Alvin Bademeca
UWT TCES 430
Final Exam
Prof. Sheng

Q1 Part 1.

1. (30%) Assume the PIC18F4520 MCU is running at 8 MHz crystal frequency, consider the following delay subroutine:

```
delay: movlw valA
movwf delayreg
loop: nop
nop
decfsz delayreg, 1
goto loop
return
```

a. What's the delay this subroutine actually introduces, considering the time taken by every instruction?

```
delay: movlw valA     //Takes 1 instruction cycle
       movwf delayreg      //Takes 1 instruction cycle

loop:  nop           //Takes 1 instruction cycle
       nop           //Takes 1 instruction cycle
       decfsz delayreg, 1 //Takes 1 instruction cycle if result in not zero
                          //otherwise takes 3 according to datasheet
                          //3 cycles if skip and followed
                          //by a 2-word instruction.

       goto loop     //Takes 2 instruction cycle
return
```

Each instruction takes 4 clock cycles so at 8 MHz we can have (8MHz*1sec)÷(4Hz/Instruction cycle) = 2M instruction-cycles per second. Each instruction cycle takes :
$1second/2MHz = 500nano-seconds$ (5 x 10^{-7}sec/instruction-cycle)

The loop takes 5 instruction-cycles so the delay introduced is approximately:
5 instruction cycle multiplied by the value of `valA`
delay $\approx$ 5 instruction-cycle $\times$ (5 x 10^{-7} sec/instruction-cycle) $\times$ `valA`

if we want to be exact we can count the two instructions in `delay`

delay = 5 instruction-cycle $\times$ $\left(\left((5 \times 10^{-7}\text{sec/instruction-cycle}) \times \texttt{valA}\right) + 2\right)$

b. What number should be loaded into delay register delayreg to make the generated delay as close as possible to 500 μs?

valA is loaded into delayreg
Using the equation
delay ≈ 5 instruction-cycle × (5 x 10^{-7} sec/instruction-cycle) × valA
valA = dealy /(5 instruction-cycle × (5 x 10^{-7} sec/instruction-cycle)
For delay = 500 μs

valA ≈ 500 μs /(5 instruction-cycle × (5 x 10^{-7} sec/instruction-cycle)
≈ **200**

so **200** should be loaded into delayreg

c. What modification needs to be taken in order to introduce a time delay of 10 ms?

Using the reasoning in the part b we need a value of :

valA ≈ 10ms /(5 instruction-cycle × (5 x 10^{-7} sec/instruction-cycle)
≈ 4000

The problem with this is that we only have 8-bits which could have a max value of 255. Notice that for part b we looped for 200 times. If we perform this loop 20 times (200 × 20 = 4000) we can accomplish our goal of creating a delay of 10ms.

```
//valB = 20 valA = 200
 movlw valB
 movwf delayregB

 delay: movlw valA
 movwf delayreg
 loop: nop
 nop
 decfsz delayreg, 1
 goto loop

 decfsz delayregB,1
 bra delay
 return
```

The assembly code above is approximately 10ms. We are off by a little because we essentially added 4 instruction-cycle:

decdfsz delayregB, 1
bra delay
delay: movlw valA
movwf dealyreg

for 20 times plus 2 insctruction-cycle:
movlw valB
movwf dealyregB

for a total of 22 instruction-cycle adding 11 μs more of delay:
5 x 10^{-7}sec/instruction-cycle* 22 instruction-cycle = 11 μs

so the assembly code below has approximately 10.011ms of delay

```
movlw valB
movwf delayregB

delay: movlw valA
movwf delayreg
loop: nop
nop
decfsz delayreg, 1
goto loop

decfsz delayregB,1
bra delay
return
```

Is this acceptable? If not we have to modify this code to ensure that we have 4000 instruction-cycles and not 4022.

Q2 Part 2.

i. An explanation of your design and the calculation involved (see more details in the problem description).

The state machine is implemented using switch statement. Timer0 was used to turn the LED off and on. Timer0 is set-up to interrupt every 262 ms (milliseconds) normally, when its blinking rapidly the interrupt happens every 131 ms this is accomplished by changing timer0's prescale from 256 to 128. The calculations for the timer0 interrupt are as follows:

Timer0 is set to interrupt at a frequency of (FOSC / 4)/(prescaler * 8bitmode)
interrupt frequency = (1Mhz / 4)/ (256*256) = 3.815Hz the period is the inverse which gives us **262ms**. When prescaler is changed to 128 ,
interrupt frequency = (1Mhz / 4)/ (128*256) = 7.629Hz the period becomes **131ms**

The interrupt service routine for timer0 is as follows:

```
if(TMR0IF)
    {
        TMR0IF = 0;
        tmr0flag = 1; //Set timer0 flag used to count toggles or
blinks
        PORTBbits.RB1 ^= 1; //Toggle LED output
    }
```

`tmr0flag` is used in state 5 ensure that the number of blink is just five. Blinking is disabled by turning off timer0 (disabling the interrupt is another option but was not used here) . The state machine implemented have states 1 through 5 and mirror the description in 2013final.pdf :

State(1) Turn LED off. On press and release of RB0, turn LED on, goto (2).
State(2) After press and release of RB0, begin blinking.
State(3) After press and release of RB0, halt blinking. If select input RA4 = 1 goto (1), else goto next step.
State(4) Turn LED off. On press of RB0 only, goto (5).
State(5) Blink rapidly 5 times while button held down. If button RB0 released before all 5 blinks complete, go to (1). If all 5 blinks complete, freeze off, and release of RB0 remains in this state. A subsequent press of RB0 repeats (5).

ii. The header file you included in your C source code.

```
// A separate header file was not created
 #include <p18f4520.h>
 #include <xc.h>
```

iii. The C source code.

The whole mplabx project is included in the submission below is the code for finalmain.c

```
/*
 * finalmain.c
 * Author : Alvin Baldemeca
 *
 * Alvin Baldemeca
 * Final Exam Problem 2
 * University of Washington Tacoma
 * TCES 430, Prof. Sheng
 * 12.10.2013
 *
 * LED is on when RB1 is HIGH off otherwise
 * Normal Rate of Blinking : 0.524ms (262ms(ON) + 262ms(OFF)
 * Rapid Rate of Blinking : 262ms (131ms(ON) + 131ms(OFF)
 * Clock : internal 1MHZ (FOSC)
 * Timer : Timer0 which is configured for high priority interrupt.
 *
 * This project only has one source C file the header file included are the
 * p18f4520.h (micro-controller specific header) and xc.h (XC compiler header).
 *
 * The state machine is implemented using switch statement.  Timer0 was used to
 * turn the LED off and on.  Timer0 is set-up to interrupt every 262 ms
 * (milliseconds) normally, when its blinking rapidly the interrupt happens
 * every 131 ms this is accomplished by changing timer0's prescale from 256 to
 * 128.  The calculations for the timer0 interrupt are as follows:
 *
 * Timer0 is set to interrupt at a frequency of (FOSC / 4)/(prescaler * 8bitmode)
 * interrupt frequency = (1Mhz / 4)/ (256*256) =  3.815Hz the period is the
 * inverse which gives us 262ms. When prescaler is changed to 128 ,
 * interrupt frequency = (1Mhz / 4)/ (128*256) =  7.629Hz the period becomes 131ms
 *
 * The interrupt service routine for timer0 is as follows:
 *
 * if(TMR0IF)
 *   {
 *      TMR0IF = 0;
 *      tmr0flag = 1; //Set timer0 flag used to count toggles or blinks
 *      PORTBbits.RB1 ^= 1; //Toggle LED output
 *   }
 *
 * tmr0flag is used in state 5 ensure that the number of blink is just five.
 * Blinking is disabled by turning off timer0 (disabling the interrupt is
 * another option but was not used here)
 *
 *
 * The state machine implmented here, have states 1 to 5 and mirrors the
 * description in the 2013final.pdf out as follows :
 *
 * (1) Turn LED off. On press and release of RB0, turn LED on, goto (2).
 * (2) After press and release of RB0, begin blinking.
 * (3) After press and release of RB0, halt blinking. If select input RA4 = 1
 *     goto (1), else goto next step.
 * (4) Turn LED off. On press of RB0 only, goto (5).
 * (5) Blink rapidly 5 times while button held down. If button RB0 released
```

```
 *     before all 5 blinks complete, go to (1). If all 5 blinks complete, freeze
 *     off, and release of RB0 remains in this state. A subsequent press of RB0
 *     repeats (5).
 *
 * The code in this file is well commented, each
 * has a description of what the code is trying to accomplish and each line is
 * commented except for the trivial.
 */

#include <p18f4520.h>
#include <xc.h>
#pragma config WDT = OFF  // Set watch dot timer off
#pragma config LVP = OFF  // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)
#pragma config PBADEN = OFF // Turn off PortB A/D
#pragma config OSC = INTIO67  //Internal frequency is 1MHz by default
#define _XTAL_FREQ 1000000  //used by __delay_ms this should match what we set our clock
to
#define FIVE_BLINK 10
#define ON 1
#define OFF 0

volatile unsigned char tmr0flag = 0;
void init_pwm();
void init_timer0();

void high_priority interrupt high_isr()
{
    if(TMR0IF)
    {
        TMR0IF = 0;
        tmr0flag = 1; //Set timer0 flag used to count toggles or blinks
        PORTBbits.RB1 ^= 1; //Toggle LED output
    }
}

void main()
{
    unsigned char state = 1;
    int count; //Count the number of toggles count = 2 for an on-off or off-on toggle
    TRISBbits.RB0 = 1; //Set switch 3 to input
    TRISBbits.RB1 = 0; //Set to output for LED
    TRISAbits.RA4 = 1; //Set switch 4 to input
    init_timer0();    //Initialize timer 0 interrupt (timer0 if off)

    while(1)
    {
        /*
         * The state machine implemented as switch statements
         */
        switch(state)
        {
            /*
             * state(1) Turn LED off. On press and release of RB0, turn LED on,
             * goto state(2).
             */
            case 1 :
```

```
    T0CONbits.TMR0ON = OFF;   //Ensure timer0 is off
    T0CONbits.T0PS = 0b111; //Prescale timer0 to 256;
    PORTBbits.RB1 = OFF;      //turn of led
    while(PORTBbits.RB0); __delay_ms(10); //Wait till pressed
    while(!PORTBbits.RB0); __delay_ms(10);//Wait till released
    PORTBbits.RB1 = ON;  //Once release turn led on
    state = 2;
    break;

/*
 * state(2) After press and release of RB0, begin blinking.
 */
 case 2 :
    while(PORTBbits.RB0); __delay_ms(10); //Wait till pressed
    while(!PORTBbits.RB0); __delay_ms(10);//Wait till released
    TMR0IF = 0; //Clear timer0 interrupt flag
    T0CONbits.TMR0ON = ON; //trun on timer0 which toggles LED
    state = 3;
    break;

 /*
  * state(3) After press and release of RB0, halt blinking. If select
  * input RA4 = 1 goto state(1), else goto next step (state 4).
  */
 case 3 :
    while(PORTBbits.RB0); __delay_ms(10); //Wait till pressed
    while(!PORTBbits.RB0); __delay_ms(10);//Wait till released
    T0CONbits.TMR0ON = OFF; //turn off blinking (turn off timer0)
    TMR0IF = 0; //clear timer0 interrupt flag

    if(PORTAbits.RA4) //If S2 (switch 2/RA4) not pressed
    {
        state = 1;
    }else //If S2 is pressed
        state = 4;
    break;

/*
 * state(4) Turn LED off. On press of RB0 only, goto state(5).
 */
 case 4 :
    PORTBbits.RB1 = OFF; //Turn LED off
    while(PORTBbits.RB0); __delay_ms(10); //Wait till RB0 is pressed
    state = 5;
    break;

/*
 * state(5) Blink rapidly 5 times while button held down. If button
 * RB0 released before all 5 blinks complete, go to state(1). If all
 * 5 blinks complete, freeze off, and release of RB0 remains in this
 * state. A subsequent press of RB0 repeats state(5).
 */
 case 5:
    T0CONbits.T0PS = 0b110; //Speed up the blinking (rapily 2X faster)
    T0CONbits.TMR0ON = ON;   //Turn blinking on
    __delay_ms(10);
```

```
                /*
                 * count is used for the number of toggles on the LED a count of
                 * 10 toggles represents 5 blinks.
                 */
                count = 0;

               /*
                * If button RB0 released before all 5 blinks while loop exits
                * if not it waits till count goes to 10 before exiting
                */
                while(!PORTBbits.RB0 && (count < FIVE_BLINK))
                {
                    if(tmr0flag)
                    {
                        tmr0flag = 0;
                        count++;
                    }
                }

                T0CONbits.TMR0ON = OFF; //After 5 blinks turn off blinking
                __delay_ms(10);
                TMR0IF = 0;          //Clear timer0 interrupt flag
                PORTBbits.RB1 = OFF; //Turn off LED
                while(count >= 10)//If five blinks complete freeze off
                {
                     while(!PORTBbits.RB0); __delay_ms(10); //wait till RB0 is released
                     break;
                }

                if(count < 10)
                {
                  /*
                   * If button  RB0 released before all 5 blinks complete, go to
                   * state(1)
                   */
                    state = 1;
                }else
                {
                   /*
                    * A subsequent press of RB0 repeats state(5).
                    */
                    while(PORTBbits.RB0); __delay_ms(10);
                    state = 5;
                }
                break;

            default :
                state = 1;
        }

    }
}

/**
 * Timer0 is set to interrupt at a frequency of (FOSC / 4)/(prescaler * 8bitmode)
 * interrupt frequency = (1Mhz / 4)/ (256*256) =  3.815Hz the period is the
 * inverse which gives us 262ms.
```

```
 */
void init_timer0()
{
    //T0CON
    T0CONbits.T0CS = 0; //Internal Freequency divided by four (FOSC/4)
    T0CONbits.PSA = 0; // Assign Prescale
    T0CONbits.T0PS = 0b111; //Prescale timer 0 to 256;
    T0CONbits.T08BIT = 1;  //Use 8 bit count
    T0CONbits.T0SE = 0; // Source Edge, 0 = Increment on low-to-high transition on T0CKI
pin

    //Set up interrupt
    INTCONbits.GIE_GIEH = 1; //enable global interrupt
    INTCONbits.TMR0IE = 1; //enable timer0 interrupt
    INTCONbits.T0IF = 0; //clear timer0 flag
    INTCON2bits.TMR0IP = 1; //Set timer0 for priority.

    //Turn off timer 0
    T0CONbits.TMR0ON = 0;

}
```