Quick Overview of PIC16F8X

Version 3.0
Date: August 12, 2010

1. Introduction

This document provides a quick overview of the PIC16F84A microcontroller by Microchip. This is an 18-pin chip as shown in Figure 1.

Features of the chip are

- 5-bit I/O port PORTA (pins RA4-RA0)
- 8-bit I/O port PORTB (pins RB7-RB0)
- Interrupts, including RB0/INT pin and PORTB<7:4>.

*Interrupt:* An "interrupt" is a mechanism that allows a microprocessor to execute an alternate job from its main routine using an interrupt handler. The interrupt handler may be invoked through hardware or software. If it is by hardware, it often corresponds to an electrical signal on a pin. If it is by software, then it often
corresponds to a machine instruction. Interrupts will be covered in Lab 2.3, so we will describe them in more detail then.

Other pins of the chip include
- **MCLR**: Master clear (reset) input/programming voltage input. This pin is an active low reset to the device.
- **VSS**: Ground reference for logic and I/O pins.
- **VDD**: Positive supply for logic and I/O pins.
- **RA4/TOCKI**: An interrupt input.
- **OSC1/CLKIN**: Oscillator crystal input/external clock source input.
- **OSC2/CLKOUT**: oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT, which has 1/4th frequency of OSC1, and denotes the instruction rate (for this processor, each instruction takes four clock cycles to execute).

Important registers of the PIC 16F84A architecture are provided below. They are described in the next section.
- **TMRO**: 8-bit real-time clock/counter. This is covered in Lab 2.2.
- **OPTION**: Used to select options for controller
- **TRISA**: Selects data direction of PORTA
- **TRISB**: Selects data direction of PORTB
- **PORTA**: RA4-RA0
- **PORTB**: RB7-RB0
- **INTCON**: Used to select interrupt options
- **Configuration word**: Used to select options for the controller

Consider the following example. The PIC micro-controller is connected to a 7-segment display. The display counts "0", "1",..., "9", "0", "1",... and changes value every second. Figure 2 shows the block diagram of the system. Let us design the system such that the RB pins of the Micro-controller act as outputs and are connected to the 7-segment LED.
As you can see, the micro-controller acts as a state machine here, and is the ‘brains’ of the system. The outputs of the micro-controller basically control the display shown by the LED. For the example, the source code for programming the Micro-controller is shown below. Have a close look at the source code and observe the keywords TRISB and PORTB used extensively in the code. Refer to Section 2 for their significance.

```c
#include <pic1684.h>

// ver 1.0
// Created on 09/03/01 by Ashok B.
// This C program will make a 16F94 PIC microcontroller drive a 7-segment display so
// that the display counts "0", "1",..., "9", "0",... The display changes value every second.
// Thus, the program has delay functions that are loops. The number of times a loop is
// executed is based upon the clock rate to get the exact delay.

t void delay_10us(unsigned char t); // t*10us delay
t void delay_ms(long t); // t ms delay
t void delay1sec(void); // 1 sec delay

unsigned char delay; // global used in the Assembly code

// code continues on the next page
```
void main(void)
{
    int count = 0; // The value that is to be displayed on the 7 segment display.
    TRISB = 0b00000000; // Setting PORTB as an output
    while(1) {
        switch(count) // increment count and then output the value to PORTB
            // Seven segment LED display is used. RB7 corresponds to “a”
            // of the LED, RB6 corresponds ‘b’ of the LED,..RB1 to ‘g’ of the LED
        {
            case 0:
                PORTB = 0b00000011;
                break;
            case 1:
                PORTB = 0b11110011;
                break;
            case 2:
                PORTB = 0b00100101;
                break;
            case 3:
                PORTB = 0b01100001;
                break;
            case 4:
                PORTB = 0b11010001;
                break;
            case 5:
                PORTB = 0b01001001;
                break;
            case 6:
                PORTB = 0b00011001;
                break;
            case 7:
                PORTB = 0b11100011;
                break;
            case 8:
                PORTB = 0b00000001;
                break;
            case 9:
                PORTB = 0b01000001;
                break;
        }
        delay1sec();
        count++;
        if (count == 10) count = 0;
    }
} // code continues on the next page
// The following are functions to create real-time delay.

void delay_10us(unsigned char t)
// provides t * 10 usecs of delay.
// Max of t is 255 which corresponds to 2550 usecs
{
delay = t; // This is a global variable.
// Below is an assembly language fragment to create a 10 microsecond delay,
// Note that in this fragment the C integer variable "delay" is written as "_delay".
#asm
DELAY_10US_1:
    CLRWDT
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    DECFSZ _delay,f
    GOTO DELAY_10US_1
#endasm
}

void delay_ms(long t) // delays t millisecs
{
do {
delay_10us(100);
} while(--t);
}

void delay1sec(void)
{
delay_ms(1000);
}

2. Registers

The following is a description of some of the important registers of the PIC 16F84A architecture. Some of this is review. **The value of the registers usually needs to be programmed in order for the micro-controller to perform desired functions.** For example, pins RB7 to RB0 can be used both as input pins and output pins. In a particular design, certain pins of the PORTB of the PIC might be used as input. In order to enable those
pins of the port to be inputs, the corresponding register has to be initialized. Included in
the description are also the values on Power-on Reset (when you power up the system)
and value on all other resets. Also included are the addresses of the register in
hexadecimal (prefix "0x").

Legend:  \(x = \text{unknown}\) and \(u = \text{unchanged}\).

**TMR0**

Description:  8-bit real-time clock/timer.

Value on Power-on Reset:  xxxx xxxx

Value on all other resets:  uuuu uuuu

Address:  0x01

**OPTION_REG**

Description:  Used to select options for microcontroller

- \(bit\ 7, \ \text{RBPU}: \ \text{PORTB}\ \text{Pull-up enable/disable bit, where} \ 0/1 = \text{enable/disble}\)
- \(bit\ 6, \ \text{INTEDG}: \ \text{Interrupt edge select bit, where} \ 0/1 = \text{interrupt on falling/rising}\)
  edge of RBO/INT pin
- \(bit\ 5, \ \text{TOCS}: \ \text{TMR0 clock source select bit. This determines where the instruction}\)
  \(\text{clock will come from. If the bit = 0 then the clock will come internally, and} \)
  \(\text{CLKOUT will be an output of this clock signal. If bit = 1 then the clock will be from} \)
  \(\text{an external source that will be input the RA4/TOCKI pin.}\)

Value on Power-on Reset:  1111 1111

Value on all other resets:  1111 1111

Address:  0x81
**TRISA**

**Description:** This determines the data direction of PORTA (pins RA4-RA0). Note that TRISA.n (n = 0, 1, ..., 4) corresponds to pin RAn. TRISA.n corresponds to bit n of the TRISA register. If TRISA.n = 0 then PORTA.n is an output, and if TRISA.n = 1 then PORTA.n is an input.

- **Value on Power-on Reset:** ---1 1111
- **Value on all other resets:** ---1 1111
- **Address:** 0x85

**TRISB**

**Description:** This determines the data direction of PORTB (pins RB7-RB0). TRISB.n corresponds to bit n of the TRISB register. If TRISB.n = 0 then PORTB.n is an output, and if TRISB.n = 1 then PORTB.n is an input.

- **Value on Power-on Reset:** 1111 1111
- **Value on all other resets:** 1111 1111
- **Address:** 0x86

**PORTA**

**Description:** This is a 5-bit port corresponding to pins RA4-RA0. RA.n (n = 0, 1, ..., 4) corresponds to pin RAn.

- **Value on Power-on Reset:** ---x xxxx
- **Value on all other resets:** ---u uuuu
- **Address:** 0x05
PORTB

**Description:** This is an 8-bit port corresponding to pins RB7-RB0. RB.n (n = 0, 1, ..., 7) corresponds to pin RBn.

**Value on Power-on Reset:** xxxx xxxx

**Value on all other resets:** uuuu uuuu

**Address:** 0x06

INTCON

**Description:** The user may enable/disable and check the status of interrupts with this register. For Lab 2.1, we won’t use the interrupts, so all interrupts should be disabled.

- **GIE (bit 7):** Global interrupt enable bit. 0 = disables all interrupts. 1 = enables all unmasked interrupts.
- **EEIE (bit 6):** EE write complete interrupt enable bit. 0/1 = disables/enables interrupt
- **TOIE (bit 5):** TMR0 overflow interrupt enable bit. 0/1 = disables/enables TMR0 interrupt
- **INTE (bit 4):** RB0/INT interrupt enable bit. 0/1 = disables/enables RB0/INT interrupt
- **RBIE (bit 3):** RB port change interrupt enable bit. 0/1 = enables the RB port change interrupt
- **TOIF (bit 2):** TMR0 overflow interrupt flag bit. 0/1 = TMR0 has didn't/did overflow.
- **INTF (bit 1):** RB0/INT interrupt flag bit. 0/1 = the RB0/INT interrupt didn't/did overflow.
- **RBIF (bit 0):** RB port change interrupt flag bit. 0 = none of the RB7:RB4 pins have changed state. 1 = at least one of the RB7:RB4 pins changed state.

**Value on Power-on Reset:** 0000 000x

**Value on all other resets:** 0000 000u

**Address:** 0x0b, 0x8b
Note that the flag bits must be cleared in software. Also note that interrupt flags get set regardless of whether an interrupt is enabled or disabled.

**Configuration Word**

*Description:* There are 12 bits in this word

- **bits 13:4,** Code protection bit: 0/1 = code protection on/off. For our purposes, these should be 1 for now.
- **bit 3, PWRTE:** Power-up timer enable bit. 0/1 = power-up timer is enabled/disabled. For our purposes, this should be disabled for now.
- **bit 2, WDTE:** Watchdog timer enable bit. 0/1 = WDT disabled/enabled. For our purposes, this should be disabled for now. We will discuss watch dog timers in Lab 2.3.
- **bit 1:0,** FOSC1:FOSC0: Oscillator selection bits
  - 11 = RC oscillator
  - 10 = HS oscillator
  - 01 = XT oscillator
  - 00 = LP oscillator

*Value on Power-on Reset:* uuuu uuuu
*Value on all other resets:* uuuu uuuu
*Address:* 0x2007. This is beyond user program memory. The bits of this word are set at the time the chip is being programmed.
3. C Programming Using PICC Lite

We would be using the PICC Lite compiler for programming the PIC 16F84A. The following are some C programming structures for the PICC Lite compiler.

3.1 Setting the Configuration Word

#include <pic.h>
CONFIG(x); // This is a macro (which is explained below).
            // x is the configuration word value.

What’s a macro? Basically, it’s a single line or word of program code that represents a collection of lines of code. The macro is defined somewhere to represent a set of lines of code. Then a programmer wants that set of lines, he/she can write the macro rather than the whole set of lines. A example of a macro is as follows (note this is for a fictitious programming language):

MACRO ExampleMacro
Line 1;
Line 2;
Line 3;
ENDMACRO

Every instance of “ExampleMacro” in the program is actually the three lines Line 1, Line2, and Line 3. Macros can also have parameters. The macro _CONFIG(x) has the parameter x. Examples on the next page are macros with parameters.

We’ll discuss macros in lecture later in the semester.
3.2 Bit setting

To set and clear bits, the following macros may be used (this are actual macros in C).

#define bitset(var,bitno)  ((var) |= 1 << (bitno))
#define bitclr(var,bitno)  ((var) &= ~(1 << (bitno)))

bitset(foo,6); will set bit 6 of foo to 1.

These macros make it easier to read the program. These operations set and clear bits in variable “var”. The bit position to be set or cleared is “bitno”.

3.3 TRIS and PORT

The TRIS and PORT registers may be specified using TRISA, TRISB, PORTA, and PORTB. To specify individual bits of TRISA or TRISB, you may use TRISAn or TRISBn, where n = 0, 1, ...., 7. For example, to set bit 3 of TRISA to 1, you can use TRISA3 = 1;

To specify individual bits of PORTA or PORTB, you may use RA.n or RB.n, where n = 0, 1, ..., 7. For example, to clear bit 5 of PORTB to 0, you can use RB5 = 0.