Examples of Timer2 Interrupts:

Once you can keep track of time, there's lots of things you can do. A short list is:

- Build a better wait routine
- Build a stopwatch that's accurate to 0.0001 second (N=1,000)
- Generate musical notes
- Drive a stepper motor (step every 20ms)
- Vary the light output (pulse width modulation)

Better Wait Routine:

Objective:

Write a routine, Wait(unsigned int X), which waits X milliseconds and then returns.

Solution:

1) Set up Timer2 for 1ms (a nice round number)

a) Compute how many clock ticks equals 1ms

- 1ms * (10,000,000 clock / second) = 10,000 clocks

b) Find A, B, C

- A * B * C = 10,000

Let C = 4

- A * B = 2500

Let A = 10, B = 250

c) Find PR2 and TMR2CON

- PR2 = 249 (B = PR2+1)
- (A3:A2:A1:A0) = 9 = b1001 (A = # + 1)
- (C1:C0) = b01
- TMR2CON = b 0100 1101 = 0x4D

T2CON 0x4D

7	6	5	4	3	2	1	0
-	A3	A2	A1	A0	TMR2ON	C1	C0
0	1	0	0	1	1	0	1
	A = 10 (9 + 1)					C = 4	

d) Intialization routine:

```
PR2 = 249;
TMR2CON = 0x4D;
TMR2IE = 1;
PEIE = 1;
```

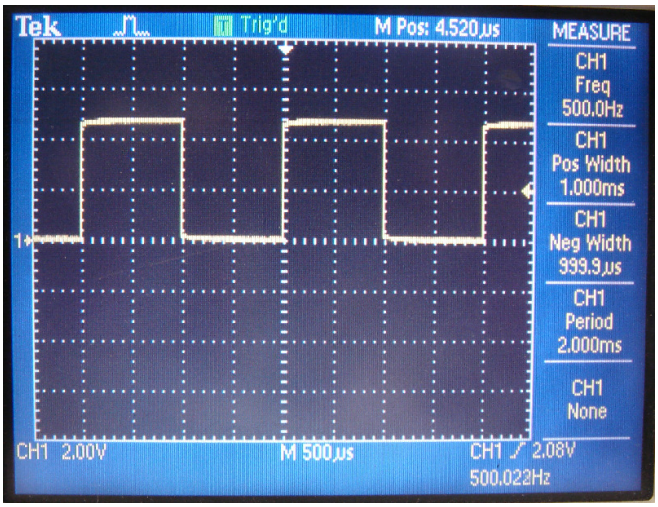
2) Set up the interrupt service routine to do stuff, such as decrement a number to zero stopping at zero:

```
// Global Variables

unsigned int DELAY;

void interrupt IntServe(void)
{
   if (TMR2IF) {
      RA0 = !RA0;
      if (DELAY) DELAY -= 1;
      TMR2IF = 0;
      }
   }
```

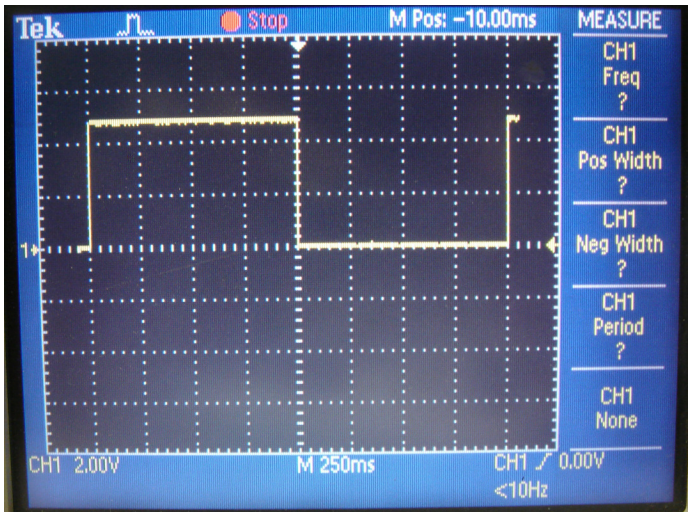
Example: Build a binary clock where PORTC counts in seconds.

```
#include <pic18.h>

// Global Variables
unsigned int DELAY;

// Subroutine Declarations

void interrupt timer2(void)
{
   RA0 = !RA0;
   if (DELAY) DELAY -= 1;
   TMR2IF  = 0;
   }

void main(void)
{
   TRISA = 0;
   TRISB = 0;
   TRISC = 0;
   TRISD = 0;
   ADCON1 = 15;

// initialize Timer2

   T2CON   = 0x4D;
   PR2     = 249;
   TMR2IE  = 1;
   PEIE    = 1;
   TMR2ON  = 1;
   TMR2IP  = 1;

// Turn on all interrupts

   GIE = 1;

   PORTC = 0;

   while(1) {
      while(DELAY);
      DELAY = 1000;

      PORTC = PORTC + 1;
      }
   }
```

Pin RA0 toggles every Timer2 interrupt: this lets you check that the interrupts are running correctly:



RA0 toggles every Timer2 interrupt: Timer2 is set up for 1.000 ms



RC0 counts every 1.000 second (1000 interrupts)

Bulid a Stopwatch, accurate to 0.0001 second

Use the same initialization as before to interrupt every 100us. Each time you interrupt, check the push buttons:

- If RB0 is pressed, stop the stopwatch.
- If RB1 is pressed, start the stopwatch.
- If RB2 is pressed, clear the time.

To set N = 1000, one combination that works is

- A = 1, B = 250, C = 4
- PR2 = 249

T2CON 0x05

7	6	5	4	3	2	1	0
-	A3	A2	A1	A0	TMR2ON	C1	C0
0	0	0	0	0	1	0	1
	A = 1 (0 + 1)					C = 4	

A global variable, TIME, is used to keep track of the current time (1 count = 1ms).

```
#include <pic18.h>
// Global Variables

unsigned int TIME;
unsigned char RUN;

// Subroutines
#include "LCD_PortD.C"

void interrupt IntServe(void)
{
   if (TMR2IF) {
      PORTA += 1;
      if (RB0) RUN = 0;
      if (RB1) RUN = 1;
      if (RB2) TIME = 0;
      if (RUN) TIME += 1;
      TMR2IF = 0;
      }
   }

// main routine

void main(void)
{
   TRISA = 0;
   TRISB = 0xFF;
   TRISC = 0;
   TRISD = 0;
   ADCON1 = 15;
   LCD_Init();

 // initialize Timer2 for 1ms (10,000 clocks)
   PR2 = 249;
   T2CON = 0x4D;
   TMR2IE = 1;
```

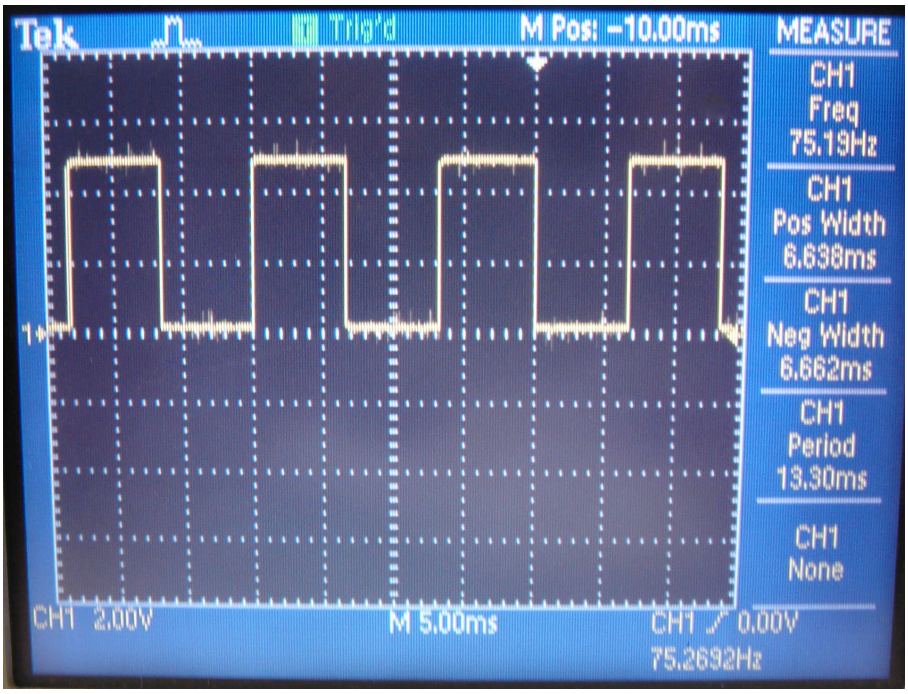
```
        PEIE = 1;
        TMR2IP = 1;

        TIME = 0;
        RUN = 0;
        GIE = 1;

        while(1) {
           RC1 = !RC1;
           LCD_Move(1,0);
           LCD_Out(TIME, 5, 3);
           }
        }
```

If you look at pin RA0, you should see a 5000Hz square wave. For the Piano Tuner app, this is too high in frequency. By counting on PORTA, each pin acts like a divide by 2

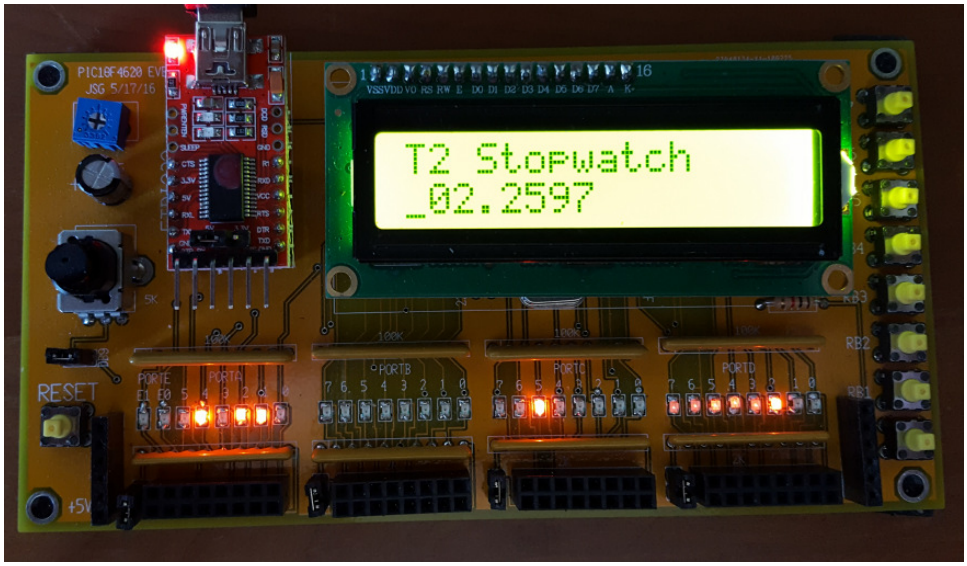
- RA0: 5000Hz
- RA1: 2500Hz
- RA2: 1250Hz
- RA3: 625Hz
- RA4: 312.5Hz

Piano Tuner can then be used for something in the audio rang (RA2 or 1250Hz shown here)

Frequency seen on RA2 (Piano Tuner app)



Signal on RC1: The main routine loops every 6.38ms (doesn't tell you much other than that's how often the LCD display is updated)



LCD Display of Time, Accurate to 1.000ms (one interrupt)

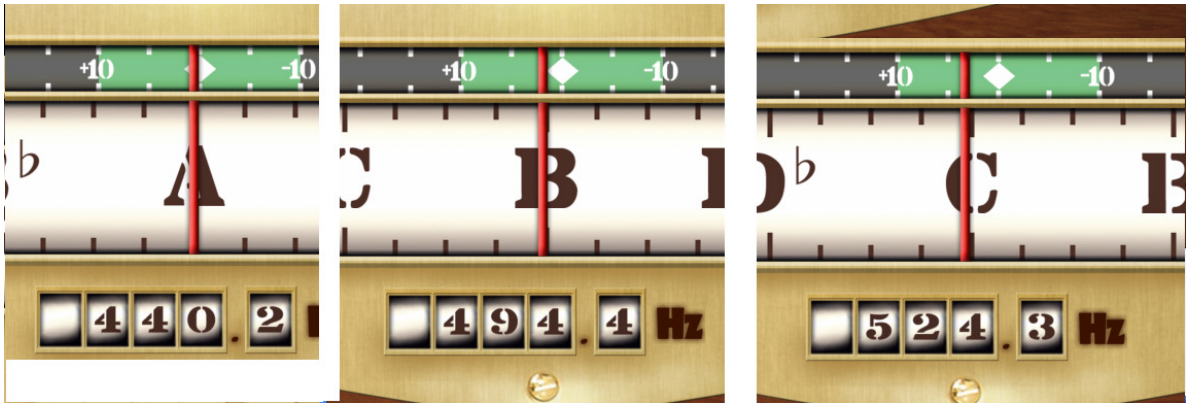
Build a 3-Key piano:

$$N = \left(\frac{10{,}000{,}000}{2 \cdot Hz}\right)$$ Number of clocks between interrupts

Note	A4	B4	C5
Hz	440	493.88	523.25
N	11,363.64	10,123.92	9,555.66
A	12	12	12
B	236.74	210.91	199.08
C	4	4	4

By changing PR2, you change the rate at which the Timer2 interrupt is called. This lets you change the frequency of the square wave output of RA0, creating different notes.

To turn the note of and off, TMR2ON is used to turn on and off TIMER2. When it's off, TIMER2 quits playing. You could also switch RA0 to input to turn off the output as well.



Frequency of music notes (Piano Tuner app)

```
// Global Variables
const unsigned int A4 = 236;
const unsigned int B4 = 210;
const unsigned int C5 = 198;


// Subroutine Declarations
#include <pic18.h>
#include "lcd_portd.c"

void interrupt IntServe(void)
{
 if (TMR2IF) {
    if(PORTB) RA1 = !RA1;
    else RA1 = 0;
    TMR2IF = 0;
    }
 }

// Main Routine
void main(void)
{
 unsigned char i, j;
 TRISA = 0;
 TRISB = 0xFF;
 TRISC = 0;
 TRISD = 0;
 TRISE = 0;
 ADCON1 = 15;

// Timer2 Initialize
 TMR2ON = 1;
 TMR2IE = 1;
 PEIE = 1;
 T2CON = 0x5D; // A=12, C=4 0 1011 1 01
 PR2 = 49;
 GIE = 1;

 while(1) {
    if (RB0) PR2 = A4;
    if (RB1) PR2 = B4;
    if (RB2) PR2 = C5;
    };
 }
```

Stepper Motor:

Write a program which

- Steps the stepper motor every 20ms (200,000 clocks), and
- Displays the total number of steps on the LCD display.

Note that 20ms is a problem with Timer2: it max's out at 6.55ms. Instead of stepping e*very* interrupt, you could step every 100th interrupt. This results in

- N = 2,000
- A = 10, B = 200, C = 1

T2CON 0x4C

7	6	5	4	3	2	1	0
-	A3	A2	A1	A0	TMR2ON	C1	C0
0	1	0	0	1	1	0	0
	A = 10 (9 + 1)					C = 1	

The interrupt service routine steps the motor every 100th interrupt:

```
void interrupt IS(void)
{
   N = (N + 1) % 100;
   if(N == 0) {
      STEP += 1;
      PORTC = TABLE[STEP % 4];
      RA1 = !RA1;
      }
   TMR2IF = 0;
   }
```

The main routine just displays the total number of steps

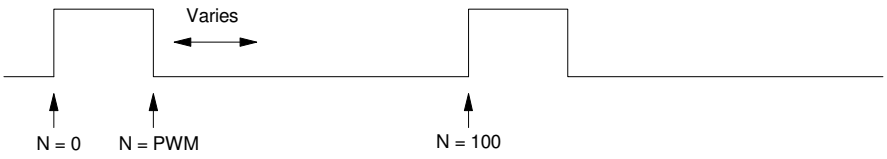
```
while(1) {
   LCD_Move(0,8);
   LCD_Write(STEP, 5, 0);
   }
```

With 200 steps per rotation and 20ms per step, this results in the stepper motor making one rotation (200 steps) every 4.00 seconds.

Pulse Width Modulation

Finally, let's write a program which allows you to adjust the brightness of the LEDs with 64 levels of grey. To do this, use pulse-width modulation

The idea behind PWM is to create a square wave whose duty cycle varies from 0% to 100%. The DC (average) votlage then varies from 0V (0%) to 5.00V (100%).



PWM: Vary the duty cycle to approximate an analog (DC) voltage

One way to do this is as follows:

Setp up Timer2 to interrupt every 200 clocks

- A = 1
- B = 200
- C = 1

The interrupt service routine then

- Counts mod 64,
- Sets PORTC for PWM counts, and
- Clears PORTC for the remainder

```
void interrupt IS(void)
{
   N = (N + 1) % 64;
   if(N < PWM) PORTC = 0x3F;
   else PORTC = 0;

   if(N == 0) RA1 = !RA1;

   TMR2IF = 0;

   }
```

The main routine can then communicate with the interrupt service routine through the global variable, PWM:

```
while(1) {

   if(RB0) PWM = 0;
   if(RB1) PWM = 9;
   if(RB2) PWM = 18;
   if(RB3) PWM = 27;
   if(RB4) PWM = 36;
   if(RB5) PWM = 45;
   if(RB6) PWM = 54;
   if(RB7) PWM = 64;

   LCD_Move(1,0);
   LCD_Write(PWM, 3, 0);
   }
```

To verify that this is working, you can

- Check the frequency of the PWM signal (pin RA1)
- Check the DC votlage seen on PORTC

Signal or RA1.

- N = 200 * 64 = 12,800
- f = (10,000,000) / 2N
- f = 390.625Hz

Frequency of signal on RA1 for PWM.

Votlage on PORTC:

- RB7: 4.55V
- RB6: 3.84V
- RB5: 3.20V
- RB4: 2.56V
- RB3: 1.92V
- RB2: 1.28V
- RB1: 0.64V
- RB0: 0.00