Timer 0/1/2/3 Interrupts

ECE 376 Embedded Systems

Jake Glower - Lecture #21

Please visit Bison Academy for corresponding
lecture notes, homework sets, and solutions

Timer 0/1/2/3 Interrupts

With a single processor, you can only do one thing at a time:

- You can drive the LCD display, or
- You can measure time, or
- You can output a precise freqeucy.

With Timer2 interrupts, you can now do two things at the same time:

- One using the Timer2 interrupt (measure time or output a precise frequency), and
- One using the main routine (drive the LCD display, read the buttons, etc.)

With Timer0/1/2/3 interrupts, you can do five things at the same time.

- It can get confusing, but if you can figure out interrupts, they make some problems a *lot* easier to solve

Timer 0/1/2/3 Set Up

To get each interrupt to run, you need to

- Set the condition of the interrupt (input, N),
- Enable the interrupt, &
- Acknowledge the interrupt (clear flag) when done

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 = 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 = 0xB1: PS = 8 TMR1 = -Y	TMR1ON = 1 TMR1IE = 1 TMR1IP = 1 PEIE = 1	TMR1IF
Timer2	Interupt every N clocks 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 .. 219 100ns to 0.52 sec	RC1 TMR3CS = 1 OSC/4 TMR3CS = 0	N = (PS)(Y) T3CON = 0x81: PS = 1 T3CON = 0xB1: PS = 8 TMR3 = -Y	TMR3ON = 1 TMR3IE = 1 TMR3IP = 1 PEIE = 1	TMR3IF

Chords (Chord.c)

Do five things at the same time:

- Play note A3/B3/C4 on RC0
- Play note C4/D4/E4 on RC1
- Play note E4/F4/G4 on RC2,
- Monitor the push buttons every 1ms, and
- Display the note being played on the LCD display.

Step #1: Assign interrupts and what note you play on the output pin

	Interrupt	Timer0	Timer1	Timer3	Timer2
	Output Pin	RC0	RC1	RC2	-
Button Pressed	RB0	A3	C4	E4	monitor PORTB T = 1ms
	RB1	B3	D4	F4	
	RB2	C4	E4	G4	

Step 2: Determine N (# clocks between interrupts)

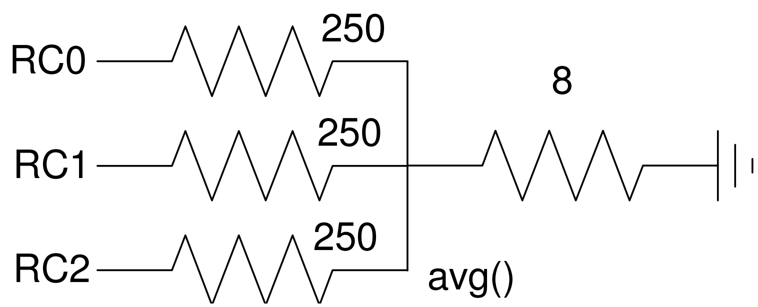
	A3	B3	C4	D4	E4	F4	G4	A4
Hz	220	246.94	261.63	293.66	329.63	349.23	392	440
N	22,727.27	20,247.83	19,110.96	17,026.49	15,168.52	14,317.21	12,755.1	11,363.64

Timer2: N = 10,000

- A = 10, B = 250, C = 4

Step 3 Hardware: Send three different notes to one (or more) speakers

- Limit the current to 20mA



Step 4: Software

Global Variables

```
// Global Variables

const unsigned char MSG0[21] = "Chord.C             ";
const unsigned char MSG1[21] = "Timer 0/1/2/3       ";

const unsigned int A3 = 22727;
const unsigned int B3 = 20247;
const unsigned int C4 = 19110;
const unsigned int D4 = 17026;
const unsigned int E4 = 15168;
const unsigned int F4 = 14317;
const unsigned int G4 = 12755;
const unsigned int A4 = 11363;

unsigned int N0, N1, N3;
```

Interrupt Service Routines

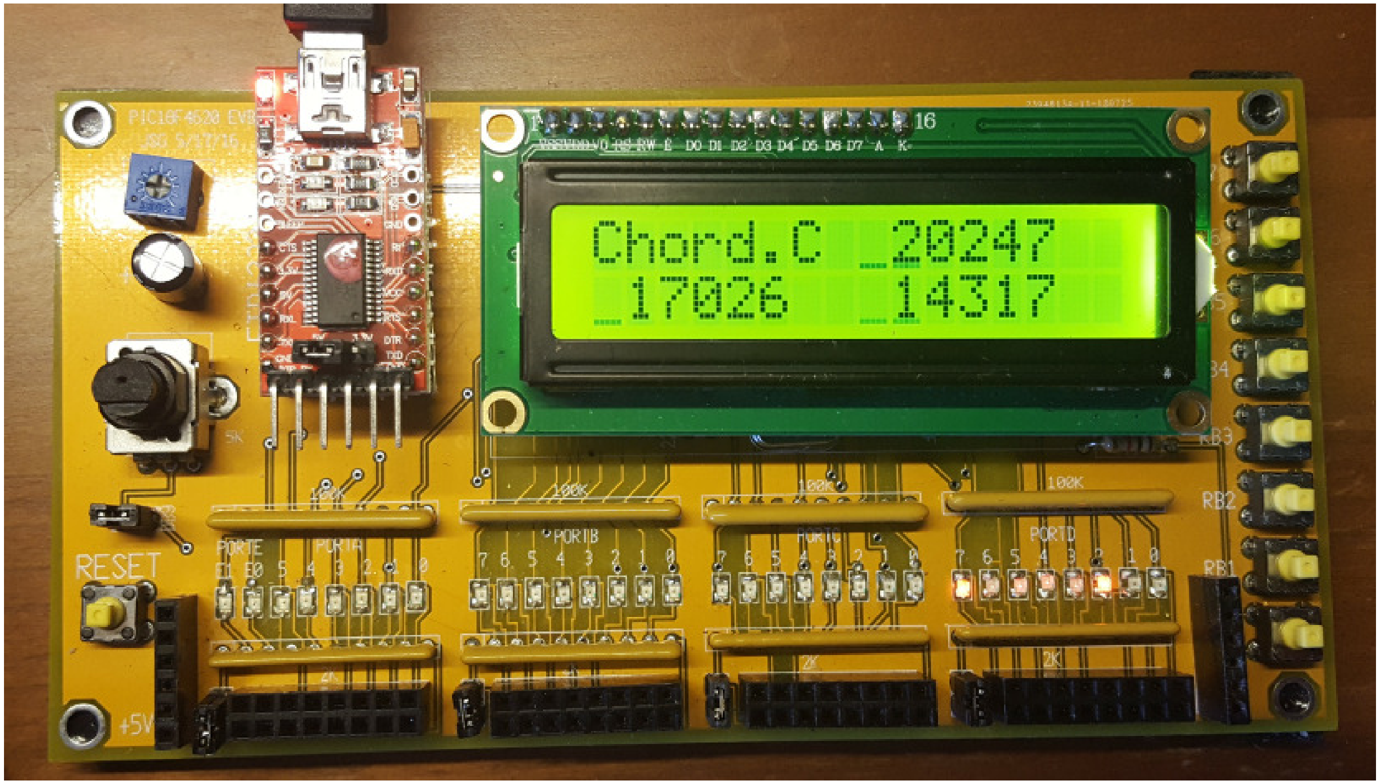
```
// Interrupt Service Routine
void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -N0;
      if (PORTB) RC0 = !RC0;
      TMR0IF = 0;
      }
   if (TMR1IF) {
      TMR1 = -N1;
      if (PORTB) RC1 = !RC1;
      TMR1IF = 0;
      }
   if (TMR2IF) {
      if (RB0) { N0 = A3;  N1 = C4;  N3 = E4; }
      if (RB1) { N0 = B3;  N1 = D4;  N3 = F4; }
      if (RB2) { N0 = C4;  N1 = E4;  N3 = G4; }
      if (RB3) { N0 = D4;  N1 = F4;  N3 = A4; }
      TMR2IF = 0;
      }
   if (TMR3IF) {
      TMR3 = -N3;
      if (PORTB) RC2 = !RC2;
      TMR3IF = 0;
      }
   }
```

Main Routine: Enable all four interrupts

```
// set up Timer0 for PS = 1
   T0CON = 0x88;
   T0CS = 0;
   TMR0ON = 1;
   TMR0IE = 1;
   TMR0IP = 1;
   PEIE = 1;
// set up Timer1 for PS = 1
   T1CON = 0x81;
   TMR1CS = 0;
   TMR1ON = 1;
   TMR1IE = 1;
   TMR1IP = 1;
   PEIE = 1;
// set up Timer2 for 1ms
   T2CON = 0x4D;
   PR2   = 249;
   TMR2ON = 1;
   TMR2IE = 1;
   TMR2IP = 1;
   PEIE = 1;
// set up Timer3 for PS = 1
   T3CON = 0x81;
   TMR3CS = 0;
   TMR3ON = 1;
   TMR3IE = 1;
   TMR3IP = 1;
   PEIE = 1;
```

Main Routine: Display the note being played

```
while(1) {
   LCD_Move(0,9);   LCD_Out(N0, 4);
   LCD_Move(1,0);   LCD_Out(N1, 4);
   LCD_Move(1,9);   LCD_Out(N3, 4);
   }
```



Quad Copter Motor Controller (Quad.C)

Interrupts can change the condition of other interr

Example: Quad Copter controller.

Hardware:

- Blue Wires: Motor A,B,C
- Power (black / red wires):

```
Red =  +6 to +12V DC, capable of 1A
Black = ground
```

- Signal: (3-wire black / red / white)

```
Black: ground
Red:   +5V
White: Signal:
  0.9ms to 2.0ms pulse @ 50Hz
```

Software: Generate a 50Hz, 0.9mm to 2.0mm 0V/5V pulse

- The white signal wire is TTL logic levels (0 / 5V)
- The frequency of the pulse should be 50Hz
- The pulse width determines the motor speed

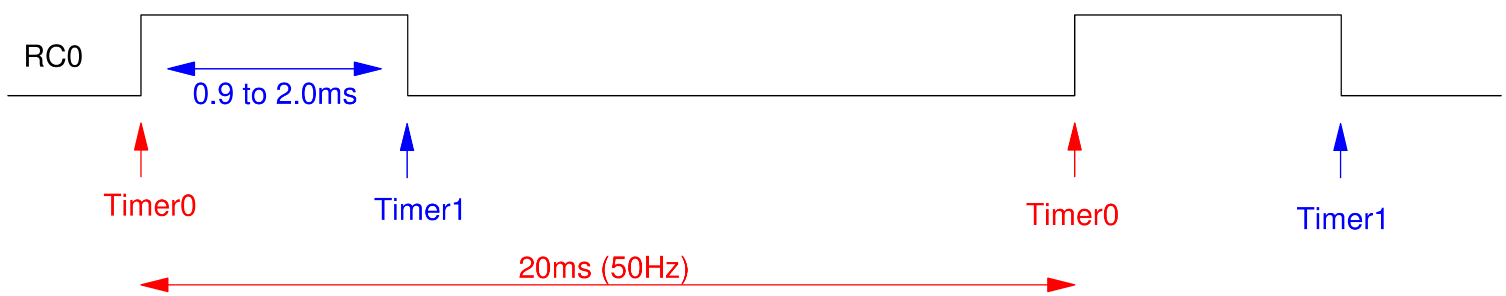
```
0.9ms = Idle  & power on
1.2ms = Slow
2.0ms = Fast
```

Timer0:

- Triggered ever 20ms
- Sets RC0
- Sets up Timer1 interrupt, 0.9ms to 1.2ms later

Timer1:

- Clears RC0



Coding

Timer0 (PS = 4)

```
if (TMR0IF) {
   TMR0 = -50000;
   TMR1 = -N;
   T0 += 1;
   RC0 = 1;
   TMR1IF = 0;
   }
```

Timer1 (PS = 1)

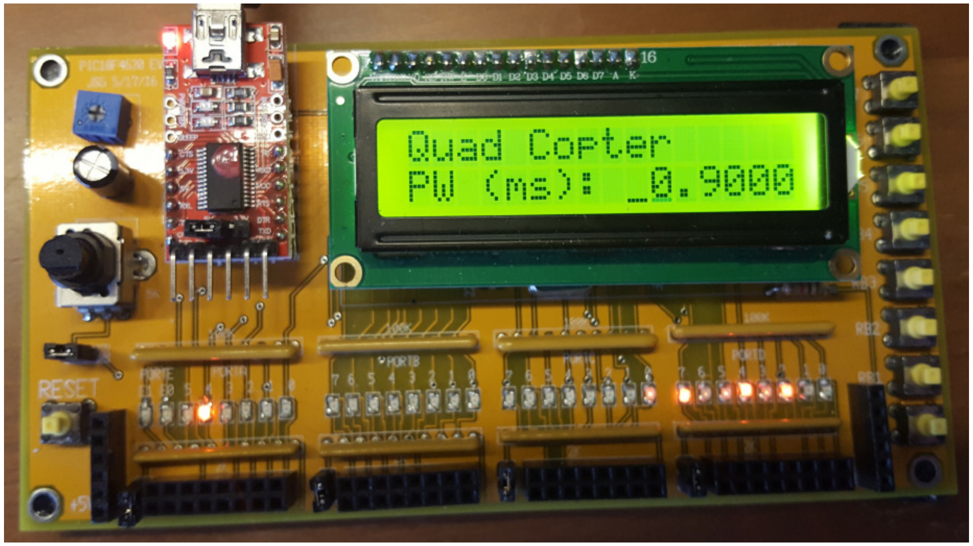
```
if (TMR1IF) {
   RC0 = 0;
   T1 += 1;
   TMR1IF = 0;
   }
```

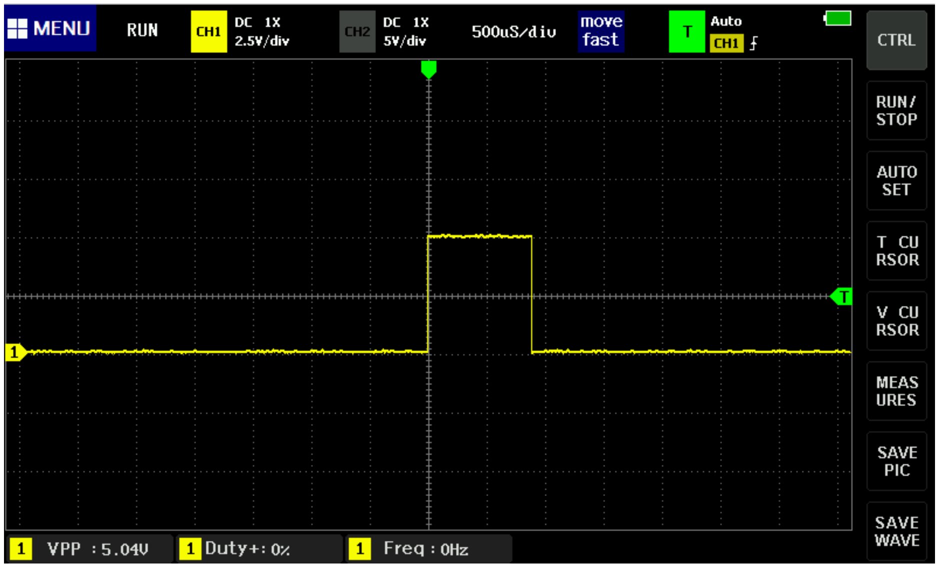
Main

```
while(1) {
   if (RB0) N =  9000;
   if (RB1) N = 12000;
   if (RB2) N = 13000;
   if (RB3) N = 14000;
   if (RB4) N = 15000;
   if (RB5) N = 16000;
   if (RB6) N = 17000;
   if (RB7) N = 18000;
   LCD_Move(1,9);
   LCD_Out(N, 4);
   }
```

Quad Copter Results:

- 50 Hz square wave (Timer0)
- 0.9ms to 1.2ms pulse (Timer1)

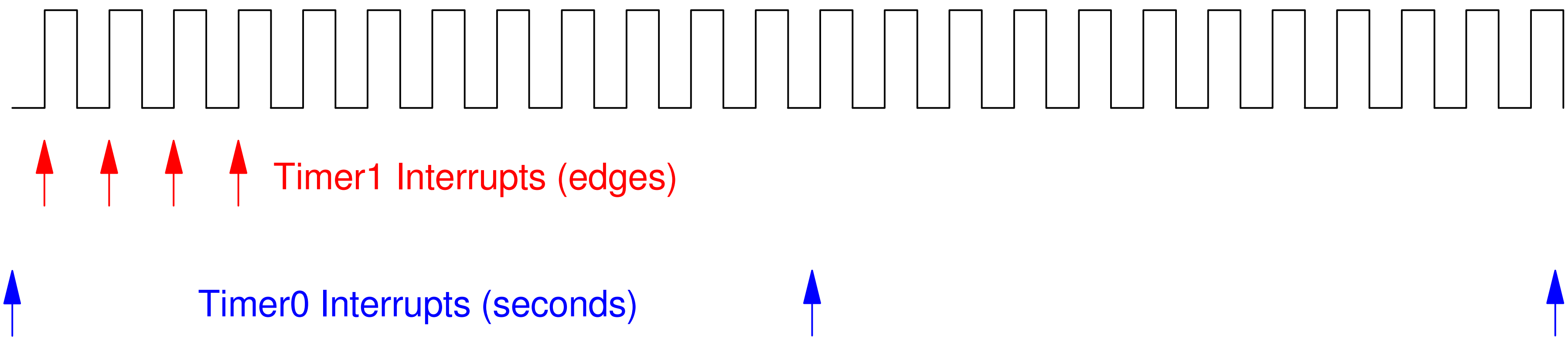




Frequency Counter (Freq.C)

Measure the frequency of a square wave

- Timer1: Cycles
- Timer0: Seconds
- Cycles / Second = Hz



Why? Optical Encoders:

- Convert motor speed to a frequency
- (100 pulses / rotation) * (N rotations / second) = 100N Hz

Wiring:

- +5V & Ground
- A: 100 pulses / rotation
- B: 100 pulses / rotation

A & B are 90 degrees out of phase

5-PIN SINGLE-ENDED (1)	
Pin	Description
1	Ground
2	Index
3	A channel
4	+5VDC power
5	B channel

Interrupt Selection

Timer0:

- Time in seconds
- N = 10,000,000
- = 256 * 39250

Timer1:

- Counts edges
- Assume f < 65,536 Hz

Timer2:

- Test signal
- Output 500Hz

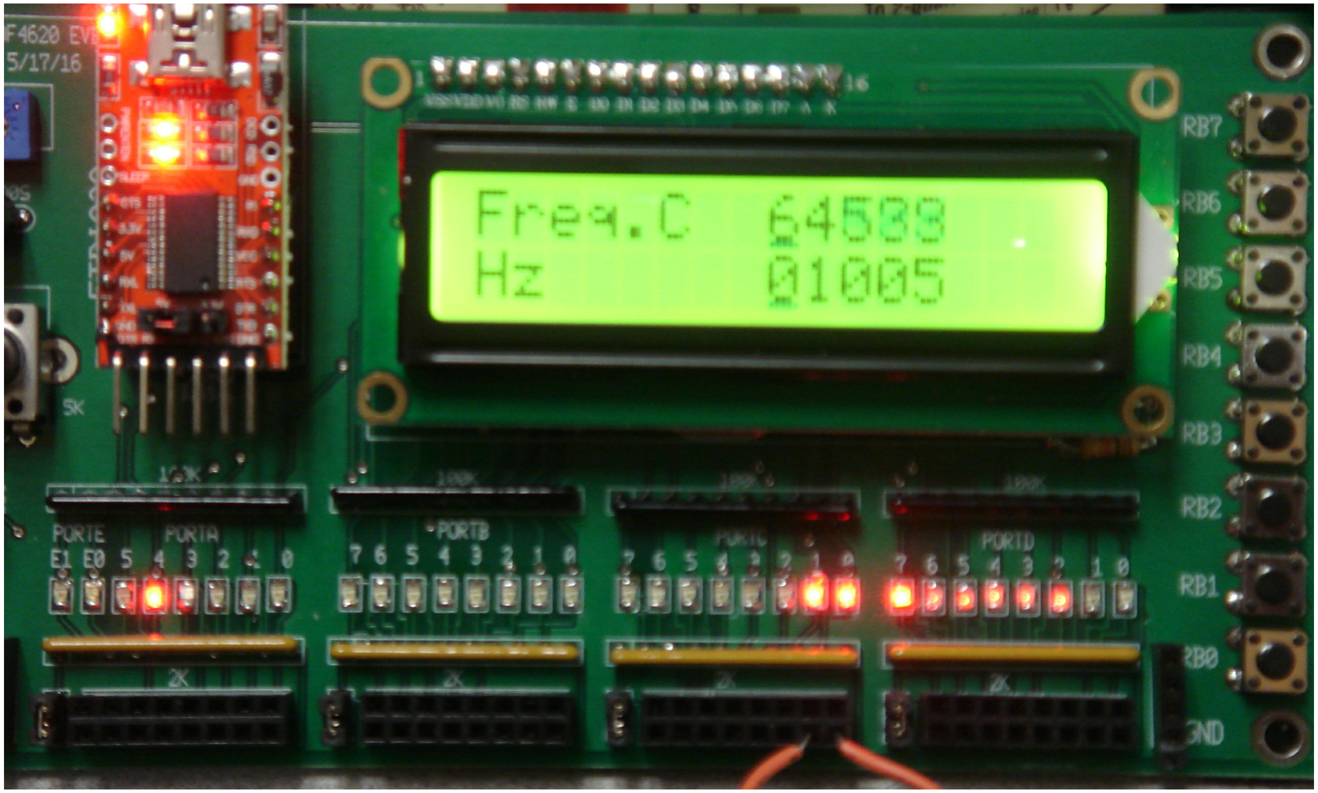
```
// Global Variables
unsigned int T0, T1, T2;

// Interrupt Service Routine
void interrupt IntServe(void)
{
   if (TMR0IF) {
      TMR0 = -39250;
      T0 += 1;
      T2 = T1;
      T1 = TMR1;
      TMR0IF = 0;
      }
   if (TMR1IF) {
      TMR1IF = 0;
      }
   if (TMR2IF) {
      RC0 = !RC0;
      TMR2IF = 0;
      }
   }
```

Interrupt Initialization

```
// set up Timer0 for PS = 256, input = clock
   T0CS = 0;
   T0CON = 0x87;
   TMR0ON = 1;
   TMR0IE = 1;
   TMR0IP = 1;
   PEIE = 1;
// set up Timer1 for PS = 1, input = RC0
   T1CON = 0x81;
   TMR1ON = 1;
   TMR1IE = 0;
   TMR1IP = 0;
   PEIE = 0;
   TMR1CS = 1;
   TRISC0 = 1;
// set up Timer2 for 0.5ms (1kHz reference signal on RC0)
   T2CON = 0x4D;
   PR2   = 124;
   TMR2ON = 1;
   TMR2IE = 1;
   TMR2IP = 1;
   PEIE = 1;
// turn on all interrupts
   GIE = 1;
```

Main Loop:

```
while(1) {
    Hz = T1 - T2;
    LCD_Move(1,8);  LCD_Out(Hz, 0);
    }
```

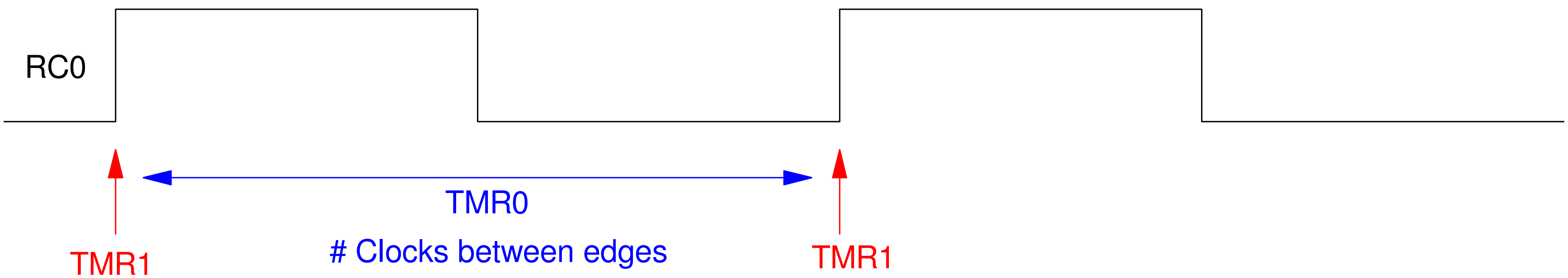


Frequency Counter (take 2: Tach.C)

Measure the period of a square wave

- More accurate for low-frequency signals
- 1000 Hz = 1000 edges / second
- 1000 Hz = 10,000 clocks between edges

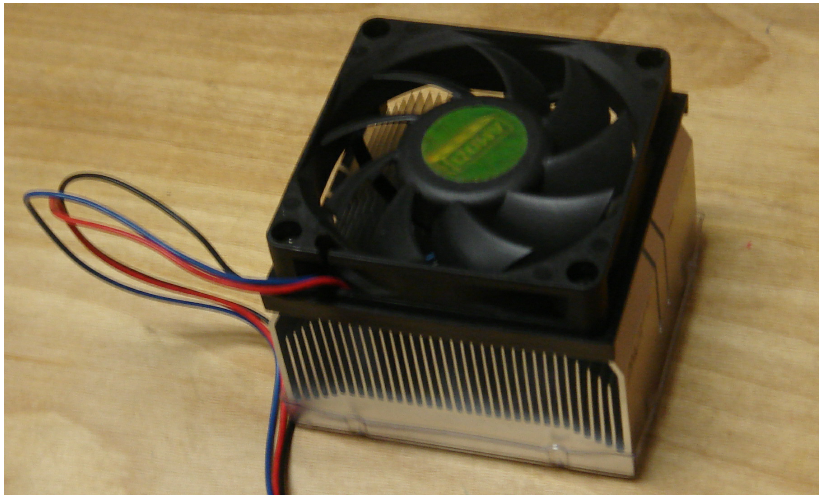
It's actually more accurate to measure clocks per edge than edges per second. Plus, you get a reading every 1ms.

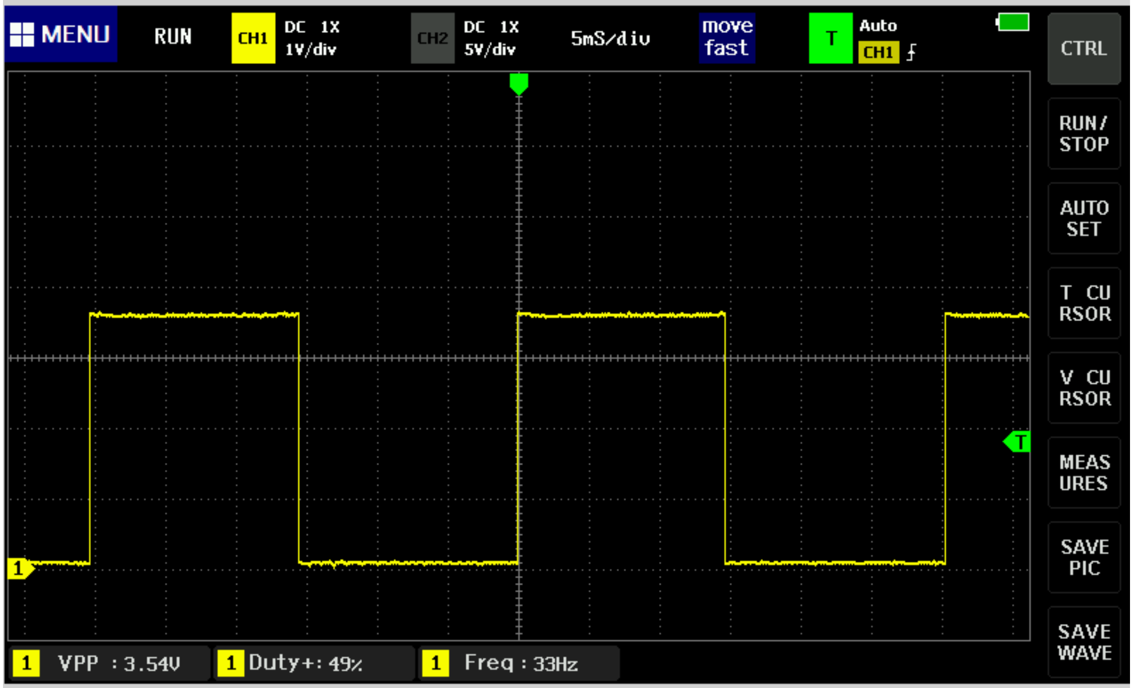


Timer1 interrupts on rising edges on RC0. Timer0 measures the time between interrupts to 100ns

Example: Measure the speed of a 12V DC motor

- Black: Ground
- Red: Power (0V to +12V DC, capable of 10mA)
- Blue: Tachometer output (add a 1k pull-up resistor to +5V to read this signal)
- Output = 33Hz @ 5.00V DC





Software:

Timer0

time accurate to 100ns

```
if (TMR0IF) {
  TIME = TIME+0x10000;
  TMR0IF = 0;
  }
```

Timer1

time of rising edges

```
if (TMR1IF) {
   TMR1 = -1;
   T2 = T1;
   T1 = TIME + TMR0;
   PERIOD = T1 - T2;
   TMR1IF = 0;
   }
```

Timer2

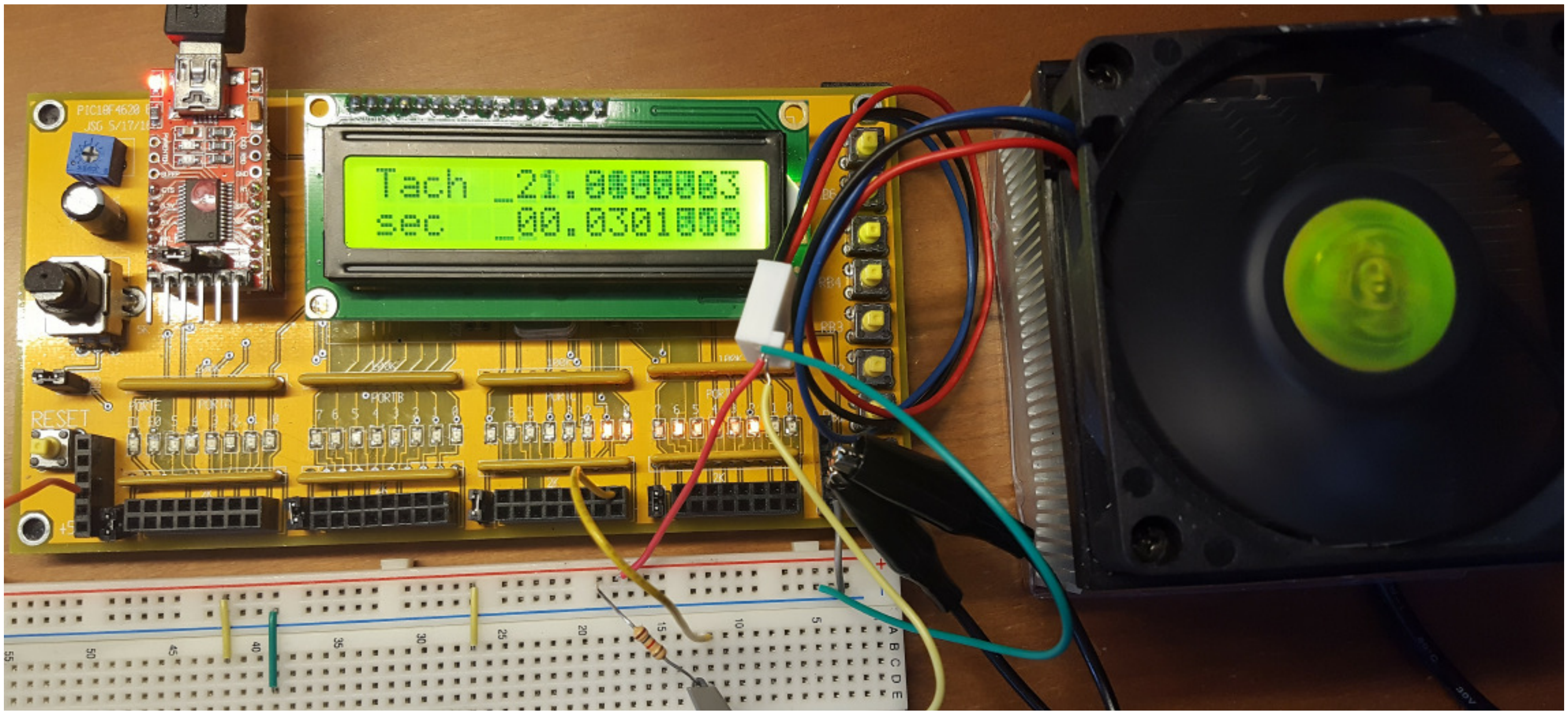
1kHz test signal

```
if (TMR2IF) {
   RC1 = !RC1;
   TMR2IF = 0;
   }
```

Main Loop:

- Period = 301,536 clocks

```
while(1) {
   LCD_Move(0,5);   LCD_Out(TIME + TMR0, 7);
   LCD_Move(1,5);   LCD_Out(PERIOD, 7);
   }
```



Pulse Width Modulation (PWM.C)

Objective: Turn on and off a motor / light / heater from 0% on to 100% on

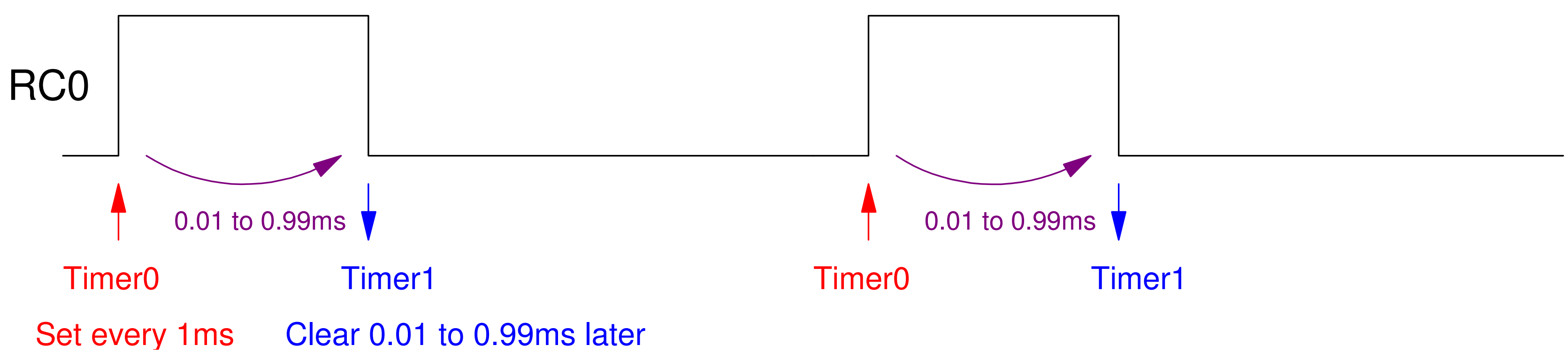
- With 10,000 levels of grey, and
- At 1kHz

Timer0 sets RC0

- Timer0 interrupts every 1ms for 1kHz
- When called, it sets up a Timer1 interrupt from 100 to 9900 clocks in the future

When Timer0 kicks in

- Timer1 clears RC0



Software:

Timer0
Set RC0
Set up Timer1 0..1ms later

```
if (TMR0IF) {
   TMR0 = -10000;
   TMR1 = -PWM;
   TIME += 1;
   RC0 = 1;
   TMR0IF = 0;
   }
```

Timer1
Clear RC0

```
if (TMR1IF) {
   RC0 = 0;
   TMR1IF = 0;
   }
```

Main Loop
Define PWM
Drive LCD display

```
while(1) {
  if (RB0) PWM = 100;
  if (RB1) PWM = 1000;
  if (RB2) PWM = 2000;
  if (RB3) PWM = 3000;
  if (RB4) PWM = 4000;
  if (RB5) PWM = 5000;
  if (RB6) PWM = 6000;
  if (RB7) PWM = 9900;
  LCD_Move(0,7);
  LCD_Out(TIME, 3);
  LCD_Move(1,7);
  LCD_Out(PWM, 2);
  }
```

Result

- 1kHz
- 0.5% to 99.5% PWM