CSE 3442: Alarm System Project

Brandt Frazier

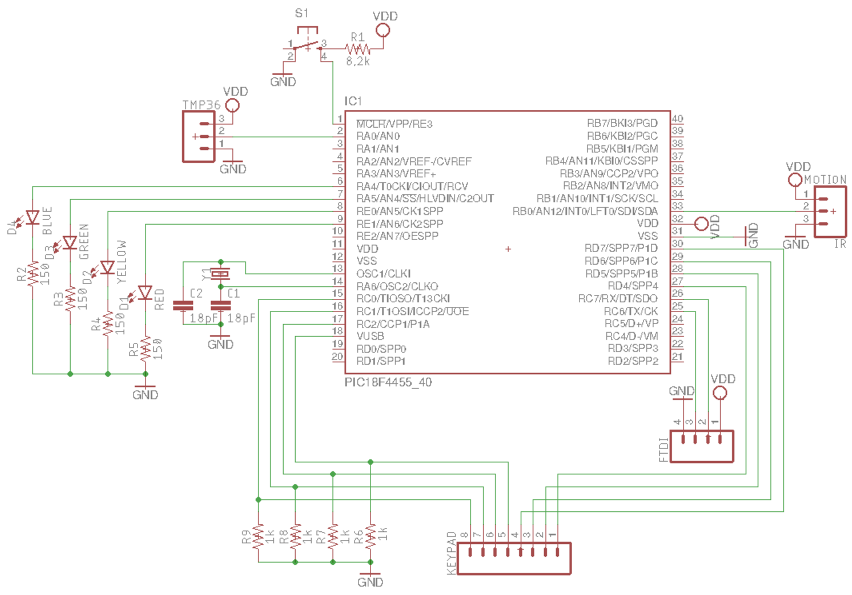
2017-12-17

Contents

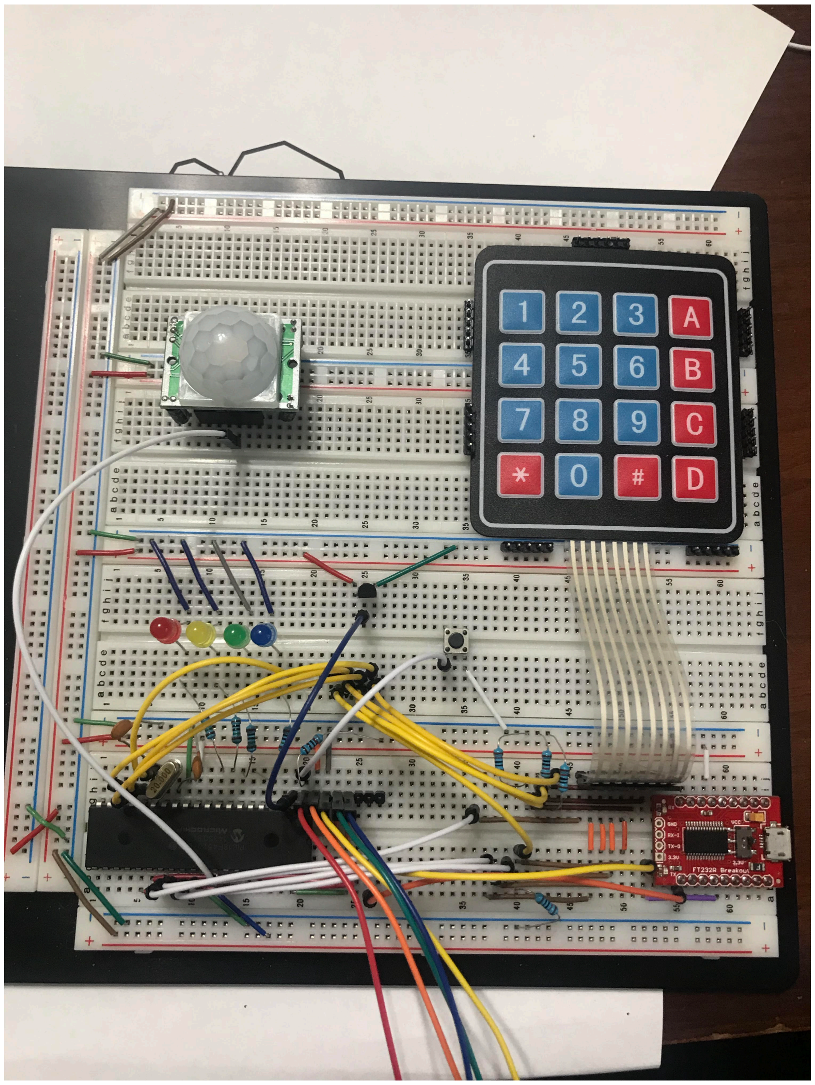
1 Overview

This project is a microcontroller-based alarm system intended to demonstrate the integration of many of the PIC 18F4520's peripheral features. It is can be configured using a serial terminal or (following initial configuration) using a 16-button keypad attached to the device. It detects and responds to motion using an infrared discrete motion sensor and to temperature using a linear analog temperature sensor. It supports multiple users and a rudimentary user privelidge scale.

1.1 Schematic



1.2 Breadboard



2 Requirements

- Use of USART to communicate with serial terminal.

  In order to abstract the complexity of serial communication, the AlarmUSART component includes functions to clear the RX buffer, to operate the client terminal using VT-100 terminal commands, and to provide system calls in order for C library I/O functions to reliably interact with USART.

- Use of EEPROM to save and recall user information.

  The alarm system should not reset to its initial configuration when it loses power. Therefore, the AlarmEEPROM component includes functions to read and write to persistent flash memory.

- Use of digital I/O to handle keypad.

  They keypad should be a reliable input method when enabled, and should not interfere with the interpretation of input over serial. The AlarmKeypad component manages the keypad interface by maintaining a ring buffer of keypad strokes that other components, including interrupts, can access when required. It also contains functions for retrieving and interpreting keys from the keypad.

- Use of digital I/O to handle motion sensor.

  The AlarmMotion component contains functions for initializing, detecting, and responding to input from the motion sensor.

- Use of analog I/O to handle temperature sensor.

  The AlarmTemp component contains function for initializing, detecting, and responding to input from the ADC module of the PIC18.

- User management

  The project supports up to eight users. Only the Master user may create or delete users, but users may change their own passcodes after the initial code is set by logging in with the Master. The AlarmUser component also contains functions for handling user menus.

3 Interrupts

The Alarm System makes use of high- and low-priority interrupts to accomplish the following:

1. If the motion sensor has been initialized, a rising edge detected on the sensor's output pin will cause an interrupt (high priority #1). This interrupt service routing (ISR) checks to see whether the motion alarm has been enabled and, if so, triggers and alarm. This notifies the user over serial and sets a red LED. This condition will not change unless and until a valis user passcode is entered.

2. When an ADC conversion has completed, an interrupt is triggered (high priority #2). The ISR saves the analog value measured from the temperature sensor and compares it to a user-defined threshhold temperature (default: 20 degC). If it is higher than the threshold and the temperature alarm is enabled, the temperature alarm will be triggered. This notifies the user over serial and sets a yellow LED. This condition will not change unless and until a valid user passcode is entered. This interrupt then starts a new ADC conversion.

3. A 16-bit timer provides a nearly-constant tick component to the system. On overflow, an interrupt is triggered (low priority #1). The ISR temporarily disables the timer and updates a few of the badge's components:
   (a) Signals AlarmMotion that a tick has passed (after reset or startup, a full minute must pass before output from the motion sensor can be relied upon).
   (b) Calls an AlarmKeypad function to check for a new keypad entry.
   (c) If a new key press is found, its character is saved to the AlarmKeypad ring buffer.

4 Timer Calculations

A single 16-bit timer is used to provide a reliable tick to the program. While absolute precision was not required, the timer needed to cause interrupts often enough for the keypad input to feel responsive to the user but not so often that regular code execution became obviously blocked.

All of the features of Timer 0 can be accessed via the registers TRISA, T0CON, INTCON, TMR0H, and TMR0L. For a 16-bit timer with without preloading, initialization did not require access to the timer count registers TMR0H and TMR0L. The T0CON register is set on startup to the following:

```
TOCON = 0000011;
```

- Bit 7 (MSB) is cleared initially, which ensures the clock is disabled while it is being configured.
- Bit 6 is cleared to configure Timer 0 as a 16-bit timer–without preloading, this means that the timer will count to $2^{16} = 65\,536$.
- Bit 5 is cleared to select the internal instruction clock as the signal for the clock to increment its count. The PIC18 has an average of 4 cycles per instruction, so this would result in a count frequency of $1/\mathrm{F}_{\mathrm{osc}}$ ($\mathrm{F}_{\mathrm{osc}} =$ 20 MHzfor this project) if no prescalar were set.
- Bit 4 is cleared so that the clock will increment on the rising edge of its source.
- Bit 3 is cleared to assign the prescalar value given by bits 2-0.
- Bits 2-0 are set to 0b011, which corresponds to a prescalar of 1/16. Thus, the frequency of overflow is given by:

$$F_{ovf} = \frac{F_{osc}}{Cycles\,Per\,Instruction} \times \frac{Prescalar}{2^{Clock\,Count\,Bits}} = \frac{20\,\mathrm{MHz}}{4\,CPI} \times \frac{1}{16} \times \frac{1}{2^{16}} = 4\,\mathrm{Hz}$$

To enable interrupts when Timer 0 overflows, the INTCON register was modified by the following:

```
    INTCONbits.TMR0IE = 1;
    INTCONbits.TMR0IF = 0;
```

NOTE: The author only just now realized that Timer0 cannot have a priority assigned to it, and is therefore interpreted as a high priority interrupt any time it is enabled. This almost certainly led to the majority of headaches during the later stages of this project's development. With Timer 0 set to cause interrupts, the ISR should be called close to 4 times per second. This is fast enough for the keypad to feel responsive, but not so fast that other functionality will be noticably blocked while the ISR handles the overflow signal.

5 Temperature Calculations

The TMP36 linear analog temperature sensor produces an output voltage which is near-linear with air temperature over a range of temperatures that humans are likely to experience. The data sheet provides that V_{out}increases 10 mV/°C and should be factory-calibrated to output 0.750 Vat 25 °C. In point-slope form, this yields the linear relation

$$V_{out} - 0.750\,\mathrm{V} = 0.010\,\frac{\mathrm{V}}{^\circ\mathrm{C}} \times (T_{\text{measured}} - 25\,^\circ\mathrm{C})$$

$$\Rightarrow T_{\text{measured}} = \frac{V_{out} - 0.750\,\mathrm{V} + 0.010\,\frac{\mathrm{V}}{^\circ\mathrm{C}} \times 25\,^\circ\mathrm{C}}{0.010\,\frac{\mathrm{V}}{^\circ\mathrm{C}}}$$

$$\Rightarrow T_{\text{measured}} = \frac{V_{out}}{0.010\,\frac{\mathrm{V}}{^\circ\mathrm{C}}} - 50\,^\circ\mathrm{C}$$

Mapping voltages to the range of available ADC values ($Val_{ADC} = \frac{V_{out}\times 1023}{5\,V}$) gives:

$$T_{\text{measured}} = \frac{\frac{Val_{ADC}\times 5\,V}{1023}}{0.010\,\frac{\mathrm{V}}{^\circ\mathrm{C}}} - 50\,^\circ\mathrm{C}$$

$$\Rightarrow T_{\text{measured}} = 0.49 \times Val_{ADC} - 50\,^\circ\mathrm{C}$$

6 Keypad Management

The 16-key input pad is a simple resistive matrix which connects conducting paths between the row and a column of a key when it is pressed. The author spend a great deal of time attempting to determine key presses using voltage dividers to input different voltages to each row, and then to detect key presses using a 4-bit R-2R ladder DAC circuit. Unfortunately, the resistance of each of the 16 possible connections can vary by as much as 20%, and choosing voltage divider resistances that would produce unique outputs once divided by the DAC briefly turned the project into an experiment in number theory.

The next solution was the recommended one: the AlarmKeypad component "sweeps" across the rows by exclusively outputting 5 V to each column in succession once every time Timer 0 overflows (roughly 4 Hz). If it detects a connection at one of the columns, it maps the row and column number to the relevant character and returns this character to the ISR. If the character is valid, the ISR saves the key press to the keypad ring buffer. The AlarmKeypad component keeps pointers to read and write positions in the ring buffer, so that other components can determine if a new character has been written and hasn't yet been read by another component.

It is then up to the components to use the buffer however it suits them. For instance, the main loop spends most of its time waiting for a series of four valid character from either USART or the keypad, whichever completes first. The user menus flush the key buffer, then wait for a new key press or a new byte over serial. Whichever character comes first is checked against a list of valid menu options, and a new menu is drawn if the key press is valid.