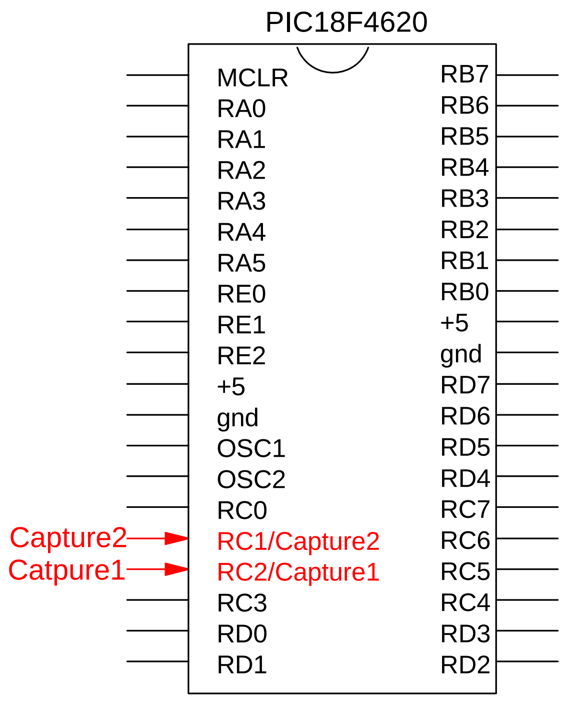
Timer1 Capture Mode:

Interrupt	Description	Input	Conditions	Enable	Flag
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
Timer 1 Capture Mode 1	On an event, record the TIMER1 counter and trigger an interrupt. Time of the event is stored in CCPR1	RC2	CCP1CON = 0x04: every rising edge CCP1CON = 0x05: every falling edge CCP1CON = 0x06: 4th rising edge CCP1CON = 0x07: 16th rising edge	TMR1ON = 1 CCP1IE = 1 PEIE = 1;	CCP1IF
Timer 1 Capture Mode 2	On an event, record the TIMER1 counter and trigger an interrupt. Time of the event is stored in CCPR2	RC1	CCP2CON = 0x04: every rising edge CCP2CON = 0x05: every falling edge CCP2CON = 0x06: 4th rising edge CCP2CON = 0x07: 16th rising edge	TMR1ON = 1 CCP2IE = 1 PEIE = 1;	CCP2IF

A PIC processor is able to measure time to 100ns. If you want to record the time of an event with time known to the clock, use Timer1 Capture Mode.

There are two Timer1 Capture interrupts:

- Capture Mode 1 records the time of an event on pin RC2
- Capture Mode 2 records the time of an event on pin RC1 (yes, it's backwards)

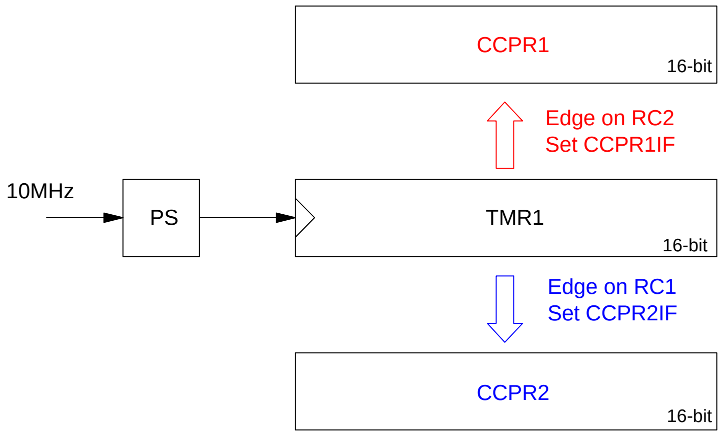


Capture Interrupts Record the time of an event on RC2 (Capture1) and RC1 (Capture2)

Timer1 Capture Description

What a Time1 capture interrupt does is

- When an event is observed on RC1 or RC2 (rising or falling edge),
  - The time stored in TMR1 is copied to a register, and
  - A Timer1 Capture interrupt is triggered.



Timer1 Capture1 and Capture2: On an event, the time stored in TMR1 is saved

To get Timer1 Capture to work

i) You have to turn on Timer1.

ii) You have to set the condition for Timer1 Capture interrupt:

• Capture every falling edge:	CCP1CON = 0x04
• Capture every riding edge:	CCP1CON = 0x05
• Capture every 4th rising edge:	CCP1CON = 0x06
• Capture every 16th rising edge:	CCP1CON = 0x07

iii) Pin RC1 or RC2 have to be input

iv) You have to enable the Timer1 Capture interrupt (various flags)

At that point, whenever the corresponding edge is detected on RC1 (Capture2) or RC2 (Capture1),

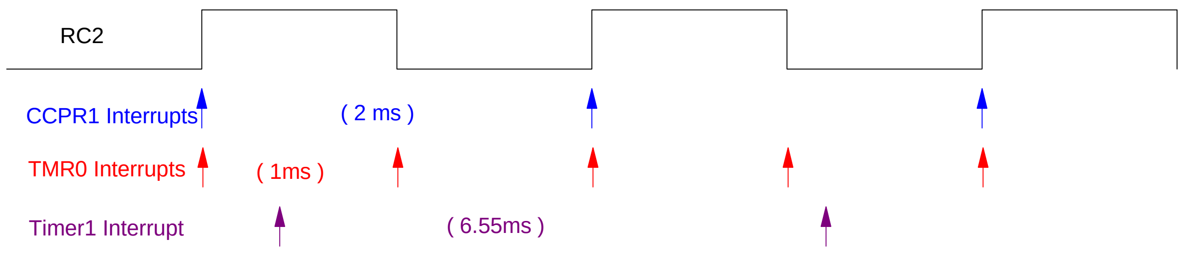
- The value of TMR1 is copied to CCPR1 or CCPR2
- A Timer1 Capture interrupt is triggered.

Example 1: Measure the frequency of a square wave (Capture1.C)

- Generate a 500Hz square wave on RC0 using Timer0.
- Connect RC0 to RC2
- Measure the period of this square wave using Capture1 interrupts.

Here, we have three interrupts running in parallel:

- Timer0 interrupts every 10,000 clocks (1ms)
- Timer1 interrupts every 65,536 clocks (6.55ms)
- Capture1interrupts trigger every rising edge on RC2 (2ms)



Three interrupts running in parallel:

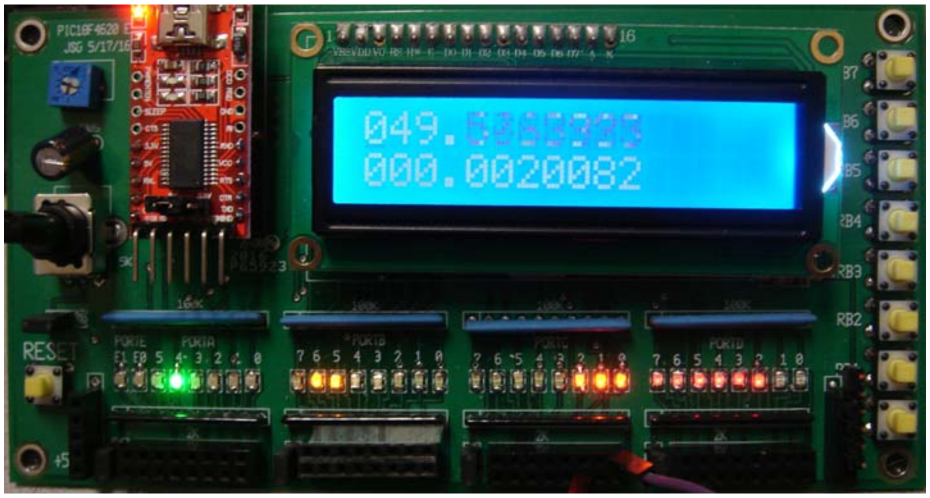
Timer1 is turned on with a prescalar of 1 - but there is no need to trigger a Timer1 interrupt. The event we're measuring takes less than 65,536 clocks (6.5ms) so we don't have to worry about Timer1 wrapping around.

Code: (main sections in larger font - smaller font is mundane code that you need for it to compile)

```
#include <pic18.h>

// Global Variables
unsigned long int TIME, TIME0, TIME1;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -10000;
      RC0 = !RC0;
      TMR0IF = 0;
      }
   if (TMR1IF) {
      TIME = TIME + 0x10000;
      TMR1IF = 0;
      }
   if (CCP1IF) {
      TIME0 = TIME1;
      TIME1 = TIME + CCPR1;
      CCP1IF = 0;
      }
   }
```

```
// Subroutines
#include        "lcd_portd.c"
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');
   }
}
// Main Routine

void main(void)
{
   TRISA = 0;
   TRISB = 0;
   TRISC = 0x04;     // capture every rising edge
   TRISD = 0;
   ADCON1 = 0x0F;

// set up Timer0 for PS = 1
   T0CS = 0;
   T0CON = 0x88;
   TMR0ON = 1;
   TMR0IE = 1;
   TMR0IP = 1;
   PEIE = 1;
// set up Timer1 for PS = 1
   TMR1CS = 0;
   T1CON = 0x81;
   TMR1ON = 1;
   TMR1IE = 1;
   TMR1IP = 1;
   PEIE = 1;
// set up Capture1 for rising edges
   TRISC2 = 1;
   CCP1CON = 0x05;
   CCP1IE = 1;
   PEIE = 1;

   LCD_Init();
   Wait_ms(100);

   TIME = 0;

// turn on all interrupts
   GIE = 1;

   while(1) {
      LCD_Move(0,0);  LCD_Out(TIME + TMR1, 7);
      LCD_Move(1,0);  LCD_Out(TIME1 - TIME0, 7);
      }
   }
```

What appears on the display is as follows:



Program Capture1.C Row #1 displays time since reset (TMR1) showing that Timer1 is working.
Row 2 displays the period of the square wave on RC2. Note that a wire shorts RC0 to RC2.

What this tells you is the period is 20,082 clocks.

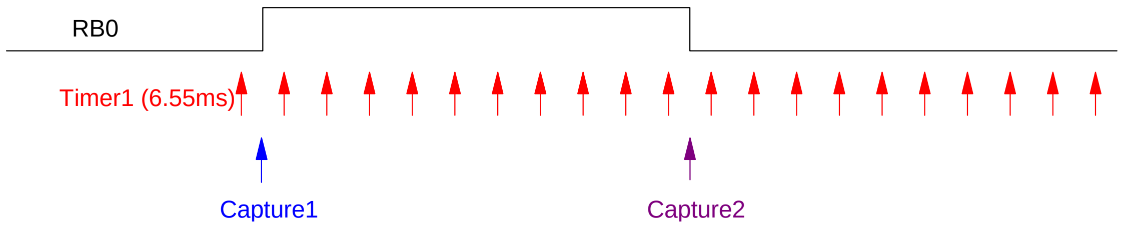
- Timer0 has to interrupt twice to get another rising edge
- Timer0 apparently interrupts every 10,041 clocks
- It takes 41 clocks to trigger a Timer0 interrupt

Example 2: Measure a pulse width (Capture2.C)

Tap button RB0. Measure how long that button was pressed (falling edge vs. rising edge).

To do this, use three interrupts:

- Timer1 keeps track of time to 100ns
- Capture1 records the time of the rising edge
- Capture2 records the time of the falling edge

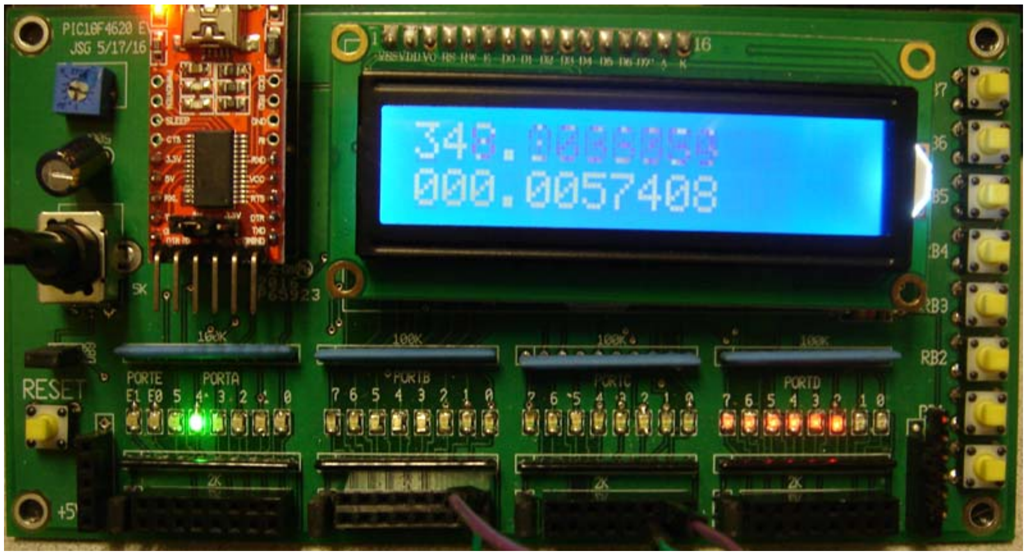


Three Interrupts are Running: Timer1 every 6.55ms, Capture1 on rising edges, Capture2 on falling edges

Code: (Capture2.C - normal stuff in 8-point font. Stuff to note is in color and 10-point font)

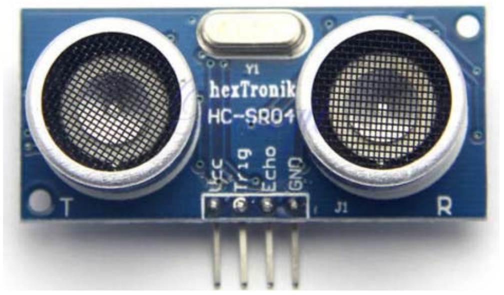
```
#include <pic18.h>

// Global Variables
unsigned long int TIME, TIME0, TIME1;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR1IF) {     // 6.55ms
      TIME = TIME + 0x10000;
      TMR1IF = 0;
      }
   if (CCP1IF) {    // rising edge
      TIME0 = TIME + CCPR1;
      CCP1IF = 0;
      }
   if (CCP2IF) {    // falling edge
      TIME1 = TIME + CCPR2;
      CCP2IF = 0;
      }
   }

// Subroutines
#include        "lcd_portd.c"


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');
   }
}
```

```
// Main Routine

void main(void)
{
   TRISA = 0;
   TRISB = 0xFF;
   TRISC = 0x04;     // capture every rising edge
   TRISD = 0;
   ADCON1 = 0x0F;

   LCD_Init();
   Wait_ms(100);

   TIME = 0;

// set up Timer1 for PS = 1
   TMR1CS = 0;
   T1CON = 0x81;
   TMR1ON = 1;
   TMR1IE = 1;
   TMR1IP = 1;
   PEIE = 1;
// set up Capture1 for rising edges
   TRISC2 = 1;
   CCP1CON = 0x05;
   CCP1IE = 1;
   PEIE = 1;
// set up Capture2 for falling edges
   TRISC1 = 1;
   CCP2CON = 0x04;
   CCP2IE = 1;
   PEIE = 1;

// turn on all interrupts
   GIE = 1;

   while(1) {
      LCD_Move(0,0);  LCD_Out(TIME + TMR1, 7);
      LCD_Move(1,0);  LCD_Out(TIME1 - TIME0, 7);
      }
   }
```



Display from Capture2.C. Row #1 displays the time since reset (TMR1)
Row #2 displays the duration of the last pulse (5.7408ms in this case)
Note that pin RB0 is tied to RC1 and RC2 with jumper wires.

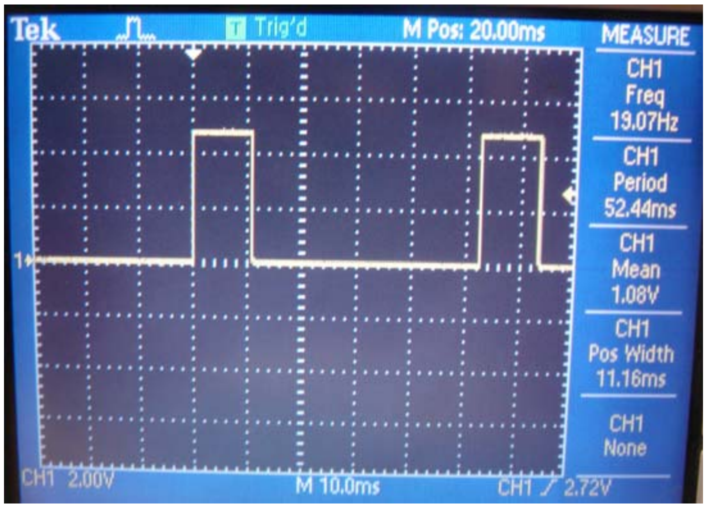
Example 3: Range Sensor (Range.C)



In your lab kit is an ultrasonic range sensor. This device has four pins:

- Vcc: input: +5V
- Trig: input: 0V/5V pulse from the PIC
- Echo: output: 0V/5V pulse to the PIC.
- Gmd: input: 0V

Each time you sent from the range sensor. The time it takes for the sound to return is the duration of the pulse on Echo. For example, if Trig is a 19Hz square wave, the signal on Echo might look like this:



Singal on Echo pin. The pulse width is proportional to distance

To compute to distance, do the following:

The speed of sound at 1atm at 20C with dry air is 343 m/s. (www.wikipedia.com)

v = 343 m/s

The distance sound travels in 50ns (50ns there plus 50ns return = 100ns clock) is

$$d = (343\tfrac{m}{s}) \cdot (50ns) = 17.15\mu m$$

$$1 \text{ clock} = 17.15\mu m$$

Hence, if you measure the duration of the pulse on Echo, the distance is then d = 17.15 * dT units = microns,

d = 0.01715 * dT units = mm

where dT = duration of Echo pulse in clock tics. For comparison,

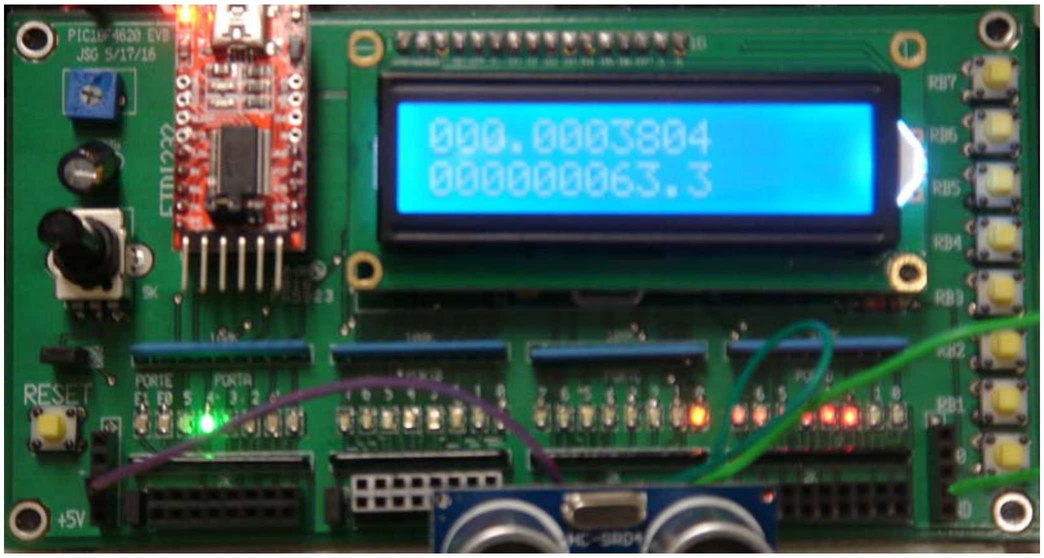
- Human Hair: 17 to 180um
- Thickness of a Piece of Paper: 70 to 180um
- Paramecium: 50 to 330um
- Bacteria: 0.5 to 5.0 um
- Hydrogen Atom: 0.0001 um

With a resolution of 17um, you can't see an atom but you can detect if a single piece of paper was removed from a stack.

Using the previous code, you can turn your range sensor into a meter stick:

```
#include <pic18.h>

// Global Variables
unsigned long int TIME, TIME0, TIME1, dT;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR0IF) {
      RC0 = !RC0;
      TMR0IF = 0;
      }
   if (TMR1IF) {
      TIME = TIME + 0x10000;
      TMR1IF = 0;
      }
   if (CCP1IF) {
      if (CCP1CON == 0x05) {  // rising edge
         TIME0 = TIME + CCPR1;
         CCP1CON = 0x04;
         }
      else {
         TIME1 = TIME + CCPR1;
         dT = TIME1 - TIME0;
         CCP1CON = 0x05;
      }
      CCP1IF = 0;
      }
   }
// Subroutines
#include        "lcd_portd.c"

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');
   }
}
```

```
// Main Routine

void main(void)
{
   int mm;

   TRISA = 0;
   TRISB = 0xFF;
   TRISC = 0x04;     // capture every rising edge
   TRISD = 0;
   ADCON1 = 0x0F;

   LCD_Init();
   Wait_ms(100);

   TIME = 0;

// set up Timer0 for PS = 1
   T0CS = 0;
   T0CON = 0x81;
   TMR0ON = 1;
   TMR0IE = 1;
   TMR0IP = 1;
   PEIE = 1;
// set up Timer1 for PS = 8
   TMR1CS = 0;
   T1CON = 0x81;
   TMR1ON = 1;
   TMR1IE = 1;
   TMR1IP = 1;
   PEIE = 1;
// set up Capture1 for rising edges
   TRISC2 = 1;
   CCP1CON = 0x05;
   CCP1IE = 1;
   PEIE = 1;

// turn on all interrupts
   GIE = 1;

   while(1) {
      mm = dT * 0.1715;                    // 1 count = 1/10 mm

      LCD_Move(0,0);  LCD_Out(dT, 7);
      LCD_Move(1,0);  LCD_Out(mm, 1);

      }
   }
```



Typical display from Range.C. The pulse width of Echo was 380.4us. The corresponding distance was 63.3mm

Example 3: Measure how high you can jump (Jump.C)

Place a laser beam on the ground and shine it on a light sensor. In hardware, output 5V when the light is shining on this sensor, 0V when not. If you stand, breaking the beam of light, then jump in the air, the time you're in the air is equal to the pulse width seen by the PIC.

The previous program allows you to record your hang time to 100ns. To compute how high you jumped, use

$$d = \tfrac{1}{2}at^2$$

t is the time from the apogee to the ground - equal to 1/2 the time we measured.

$$d = \tfrac{1}{2}a \cdot \left(\tfrac{1}{2}(t_1 - t_0)\right)^2$$

or

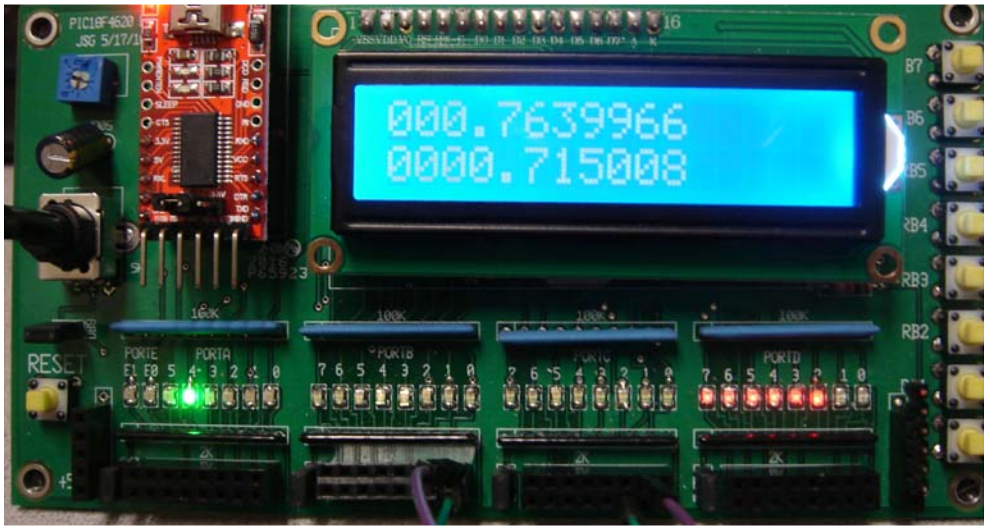
$$d = \tfrac{1}{8}a(t_1 - t_0)^2$$

The interrupt service routine:

```
#include <pic18.h>

// Global Variables
unsigned long int TIME, TIME0, TIME1, dT;

// Interrupt Service Routine

void interrupt IntServe(void)
{
   if (TMR1IF) {
      TIME = TIME + 0x10000;
      TMR1IF = 0;
      }
   if (CCP1IF) {
      TIME0 = TIME + CCPR1;
      CCP1IF = 0;
      }
   if (CCP2IF) {
      TIME1 = TIME + CCPR2;
      dT = TIME1 - TIME0;
      CCP2IF = 0;
      }
   }
```

Main Loop:

```
   while(1) {
      time = 0.0000001 * dT;
      meters = 0.125 * 9.8 * time * time;

      LCD_Move(0,0);  LCD_Out(dT, 7);
      LCD_Move(1,0);  LCD_Out(meters*1000000.0, 6);

      }
   }
```



Result of program Jump.C. Row #1 displays your hang time in seconds.
Row #2 displays your computed vertical leap in maters

To find the resolution,

- The smallest change in time you can see is 100ns
- Perturb time by 100ns and compute your vertical leap again
- The difference is the resolution of this device in meters

For example

- 0.763 996 6 second corresponds to 0.715 021 236 meters
- 0.763 996 7 seconds corresponds to 0.725 021 423 meters

The difference is 0.000 000 187 meters (187 nm)

This sensor has a resolution of 187nm

If someone jumped 187nm higher than you, this sensor could see it.