Name _____

Lab Section _____

PIC – Periodic Pulse with Timer and Interrupt

Lab 5

Introduction: In last week's lab you controlled the blink rate of an LED by using a time wasting loop. The lab was a good way to get started working with the PIC microcontroller without learning about any of the microcontroller's peripherals. The "time wasting" method is an inefficient use of the PIC's processor. In this week's lab you will solve a similar problem using one of the microcontroller's built in timer modules. Your task is to create a short periodic pulse by utilizing the microcontroller's timer and interrupt structure.

Lab Requirements:

1. Demonstration of a periodic pulse with a width equal to the last digit of your RedID in uS (±0.1uS) and a repetition rate of the last digit of your RedID in mS (±0.1mS). If your RedID ends in "0", make a 10uS pulse repeating every 10mS.

2. Submission of your neatly formatted source code.

Last Digit of Red ID_____ Demo Check (JK)_____

About the TIMER 0 Module: The Timer 0 (TMR0) module provides a useful method of tracking time without creating inefficient time wasting loops. TMR0 can operate as either an 8-bit timer/counter with a programmable period or as 16-bit timer/counter. The Prescaler provides a method for slowing the timer count rate to yeild longer overflow or match cycles. For instance, if we configure TMR0 as an 8-bit timer without the Prescaler, the timer would overflow at the following rate.

$$^{4}/_{F_{OSC}} \times 2^{8} = T_{OVERFLOW}$$

When running the PIC16F18324 from a 4MHz clock an 8-bit overflow cycle of TMR0 would yield a 256uS delay as follows:

$$^{4}/_{4MHz} \times 2^{8} = 256uS$$

This is overflow cycle would be too frequent for many useful applications such as creating a basic system tick scheduler. In order to consume longer durations of time we could use the timer in 16-bit mode or utilize the timer prescaler or postscaler.

There are several ways to configure TMR0 to generate an interrupt a desired interval. One method is to run the timer in 8-bit mode using the prescaler and a programmed period value so the Timer 0 Interrupt Flag (T0IF) is asserted on a match. To get started look at the 8-bit Timer 0 block diagram shown below:

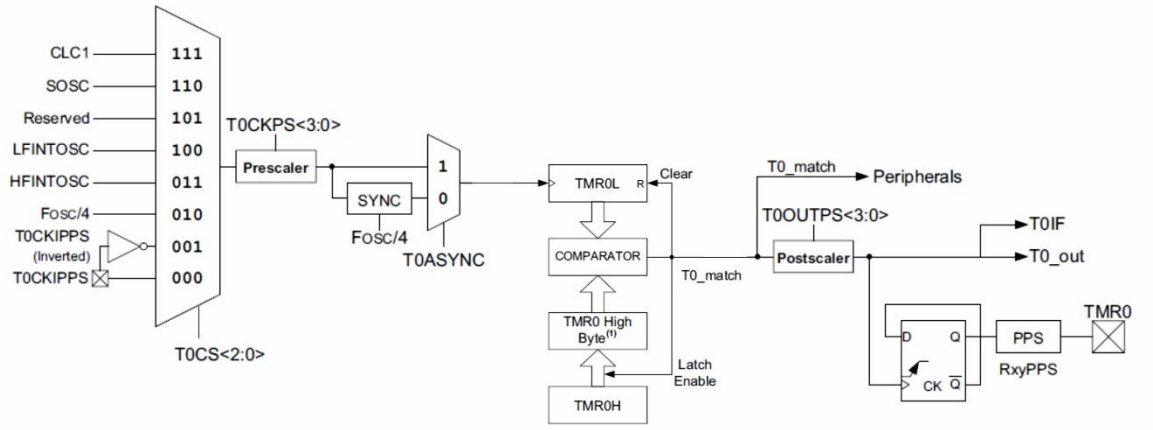

Figure 1 Block Diagram of TMR0 8-bit mode

The input to the timer/counter can be selected by setting the Timer 0 Clock Source bits (T0CS<2:0>) in the T0CON1 register. For today's lab I sugest you configure the clock rate for Fosc/4. Next you might want to slow the clock down by using the prescaler.

$$^{4}/_{F_{OSC}} \times Prescaler\ Ratio \times\ 2^{8} = T_{OVERFLOW}$$

The optimum prescaler ratio depends on the rate at which you wish to trigger interrupts. Select the smallest prescaler that creates an interval just longer than you need when the timer is counting its full range. The count can then be shortened by setting the match value to get close to the desired interval.

If for instance; if you select a prescaler ratio of 1:64 with a 4MHz clk in 8-bit mode the full span of the timer would be:

$$^{4}/_{4MHz} \times 64 \times 2^{8} = 16.384mS$$

If an interval of 15mS is desired then reduce the time span by writing a match value to the TMR0H register. When the TMR0 count is equal to the TMR0H register the output will be asserted for one cycle and a new cycle will begin.

$$^{4}/_{4MHz} \times 64 \times (?+1) \approx 15mS$$

For detailed information about the registers associated with TMR0 see section 26 of the PIC16f18324 datasheet.

About Interrupts: The usefulness of the timer modules in a microcontroller is greatly enhanced by its ability to trigger an interrupt when the count overflow or match occurs. To enable interrupts for a TMR0 event, set the TMR0 overflow interrupt enable bit (TMR0IE) in the PIE0 register and the Peripheral Interrupt Enable (PEIE) and Global Interrupt Enable (GIE) bits in the INTCON register.

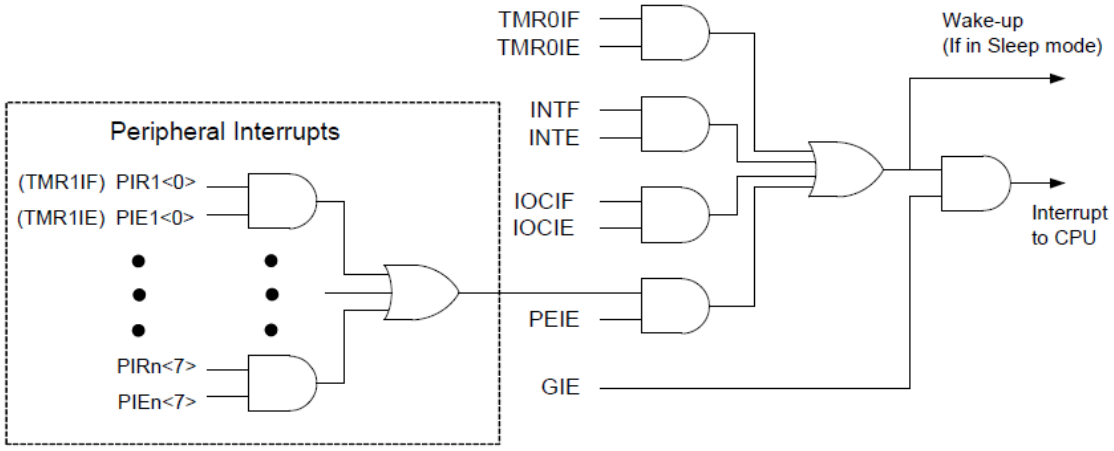

Figure 2 Interrupt Logic

Interrupts on a Mid-Range 8-bit PIC microcontroller are very basic due to their single interrupt vector address. To specify an interrupt using the XC8 compiler you create a function using the ***interrupt*** qualifier. If you are using multiple interrupt sources it is essential that you test for what caused the interrupt by performing interrupt source checks.

```
void interrupt my_isr (void)
{
  If (TMR0IF && TMR0IE)        // Source Check for TMR0
  {
    TMR0IF = 0;                // Clear TMR0 Flag

    // Put your interrupt service code here

  }
}
```

To shape the pulse you can use the _delay() function to make short delays at a rate of Fosc/4. When called, this routine expands to an in-line assembly delay sequence. The sequence will consist of code that delays for the number of instruction cycles that is specified as the argument. From my tests, the delay time appears to be off by one cycle. Use **_delay(0)** to provide a delay of one cycle.

```
LATC5 = 1;       // Start 15uS Pulse (4MHz clk)
_delay(13);      // Delay 14uS
LATC5 = 0;       // End Pulse
```

Registers: Here are some registers you may be working with today.
Open the Datasheet!

REGISTER 25-3: T0CON0: TIMER0 CONTROL REGISTER 0

R/W-0/0	U-0	R-0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
T0EN	—	T0OUT	T016BIT	T0OUTPS<3:0>			
bit 7							bit 0

REGISTER 25-4: T0CON1: TIMER0 CONTROL REGISTER 1

R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0	R/W-0/0
T0CS<2:0>			T0ASYNC	T0CKPS<3:0>			
bit 7							bit 0

REGISTER 25-2: TMR0H: TIMER0 PERIOD REGISTER

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
TMR0H<7:0> or TMR0<15:8>							
bit 7							bit 0

REGISTER 7-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0/0	R/W-0/0	U-0	U-0	U-0	U-0	U-0	R-1/1
GIE	PEIE	—	—	—	—	—	INTEDG
bit 7							bit 0

REGISTER 7-2: PIE0: PERIPHERAL INTERRUPT ENABLE REGISTER 0

U-0	U-0	R/W/HS-0/0	R-0	U-0	U-0	U-0	R/W/HS-0/0
—	—	TMR0IE	IOCIE	—	—	—	INTE
bit 7							bit 0

REGISTER 7-7: PIR0: PERIPHERAL INTERRUPT STATUS REGISTER 0

U-0	U-0	R/W/HS-0/0	R-0	U-0	U-0	U-0	R/W/HS-0/0
—	—	TMR0IF	IOCIF	—	—	—	INTF(1)
bit 7							bit 0

REGISTER 11-18: TRISC: PORTC TRI-STATE REGISTER

R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1	R/W-1/1
TRISC7(1)	TRISC6(1)	TRISC5	TRISC4	TRISC3	TRISC2	TRISC1	TRISC0
bit 7							bit 0

REGISTER 11-19: LATC: PORTC DATA LATCH REGISTER

R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u	R/W-x/u
LATC7(1)	LATC6(1)	LATC5	LATC4	LATC3	LATC2	LATC1	LATC0
bit 7							bit 0

PIC16F18324 Pinout: