Introduction to PIC Programming

Programming Midrange PICs in C

by David Meiklejohn, Gooligum Electronics

Lesson 10: Using Timer2

Midrange assembler lesson 16 introduced a simple 8-bit timer, Timer2, showing how its *period register* can be used to generate a specific timer period, and how its *postscaler* can be used to reduce the rate that timer interrupts are triggered. We saw that these features make Timer2 ideal for generating a specific time-base for interrupt-driven tasks.

This lesson revisits that material, showing how to use C to control and access Timer1, re-implementing the examples using the free HI-TECH C[1] (in "Lite" mode) and PICC-Lite compilers.

In summary, this lesson covers:

- Introduction to the Timer2 module
- Using the Timer2 postscaler to drive an interrupt at a reduced rate
- Using the Timer2 period register to generate a specific time-base

with an example program for HI-TECH C and PICC-Lite.

Timer2 Module

The current value of Timer2 is held in a single 8-bit register: TMR2.

Associated with TMR2 is an 8-bit *period register*: PR2, which specifies the maximum value that TMR2 will reach before it is reset (rolls over) to zero on the next timer increment.

Thus, Timer2's period is equal to the value stored in PR2, plus one.

As usual, the HI-TECH C compilers make these available as `unsigned char` (8-bit) variables, `TMR2` and `PR2`.

Timer2 can only be driven by the instruction clock (FOSC/4). Thus, Timer2 can only be used as a timer; it cannot be used to count external events.

Timer2 is configured using the T2CON register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
T2CON	–	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0

[1] PICC-Lite was bundled with versions of MPLAB up to 8.10. HI-TECH C (earlier known as "HI-TECH C PRO") was bundled with MPLAB 8.15 and later, although you should download the latest version from www.microchip.com.

Like Timer1, Timer2 can be turned on or off, using the TMR2ON bit: setting it to '1' to enables Timer2, and clearing it to '0' stops it.

Prescaler

Like the other timers, Timer2 includes a prescaler, so that the timer does not have to increment on every clock cycle.

The prescale ratio is set by the T2CKPS<1:0> bits, as shown in the following table:

T2CKPS<1:0> bit value	Timer2 prescale ratio
00	1 : 1
01	1 : 4
10 or 11	1 : 16

T2CKPS<1:0> = '00' means that no prescaling will occur, and TMR2 will increment at the instruction cycle rate.

T2CKPS1 = '1' (regardless of the value of T2CKPS0) selects the maximum prescale ratio of 1:16, meaning that TMR2 will increment every 16 instruction cycles. Given a 1 MHz instruction cycle rate, the timer would increment every 16 µs.

Postscaler and Timer2 interrupts

Unlike the other timers, Timer2 also includes a *postscaler*.

A postscaler does not affect how quickly the timer increments.

Instead, it affects how often the Timer2 interrupt flag, TMR2IF, flag in the PIR1 register, is set:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PIR1	EEIF	ADIF	CCP1IF	C2IF	C1IF	OSFIF	TMR2IF	TMR1IF

This is similar to the other timers' interrupt flags, except that, instead of indicating a timer overflow, the TMR2IF flag indicates that a match between TMR2 and PR2 has occurred.

If the postscaler is active (TOUTPS<3:0> ≠ '0000'), this match has to occur some number of times before TMR2IF is set.

The postscale ratio is equal to the value of TOUTPS<3:0> plus one.

Thus, the postscale ratio ranges from 1:1 (TOUTPS<3:0> = '0000') to 1:16 (TOUTPS<3:0> = '1111').

So, if TOUTPS<3:0> = '1001' (binary) = 9, the postscale ratio will be 1:10, and the match between TMR2 and PR2 will have to occur 10 times before TMR2IF is set.

The Timer2 interrupt is enabled by setting the TMR2IE enable bit is in the PIE1 register:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PIE1	EEIE	ADIE	CCP1IE	C2IE	C1IE	OSFIE	TMR2IE	TMR1IE

And of course, being a peripheral interrupt, to enable Timer2 interrupts you must also set the peripheral and global interrupt enable bits (PEIE and GIE) in the INTCON register.

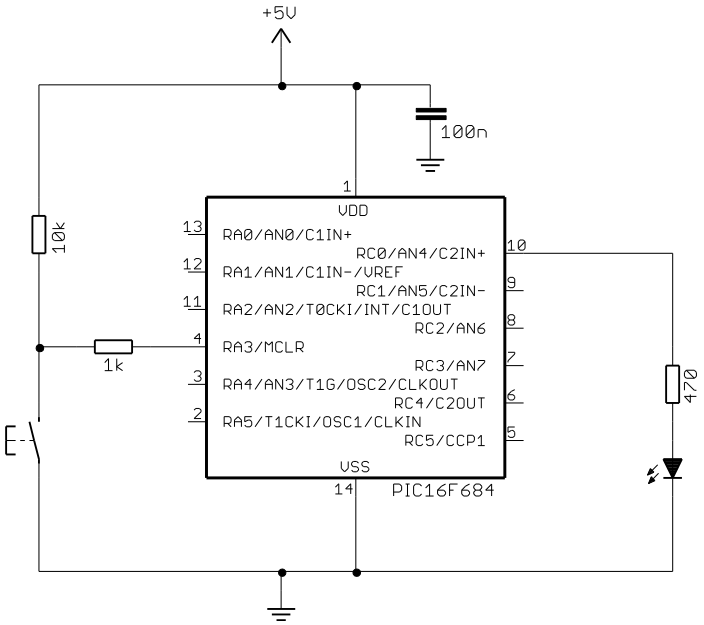
The postscaler effectively slows the interrupt rate, allowing Timer2 to generate a longer time-base.

An example will help to illustrate these concepts…

Example 1: Flash an LED at exactly 1 Hz

Lesson 3 included an example where an LED was flashed at exactly 1 Hz, using a Timer0 interrupt. To do so, the interrupt handler had to add an appropriate offset to the timer, and, to ensure accuracy, the prescaler could not be used.

In this example, we will see that it is much simpler to achieve the same result using Timer2.



We will use the circuit shown on the right, based on a PIC16F684 with an LED on RC0, which we will flash at 1 Hz.

As usual, you can use Microchip's Low Pin Count Demo Board, where an LED (labelled DS1) is already connected to RC0.

Using Timer2's period register, we can have TMR2 reset automatically, with a period that divides evenly into 1 s.

Given the default 4 MHz processor clock, we'll need to toggle the LED every 500,000 instruction cycles.

The largest value, less than or equal to 256, that divides exactly into 500,000 is 250.

To generate a period of 250 counts, we need to load the value 249 (= 250 – 1) into PR2:

```
PR2 = 249;                  // Timer2 period = 250 clocks (PR2 = period-1)
```

If we set the prescaler to 1:4, TMR2 increments every 4 µs, and will match PR2 every 250 × 4 µs = 1 ms.

And if we set the postscaler to 1:10, the interrupt will be triggered on every 10th match, i.e. after 10 ms.

To configure Timer2 in this way, we can use:

```
T2CON = 0b01001101;             // configure Timer2:
        //-1001---                   postscale = 10 (TOUTPS = 1001)
        //-----1--                   enable Timer2 (TMR2ON = 0)
        //------01                   prescale = 4 (T2CKPS = 01)
```

or, if using HI-TECH C version 9.81, you could express this as:

```
                                // configure Timer2:
T2CONbits.T2CKPS = 1;           //   prescale = 4
T2CONbits.TOUTPS = 9;           //   postscale = 10
T2CONbits.TMR2ON = 1;           //   enable Timer2
```

We can then enable the Timer2 interrupt:

```
TMR2IE = 1;                     // enable Timer2 interrupt
PEIE = 1;                       // enable peripheral
ei();                           //   and global interrupts
```

The ISR needs to maintain a count, so that it can toggle the LED after it has been run 50 times, to generate a total period of 500 ms – and for that we need a static variable, defined at the start of the interrupt function:

```
    static unsigned int     cnt_10ms = 0;   // counts 10 ms periods
```

The ISR can then increment this variable each time it is called, toggling the LED after 500 ms (when the count = 50):

```
    TMR2IF = 0;                     // clear interrupt flag

    // toggle LED every 500 ms
    ++cnt_10ms;                     // increment 10 ms period count
    if (cnt_10ms == FlashMS/10)     // if we've counted for 500 ms,
    {
        cnt_10ms = 0;               //   reset count
        sPORTC ^= 1<<nF_LED;        //   toggle LED (using shadow register)
    }
```

where the constant `FlashMS` is defined by:

```
#define FlashMS 500                 // LED flash toggle time in milliseconds
```

Note that the interrupt flag is cleared at the beginning of the interrupt handler, and, as we've done before, a shadow register is used when toggling the LED port bit, to avoid potential read-modify-write issues.

That means that all the main loop has to do is copy this shadow register to the port:

```
    // Main loop
    for (;;)
    {
        // copy shadow register (updated by ISR) to port
        PORTC = sPORTC;
    }
```

Complete program

This is how these pieces fit together, for HI-TECH C v9.81:

```
/************************************************************************
*                                                                       *
*   Description:    Lesson 10, example 1                                *
*                                                                       *
*   Demonstrates use of Timer2 to perform generate a specific time-base *
*   for an interrupt-driven background task                             *
*                                                                       *
*   Flash an LED at exactly 1 Hz (50% duty cycle).                      *
*                                                                       *
************************************************************************
*                                                                       *
*   Pin assignments:                                                    *
*       RC0 - flashing LED                                              *
*                                                                       *
************************************************************************/

#include <htc.h>

/***** CONFIGURATION *****/
//  ext reset, no code or data protect, no brownout detect,
//  no watchdog, power-up timer, int clock with I/O,
```

```
//  no failsafe clock monitor, two-speed start-up disabled
__CONFIG(MCLRE_ON & CP_OFF & CPD_OFF & BOREN_OFF & WDTE_OFF & PWRTE_ON &
         FOSC_INTOSCIO & FCMEN_OFF & IESO_OFF);

// Pin assignments
#define nF_LED  0                       // flashing LED on RC0


/***** CONSTANTS *****/
#define FlashMS 500                     // LED flash toggle time in milliseconds


/***** GLOBAL VARIABLES *****/
volatile unsigned char  sPORTC;         // shadow copy of PORTC


/***** MAIN PROGRAM *****/
void main()
{
    // Initialisation

    // configure ports
    PORTC = 0;                  // start with PORTC clear (LED off)
    sPORTC = 0;                 //   and update shadow
    TRISC = ~(1<<nF_LED);       // configure LED pin (only) as an output

    // configure timers
    PR2 = 249;                  // Timer2 period = 250 clocks (PR2 = period-1)
                                // configure Timer2:
    T2CONbits.T2CKPS = 1;       //   prescale = 4
    T2CONbits.TOUTPS = 9;       //   postscale = 10
    T2CONbits.TMR2ON = 1;       //   enable Timer2
    TMR2IE = 1;                 // enable Timer2 interrupt

    // configure interrupts
    // enable interrupts
    PEIE = 1;                   // enable peripheral
    ei();                       //   and global interrupts

    // Main loop
    for (;;)
    {
        // copy shadow register (updated by ISR) to port
        PORTC = sPORTC;
    }
}


/***** INTERRUPT SERVICE ROUTINE *****/
void interrupt isr(void)
{
    static unsigned int     cnt_10ms = 0;   // counts 10 ms periods

    // *** Service Timer2 interrupt
    //
    // Runs every 10 ms
    //  (every 10th TMR2/PR2 match;
    //   TMR2 matches PR2 every 250 x 4 clocks = 1000 us)
    //
    // Flashes LED at 1 Hz
    //  by toggling on every 50th interrupt (every 500 ms)
```

```
    //
    //  (only Timer2 interrupts are enabled)
    //
    TMR2IF = 0;                     // clear interrupt flag

    // toggle LED every 500 ms
    ++cnt_10ms;                     // increment 10 ms period count
    if (cnt_10ms == FlashMS/10)     // if we've counted for 500 ms,
    {
        cnt_10ms = 0;               //   reset count
        sPORTC ^= 1<<nF_LED;        //   toggle LED (using shadow register)
    }
}
```

We really only need this one example to demonstrate Timer2's features. But despite its simplicity, hopefully this lesson has shown that Timer2 is a useful, and easy-to-use, timer.

Now that we've seen all the timers, we can turn our attention to one of the most powerful midrange PIC peripherals, the Capture/Compare/PWM module, which we'll introduce in the next lesson, beginning with its capture and compare modes.