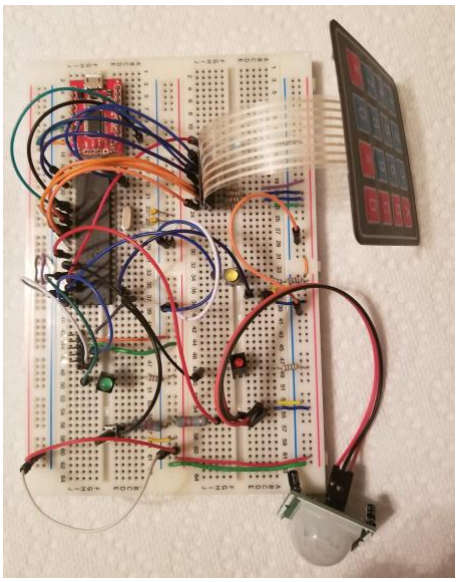
Building a PIC18F4520 Standalone Alarm System with EUSART Communication



JANUARY 30, 2018

PRABESH KHADKA

Alarm System Description

This standalone hardware/firmware Alarm system consisting of embedded PIC18F4520 microchip, serial communication device, PIR motion sensor, temperature sensor, LEDs, resistor, capacitor, push button reset and 4x4 keypad connected together with wires in a bread board.

The alarm system can detect motion and trigger a temperature alarm. It incorporates a PIR motion sensor that is running throughout the program with a high priority interrupt running in the background. The Red LED turns on whenever a motion is detected. On the other hand, temperature sensor (TMP36) driven by timer interrupt measures the current temperature and displays it to the user and also triggers the alarm once the current temperature is equal to or greater than the provided threshold temperature. The Yellow LED toggles as it gets a new temperature reading and remains solid whenever the temperature alarm is triggered. It is driven by a low priority interrupt stored at the memory of the microchip.

FT232RL is used to create a bi-directional serial communication between the PIC and the PC. It also provides a common ground and also powers the PIC through the VCC output. A terminal on a PC can be used to interact with the output the message coming from the PIC and to communicate with it. External frequency of 20MHz is provided to the PIC through an external crystal oscillator. The Green LED turns on as system has started successfully and message coming from the PIC is displayed in the terminal of the PIC.

The alarm system also allows user to select a programmed 4x4 matrix keypad as the input method to the PIC. The blue LED turns on whenever the embedded keypad is selected as method of input. Another feature of this alarm system includes a four-digit Passcode which helps to safely log in to the main menu. The passcode is required each time the interrupts triggers or each time the MCLR Reset button is pressed. User is able to set the passcode as the system is initially programmed using the PICkit3 device. User is also able to change the current passcode through the menu option. If the MCLR Reset button is pushed, all the previous states are saved in PIC's EEPROM and remains the same as the system restarts.

Overview of the Process

In order to design this alarm system a proper planning was required. After carefully reading the instruction for the project, a schematic was created that gave the idea of how each component had to be connected into the breadboard. Using the schematic, every component was carefully connected together in the board using wires. The LEDs were connected to the ground using a 150 ohms resistor. The PIC18F4520 was connected to the FT232RL using four pins (Rx, Tx, VCC and GND) in order to transfer data and power up the PIC. The PIR sensor and Push Button Reset was connected to the PIC using 8.2K ohms resistors. Similarly, four 10K ohms resistors were used while connecting the keypad to the pic. Capacitors were used with the oscillator and temperature sensor. The connection was tested in order to avoid any short-circuits.

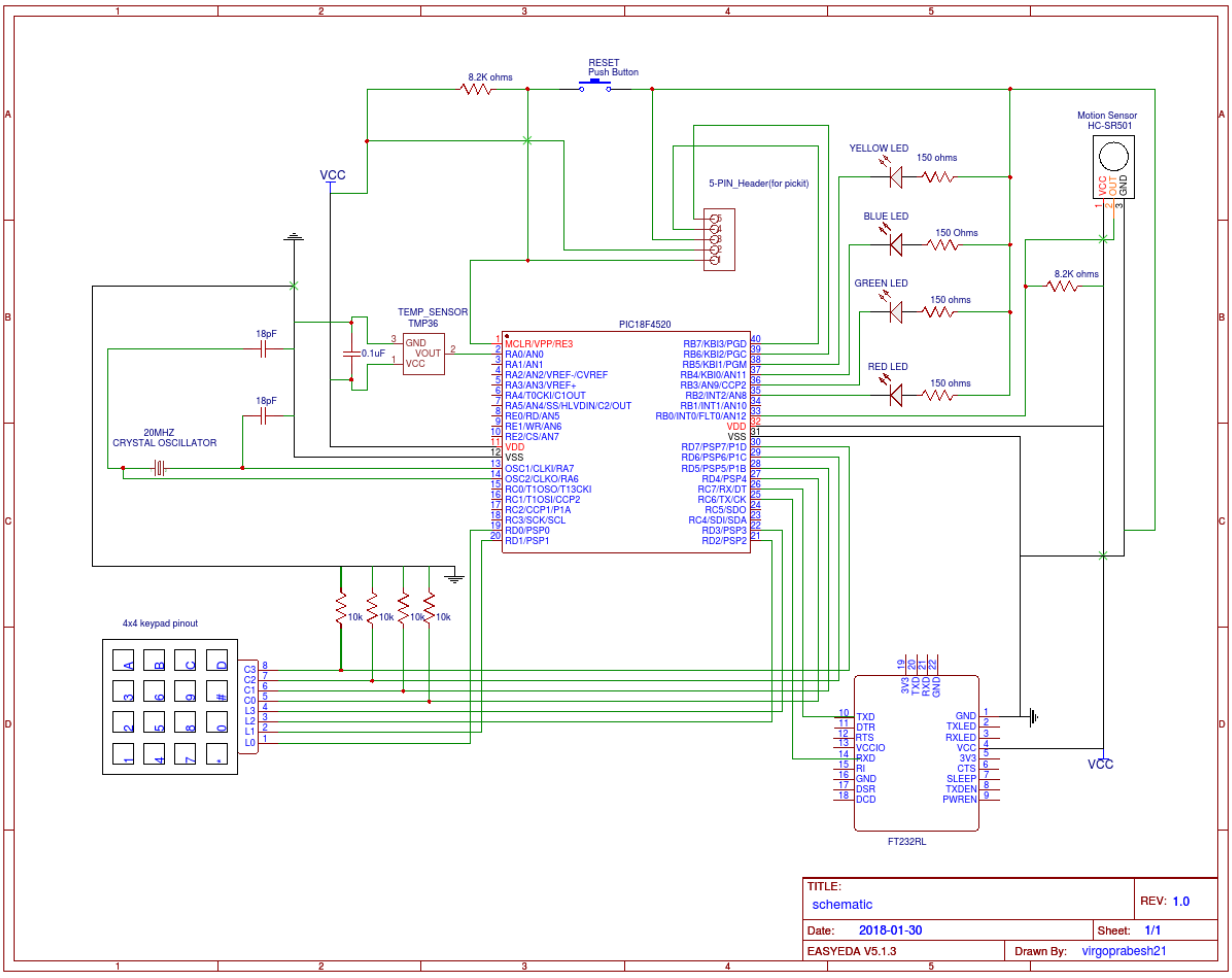


Fig: Schematic of the Hardware Connection

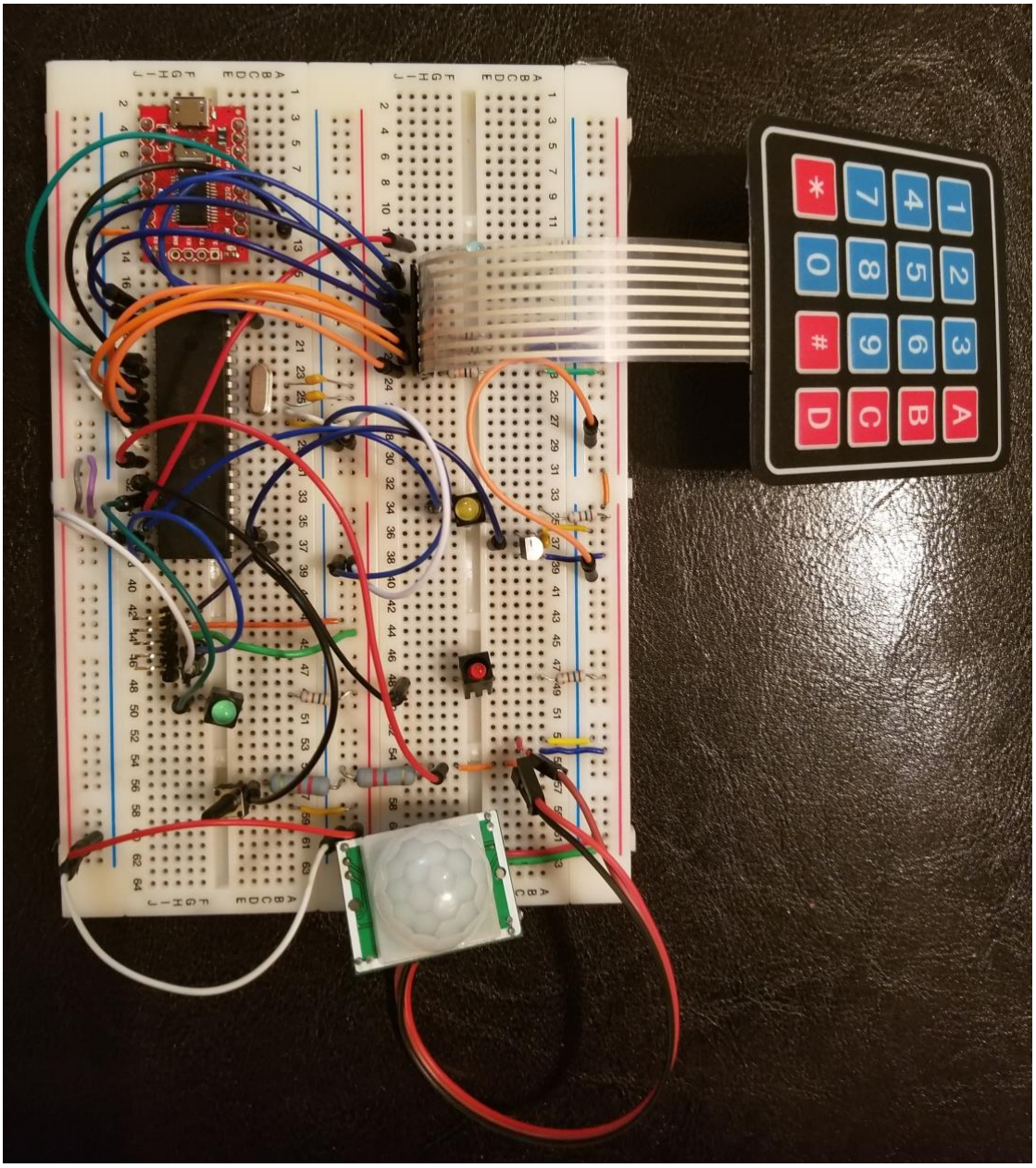


Fig: Hardware implementation on the Bread board

After completing the hardware design, the source code for system was written using the MPLAB X IDE in the PC and compiled using XC8 compiler. Small features were implemented and then it was programmed in the alarm system's PIC using the PICkit3 device for testing. The PIC's

EEPROM was used to store important states of the system. Debugging was done to make sure the feature is implemented properly before moving onto another feature. The serial communication was emulated in the terminal of the PC. Interrupts were used to trigger the alarm and ADC feature was used to measure and calculate the temperature. Finally, complete system was created and tested as whole in order to verify the complete functionality of the system.

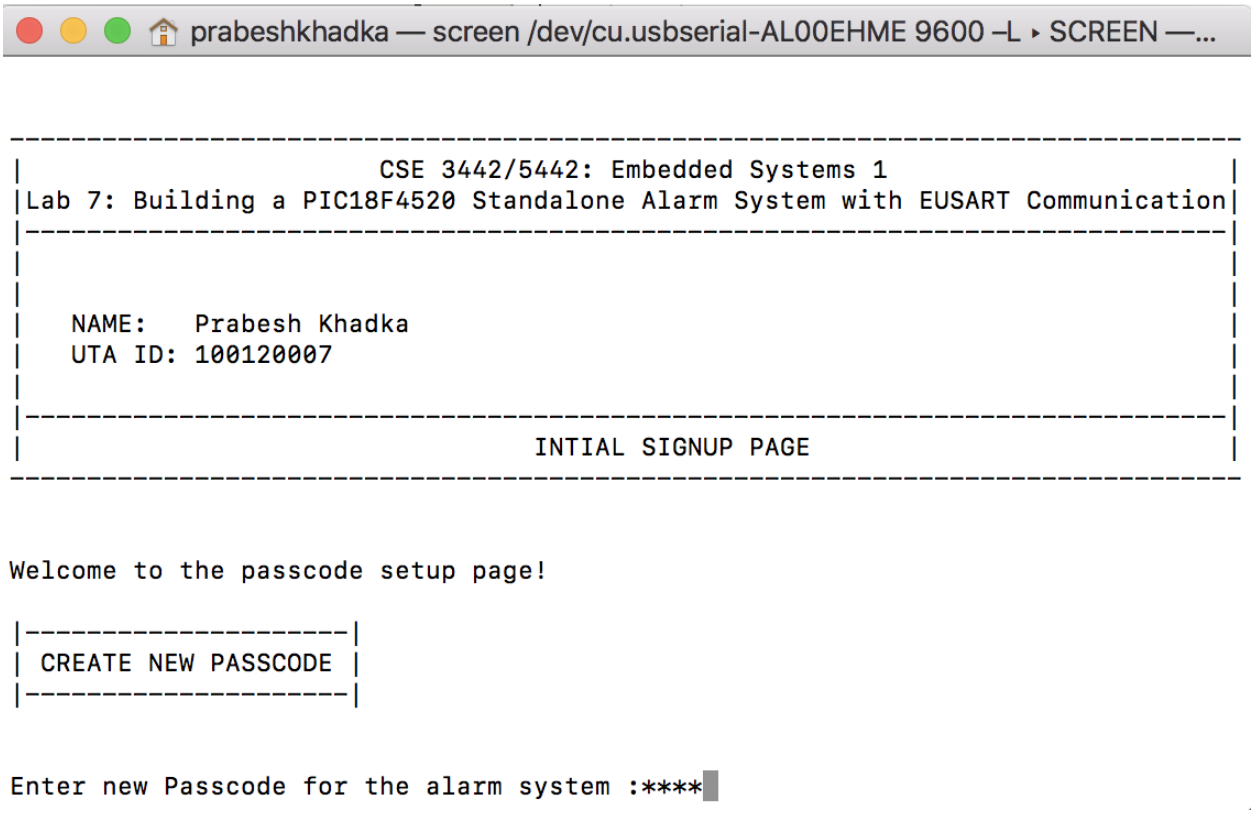


Fig: Preview of the User Interface

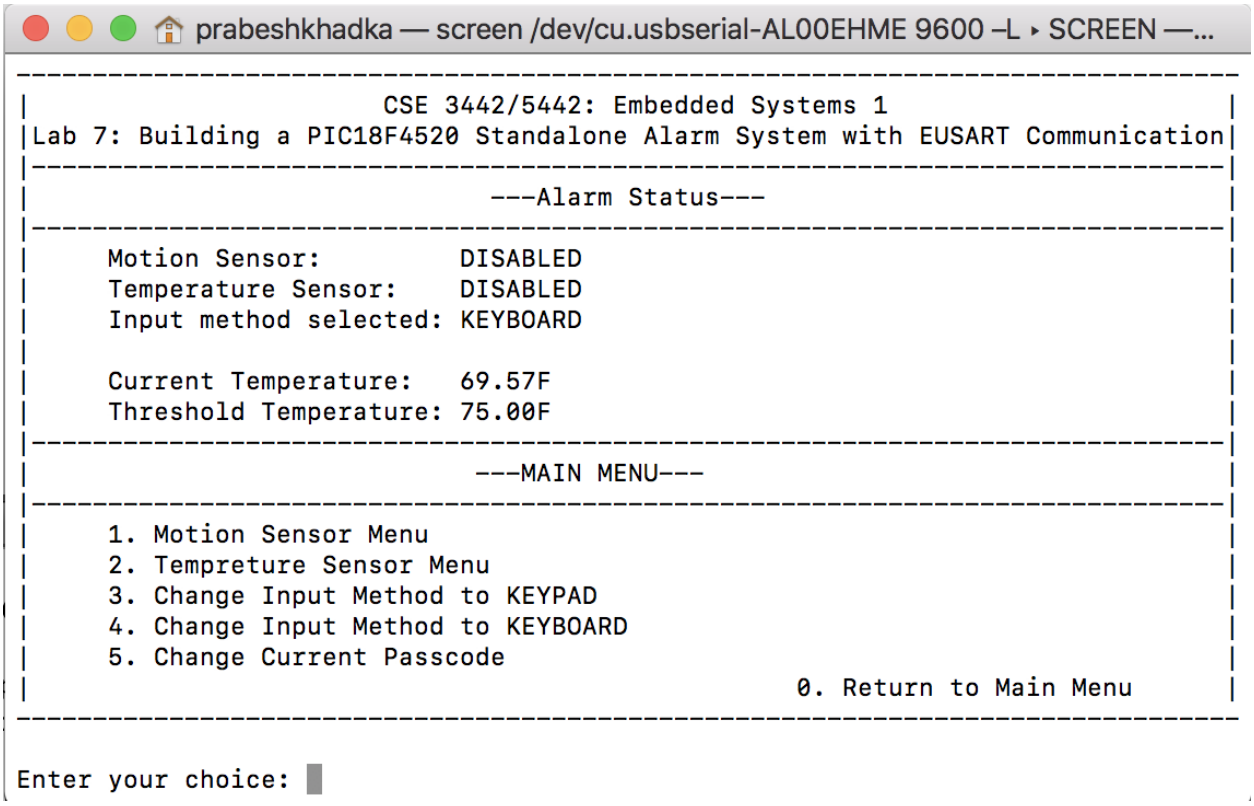


Fig: Preview of the Alarm System's Main Menu

Detailed Description of PIC Interrupts

PIC interrupts handle the alarm triggering process. The proper bits settings for the interrupts are executed in the program as soon as the user turns on the alarm function. The interrupts are disabled by default and user can also disable the interrupts using the menu function. The detection of motion is handled by using high priority interrupt function of the PIC. Whenever turned on, the high priority interrupt function constantly checks if the INT0 Interrupt Flag is set high due to the interrupt caused by the PIR sensor. If found high, the Red LED is turned ON and user is redirect

to a function which prompts user to enter passcode and reset the alarm. The Interrupt Flag resets and the interrupt function checks for interrupts again.

Similarly, the low priority function handles the temperature alarm. The ADC is used for periodic sampling of the temperature reading. Once, the temperature alarm is enabled, the low priority function check for TMR0 flag is set high. If it is high, it stops TMR0 and gets the current temperature reading and starts the timer again. The Yellow LED toggles as it gets new temperature. Meanwhile, the low priority function also checks for ADC completion through ADIF flag bit. While set high, it compares the current temperature reading with the threshold temperature. If the current temperature is greater, it turns ON the Yellow LED and redirects the user to the enter passcode and reset the alarm. The function clears the ADIF flag and repeats the process again until turned OFF.

Detailed Description of Timer Modules

This Alarm system uses an external 20MHz Crystal Oscillator connected with the PIC. The proper configuration bits were selected for the oscillator type in order for the interrupt and serial communication to properly function. In the program, the timer settings are initialized as the program executes. The TMR0 Interrupt Flag is cleared initially and the Enable Bit is set high, the Interrupt Priority bit is set as low priority and the TMRL and TMRH bits are properly assigned with the calculated preload values. As the temperature alarm is enabled, the timer module carries out the ADC and displays the current temperature to the user. This interrupt is used to trigger the

temperature threshold alarm. The timer settings are stored in the EEPROM of the pic to prevent form erasing after the MCLR Reset button is pressed.

Calculation for ADC and Timer values

The preload values for the timer was calculate as below:

Time period (d) = 1 sec

Oscillator Frequency (F_{osc}) = 20 MHz

We have, X $= d \times F_{osc}$

$= 1 \times (20/4)$ MHz

= 5 MHz

= 5000000 cycles per oscillator

Using the prescalar 1:128 to fit it into 16-bit TMRO register (0 – 65536)

= 65536 – (5000000/128)

= 65236-39063

= 26473

= 0x6769

Therefore,

TMR0H = 0x67

TMR0L = 0x69

The ADC calculation was done as follows:

The result from the ADC is used and the value of the ADRESH register is taken and the address was shifted 8 bits left and the value of ADRESL was added to it and the result was 10 bits and the

result was stored in a variable named “tempr’. Following calculation are performed to get the current temperature.

Firstly, it was converted into the range of 0 to 5V, subtract the device offset and convert it to degree centigrade,

$$\text{Current temperature} = \frac{\left(\left(\text{tempr}\frac{}{1023}\right)\times 5000\right) - 500}{10} {}^{\circ}\text{C}$$

Finally, convert it to Fahrenheit scale,

Temperature in F = ((temperature in $^{\circ}$C) × 1.8) + 32

Description of Keypad’s Reading Method

The 4x4 keypad is connected to the port D (from port RD0 – RD7). The first four ports are selected as output and the rest four port are selected as input. The first port represents the rows of buttons in the keypad and the remaining bit represents columns of the keypad buttons. A function that continuously checks for the key pressed is created that returns the character pressed using the keypad. The Blue LED is turned on when keypad is used as the input method.

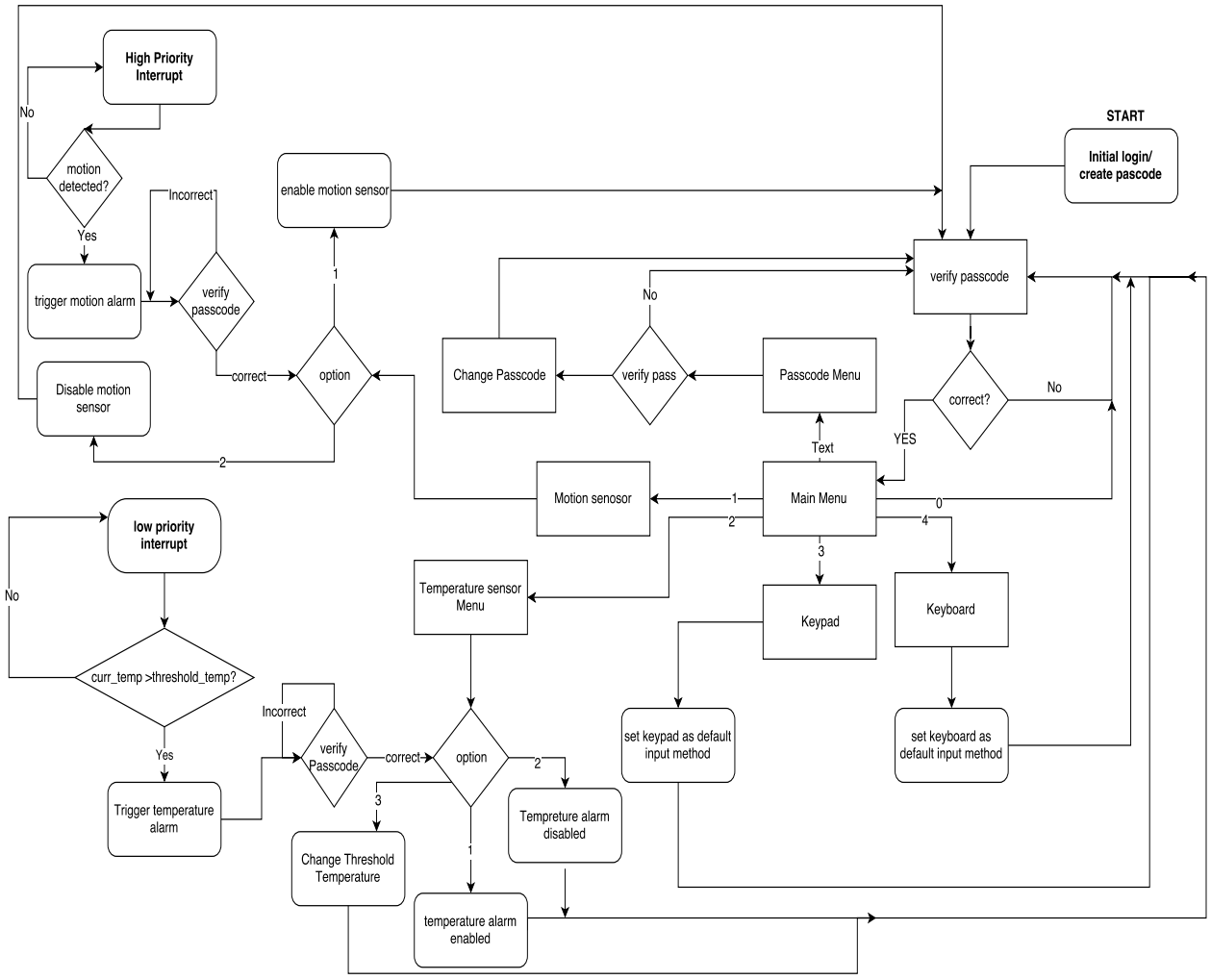
The function works by setting each of the bits of the row’s button high and checking for the port of the column’s button whose bits are also high and then returning the respective character pressed.

For example, RD0, which represent the first rows of buttons (1, 2, 3 and A) is set high and the function checks for high bits in each column. Here, if a user pressed the button in the first column

then the function returns ‘1’, for second column bits found high, it returns ‘2’. For the third column, it returns ‘3’ and for the fourth column it returns ‘A’. Similarly, the second, third and fourth rows are now set high one after another and checked for column bits that are high and then return the respective character. While the function checks for column bits high, it also toggles the blue LED to denote that key can be pressed again in the keypad.

The current state of the input used is stored in the EEPROM of the PIC so that the data is not lost after the MCLR Reset.

Process Flow Chart



Problems Encountered

Various problem had arisen while designing and implementing the alarm system. First of all, mistake in the hardware connection would cause the system to work improperly. The connections

had to be changed and rearranged for properly. Similarly, there was problem in serial communication due to improper baud rate setting. It would generate random garbage values on the terminal while displaying the message coming from the PIC. Similarly, lot of debugging was required to properly implement a functionality into the alarm system. Out of them all, the most difficult problem was the memory management of the PIC. Calling a bunch of printf in the interrupt function caused the program to be oversized in comparison to the device flash memory. This issue was later on resolved by implementing a separate function to print characters to the terminal screen instead of just directly calling the printf function.