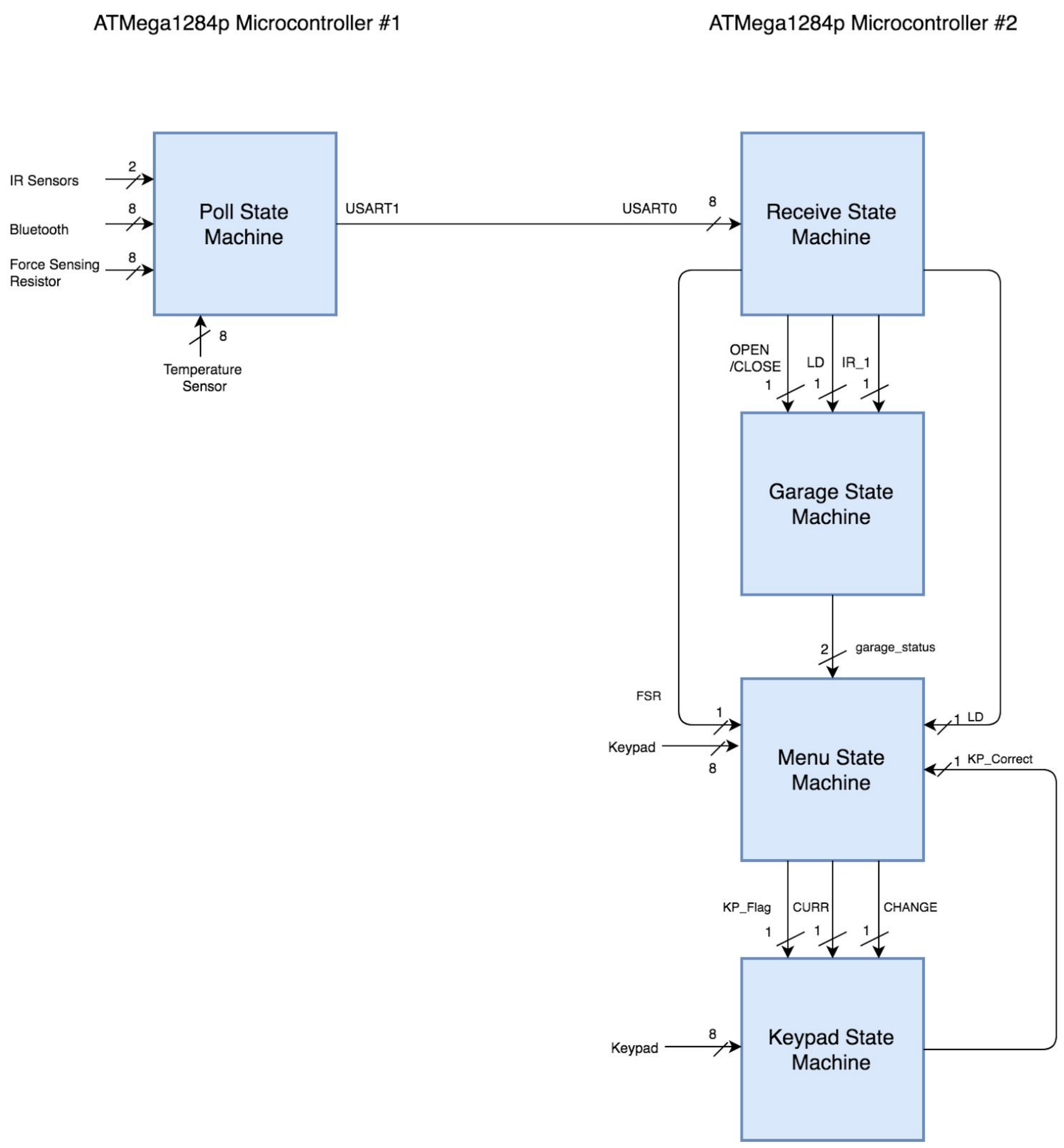


ATMega1284p Microcontroller #2:



Software

The software designed for this project was implemented using the PES standard. The overall design as a task diagram is included below.



ATMega1284p Microcontroller #1:

Polling State Machine: The **Polling** State Machine for the ATMega1284p Microcontroller #1 takes in the inputs of the infrared sensors, the force resisting sensor, temperature sensor, and bluetooth module. The **Polling** State Machine then combines the inputs into 2-8 bit numbers. The first 8-bit number contains the result of the infrared sensors, force resisting sensor, and bluetooth module, as well as the two most significant bits being flag bits (11). The second 8-bit number contains the output of the temperature sensor is degrees celsius, with the two most significant bits being flag bits (00). These 2-8 bit representations are sent over USART0.

ATMega1284p Microcontroller #2:

Keypad State Machine: The **Keypad** State Machine is used as the input between the keypad and the Nokia 5110 LCD screen. The **Keypad** State Machine will wait for its general flag to be raised from the **Display** State Machine and then it will begin grabbing the user input, which is the password. The **Keypad** State Machine will lower the flag which will signal the **Display** State Machine to determine if the password was correct or not, and allow it to move to the appropriate state.

Display State Machine: The **Display** State Machine is used as the central hub for displaying the results sent from the *ATMega1284p Microcontroller #1* and the results from the outputs of *ATMega1284p Microcontroller #2*. The **Display** State Machine consists of three main positions; the *Status Display*, *Change Password Display*, and *Lockdown Display.* The **Display** State Machine is the main interaction between the outputs of both microcontrollers.

Receive State Machine: The **Receive** State Machine is used to receive the USART data from USART0. The **Receive** State Machine will receive the data from USART0 and will mask out the bits appropriately. If the received data two most significant bits are 00, it will store the least six significant bits into the temperature variable. If the received data two most significant bits are 11, it will mask and store the received data into the appropriate variables; IR1, IR2, FSR, and BT.

Garage State Machine: The **Garage** State Machine is used to control the movement of the stepper motors, based on the data received extracted within the **Receive** State Machine. The **Garage** State Machine consists of four states; Open, Opened, Close, Closed. These four different states interact with received data differently. For example, if we are in the *Open* State or *Opened* State, if *Lockdown* is initiated, the states will transition to *Close* and cannot be accessed again until the *Lockdown* is removed.

Implementation Reflection

The project for CS122A was a lot of fun. I learned a lot of new information regarding the new components I used, such as the Temperature Sensor, Nokia 5110 LCD, RF Transmitter & Receiver, etc. I am glad I chose the project that I did because I can begin to implement the knowledge I have gained into the systems within my own household. The project was a significant upgrade from CS120B, for all of the right reasons. The process of the project was set up perfectly. The addition of the milestone was a plus because it kept me on track to complete my project. The proposed project point scale is also a positive because it allowed me to prioritize my scheduling efforts in regards to the project completion.

If I were to do this project again, I would integrate bluetooth communication between the microcontrollers because it is more stable than RF communication. I would build a mock garage which would hold the breadboard and components together. I would use more powerful stepper motors for the garage movement. The current stepper motors rotate at low speeds due to their power consumption and its use with other components on the same power source. What I like best about my project is that it implemented multiple components with complex logic into a garage that can perform various functions. I like how the wiring of my project looks simple and clean, but yet it is powerful with the internal logic that allows the Smart Garage to function.

Milestone

As for the milestone, I had mine planned out for November 16th, 2017 (Week 7). When I completed my milestone, I was done with my 70-80 point project. I had implemented the state machines for the garage, the polling of the first microcontroller, and the display of the Hitachi LCD. The milestone was perfect for my development timeline because it allowed me enough time to test all of my components and build the basic implementation of my smart garage.

Completed components

From the thre e proposed project scales, I completed my 90-100 point project. I was able to integrate the base project (70 - 80) with the Bluetooth HC-05 Module (80 - 90), as well as the Nokia 5110 LCD Display (90 - 100).

Incomplete components

I **was able to complete all** the components that I had stated in my project proposal. However, I wanted to add additional components to my project, but I was not able to get the components function properly 100% of the time. These components included the RF Transmitter and RF Receiver. The components demonstrated potential early on as reliable components. The soldering of the antenna increased the connectivity between the two module. But, due to cheap module design, the RF Transmitter and Receiver soon began to act faulty. At a point in time, if the modules were functioning properly, **any** physical interaction with either of the two antennas on the modules would result in incomplete/incorrect data transmission. The nuance of minimal physical interaction leading to errors strayed me away from perfecting the use of the RF Transmitter and Receiver. In the long run, these two components were not proposed, but rather simply for educational purposes.

Youtube Links

Make sure they are publicly viewable!

- Short video:
- Longer video:

Testing

To test the overall project, I went to three different individuals. The first individual was my cousin who recently graduated with his degree in biomedical engineering. He tested the project initially and found no obvious bugs with it. The second individual I went to was my brother, a high school student. He found two different bugs within program. The bugs were acknowledged immediately and they were solved within minutes. The third individual I went to was my girlfriend, a liberal studies major at a public university. She found an additional bug. The bugs were addressed immediately and fixed within minutes.

TMP36 Temperature Sensor

To test the TMP36 Temperature Sensor, I would print the result of its readings to a 10-LED Bar which would demonstrate that the conversion from ADC to degree celsius was working properly.

Test Case(s):

```
ADC_T = ADC * (5000 / 1024);
ADC_T = ((ADC_T - 500) / 10);

PORTA = ADC_T;
// PORTA would print the temperature in degrees celsius
```

Bluetooth HC-05 Module

To test the Bluetooth HC-05 Module, I would print the result of its input to a 10-LED Bar which would demonstrate that the correct bits were being transmitted via the usart connection into the bluetooth. If I had sent a *0x01* via bluetooth, the 10-LED bar should display *0x01*.

Test Case(s):

```
// Assuming we are sending 0x01
If usart_hasreceived(0) {
       PORTA = usart_receive(0); }
// PORTA would print 0x01
```

OSOYOO Infrared Sensor

To test the OSOYOO Infrared Sensor, I would print out the result of its output to a 10-LED Bar which would demonstrate whether or not the Infrared Sensor was sending the appropriate result.

Test Case(s):

```
// Assuming we hover our hand above the IR sensor
IR  = !(PINA & 0x01);
If (IR) {
       PORTA = 0xFF; }
// PORTA would print 0xFF
```

Keypad

To test the Keypad, I would print out the result of key that was being pressed, on the 10-LED Bar.

```
// Assuming we press ‘A’ on the keypad
key = GetKeyPadKey();
If (key == 'A') { PORTA = 0x01; } // PORTA will output 0x01 which is connected to the LED Bar.
```

Nokia 5110 84x48 Graphic LCD

To test the Nokia 5110, I would print out a series of characters to the display using the library functions. This characters would include inputs from the keypad or hard coded characters using simply for testing. To test for decimal values, I would convert the decimal values into their respective character representations then output the result to the Nokia 5110.

Test Case(s):

```
nokia_lcd_init();
nokia_lcd_clear();
nokia_lcd_writestring("Hello World", 1);
nokia_lcd_render();
// The Nokia 5110 LCD Display would display "Hello World" in size 1 font.
```

RF Transmitter & Receiver

To test the RF Transmitter & Receiver, I would send the values, that were extracted from a **working and tested** Bluetooth module, over USART0 on *ATMega1284p Microcontroller #1* and print out the received data on *ATMega1284p Microcontroller #2*.

Test Case(s):

ATMega1284p Microcontroller #1:

```
// For example, assume we sent 0x01 via bluetooth into USART0 of the first microcontroller
If (usart_hasreceived(0)) {
       bluetooth = usart_receive(0); }
// We will take the value of bluetooth (0x01) and send it over USART1 which connects to USART0
of the second microcontroller
If (usart_issendready(1)) {
       usart_send(bluetooth, 1); }
```

ATMega1284p Microcontroller #2:

```
// Once we receive the value on the second microcontroller, have PORTA output the value
// received (should be 0x01)
If (usart_hasreceived(0)) {
       PORTA = usart_receive(0); }
```

Stepper Motor

To test the stepper motor, I installed two buttons that would control the direction of the motor. If **BTN1 is being pressed**, the motor will spin counter-clockwise. If **BTN2 is being pressed**, the motor will spin clockwise. If **both BTN's are being pressed**, the motor will not both. If both **BTN's are not being pressed**, the motor will not move.

Test Case(s):

```
If ((BTN1 && BTN2) || (!BTN1 && !BTN2)) { continue; }
// cnt will be decreased in a state
Else If (BTN1) { PORTA = arr[cnt % 8]; }
// cnt will be increased in a state
Else If (BTN2) { PORTA = arr[cnt % 8]; }
```

Known Bugs

Temperature Value

There is a bug when the temperature value is greater than 96 degree celsius. The temperature will not be updated if the temperature exceeds 96 degree celsius.

- The cause of this bug is due to the implementation of **Polling** State Machine of the *ATMega1284p Microcontroller #1*. The SM sends two different 8-bit values over the same usart line; the compressed data as one 8-bit value and the temperature as another 8-bit value. The two most significant bits on each 8-bit values are used as flags. The two most significant bits for the compressed data are 11. The compressed data consists of the infrared sensor readings, force resisting sensor reading, and bluetooth input. The two most significant bits for the temperature data are 00. Therefore, if the temperature were to raise over a value above 96 degree celsius, the two most significant bits of the temperature data would mimic the two most significant bits of the compressed data. Therefore, incorrect data would be transmitted, resulting in incorrect flags being set on the *ATMega1284p Microcontroller #2*.
- To debug this error, I have tried using different flags and functions to differentiate between the two different 8-bit data transmissions, but I have been unsuccessful. As a temporary solution, I have set any result of temperature over 96 degree celsius to be converted to 0 degree celsius. The *ATMega1284p Microcontroller #2* on the receiving end will recognize 0 degree celsius as "extreme" temperature. This solution is not the correct fix to the issue, that is why it is listed as a bug.
- To address the Temperature Value bug if I continued to work on the project, I would need to limit my temperature sensor internally to only read values under 96 degree celsius.
- I have not implemented this bug as part of a feature.

Bluetooth / Garage Interaction

There is a bug when transitioning states within the **Garage** State Machine. The **Garage** State Machine has a built in feature that will result in a transition from the *Close* State to the *Open* State if the infrared sensor sends a **high** value while the garage is closing (*Close* State). However, once the infrared sensor sends a **low** value, the **Garage** State Machine will automatically transition back to the *Close* State, if the user is sending ***Close*** via the bluetooth application.

- The cause of this bug could be due to the Bluetooth HC-05 Module implementation. The Bluetooth HC-05 Module repeatedly sends values until the bluetooth is disconnected. For example, if I sent *0x01* via bluetooth, as long as the input value doesn't not deviate from *0x01* or the module becomes disconnected, *0x01* will be sent continuously.
- To debug, I have tried to use flags as well as checks to only grab the input if it is different than the last value sent.

- To address the Bluetooth / Garage Interaction bug if I continued to work on the project, I would have to design and develop a personal application that only will trigger the bluetooth module to stop sending values repeatedly, once the appropriate value was sent.
- I **have** implemented this bug as a feature. For the Smart Garage, if the infrared sensor sends a high value while the garage is closing, the garage will open completely before closing again. This can be sold as a safety feature to protect automobiles within the garage.

Keypad Input

There is a bug when entering the current password for the Smart Garage. When entering the current password, if you press and hold on a key, and click on any other key in the same row as the intended key, it will pick up the extra key presses as inputs.

- The cause of this bug is due to the design of the current keypads that were offered to CS122A students. The reason I am assuming it is a hardware issue is because no other keys will be taken as inputs, unless if they are in the same row as the key being pressed.
- To debug, I have tried to edit *keypad.h*. I have tried to rewrite the library for keypad.h using different functions.
- To address the Keypad Input bug if I continued to work on the project, I would have to implement checks and flags to ensure keys other than the key being pressed would not affect the intended result of the system.
- I have not implemented this bug as part of a feature.

Resume/Curriculum Vitae (CV) Blurb

Smart Garage Embedded Implementation *2017*

- Implemented and mimicked automated garage systems using ATMega1284p microcontroller, in relation with Embedded C Programming.
- Integrated multiple components to communicate with one another.
- Programmed logic that allows multiple amounts of 8-bit data to be sent over the same USART line.
- Developed logic that has not been implemented in modern home garages with hopes to spark thought about future garage automation.

Future work

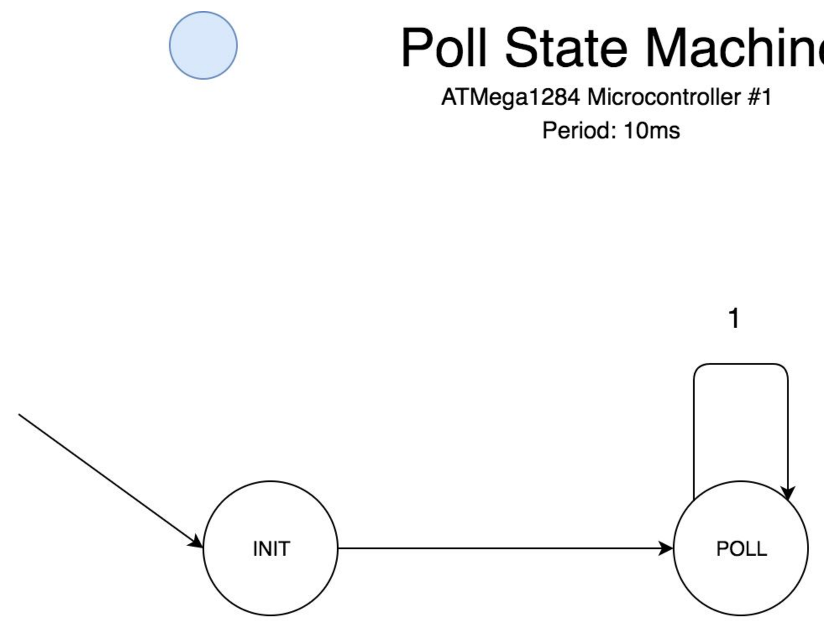
In the case that I continue to build upon and develop this project even further, I would add a few additional features. Firstly, I would implement a photoresistor into the project development. This photoresistor will let me know whether there is daylight or not. This information could be used to advance the logic of the Smart Garage. For example, if there is no daylight and the garage has been open for more than thirty-minutes with no activity, the garage will begin to automatically close. This protects the homeowners from theft if they possibly left their garage open. Another component to implement would be to implement an indoor garage security system using Raspberry Pi. The Raspberry Pi would be connected to the WiFi network of the household and the camera could only be viewed if you are a member of the household. This could be verified using a certain password that would allow you to access the camera. As for design purposes, I would indeed design a case. My wish would be to 3D print a mock garage for the project. The garage would be printed in two layers. The two layers would allow me to wire the components out of the view of the garage. The case would contain multiple inserts that would allow me to place the infrared sensors, the temperature sensor, weight sensor, and stepper motors. Custom PCB board is always a good idea in terms of IOT devices, so I think creating a custom PCB board would allow me to compact my components which in turn would allow for a cleaner looking project.

References

Nokia 5110 84x48 Graphic LCD Library:
https://github.com/LittleBuster/avr-nokia5110

TMP36 Temperature Sensor Conversion:
https://learn.adafruit.com/tmp36-temperature-sensor/using-a-temp-sensor

Appendix



Receive input from both IR Sensors
Receive input from Temperature Sensor
Receive input from Force Resisting Sensor
ADC conversion for both Temperature & Force Resisting Sensor

Receive input from Bluetooth module via USART0

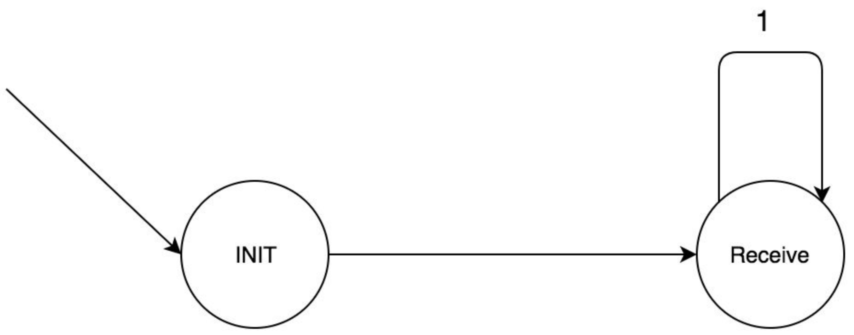
Compress bits from Bluetooth, Force Resisting Sensor, and
both IR Sensors, along with 0xC0 as a flag
Send compressed bits over USART1
Send temperature data over USART1

Output compressed data on PORTC to show what values are being
sent over USART1

Receive State Machine

ATMega1284 Microcontroller #2

Period: 1ms

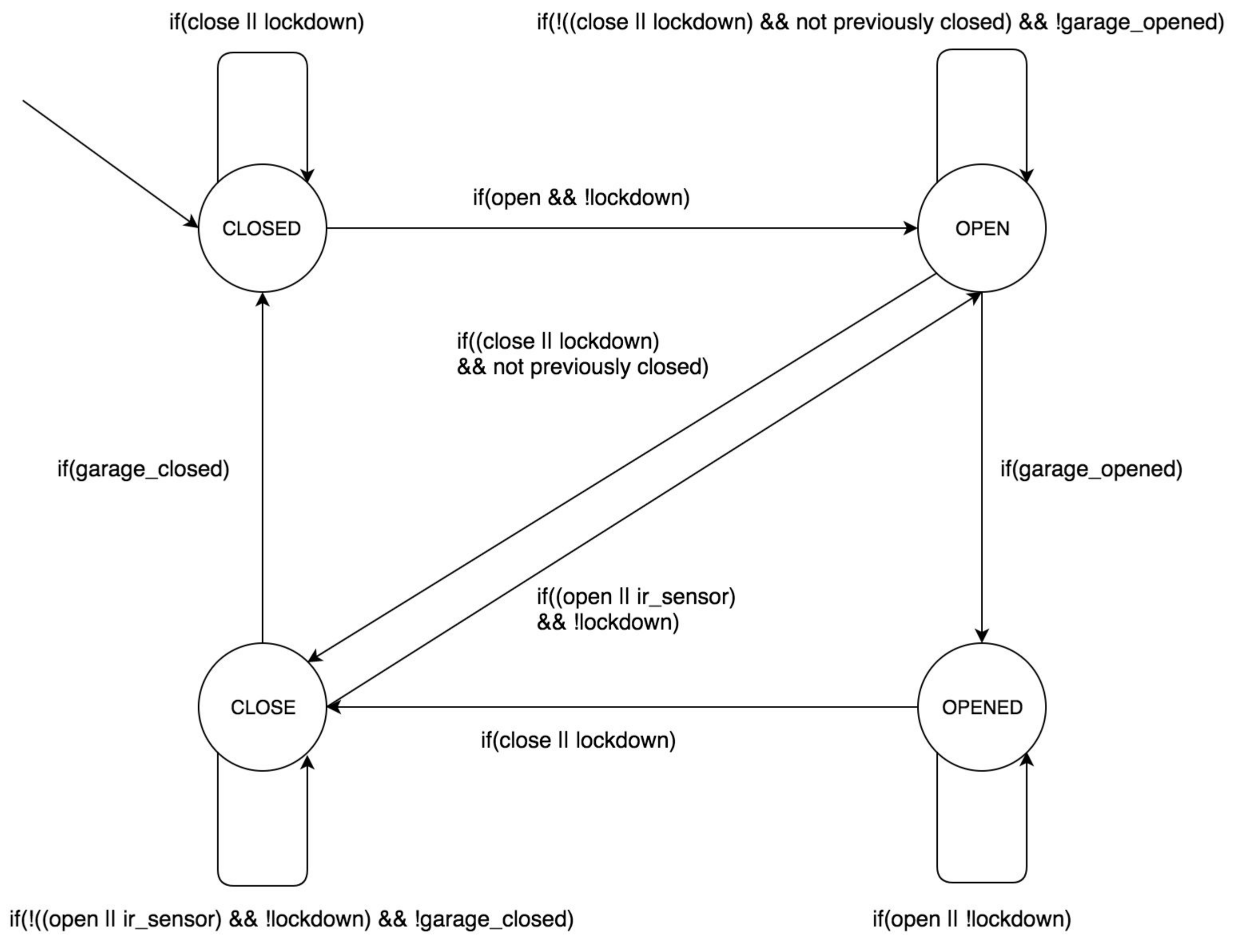


Determines data received over USART0
Based on data received, will set variables.
If data received is the compressed data, will mask the data and assign the appropriate bits to the appropriate variables.
If data received is the temperature reading, will set temperature variable to received data.
Receive State error checks bluetooth since bluetooth module sends repeated inputs; will not update values unless there was a previous change.

Garage State Machine

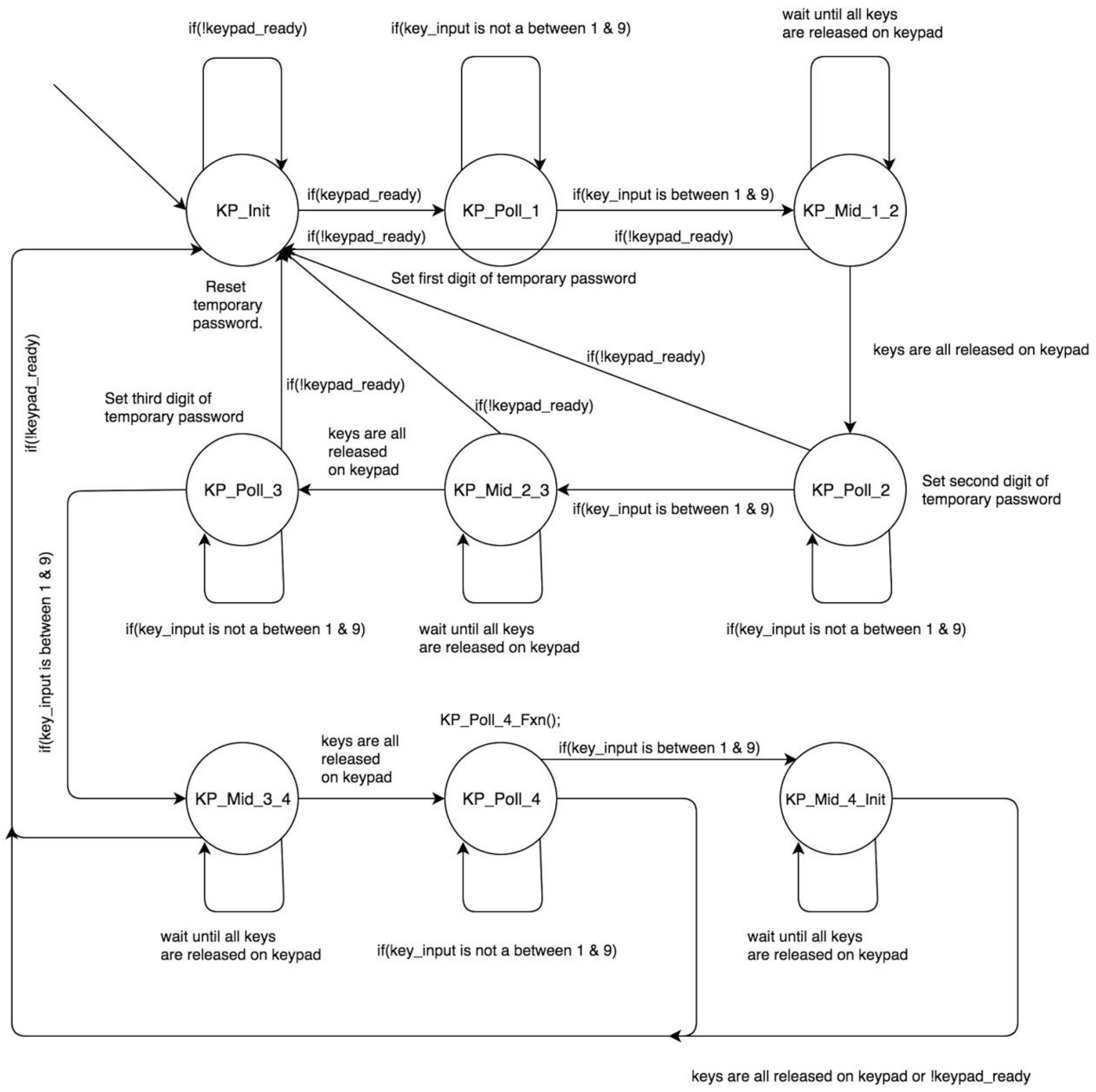
ATMega1284 Microcontroller #2

Period: 3ms



Keypad State Machine

ATMega1284 Microcontroller #2
Period: 1ms



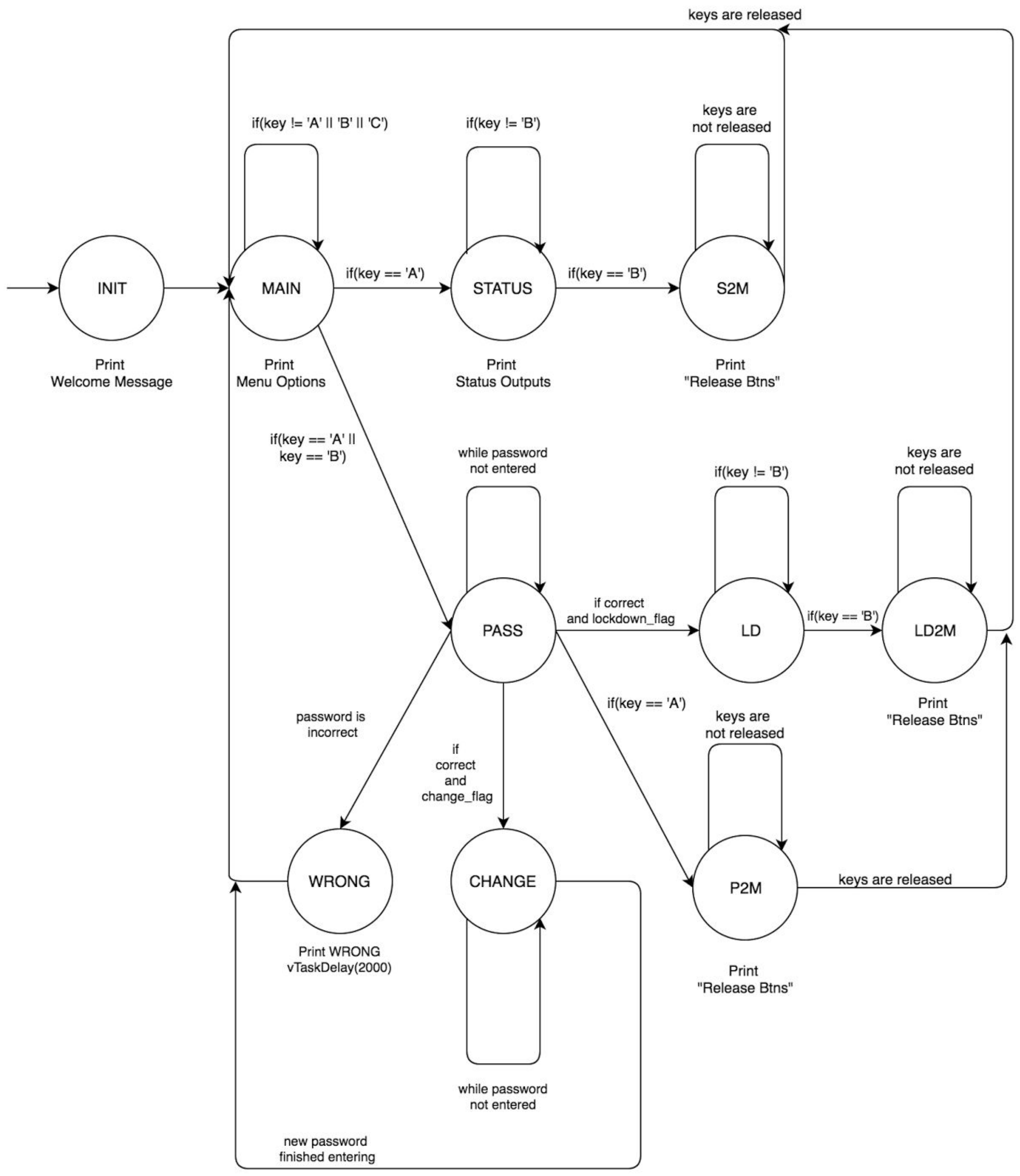
if(!keypad_ready)
if(key_input is not a between 1 & 9)
wait until all keys are released on keypad
KP_Init
if(keypad_ready)
KP_Poll_1
if(key_input is between 1 & 9)
KP_Mid_1_2
if(!keypad_ready)
if(!keypad_ready)
Set first digit of temporary password
Reset temporary password.
keys are all released on keypad
if(!keypad_ready)
if(!keypad_ready)
if(!keypad_ready)
if(!keypad_ready)
Set third digit of temporary password
keys are all released on keypad
KP_Poll_3
KP_Mid_2_3
KP_Poll_2
Set second digit of temporary password
if(key_input is between 1 & 9)
if(key_input is between 1 & 9)
if(key_input is not a between 1 & 9)
wait until all keys are released on keypad
if(key_input is not a between 1 & 9)
KP_Poll_4_Fxn();
keys are all released on keypad
if(key_input is between 1 & 9)
KP_Mid_3_4
KP_Poll_4
KP_Mid_4_Init
wait until all keys are released on keypad
if(key_input is not a between 1 & 9)
wait until all keys are released on keypad
keys are all released on keypad or !keypad_ready

KP_Poll_4_Fxn() {
Set fourth digit of temporary password.
Determine if we are checking for current password or changing the password
If we are checking password, check the temporary password compared to the password stored in eeprom.
If we are changing password, store password in memory 0, memory 1, memory 2, and memory 3 in eeprom }

Menu State Machine

ATMega1284 Microcontroller #2

Period: 1ms



keys are released
if(key != 'A' || 'B' || 'C')
if(key != 'B')
keys are not released
INIT
MAIN
if(key == 'A')
STATUS
if(key == 'B')
S2M
Print Welcome Message
Print Menu Options
Print Status Outputs
Print "Release Btns"
if(key == 'A' || key == 'B')
while password not entered
if(key != 'B')
keys are not released
PASS
if correct and lockdown_flag
LD
if(key == 'B')
LD2M
if(key == 'A')
Print "Release Btns"
password is incorrect
keys are not released
if correct and change_flag
WRONG
CHANGE
P2M
keys are released
Print WRONG vTaskDelay(2000)
Print "Release Btns"
while password not entered
new password finished entering