ECE 218

EXERCISE 4. TIMERS AND INTERRUPTS

In this exercise you will learn how to use timers to cause events to happen periodically without using software delays. We will use the timer to generate a delay between LED blinks, both in the main program, and in an interrupt service routine. We will then write a program to use interrupts to sample the state of a switch input.

HARDWARE SETUP

To keep things simple, we will use the LED on the Microstick that is connected to pin RA0 on the PIC24 for the output, and we will connect a switch to RB2 as we did in Exercise 1.

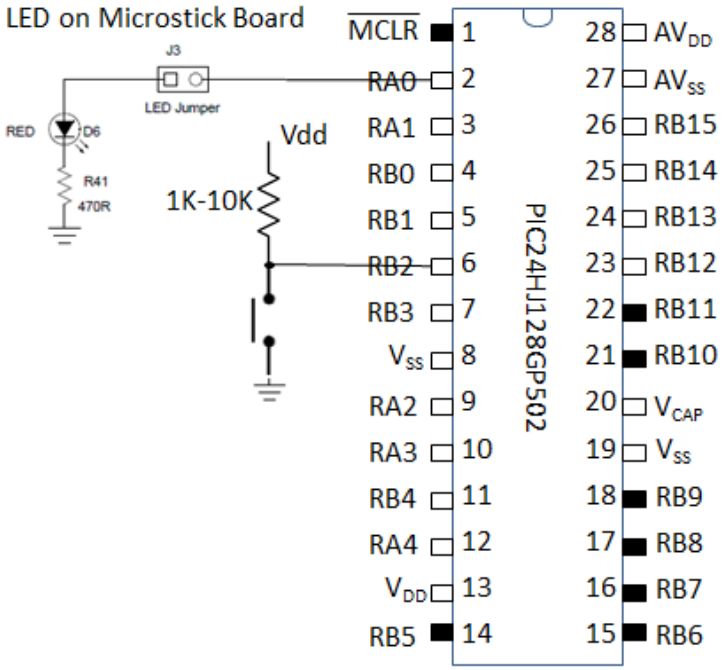


Figure 1. Schematic for Switch Connection to RB2.

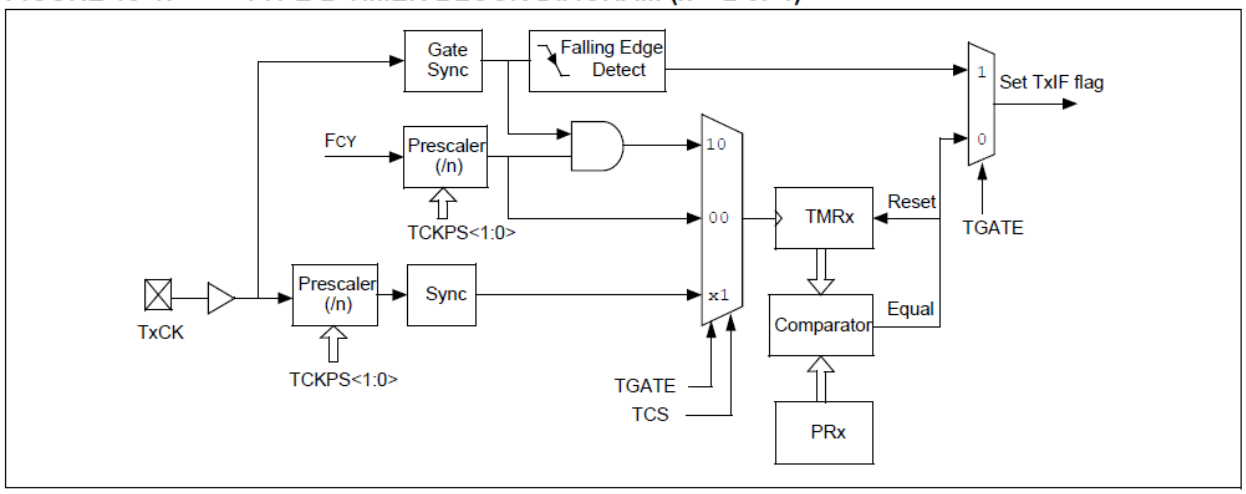
TIMER REVIEW

The PIC24 has five 16-bit timers. Timer 1 has a few extra features from the others. Timers 2-3 and 4-5 can be grouped to make up to two 32-bit timers. A block diagram of Timer B (2 or 4) from the PIC24 datasheet is shown in Figure 13-1[1]. The other timers are similar. Remember that the TMR2 is basically a counter. The preset register, PR2, can be used to set a terminal count, where the T2IF flag gets set once the timer and preset are equal.

[1] MicroChip 16-bit Microcontrollers (up to 128 KB Flash and 8K SRAM) with Advanced Analog, © 2007-2012 Microchip Technology Inc., DS70293G

Find the TMR2 (TMRx) register and the PR2 register in the block diagram and notice how their outputs are connected to the comparator. If the TGATE bit is 0, what two things happen when TMR2 = PR2? _____ _____

FIGURE 13-1: TYPE B TIMER BLOCK DIAGRAM (x = 2 or 4)



Timer 2 has three modes, depending on the clock source used for TMR2:

- Timer mode
- Gated Timer mode
- Synchronous Counter mode

In this exercise we will use the Timer mode, where FCY (the instruction clock frequency) is the TMR2 clock. FCY, can be used directly for the TMR2 clock, or it can be scaled down using the prescaler module, selected by the two TCKPS bits as follows:

TCKPS<1:0>: Timer2 Input Clock Prescaler Select bits
11 = 1:256
10 = 1:64
01 = 1:8
00 = 1:1

The prescale ratio is the amount of clock division, so a 1:64 prescale divides the clock frequency by 64. For example prescaling a 40MHz clock 1:8 results in a 5MHz clock source for the timer.

When we use the *configClock()* function with the textbook library, the internal instruction clock, FCY, is 40MHz. What are the four timer clock frequencies possible after prescaling?

TCKPS<1:0>: Timer2 Input Clock Prescaler Select bits
11 = 1:256 F_{tr2} = ____156.25 KHz__
10 = 1:64 F_{tr2} = ____625 KHz____
01 = 1:8 F_{tr2} = ____5 MHz_____
00 = 1:1 F_{tr2} = _____40MHz____

Recall how to calculate the period of a square wave. If the TMR2 clock frequency, F_{tr2}, is 156.25 KHz, what is the TMR2 clock period, T_{tmr2}? 0.0064 ms

The timer "flag" T2IF bit will be set when the timer is the same as PR2+1, so the timer delay (time interval between when the flag goes high) can be found as:
$T2delay = (PR2 + 1) * T_{tmr2}$.
What is the timer delay (for the timer with 156.25 KHz clock) if PR2 is set to 9? 0.064 ms

Assuming FCY=40MHz, let's find the minimum and maximum Timer 2 delay.

Minimum delay: Prescale 1:1 (TCKPS = 00), PR2 = 0x0000
T2delay = 0.025 micro s

Maximum delay: Prescale 1:256 (TCPS = 11) , PR2 = 0xFFFF
T2delay = 0.419 s

Recall that we can configure T3 and T2 to be a 32-bit timer. What is the maximum delay for a 32-bit timer? 7.65 hours

If the prescale is 1:256, what is the PR2 value for a 0.25s delay?
T2delay = 0.25s PR2 = 39,062

SOFTWARE PROGRAM 1: BLINK LED USING TIMER DELAY IN MAIN PROGRAM

You will need to consult the PIC24HJ128GP502 family datasheet. Download this file from Nexus to have handy. Find the documentation on the T2CON register (section 13.4 or search for TXCON).

You will start a new project using the textbook library, and we will create 3 different main programs in this exercise, one for each of the tasks. Starting with the *main_template.c* file on Nexus, we will first write a main program, *main_t2_blink.c*, to toggle the LED on the Microstick board every 0.25 second in the main function using Timer 2, with FCY divided by 256 for the timer clock input.

1 .Define the LED variable as bit RA0 (already in the template).

2. Write a global function called configTimer2() that sets Timer2 up to set the flag every 0.25s. This function type is void (it does not return a value) and it requires no parameters. This will require

a. Setting the T2CON control register to turn off the timer initially, use the internal instruction clock prescaled 1:256, and use the default TSIDL and T32 settings of 0. Remember that you can either write all of the bits to the T2CON at once (T2CON = 0xXXXX), or you can set the bits individually using instructions like T2CONbits.TON = 1. Unused bits can be set to zero (their default value). See the PIC24HJ128GP502 datasheet (Section 13.4) for the location of the T2CON bits,
b. Setting the preset (PR2) register for a 0.25s delay (you calculated this earlier),
c. Initialize TMR2 to 0,

d. Initialize the flag, T2IF to 0,
e. and finally turn Timer2 on before exiting.

3. Main program

Configuration and initializations

a. Configure the clock and Timer2 by calling the configuration functions.
b. Initialize Port A to be all digital (AD1PCFGL) all outputs (TRISA), and initialized to zero(LATA). Note this is overkill since we are only using bit RA0.

Main program – loop

In the main infinite *while* loop we poll the timer flag, and when high, toggle the led and clear the flag.

a. wait for the _T2IF flag to be set (this happens every .25s)
    i. toggle the led
    ii. clear the _T2IF

Once your code is written, build the project. Download to the board to confirm visually that you have a light that changes state about every 0.25 second, or about 4 times per second. Demonstrate to your instructor.

SOFTWARE PROGRAM 2: BLINK LED USING TIMER 2 INTERRUPT

Next we will modify our program to use the Timer 2 T2IF flag to interrupt the main program, and we will write an interrupt service routine that blinks the light. Save the current version of your program, and then use "save as" to save it to a different name, like *main_t2_blink_interrupts.c*. Replace the current main program with this new one in your project.

An *interrupt service routine,* ISR, is a function that is written to run when a particular interrupt occurs. Timer 2 is one of the many possible sources of interrupts on the PIC24. Each interrupt source has a particular interrupt vector number and address. In the XC16 compiler, names are given to each interrupt source. For Timer 2, that name is ***_T2Interrupt***.

The following code shows how this name is used to declare the Timer 2 ISR.

void _ISR _T2Interrupt(void)

This line specifies that function is an interrupt service routine (_ISR), the name of the ISR is T2Interrupt, that it takes no parameters, and returns no values (the two occurrences of void). The _ISR attribute informs the compiler to save and restore registers and to return from interrupt using the special assembler instruction ***retfie***.

Within the ISR, we will cause the LED to change state, and then reset the Timer 2 interrupt flag, T1IF to zero so interrupts can happen again. Here are the instructions to do this:

```
LED = ~LED;  //toggle LED
_T2IF = 0;   //clear Interrupt flag, the last instruction in ISR
```

Add this ISR to your program, in the global function definitions section.

Now that we have an ISR to toggle the LED, there is nothing for the main program to do except to initialize the ports and timer (already done in previous program) and **enable interrupts. This is done by setting the Timer 2 interrupt enable (_T2IE) to 1.** (Note that in a practical embedded system program, there would be many other things to do.) Your main program loop will be simply an infinite while loop with empty body.

Once your code is written, be sure that you have replaced the previous main program with this new interrupt-based one, and then build the project and download to the board to confirm visually that your LED still changes state about 4 times per second. Demonstrate to your instructor

How would you modify your programs to make the LED blink faster or slower?
Modify PR2

Next, we will introduce some timer library functions. See Appendix for code for these functions.

```
msToU16Ticks(desired_delay_in_ms,  prescale_setting); //Convert desired delay to the PRx setting
getTimerPrescale(TxCONbits);   //Returns the timer prescale setting
```

These could be used to set the PRx registers for particular delays, without doing the calculations in the previous section. For example to set PR2 for a 25ms delay:

```
PR2 = msToU16Ticks(25, getTimerPrescale(T2CONbits)) - 1;
```

Modify your *configTimer2* routine to use the timer library functions to set PR2. This will allow you to modify the timer delay without calculating the PR2 value each time.

Upload your final code for both *main_t2_blink.c*, and *main_t2_blink_interrupts.c* to Nexus.

APPENDIX

Text timer library functions:

```
//Converts milliseconds to 16-bit timer ticks.
uint16_t msToU16Ticks(uint16_t u16_ms, uint16_t u16_pre) {
// Use a float internally for precision purposes to accommodate wide range of FCY
  float f_ticks = FCY;
  uint16_t u16_ticks;
  f_ticks = (f_ticks*u16_ms)/u16_pre/1E3;
  ASSERT(f_ticks < 65535.5);  //Check that the f_ticks result is in range of 16-bits
  u16_ticks = roundFloatToUint16(f_ticks);  //Convert back to integer
  return u16_ticks;
}

// Converts microseconds to 16-bit timer ticks.
uint16_t usToU16Ticks(uint16_t u16_us, uint16_t u16_pre) {
  // Use a float internally for precision purposes to accommodate wide range of FCY
  float f_ticks = FCY;
  uint16_t u16_ticks;
  f_ticks = (f_ticks*u16_us)/u16_pre/1E6;
  ASSERT(f_ticks < 65535.5);  //Check that the f_ticks result is in range of 16-bits
  u16_ticks = roundFloatToUint16(f_ticks);  //back to integer
  return u16_ticks;
}

/** Given the TCKPS bitfield, return the timer prescale encoded
 *  by these bits. Use \ref getTimerPrescale as a convenient
 *  way to extract the TCKPS bitfield from a TxCONbits SFT
 *  then call this function.
 *  \param u8_TCKPS TCKPS bitfield from the timer in question
 *  \return Prescale value.
 */
uint16_t getTimerPrescaleBits(uint8_t u8_TCKPS) {
  const uint16_t au16_prescaleValue[] = { 1, 8, 64, 256 };
  ASSERT(u8_TCKPS <= 3);
  return au16_prescaleValue[u8_TCKPS];
}
```