

PIC16f877a Timer

Contents ▾

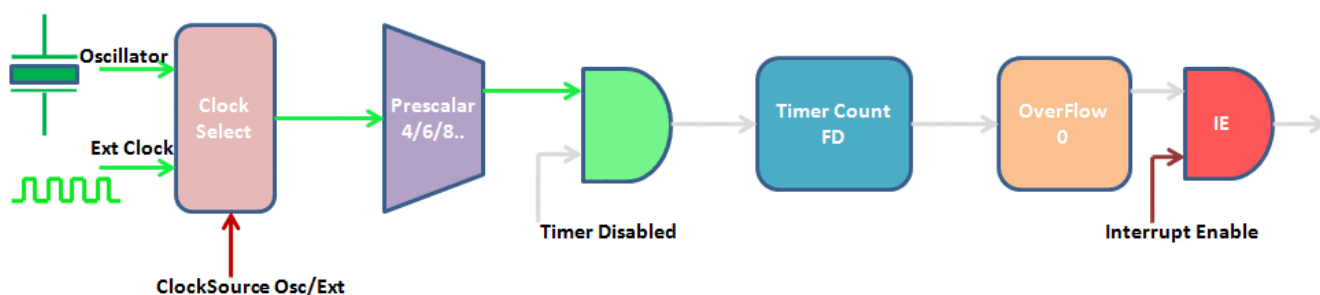
In this tutorial we are going to discuss the Timer module of Pic16f877a. First we will see what are timers, their working and later we will configure the PIC16f877a timers to generate delay of 100ms and 500ms respectively. At the end we will see how to use the ExploreEmdedded Timer library.

Timer Basics

As the name suggests these are used to measure the time or generate the accurate time delay. The microcontroller can also generate/measure the required time delays by running loops, but the timer/counter relieves the CPU from that redundant and repetitive task, allowing it to allocate maximum processing time for other tasks.

Timer is nothing but a simple binary counter that can be configured to count clock pulses(Internal/External). Once it reaches the Max value, it will roll back to zero setting up an **OverFlow** flag and generates the interrupt if enabled.

Timer Block Diagram



ExploreEmbedded

(/wiki/File:Timer.gif)

PIC Timer Module

Pic16f877a has three independent timer which can be used as timer,Counters or for PWM generation.

Below table provides the details of the three Timers.



Timer	Size	Control Register	Count Register	Min Delay	Max Delay
TIMER0	8-bit	OPTION_REG	TMR0	0.2usec	13.107ms
TIMER1	16-bit	T1CON	TMR1H,TMR1L	0.2usec	104.857ms
TIMER2	8-bit	T2CON	TMR2	0.2usec	819usec

Timer Calculation

PIC Oscillator frequency is divided by 4 and then fed to the controller, Now this this freq can be further divided by prescalar to generate the range of delays.

Time to increment the Timer count by one(timer tick) can be determined as below.

$$\text{tick} = (\text{Prescalar}/(\text{Fosc}/4))$$

$$\text{tick} = (\text{Prescalar}/(20\text{Mhz}/4))$$

$$\text{\$}\text{tick} = (\text{Prescalar} * 4)/\text{Fosc}\text{\$}$$

Now the Timer value for the required delay can be calculated as below.

$$\text{Delay} = \text{TimerCount} * \text{tick}$$

$$\text{Count} = (\text{Delay}/\text{tick})$$

$$\text{RegValue} = \text{TimerMax} - \text{Count}$$

$$\text{RegValue} = \text{TimerMax} - (\text{Delay}/\text{tick}) = \text{TimerMax} - (\text{Delay}/((\text{Prescalar} * 4)/\text{Fosc}))$$

$$\text{\$}\text{RegValue} = \text{TimerMax} - ((\text{Delay} * \text{Fosc})/(\text{Prescalar} * 4))\text{\$}$$

Below table provides the formula for all the three Timers.

Timer	Size	Formula for delay calculation
TIMER0	8-bit	$\text{\$}\text{RegValue} = 256 - ((\text{Delay} * \text{Fosc})/(\text{Prescalar} * 4))\text{\$}$
TIMER1	16-bit	$\text{\$}\text{RegValue} = 65536 - ((\text{Delay} * \text{Fosc})/(\text{Prescalar} * 4))\text{\$}$
TIMER2	8-bit	$\text{\$}\text{RegValue} = 256 - ((\text{Delay} * \text{Fosc})/(\text{Prescalar} * 4))\text{\$}$

Timer 0

The TMR0 module is an 8-bit timer/counter with the following features:

- 8-bit timer/counter
- Readable and writable
- 8-bit software programmable prescaler
- Internal or external clock select



- Interrupt on overflow from FFh to 00h
- Edge select for external clock

Timer0 Registers

The below table shows the registers associated with PIC16f877A Timer0 module.

Register	Description
OPTION_REG	This registers is used to configure the TIMER0 Prescaler, Clock Source etc
TMR0	This register holds the timer count value which will be incremented depending on prescaler configuration
INTCON	This register contains the Timer0 overflow flag(TMR0IF) and corresponding Inetrrupt Enable flag(TMROIE).

OPTION_REG							
7	6	5	4	3	2	1	0
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0

RBPU: NA for Timers

INTEDG: NA for Timers

T0CS: TMR0 Clock Source Select bit

1-Transition on T0CKI pin

0-Internal instruction cycle clock (CLKO)

T0SE: TMR0 Source Edge Select bit

1-Increment on high-to-low transition on T0CKI pin

0-Increment on low-to-high transition on T0CKI pin

PSA: Prescaler Assignment bit

1-Prescaler is assigned to the WDT

0-Prescaler is assigned to the Timer0



PS2:PS0: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

(/wiki/File:Timer0Prescalar.png)

- **Note:** There is only one prescaler available which is mutually exclusively shared between the Timer0 module and the Watchdog Timer. A prescaler assignment for the Timer0 module means that there is no prescaler for the Watchdog Timer and vice versa. This prescaler is not accessible but can be configured using **PS2:PS0** bits of **OPTION_REG**.

INTCON							
7	6	5	4	3	2	1	0
GIE	PIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF

GIE: Global Interrupt Enable bit

1-Enables all unmasked interrupts

0-Disables all interrupts

PIE: Peripheral Interrupt Enable bit

1-Enables all unmasked peripheral interrupts

0-Disables all peripheral interrupts

TMR0IE: TMR0 Overflow Interrupt Enable bit

1-Enables the TMR0 interrupt

0-Disables the TMR0 interrupt

INTE: NA for Timers

RBIE: NA for Timers

TMR0IF: TMR0 Overflow Interrupt Flag bit

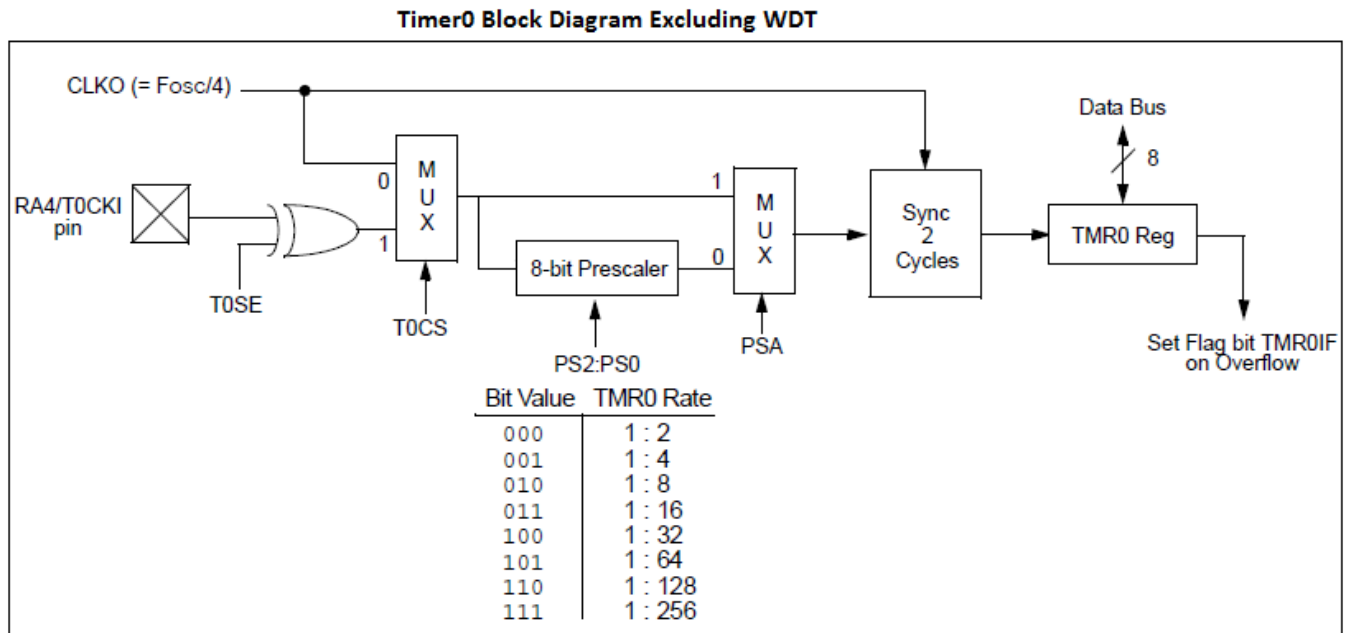
1-TMR0 register has overflowed (must be cleared in software)

0-TMR0 register did not overflow

INTF: NA for Timers



RBIF: NA for Timers



(/wiki/File:Timer0_Block_Diagram.png)

Delay Calculations for 1ms @20Mhz with Prescalar as 32:

$$\text{RegValue} = 256 - (\text{Delay} * \text{Fosc}) / (\text{Prescalar} * 4) = 256 - ((1\text{ms} * 20\text{Mhz}) / (32 * 4)) = 256 - 156 = 100$$

Code

Below are the steps for configuring and using the Timer0 for delay generation:

1. Calculate the Timer Count for the required delay.
2. Set the Presaclar bits in **OPTION_REG** as per the delay calculations.
3. Clear the **PSA** bit for using the prescalar.
4. Select the Clock Source Internal/External using **TOCS** bit.
5. Load the timer value into **TMR0** register.
6. Enable the Timer0 Interrupt by setting **TMR0IE** bit
7. Enable the Global and Peripheral interrupts by setting **GIE** and **PIE** bits

Below is the sample code to blink the LEDs with **1ms** delay.

1	#include<pic16f877a.h>
2	
3	char value = 0;
4	#define SBIT_PS2 2
5	
6	void interrupt timer_isr()
7	{

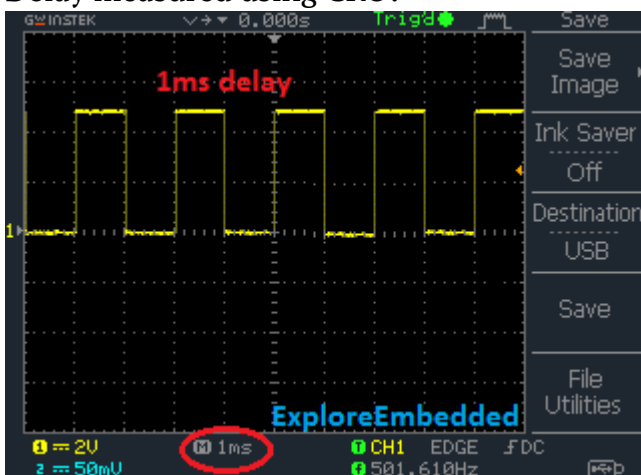
```

8      if(TMR0IF==1)
9      {
10         value=~value;    // complement the value for blinking the LEDs
11         TMR0 = 101;      /*Load the timer Value, (Note: Timervalue is 101 instaed of 100 as the
12                             TIme0 needs two instruction Cycles to start incrementing TMR0 */
13         TMR0IF=0;        // Clear timer interrupt flag
14     }
15 }
16
17
18 void main()
19 {
20     TRISD=0x00;    //COnfigure PORTD as output to blink the LEDs
21
22     OPTION_REG = (1<<SBIT_PS2); // Timer0 with external freq and 32 as prescalar
23     TMR0=100;      // Load the time value for 1ms delay
24     TMR0IE=1;      //Enable timer interrupt bit in PIE1 register
25     GIE=1;         //Enable Global Interrupt
26     PEIE=1;        //Enable the Peripheral Interrupt
27
28     while(1)
29     {
30         PORTD = value;
31     }
32 }

```

lBagwan/319888d39a82ea1ebbb1a0e29ed0754d/raw/2f956aab061911e73202c63f555cbd2f0db50c05/pic16f877a_timer0.c)
pic16f877a_timer0.c (https://gist.github.com/SaheblalBagwan/319888d39a82ea1ebbb1a0e29ed0754d#file-pic16f877a_timer0-c) hosted with ❤ by GitHub (<https://github.com>)

Delay measured using CRO:



(/wiki/File:Timer0_Cro.png)



Timer 1

The timer TMR1 module is an 16-bit timer/counter with the following features:

- 16-bit timer/counter with two 8-Bit register TMR1H/TMR1L
- Readable and writable
- software programmable prescaler upto 1:8
- Internal or external clock select
- Interrupt on overflow from FFFFh to 00h
- Edge select for external clock

Timer1 Registers

The below table shows the registers associated with PIC16f877A Timer1 module.

Register	Description
T1CON:	This registers is used to configure the TIMER1 Prescalar, Clock Source etc
TMRIH	This register holds the higher 8-bits of timer value. TMR1H and TMR1L are used in pair to increment from 0000 - FFFFh
TMRIL	This register holds the lower 8-bits of timer value. TMR1H and TMR1L are used in pair to increment from 0000 - FFFFh
PIR1	This register contains the Timer1 overflow flag(TMR1IF).
PIE1	This register contains the Timer1 Interrupt Enable flag(TMR1IE).

T1CON							
7	6	5	4	3	2	1	0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

T1CKPS1:T1CKPS0:Timer1 Input Clock Prescale Select bits

11 = 1:8 prescale value

10 = 1:4 prescale value

01 = 1:2 prescale value

00 = 1:1 prescale value



T1OSCEN: Timer1 Oscillator Enable Control bit

1-Oscillator is enabled

0-Oscillator is shut-off

T1SYNC: Timer1 External Clock Input Synchronization Control bit

1-Do not synchronize external clock input

0-Synchronize external clock input

TMR1CS: Timer1 Clock Source Select bit

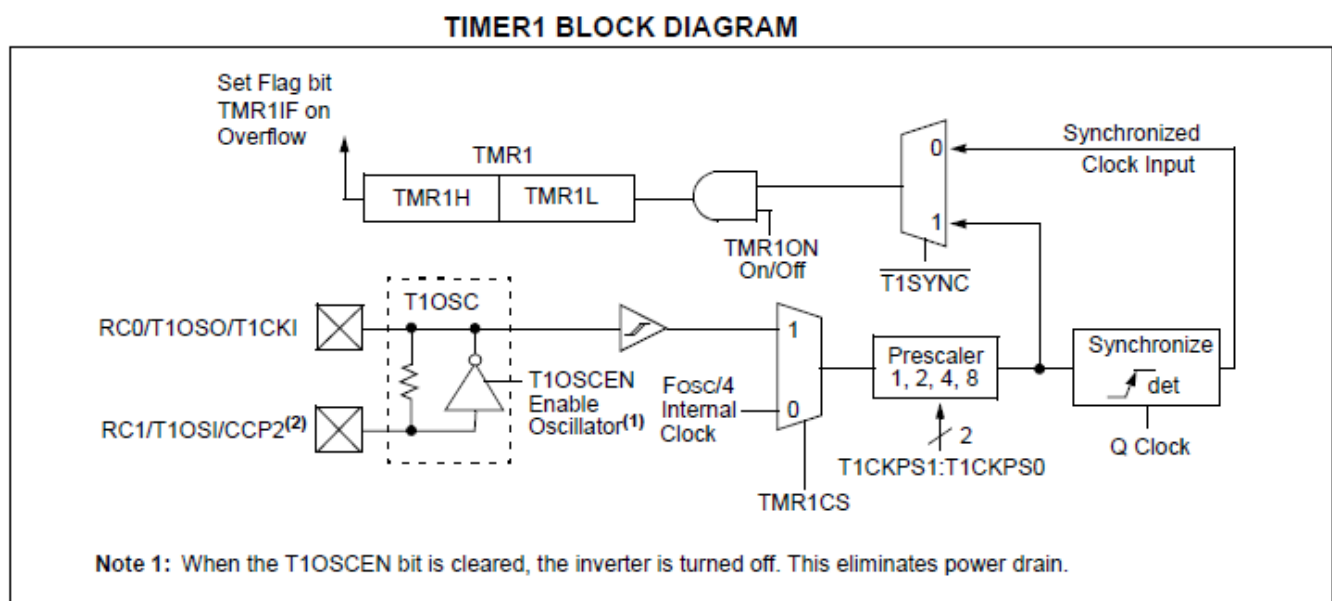
1-External clock from pin RC0/T1OSO/T1CKI (on the rising edge)

0-Internal clock (FOSC/4)

TMR1ON: Timer1 On bit

1-Enables Timer1

0-Stops Timer1



(/wiki/File:Timer1_Block_Diagram.png)

Delay Calculations for 100ms @20Mhz with Prescaler as 8:

RegValue = 65536-(Delay * Fosc)/(Prescaler*4) = 65536-((100ms * 20Mhz)/(8*4)) = 3036 = 0x0BDC

Code

Below are the steps for configuring and using the Timer1 for delay generation:

1. Calculate the Timer Count for the required delay.
2. Set the Prescaler bits in **T1CON** as per the delay calculations.
3. Select the Clock Source Internal/External using **TMR1CS** bit.
4. Load the timer value into **TMR1H,TMR1L** register.
5. Enable the Timer1 Interrupt by setting **TMR1IE** bit



6. Enable the Global and Peripheral interrupts by setting **GIE** and **PIE** bits

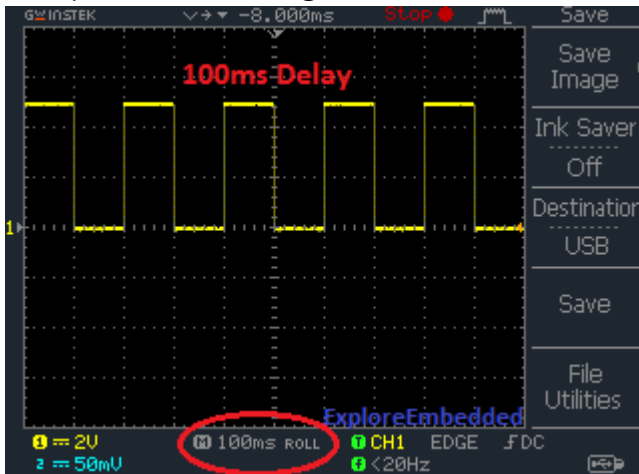
7. Finally start the timer by setting **TMR1ON** bit

Below is the sample code to blink the LEDs with **100ms** delay.

```
1  #include<pic16f877a.h>
2
3  char value = 0;
4  #define SBIT_PS1  5
5  #define SBIT_PS0  4
6
7  void interrupt timer_isr()
8  {
9      if(TMR1IF==1)
10     {
11         value=~value;    // complement the value for blinking the LEDs
12         TMR1H=0x0B;      // Load the time value(0xBDC) for 100ms delay
13         TMR1L=0xDC;
14         TMR1IF=0;        // Clear timer interrupt flag
15     }
16 }
17
18
19 void main()
20 {
21     TRISD=0x00;    //Configure PORTD as output to blink the LEDs
22
23     T1CON = (1<<SBIT_PS1) | (1<<SBIT_PS0); // Timer0 with external freq and 8 as prescalar
24     TMR1H=0x0B;    // Load the time value(0xBDC) for 100ms delay
25     TMR1L=0xDC;
26     TMR1IE=1;      //Enable timer interrupt bit in PIE1 register
27     GIE=1;         //Enable Global Interrupt
28     PEIE=1;        //Enable the Peripheral Interrupt
29     TMR1ON = 1;    //Start Timer1
30
31     while(1)
32     {
33         PORTD = value;
34     }
35 }
```



Delay measured using CRO:



(/wiki/File:Timer_1_Cro.png)

Timer 2

The TImeR2 module is an 8-bit timer/counter with the following features:

- 8-bit timer/counter
- Readable and writable
- Software programmable prescaler/PostScaler upto 1:16
- Interrupt on overflow from FFh to 00h

Timer2 Registers

The below table shows the registers associated with PIC16f877A Timer0 module.

Register	Description
T2CON	This registers is used to configure the TIMER2 Prescaler, Clock Source etc
TMR2	This register holds the timer count value which will be incremented depending on prescalar configuration
PIR1	This register contains the Timer2 overflow flag(TMR2IF).
PIE1	This register contains the Timer2 Interrupt Enable flag(TMR2IE).



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

TOUTPS3:TOUTPS0: Timer2 Output Postscale Select bits

0000 = 1:1 postscale

0001 = 1:2 postscale

0010 = 1:3 postscale

-
-
-

1111 = 1:16 postscale

TMR2ON: Timer2 On bit

1-Timer2 is on

0-Timer2 is off

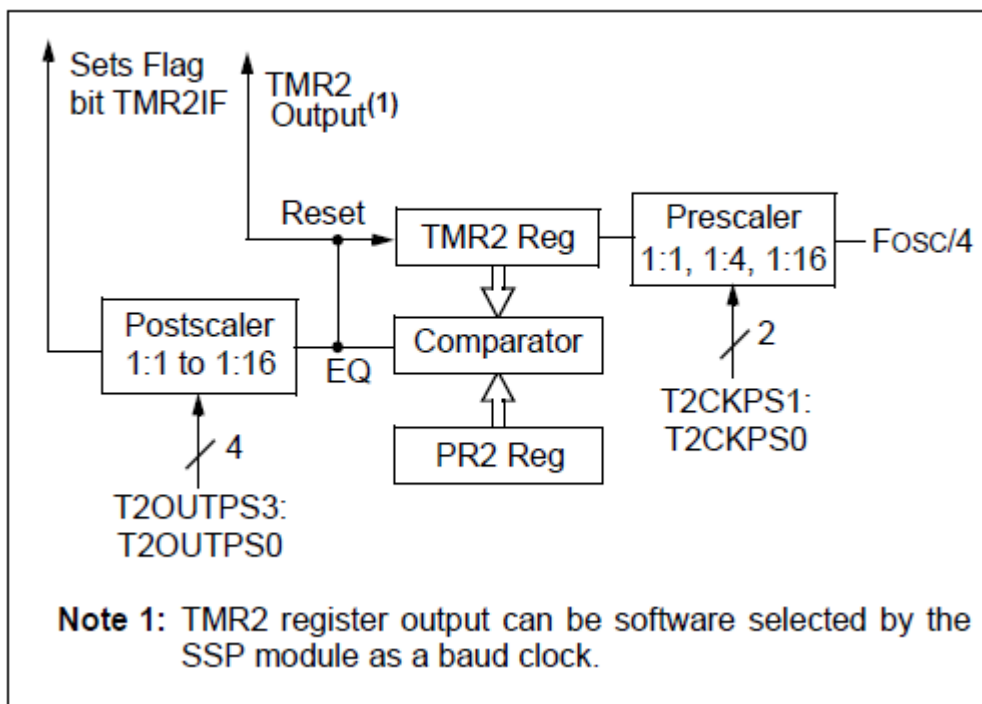
T2CKPS1:T2CKPS0: Timer2 Clock Prescale Select bits

00 = Prescaler is 1

01 = Prescaler is 4

1x = Prescaler is 16

TIMER2 BLOCK DIAGRAM



(/wiki/File:Timer2_Block_Diagram.png)

Generating 1sec delay using Timer2:

As the timer2 is 8-bit and supports 1:16 prescaler, it is not possible to directly generate the delay of 1sec. The max delay with 1:16 prescaler will be:

Delay = $256 * (\text{Prescaler} * 4) / F_{osc} = 256 * 16 * 4 / 20\text{Mhz} = 819\mu\text{s}$



Now 500us can be generated using timers which will be used to increment a counter 2000 times to get 1sec delay.

Delay Calculations for 500usec @20Mhz with Prescalar as 16:

$$\text{RegValue} = 256 - (\text{Delay} * \text{Fosc}) / (\text{Prescalar} * 4) = 256 - ((500\text{us} * 20\text{Mhz}) / (16 * 4)) = 256 - 156 = 100$$

Code

Below are the steps for configuring and using the Timer2 for delay generation:

1. Calculate the Timer Count for the required delay.
2. Set the Prescaler bits in **T2CON** as per the delay calculations.
3. Load the timer value into **TMR2** register.
4. Enable the Timer2 Interrupt by setting **TMR2IE** bit
5. Enable the Global and Peripheral interrupts by setting **GIE** and **PIE** bits
6. Finally start the Timer2 by setting **TMR2ON** bit

Below is the sample code to blink the LEDs with **1sec**(500us x2000times) delay.

1	#include<pic16f877a.h>
2	
3	#define SBIT_T2CKPS1 1
4	
5	char value = 0;
6	int count = 0;
7	
8	
9	void interrupt timer_isr()
10	{
11	if(TMR2IF==1)
12	{
13	TMR2 = 101; /*Load the timer Value, (Note: Timervalue is 101 instead of 100 as the
14	Timer2 needs two instruction Cycles to start incrementing TMR2 */
15	TMR2IF=0; // Clear timer interrupt flag
16	
17	if(count>=2000) //500us * 2000=1000000us=1sec
18	{
19	count=0;
20	value=~value; // complement the value for blinking the LEDs
21	}
22	else
23	{
24	count++; // Keep incrementing the count till it reaches 2000 to generate 1sec delay

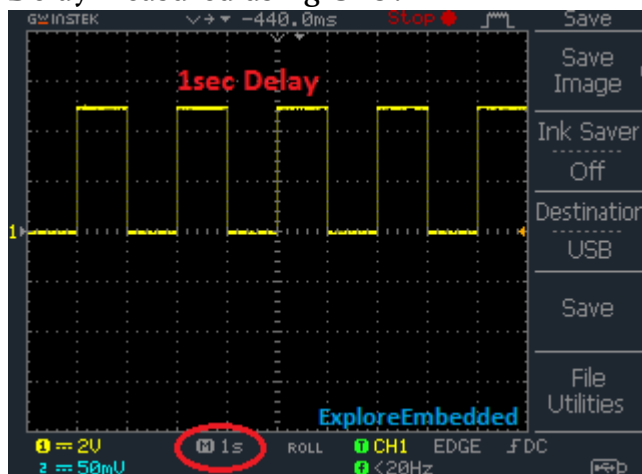
```

25     }
26 }
27 }
28
29
30 void main()
31 {
32     TRISD=0x00;    //Configure PORTD as output to blink the LEDs
33
34     T2CON = (1<<SBIT_T2CKPS1); // Timer2 with external freq and 16 as prescalar
35     TMR2=100;      // Load the timer value for 500us delay
36     TMR2IE=1;      //Enable timer interrupt bit in PIE1 register
37     GIE=1;         //Enable Global Interrupt
38     PEIE=1;        //Enable the Peripheral Interrupt
39     TMR2ON = 1;
40
41     while(1)
42     {
43         PORTD = value;
44     }
45 }

```

bagwan/93556fb952ef8f199fb302db27207230/raw/3b5aca625920e846b67c42bb8b4adab8bb0daf64/pic16f877a_timer2.c)
 pic16f877a_timer2.c (https://gist.github.com/SaheblalBagwan/93556fb952ef8f199fb302db27207230#file-pic16f877a_timer2-c) hosted with ❤ by GitHub (<https://github.com>)

Delay measured using CRO:



(/wiki/File:Timer_2_Cro.png)





(/wiki/File:00_Lpc1768_Timer.jpg)

Downloads

Download the complete project folder from the below link:

Hardware design Files and Code Library

(https://github.com/ExploreEmbedded/Pic16f877a_ExploreUltraPicDevKit/archive/master.zip)

Have a opinion, suggestion , question or feedback about the article let it out here!

ALSO ON [EXPLOREEMBEDDED.COM/WIKI](https://www.exploreembedded.com/wiki)

15 days ago

Interfacing ...

4 days ago


Difference ...




Join the discussion...


LOG IN WITH

OR SIGN UP WITH DISQUS  3 **Share****Best** **Newest** **Oldest**


C **comedor de curioso** — 
2 years ago
man, you're such a handsome guy/girl <3 thanks a lot
o o Reply • Share >



Rezaul Islam — 
3 years ago
IT IS REALLY VERY HELPFUL
o o Reply • Share >

U **Uliah Shahnovich** — 
4 years ago
Hey great tutorial! thank you very much, helped me a lot.
Not a big expert in PIC yet, in my case had to change the
interrupt syntax to run with xc8, hope that will help to sb
:)
void __interrupt () timer_isr(void)
o o Reply • Share >



Matiroce — 
5 years ago
thank you it's so helpful
o o Reply • Share >

D

Dinesh M — 
5 years ago
hai i dont know how to generate a delay using timer



...but, I don't know how to generate a delay using timer
interrupt in pic16f877a , i want to put that delay
comment anywhere in program

o o Reply • Share ›

T

Temitayo Abel

— 

6 years ago

please i dont understand the reason why you gave
TMR1H=0x0B;
TMR1L=0xDC;i was expecting digit value just like the
TMR0... please help clear the confusion.

o o Reply • Share ›

A

Aswinth Raj Balasubramani

— 

7 years ago

Hi your timero Program has a mistake.
while calculating the TMR0 value you have calculated for
100Us instead of 100msec. Please correct it as it might be
misleading.

Other than that great article !!!!

o o Reply • Share ›



Explore More



Categories (/wiki/Special:Categories): PIC Tutorials (/wiki/Category:PIC_Tutorials)
| Starter PIC16F877 (/wiki/Category:Starter_PIC16F877)

Subscribe to hear about our latest Explorations!

name@example.com

SUBSCRIBE

[Contact \(/contact\)](/contact) [About \(/about\)](/about) [Warranty \(/refund\)](/refund) [Terms & Conditions \(/terms\)](/terms) [Reward points 🎁 \(/rewards\)](/rewards)



(<https://twitter.com/exploreembedded>)



(<https://www.facebook.com/ExploreEmbedded/>)



(<https://www.youtube.com/channel/UCvXGpvPuosEI-ALxvCrSbaA>)



(<https://github.com/ExploreEmbedded>)



Now shipping worldwide from India with ♥

