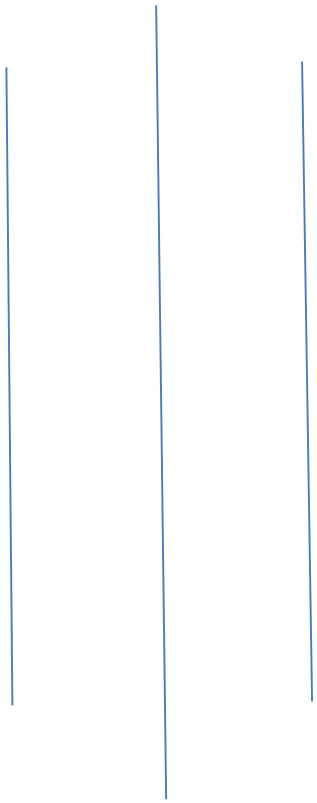
Embedded Systems I

2016 fall-CSE-3442

Instructor: Nicholas Brent Burns

University of Texas Arlington



Lab 7: Building a PIC18F4520 Standalone Alarm System with EUSART Communication

Submitted by: Suman Shrestha

Student ID: 1001162735

Date: 12/15/2016

1. Alarm System Description:

For the final project of Embedded Systems, I, a standalone Alarm System with EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transceiver) communication was built using a PIC18F4520. Communication interface adapter FT232RL was used for USB-to-serial communication. A 20MHz external crystal oscillator is used instead of the PIC18F4520's internal 8 MHz oscillator since communications peripherals are used in the project. This Alarm System consists of a temperature sensor and a PIR motion sensor. Our Standalone Alarm system supported the inputs from both keyboard and 4*4 numeric keypad. Alarm System supports the password authentication functionality for a singe user. When the PIC18F4520 is programmed for the first time, user will be asked to register with a 4-digit passcode. User will then be directed to login screen and asked to input the registered passcode. Upon successful entry, user will be allowed to access the alarm system. PIR motion sensor is interrupt driven and has the highest priority over everything. When the PIR motion sensor is tripped the red LED is turned on. The user is notified and asked to input the passcode to reset the alarm. The temperature sensor (TMP 36) is also interrupt driven and has the 2nd highest priority. Temperature sensor ADC reading is sampled every second using a timer and ADC interrupt. The user has the ability to enable or disable both temperature sensor alarm and PIR motion sensor alarm. Alarm System also provides the user with the functionality of setting and updating the threshold temperature for temperature sensor. A yellow LED represents the status of the temperature sensor. When the temperature sensor alarm is enabled, on acquisition of a new ADC sample received every second, the yellow LED is toggled. And, when the current temperature reaches

over the threshold temperature, the yellow LED turns on. The user is then notified and will be asked to enter the passcode to reset the alarm. The sensors state, input source settings, threshold temperature and pass code is saved in EEPROM so that they can survive the reset of the PIC upon power loss or after a MCLR reset.

2. Process, Planning and Solution:

The designing phase of the project started with reading the lab description provided by the instructor. With the .pdf file, there were several files included in the project folder. Data sheets of all the components used in the Alarm system were studied thoroughly as instructed in the lab description. As per the recommendation, to get some practice of programming the PIC18F452 using Pickit3 was tired. For this, a simple LED circuit was created on the breadboard and the code was written to toggle the LED. Upon success and acquiring the required knowledge of how different components of the Alarm System can be connected to achieve the desired result, the hand drawn schematic was transferred to the breadboard. A prototype of the circuit connection of sample Alarm System from the lab was of great help. Each sensor was connected and tested independently by writing a simple code to test them. I started working on the serial communication once the circuit was completed on breadboard. The lecture slides of how RCREG and TXREG was used as the reference to write the functions to read the input from user and transmit it to the PC using serial communication.

The second phase of building the alarm system was configuring the PIR Motion sensor and Temperature sensor. This was achieved using the datasheet of the components provided with the project description. However, handling the interrupts was not easy. In order to gain a depth

knowledge on interrupts, lecture slides and other online sites were referred. Majority of the time was spent on writing the Interrupt Service Routines and configurations for the sensors. Pre-load values for using the Timer0 was calculated using the procedure learned in the lab 6 so that once the temperature sensor was enabled, it would update the current temperature every one second.

Finally, the major part that was remaining was to get the input from the keypad. The description provided in the lab materials and explanation from the instructor provided a thorough understanding of how do we find which key was pressed. Even though I was struggling to get the input as the key was pressed. I had to hit enter for the character in the keypad to be displayed. This problem was finally fixed after careful observation and various debugging of the code for reading the input.

3. Detailed Description of use of PIR Sensor Interrupt:

PIR Motion Sensor is given he highest priority interrupt over every thing. The alarm pin of the PIR motion sensor is an open-collector, so it is held high. When the PIR motion sensor detects the motion, the pin is pulled to common. This fires up the interrupt and interrupt flag is set high. INT0/RB0 was used for the motion sensor because of its highest priority. Then the program counter enters the Interrupt Service Routine where check is performed for the INT0 Enable bit and the interrupt flag. If both of them are found high, red led will be turned on to notify the user that the motion has been detected. The user then has an option to keep the alarm enabled or disable it. Finally, INT0 interrupt flag was cleared explicitly.

4. Detailed Description of Timer modules:

Temperature sensor is set up as a low priority interrupt, which is generated by TIMER0 and AD. Timer0 is used to create a interval of one second every time the T0CONbots.ADIF flag bit is set high. This is used the get the current temperature to update the user. The calculation of the preload value and pre-scaler was performed was shown below:

* d = 1sec (time period)
* Fosc = 20MHz
* Fin = Fosc/4 = 5 MHz
* 16-bit Timer: 0 - 65,536
* X = d*Fin = 1s*5MHz = 5000000 i.e. 5000000 cycles/ticks occur in a 1s time span
* Using prescaler to bring down X to fit into the 16-bit Timer register(0 - 65,536)
* Using Prescaler of 1:128
* Preload = 65,536 - 5000000/128
* = 65,536 - 39063
* = 26473
* dec = 0x6769

This hexadecimal value is then loaded to TMR0H and TMR0L as shown below:

```
TMR0H = 0x67;

TMR0L = 0x69;
```

5. Timer and Temperature calculation in detail

The updated temperature is received from the ADC every second. The ADC spits out the value in the range of 0-1023. This value was then converted to degree Celsius [10 mV = 1 degree Celsius) and finally to Fahrenheit. The calculation for converting the analog value to digital, handling the offset of out temperature sensor and converting the temperature from Celsius to Fahrenheit is given below:

```
temperature = (double)((temperature/1023)*5000); //converts the ADC result to a
range of 0 to     5000 mV
temperature = temperature - 500;       // subtracts the offset of 500 mV for TMP36
temperature = temperature/10;          // converts the mV to degree celsius [10 mV = 1
degree celsius]
temperature = (temperature)*1.8+32;    // converts degree celsius to farhenheit, [F =
C*1.8 + 32]
```

If the current temperature read by the sensor exceeds the threshold temperature set by the user, ADIF is set high and the steady yellow LED turns on to notify the user that the threshold temperature has been reached. Once, the alarm is triggered, the user will be notified and asked to enter the passcode in order to reset the alarm. With successful passcode, the user will then have an options to keep the alarm enabled or disable it. And then user will hold the threshold temperature if they desire. Also the user is provided with the option to keep the alarm enabled or disable it. Incase the temperature being received every second has not reached the threshold temperature, the yellow LED toggles to notify the user that the temperature sensor is active.

6. Detailed Description of reading from Keypad:

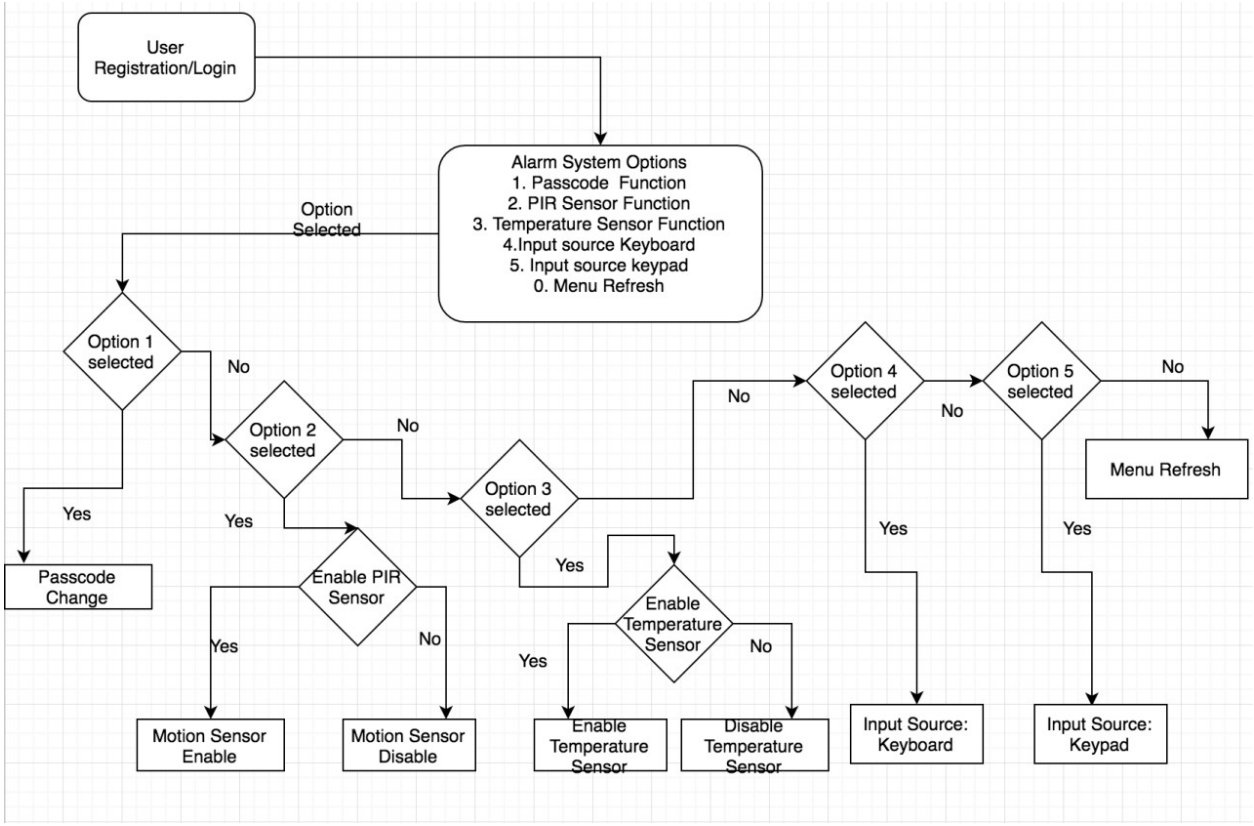
This method takes the input from the 4X4 matrix numeric keypad and stores the input character in a character variable named keypadValue. In order to do so, first of all, all the rows of TRISD ports are set as output and all the 4 columns are set as input. Then, one row of of the keypad is set high (i.e. one of the least significant bits of PORTD). Now, the each

column is checked if it is set high (i.e. Most significant bits of PORTD). Once, the column is found, the element at the corresponding row and column would be the key pressed.

7. Problems Encountered:

1. The sensor alarms would not enable the second time. Too many function calls on Interrupt Service routines kept the program counter always inside the High_ISR.
2. Finding the right serial communication terminal for Macintosh. Tried several like putty and no success. Finally, ZOC terminal came to rescue.
3. Struggled to get the EEPROM header files to perform EEPROM read and write function, which were fixed by writing my own functions to perform the read and write operations to EEPROM.

8. Flow Diagram:



I have learned a lot of things during this project. Especially, the use of interrupt driven service routines. Thus, I think this project should be kept running for coming semesters.