Timer0..Timer3

Timers are pretty useful: likewise, Microchip provides four different timers for you to use. Like all interrupts, you have to

- Enable the interrupt,
- Set the conditions of the interrupt, and then
- Do something when the interrupt happens (check the interrupt flag)

These hoops are summarized in the following table for all four timer interrupts.

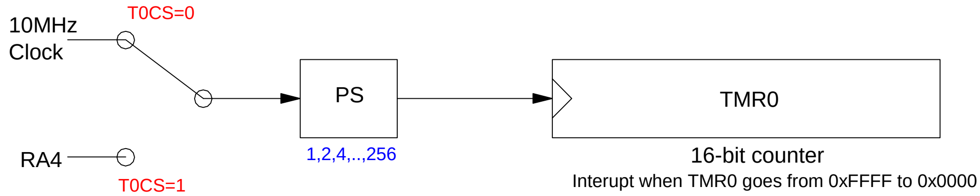
Interrupt	Description	Input	Conditions	Enable	Flag
Timer 0	Trigger after N events N = 1 .. 2^{24} 100ns to 1.67 sec	RA4: TOCS = 1 OSC/4: TOCS = 0	N = (PS)(Y) T0CON = 0x88: PS = 1 T0CON = 0x80: PS = 2 T0CON = 0x81: PS = 4 T0CON = 0x82: PS = 8 T0CON = 0x83: PS = 16 T0CON = 0x84: PS = 32 T0CON = 0x85: PS = 64 T0CON = 0x86: PS = 128 T0CON = 0x87: PS = 256 TMR0 = -Y	TMR0ON = 1 TMR0IE = 1 TMR0IP = 1 PEIE = 1	TMR0IF
Timer 1	Trigger after N events N = 1 .. 2^{19} 100ns to 0.52 sec	RC0 TMR1CS = 1 OSC/4 TMR1CS = 0	N = (PS)(Y) T1CON = 0x81: PS = 1 T1CON = 0x91: PS = 2 T1CON = 0xA1: PS = 4 T1CON = 0xB1: PS = 8 TMR1 = -Y	TMR1ON = 1 TMR1IE = 1 TMR1IP = 1 PEIE = 1	TMR1IF
Timer2	Interupt every N clocks (OSC/4) N = 1 .. 65,535 100ns to 6.55ms	OSC/4	N = A * B * C A = 1..16 (T2CON 3:6) B = 1..256 (PR2) C = 1, 4, 16 (T2CON 0:1)	T2E = 1 TMR2IE = 1 PEIE = 1	TMR2IF
Timer 3	Trigger after N events N = 1 .. 2^{19} 100ns to 0.52 sec	RC1 TMR3CS = 1 OSC/4 TMR3CS = 0	N = (PS)(Y) T3CON = 0x81: PS = 1 T3CON = 0x91: PS = 2 T3CON = 0xA1: PS = 4 T3CON = 0xB1: PS = 8 TMR3 = -Y	TMR3ON = 1 TMR3IE = 1 TMR3IP = 1 PEIE = 1	TMR3IF

Timer0, Timer1, and Timer3 are slightly different than Timer2. Each of these three are very similar, so let's take Timer0 for example.

The input to Timer0 can be either the on-board clock (10MHz) or an external pin. This lets you interrupt every N edges for an external event or every N clocks.

The input goes to a frequency divider, PS, which drops the frequency by 2^N for N = 0 to 8.

The output of the divider then goes to a 16-bit counter, TMR0. When the counter wraps around from 0xFFFF to 0x0000, a Timer0 interrupt is triggered.



Timer0 Interrupt is triggered every 100 to 2^{24} clocks (1.67 seconds)

Default Timer0 Rate with PS = 1:

For example, turn on Timer0

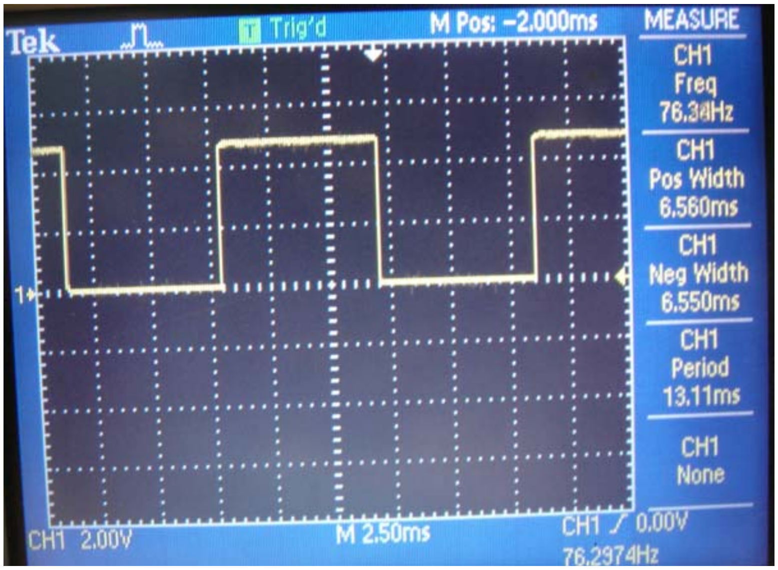
- With a prescalar of 1
- And toggle RC0 every interrupt

When Timer0 interrupts, you know that

TMR0 = 0x0000

Wrap around is what triggers the interrupt. The next interrupt won't happen again for another 65,536 clocks (216 counts). This results in Timer0 interrupting every 6.5536ms (the default rate for Timer0).

```
// Global Variables

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR0IF) {
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }

// Main Routine

void main(void)
{
   TRISA = 0;
   TRISB = 0;
   TRISC = 0;
   TRISD = 0;
   ADCON1 = 0x0F;

// set up Timer0 with PS = 1
   T0CS = 0;
   T0CON = 0x88;
   TMR0ON = 1
   TMR0IE = 1
   TMR0IP = 1
   PEIE = 1

// turn on all interrupts
   GIE = 1;

// Do nothing.  Interrupts do all the work.
   while(1) {
      PORTB = PORTB + 1;
      }
   }
```

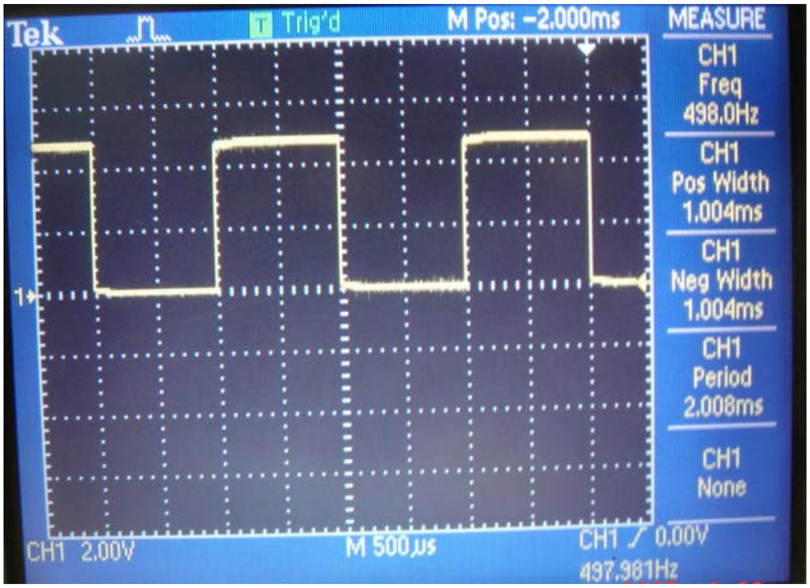


Signal at RC0: Default Timer0 Interrupt Rate with PS = 1: 65,536 Clocks

Timer0 Interrupt every 1ms

Example 2: Change the interrupt service routine to

```
void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -10000;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```



Timer0 with N = 10,000 (1.00 ms)

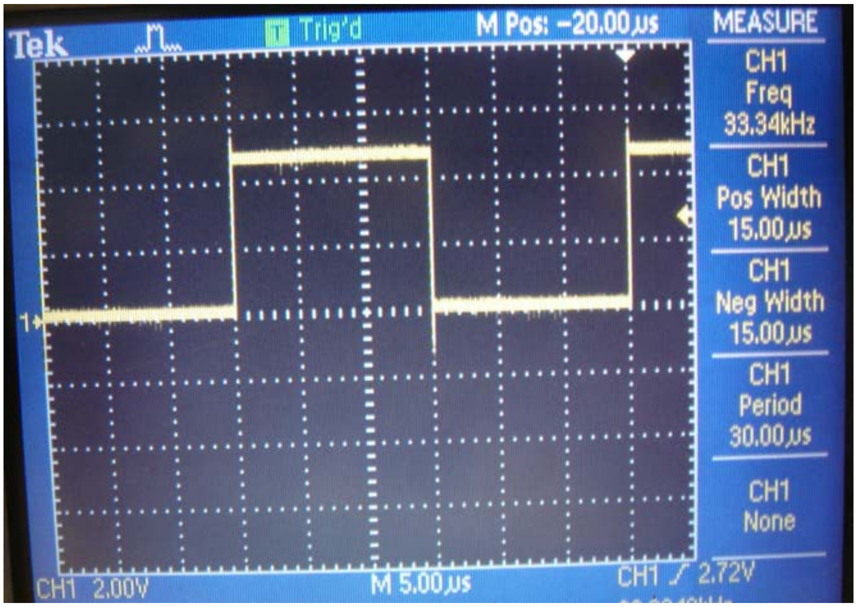
When Timer0 interrupts, TMR0 = 0x0000. Wrapping around to zero is what causes the interrupt. By changing TMR0 to -10,000, you are setting up the next interrupt to happen 1ms in the future (10,000 clocks). This results in RC0 toggling every 10,000 clocks.

Measuring the time it takes to trigger an interrupt:

Actually, TMR0 toggles every 10,000 clocks - plus however many clocks it takes to trigger the interrupt and get to that line of code. To see how long this takes, the following code is used:

```
// Global Variables
unsigned int N = 100;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -N;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```

You would expect RC0 to toggle every 100 clocks (N = 100). Actually, it toggles every 150 clocks (15us). This tells you it takes about 50 clocks to trigger an interrupt.



Timer0 with N = 100. RC0 toggles every 150 clocks - telling you that it takes 50 clocks to call the interrupt.

Playing Note D5: 587.33Hz

Example 3: Generate the note D5 (587.33Hz) on RC0.

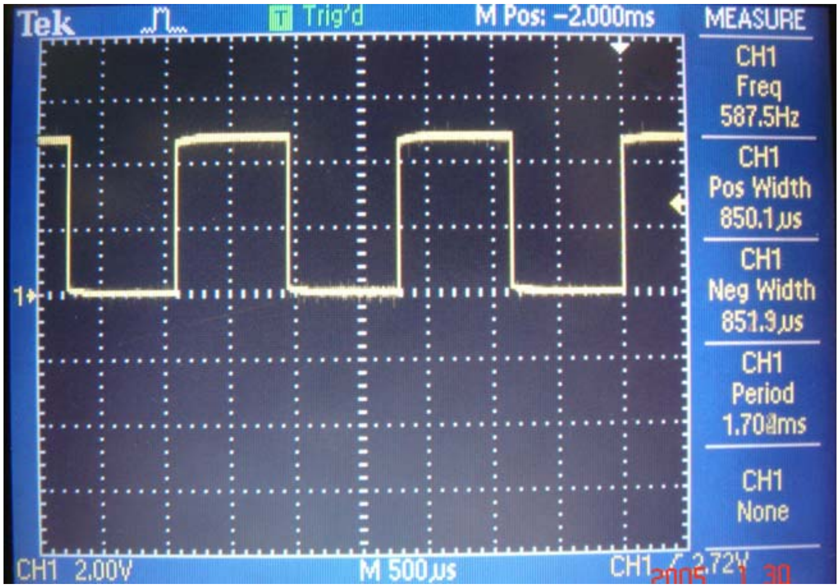
The number of clocks between interrupts should be

$$N = \left(\frac{10{,}000{,}000}{2 \cdot Hz}\right) = 8513.1017$$

Change the previous code to:

```
// Global Variables
unsigned int N = 8513 - 50;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -N;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```

and you get almost the exact frequency you wanted. Timer0 interrupts are *way* easier that Timer2 interrupts (or figuring out the timing using assembler).



Timer0 with N = 8513 - 50 (587.33Hz)

Measuring Time to 100ns

You can also use Timer0 to measure time to 100ns. For example, measure the time of the routine

```
Wait_ms(1000);
```

Here you have a problem: the longest time TMR0 can measure is 6.55ms (65,536 clocks). To measure longer times, create a 32-bit counter:

- The low 16-bits are TMR0
- The high 16-bits are stored in a variable, TIME
- Every Timer0 interrupt, add 0x10000 to TIME (one Timer0 overflow).

In the main routine then

- Record the time before calling Wait_ms(1000)
- Record the time after call it, and
- Take the difference.

Note that you also have to modify the LCD_Out() routine to display long integers (changes in red)

LCD Routine (modified)

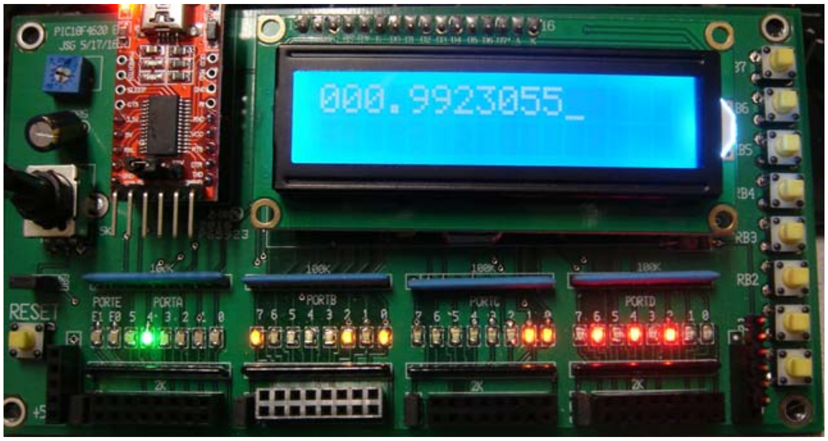
```
void LCD_Out(unsigned long int DATA, unsigned char N)
{
   unsigned char A[10], i;

   for (i=0; i<10; i++) {
      A[i] = DATA % 10;
      DATA = DATA / 10;
   }
   for (i=10; i>0; i--) {
      if (i == N) LCD_Write('.');
      LCD_Write(A[i-1] + '0');
   }
}
```

Interrupt Service Routine:

```
// Global Variables
unsigned long int TIME;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR0IF) {
      TIME = TIME + 0x10000;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   }
```

Main Routine: Main loop

```
   while(1) {

      TIME0 = TIME + TMR0;               // time before call
      Wait_ms(1000);
      TIME1 = TIME + TMR0;               // time after call
      LCD_Move(0,0);
      LCD_Out(TIME1 - TIME0, 7);         // time difference

      }
```



The time it takes to execute Wait(1000) as measured using Timer0 interrupts.

Note that you can measure time to an obscene number of decimal places using Timer interrupts.