PIC HOA 5

March 4, 2021 2:15 PM

Program Design/Psuedo code:
implement timer0

setup
    setup LATA
    setup Timer0

Main
    Heartbeat left running toggling ra7
    runtimer

userapp
    create a static var to keep track of total ms passed
    create static var to track previous LATA 0:5 values

    set LEDs to odd bits high, even bits low
    if timer is up Toggle led bits

In this activity, you will write a non-blocking function that can be used to set a variable timer to clock a period between 1us and 65535us (can you guess the variable type of the function argument already?). This is far better than the library "delay" function that a few people discovered during previous activities. That function is blocking, and you also have 0 idea how it works.

2^16 = 65,536
Will be using u16

24.3.2 Timer0 Interrupt

The Timer0 Interrupt Flag (TMR0IF) bit is set when the TMR0_out toggles. If the Timer0 interrupt is enabled (TMR0IE), the CPU will be interrupted when the TMR0IF bit is set. When the postscaler bits (T0OUTPS) are set to 1:1 operation (no division), the T0IF flag bit will be set with every TMR0 match or rollover. In general, the TMR0IF flag bit will be set every T0OUTPS +1 matches or rollovers.

This is how you will know that the timed period is up. Take a moment to find what register that bit lives in by searching for it in the user guide. Write down the name of the register.

To keep things simple, you will clock Timer0 from the same clock source as the processor's clock. As you know, the processor oscillator frequency is 64MHz and executes instructions at Fosc/4 = 16MHz. So how long is one instruction cycle in nanoseconds? Hint: T = 1/f.

To keep things simple, you will clock Timer0 from the same clock source as the processor's clock. As you know, the processor oscillator frequency is 64MHz and executes instructions at Fosc/4 = 16MHz. So how long is one instruction cycle in nanoseconds? Hint: T = 1/f.

The goal of the timing function is to provide a variable timer between 1us and 65,535us. If you clock the timer directly from Fosc/4 and the timer can count up to 16 bits, can you achieve the objective timing using Fosc/4? Hint: How many clock cycles in 1us? How many clock cycles in 65,535us?

Since the timer can only count to 65,535 (because it is 16 bits), you should see that you will need to slow down the timer's clock frequency using a prescaler. The prescaler will divide the

input clock frequency that the timer sees down to a slower value. If you're paying attention, you should already know by how much you need to prescale. If you don't know, figure it out now. The postscaler can remain 1:1 as we do not need to extend the timer any further.

So now we have the following information ready:
Timer clock mode: Asynchronous
Timer size: 16-bits
Timer clock source: Fosc/4
Timer prescaler: _____
Timer postscaler: 1:1

You'll also want to ensure that Timer0 is enabled. Using this information, look at the Timer0 registers and determine the configuration values you will need for the following registers:

Register	Binary Init Value	Hex Init Value
T0CON0		
T0CON1		

Your LED update code must work as follows:

1. Read LATA to a temporary variable
2. Use a bitmask and bitwise operation to clear the 6 LSBs
3. Use a bitwise operation to update the 6 LSBs to the new value you want
4. Write the temporary variable back to LATA
5. Make sure all of this is done using 8-bit variables.

If you didn't write your code this way, update it now and make sure it still works.

Since the UserApp is the one that is using Timer0, write all the timer code there. Start by adding two lines of code to UserAppInitialize() to load the two timer control registers with the values you identified above. Try adding some comments to help guide the reader as to what you're up to.

```
void UserAppInitialize(void)
{
    /* LED initialization */
    LATA  = 0x80;

    /* Timer0 control register initialization to turn timer on, asynch mode, 16 bit
     * Fosc/4, 1:x prescaler, 1:1 postscaler  */


} /* end UserAppInitialize() */
```