12/15/2016

# Lab 7 Report:

## Implementation of a Multi-Sensor Alarm System with EUSART Communication Using the PIC18F4520 Microcontroller

Shervin O

EMBEDDED SYSTEMS I
CSE 3442 - 001

**Table of Contents**

## Alarm System Description

The goal of this project is to create a multi-sensor alarm system, which is capable of communicating with a computer using Enhanced Universal Asynchronous Receiver/Transmitter (EUSART) protocol. The heart of the alarm is the PIC18F4520 microcontroller unit (MCU), which is manufactured by Microchip. The PIC is a very capable microcontroller family with many built-in feature and peripherals, such as four timer/counter units, analogue to digital converter (ADC), external/internal interrupt handlers, capture/compare/PWM (CCP) and USART modules.

The alarm system has a total of two sensors. A passive infrared (PIR) sensor and a temperature sensor, each of which has a corresponding LED assigned to it. Once engaged, the alarm may be triggered by either one of the sensors, with the PIR sensor having the higher priority. At this time the corresponding LED will light up. The user then needs to input the predefined pass-code to clear the alarm system. Additionally, the threshold for the temperature sensor is user defined and the user can disable or enable each sensor.

Another important feature of the alarm is that the system must remember the user preferences at startup, even after being hard reset, or encountering an unexpected shut-down as a result of losing power. This involves having the MCU's EEPROM store the user preferences. Once turned on, the MCU will retrieve and apply the settings and preferences. The saved parameters include the user pass-code, PIR alarm status, temperature sensor status and threshold as well as the preferred input method. The user should have the option of using the computer keyboard, or the system's 4 by 4 keypad as the method of input. Finally, as a bonus feature, the alarm has a passive buzzer, which is activated using a PWM signal when the PIR sensor is triggered.

## Process Overview

This was a complex project, which involved both knowledge of hardware and programming skills. Initially the requirements document was revived thoroughly to develop a general understanding of the alarm system and its requirements. Then The specific MCU's datasheet was studied to get familiarized with the features and hardware specifications. A spiral approach was taken in completing this project. At each stage of the project, first the specific hardware was wired to the circuit according to the PIC's hardware specifications. Then the code for initializing and configuring the modules involved was written. Finally, the core logic was planned and coded and the functionality was tested. In most cases, some issues were encountered and further adjustments and modifications to the hardware, configuration bits and the core logic was needed.

The first step was to wire up the MCU on the prototyping board and programming it with a test program using the Pic Kit 3 in-circuit programming tool. This code was written in the MPLAB X IDE from Microchip. The test program involved blinking an LED that was connected to one of the MCU's pins. To ensure that excessive current is not drawn through the MCU, a 150-ohm resistor was used. After successfully programming the PIC microcontroller, the master clear reset functionality was implemented. According to the datasheet's guidelines, a push button and appropriate resistors were added to the prototype board. The next step was utilizing the external crystal as the PIC's main clock. The crystal and a small-value capacitor were added to the circuit according to the MCU's datasheet. The appropriate configuration bits were modified to utilize the external clock. After this step, the LED was blinking faster, indicating the higher frequency of the external crystal. Similar procedure was followed to satisfy each of the requirements. While some of the specifications were easy to implement, most of them required careful planning and revisiting the previous steps to implement correctly.

**Interrupts**

The PIC18F4520 is capable of handling high and low priority interrupts. This alarm system utilizes both. The PIR sensor uses a high priority external interrupt. The sensor is connected to the pin 33 (INT0) of the PIC and can trigger a high priority interrupt, which then calls an interrupt service routine (ISR) function to handle the raised flag and take appropriate action. The PIR sensor outputs a pulse when it detects heat, which causes the PIC to set the flag.

The second interrupt is more complex as it utilizes a timer, a CCP module and the ADC module to monitor the temperature periodically and issue an interrupt when needed. The CCP module of the PIC has a special feature where an internal hardware interrupt is generated based on a timer, which can trigger other peripherals. This feature was utilized here to trigger an ADC conversion. First, the CCP2 module was configured for "Compare Special Event Trigger" mode. Then TMR1 module was set up as the time base for the CCP2 module. The CCP2 module automatically clears the timer 1 flag and initiates an ADC conversion when timer 1 overflows. Once the ADC module has been activated by the CCP2 module, it can then generate an interrupt once it completes the ADC conversion. In the ISR, the acquired value can be checked against the user-defined threshold to trigger the alarm if needed. The only hardware connection needed in this case is connecting the temperature sensor's output to an analogue input (in this case AN0). Figure 1 summarizes the interrupt utilization strategy.

| **Hardware/Peripheral:** | **Interrupt utilized:** |
|---|---|
| PIR Sensor | External Interrupt (INT0) – High Priority |
| Temperature Sensor | Internal Interrupt (ADC) – Low Priority |
| ADC | Special Event Trigger (CCP2 & TMR1) |

Figure 1 – Overview of the System's Interrupt Strategy

**Timers**

The PIC18F4520 MCU has a total of four timer/counter modules. For this project two of the timers are utilized. These are TMR1 and TMR2. As discussed in the *Interrupts* section, the timer 1 module was used to generate the base clock for the CCP2 module. This timer was set to 16-bit mode and the pre-scalar was set to 1:8. Timer 2 was used as the base frequency for the passive buzzer unit with a pre-scalar of 1:16. To achieve the PWM, the CCP1 module was configured as a PWM generator and timer 2 was assigned to it. In this case PR2 was set to 255, to achieve a PWM signal with a frequency of about 1.2 KHz:

$$T_{PWM} = \frac{4N(PR1 + 1)}{F_{OSC}} = \frac{4 \times 16 \times (255 + 1)}{20 \times 10^6} = 8.192 \times 10^{-4} s$$

$$F_{PWM} = \frac{1}{T_{PWM}} = \frac{1}{8.192 \times 10^{-4}} = 1.2\ KHz$$

Finally, the value of CCPR1L was set to 20 to achieve a duty cycle of about 8%:

$$CCPR1L = 255 \times 8\% \approx 20$$

The passive buzzer accepts a wide range of frequencies in order to operate properly, therefore the accuracy of the frequency and the duty cycle is not a concern. The duty cycle determines the loudness of the buzzer, as it adjusts the average voltage of the PWM signal. Finally, a resistor is utilized to prevent excessive current going through the PIC as a result of the constant average voltage level.

## Calculations

- Temperature calculations:

The temperature sensor indicates the temperature as voltage level. The temperature and the voltage have a linear relationship. However, the temperature's default unit is Celsius, so it also has to be converted to Fahrenheit. Upon receiving the value from the ADC, the temperature in degree Celsius is calculated first, then this is converted to temperature in degree Fahrenheit and compared with the pre-defined threshold. The steps for converting the ADC value to the temperature in Fahrenheit are listed below.

1. The PIC has a 10-bit ADC so the step size is calculated as follows:

$$Step\ Size = \frac{V_{ref}^{+} - V_{ref}^{-}}{2^{n}} = \frac{5V - 0V}{2^{10}} = 4.8828125\ mV$$

2. Now the actual voltage is calculated:

$$True\ Voltage = step\ size\ \times ADC\ result = \ ADC\ result\ \times\ 4.8828125\ \times 10^{-3}$$

3. The sensor has a scale of 10mV per 1 degree Celsius, so the number of steps in degree Celsius in determined as follows:

$$Steps\ in\ ℃ = \frac{True\ Voltage}{0.01} = \frac{ADC\ result\ \times\ 4.8828125\ \times 10^{-3}}{0.01}$$

$$= ADC\ result\ \times\ 0.0048828125$$

4. The absolute minimum temperature is -55 Celsius so:

$$Temperature\ in\ ℃ = Steps\ in\ ℃ - 55 = (ADC\ result\ \times\ 0.0048828125) - 55$$

5. Finally, the temperature needs to be converted to Fahrenheit for comparison:

$$T°\mathrm{F} = \left(\frac{T℃ \times 9}{5}\right) + 32$$

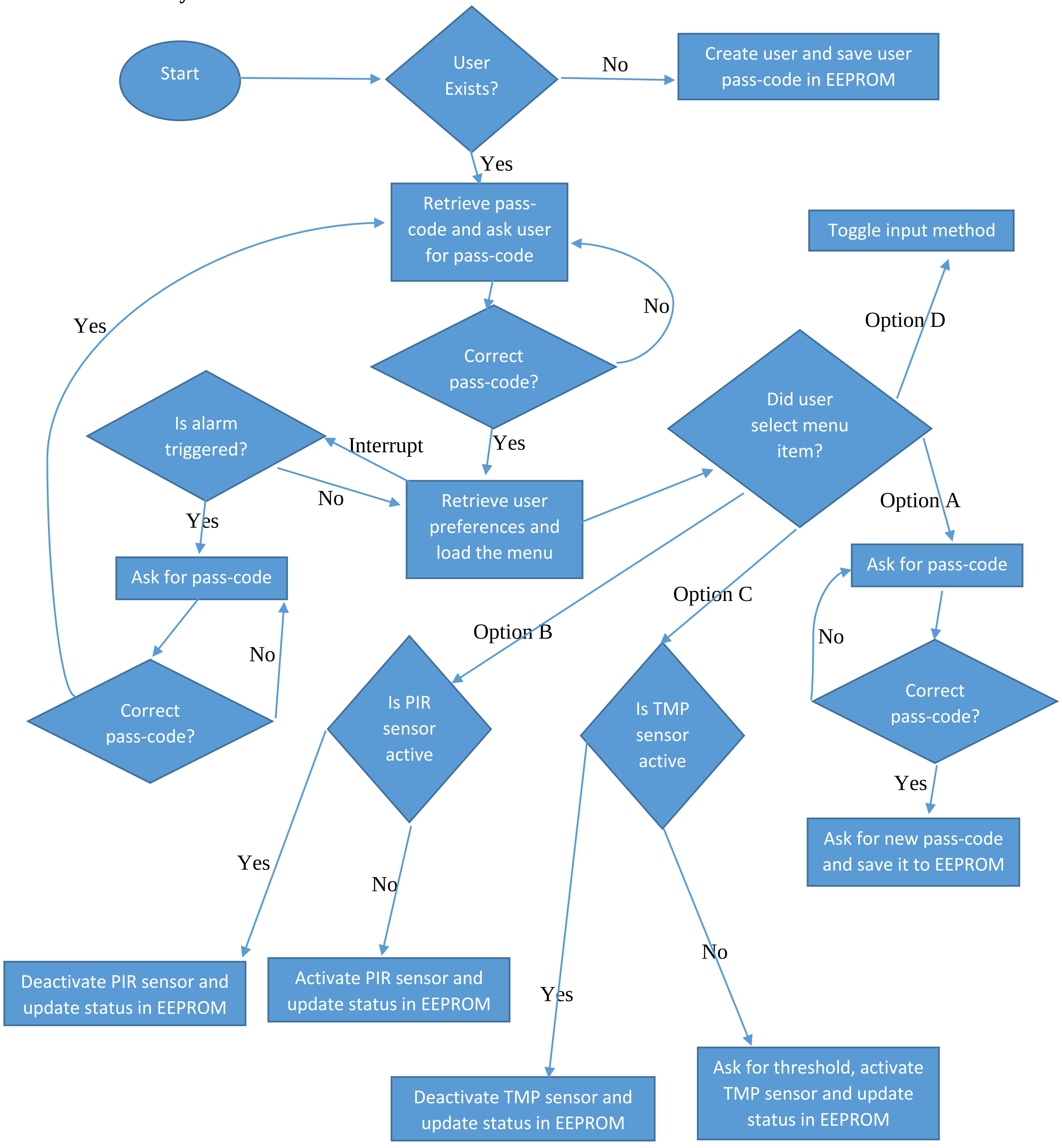
## Utilizing the Keypad

One of the most challenging tasks for this project is reading and translating the input of the 4 by 4 keypad. This number pad has a very simple design, consisting of 16 pushbuttons. Depending on what button is pressed an specific pain of wires are connected to each other. There are a total of 8 outputs which creates the possibility of 16 different couples. To read the input of the keypad, a simple for loop was used, in combination with multiple if else statements. The for loop runs four times and each time it inputs a 5-volt signal to one of the keypad's inputs, while the if else statements constantly check all four outputs of the keypad. If a voltage is detected in one of the outputs, then depending of what input was activated and what output was signal read at, it is possible to tell which push button was pressed and what the corresponding number is.

Once a signal is read from any of the four outputs, the keypad LED is turned off, and back on after a short delay. This is to indicate to the user that the input was received and to indicate when the system is ready for the next input. Following is the pseudocode.

```
1. For $i$ from 0 to 4:
    a. Turn on input[$i$]
    b. If the first output reads the voltage, set $j = 0$
    c. Otherwise read the second output, if voltage present, $j = 1$
    d. Otherwise read the third output, if voltage present, $j = 2$
    e. Otherwise read the fourth output, if voltage present, $j = 3$
    f. Otherwise $j = -1$
    g. If voltage was detected at any of the outputs (ie: $j$ != -1), do the following:
        i. Turn off LED, translate input using $i$ and $j$ (input = keypad[$i$][ $j$]), short
           delay, reset $j$ ($j = -1$) for next iteration, Turn on LED
```

**Alarm System Flowchart**


Start
User Exists?
No
Create user and save user pass-code in EEPROM
Yes
Retrieve pass-code and ask user for pass-code
Correct pass-code?
No
Yes
Toggle input method
Option D
Did user select menu item?
Is alarm triggered?
Interrupt
No
Yes
Retrieve user preferences and load the menu
Option A
Ask for pass-code
Ask for pass-code
No
Correct pass-code?
Yes
Option C
Option B
Correct pass-code?
No
Correct pass-code?
Yes
Is PIR sensor active
Is TMP sensor active
Ask for new pass-code and save it to EEPROM
Yes
No
Deactivate PIR sensor and update status in EEPROM
Activate PIR sensor and update status in EEPROM
No
Yes
Deactivate TMP sensor and update status in EEPROM
Ask for threshold, activate TMP sensor and update status in EEPROM

## Challenges Faced and Lessons Learned

One of the main challenges that was encountered during this project was having to redo some of the work. Many times the initial assumptions made were incorrect or the requirements were not all considered before tackling the problem. This caused unnecessary complexities. Carefully studying the requirements and thoroughly analyzing all the possible solutions proved to be an effective strategy in dealing with problems with such complexity. One of the specific challenges that was encountered was getting the keypad interface to work correctly. While the problem itself is relatively complex in nature, spending more time planning and thinking about the logic and less time coding would have made the problem easier to tackle.