

# How to use Timer / Counter in W7100A

Version 1.1



© 2011 WIZnet Co.,Ltd. All Rights Reserved.

For more information, visit our website at <http://www.wiznet.co.kr>

# Table of Contents

Table of Contents .....	2
1 Introduction.....	4
2 Timer0/Counter0.....	4
2.1 Timer0 8bit Timer, Software Gated .....	6
2.2 Timer0 8bit Timer, Hardware Gated .....	7
2.3 Timer0 8bit Counter, Software Gated .....	7
2.4 Timer0 8bit Counter, Hardware Gated .....	8
2.5 Timer0 16bit Timer, Software Gated .....	9
2.6 Timer0 16bit Timer, Hardware Gated.....	9
2.7 Timer0 16bit Counter, Software Gated .....	10
2.8 Timer0 16bit Counter, Hardware Gated .....	10
2.9 Timer0 8bit Auto Reload Timer, Software Gated .....	11
2.10 Timer0 8bit Auto Reload Timer, Hardware Gated .....	12
2.11 Timer0 8bit Auto Reload Counter, Software Gated.....	12
2.12 Timer0 8bit Auto Reload Counter, Hardware Gated.....	13
2.13 Timer0 Two 8-bit Timer, Software Gated .....	14
2.14 Timer0 Two 8-bit Timer, Hardware Gated .....	14
2.15 Timer0 Two 8-bit Counter, Software Gated.....	15
2.16 Timer0 Two 8-bit Counter, Hardware Gated.....	16
3 Timer1/Counter1.....	18
3.1 Timer1 8bit Timer, Software Gated .....	19
3.2 Timer1 8bit Timer, Hardware Gated .....	20
3.3 Timer1 8bit Counter, Software Gated .....	20
3.4 Timer1 8bit Counter, Hardware Gated .....	21
3.5 Timer1 16bit Timer, Software Gated .....	21
3.6 Timer1 16bit Timer, Hardware Gated.....	22
3.7 Timer1 16bit Counter, Software Gated .....	23
3.8 Timer1 16bit Counter, Hardware Gated .....	23
3.9 Timer1 8bit Auto Reload Timer, Software Gated .....	24
3.10 Timer1 8bit Auto Reload Timer, Hardware Gated .....	25
3.11 Timer1 8bit Auto Reload Counter, Software Gated.....	25
3.12 Timer1 8bit Auto Reload Counter, Hardware Gated.....	26
4 Timer2/Counter2.....	28
4.1 Timer2 16bit Auto Reload Timer .....	30
4.2 Timer2 16bit up/down Auto Reload Counter .....	30

4.3	Timer2 16bit Capture Timer .....	31
4.4	Timer2 16bit Capture Counter .....	32
4.5	Timer2 Baud Rate Generator.....	33
5	Watchdog Timer .....	34
5.1	Watchdog Timer for Interrupt Application .....	34
5.2	Watchdog Timer for Reset Application .....	35
6	Running Example .....	37
6.1	Make a KEIL project.....	37
6.2	Make a HEX file with compile .....	37
6.3	Download the HEX file to iMCU7100EVB .....	38
6.4	Run the Timer/Counter .....	38
	Document History Information .....	40

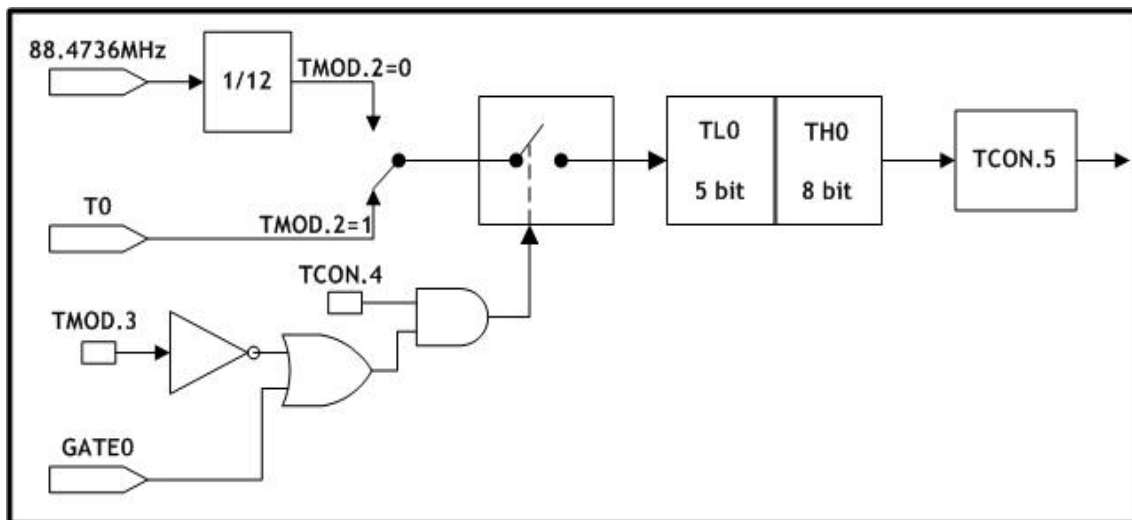
## 1 Introduction

This Application Note explains about the internal Timer/Counter\_0, 1, and 2 of W7100, Watchdog Timer, and basic example codes. Please refer to the W7100 data sheet for more details on the W7100 Timer/Counter related Register.

The examples that will appear later on shows the GPIO port P0\_3 and P0\_4 toggling by using Timer/Counter of W7100. Since P0\_3 and P0\_4 of iMCU7100EVB are connected with LED, use the example code to connect P0\_3 with LED, and LED will blink. Note that because of the rapidly working Timer, the LED looks like it is on all the time. All example codes are written based on C language and KEIL compiler. (Note: User must check the available GPIO pin because the GPIO pins of W7100A QFN 64pin package are different from the LQFP 100pin package.)

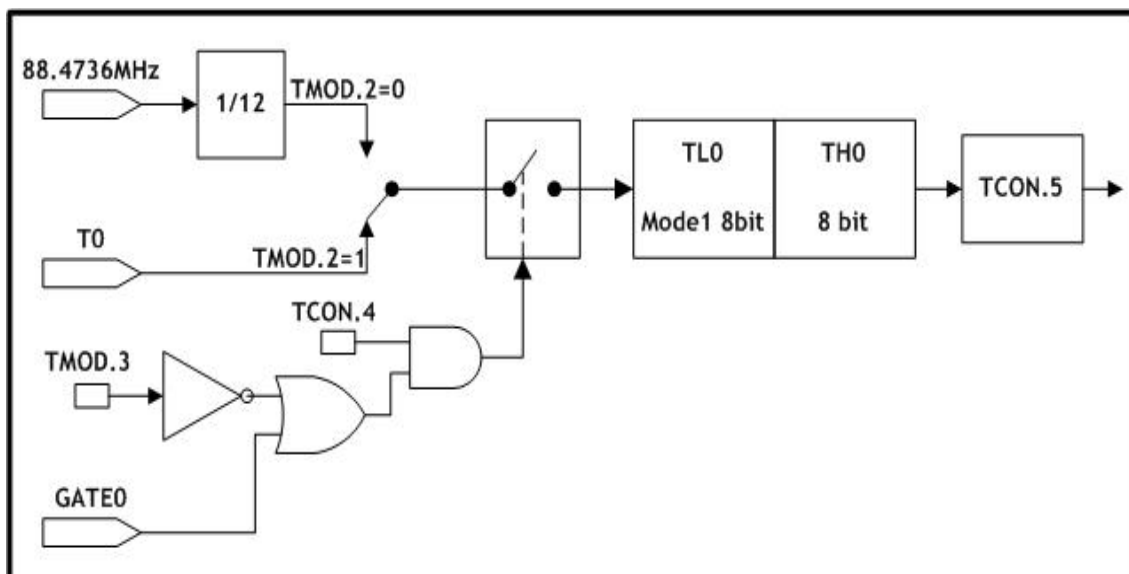
## 2 Timer0/Counter0

Before using Timer/Counter, this section will describe the structure of Timer/Counter. Fig.2.1 shows the structure of Timer0/Counter0 Mode0: 8-Bit Timer/Counter with a prescaler. The TL0's 5bit used for prescaler and TH0's 8bit is used for Timer/Counter. From the figure, 88.4736MHz is the internal clock speed, T0 is the external counter input, and TMOD.3 is the Software Gate Control bit. Also, GATE0 is the Hardware Gate Control pin, and TCON.4 is the Timer0 Start bit.



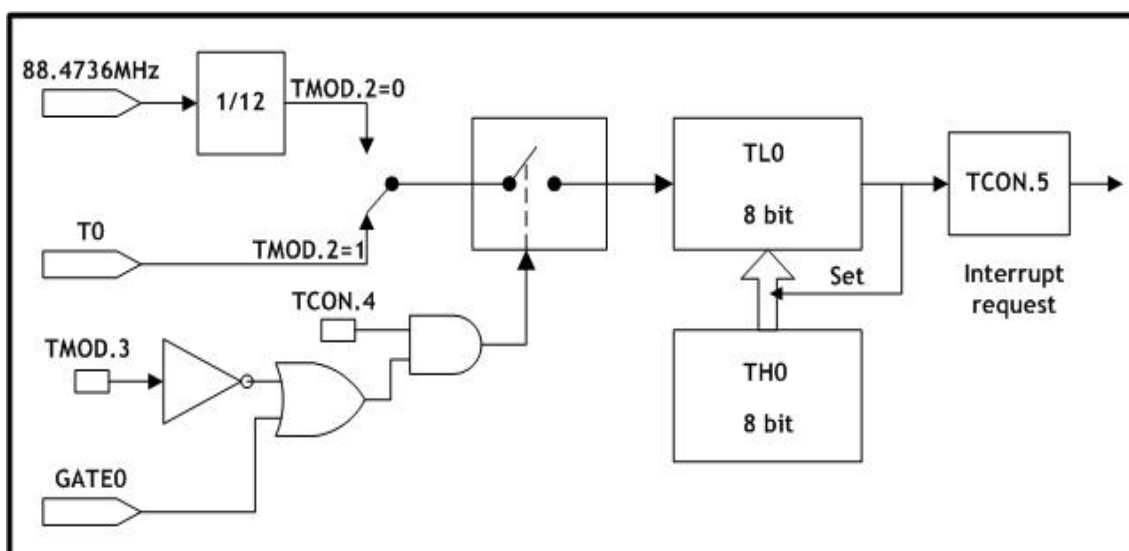
<Fig.2.1> Timer/Counter0, Mode0: 8-Bit Timer/Counter

Fig.2.2 is the structure of Timer0/Counter0 Mode1: 16-Bit Timer/Counter; it operates as a 16bit Timer/Counter using TL0's 8bit and TH0's 8bit.



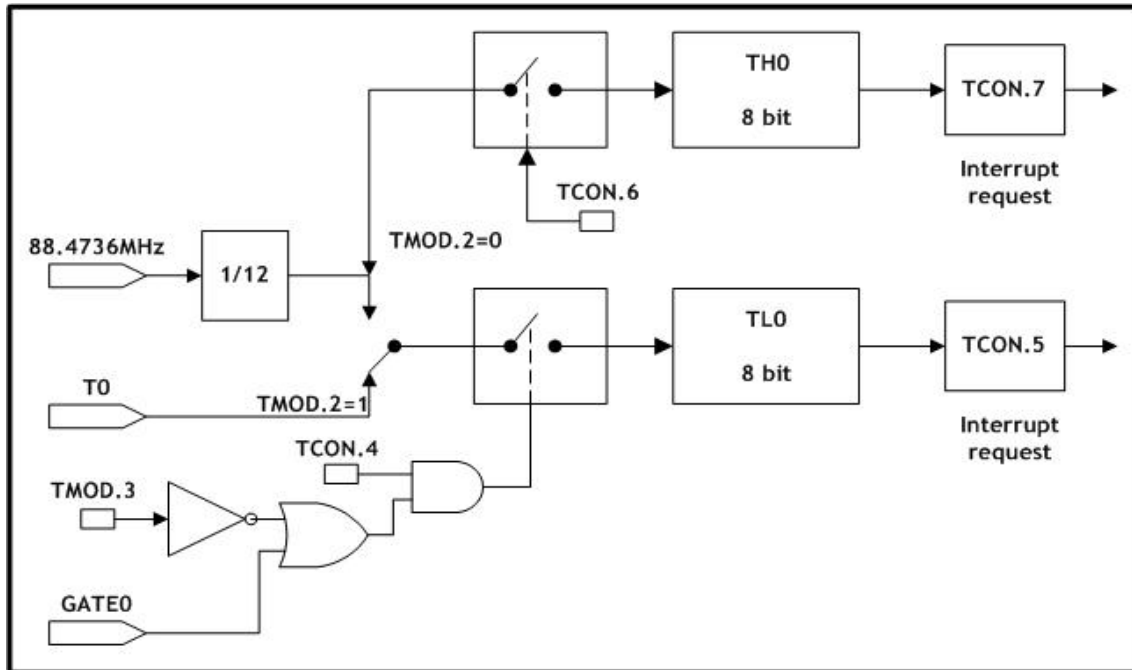
<Fig.2.2> Timer/Counter0, Mode1: 16-Bit Timer/Counter

Fig.2.3 shows the structure of Timer0/Counter0 Mode2: 8-Bit Timer/Counter with Auto-Reload. If an interrupt occurs during this mode, reload the saved value of TH0 register to the TL0 register. Input the initial value in TL0, and for TH0 input the value that will reload when an interrupt is occurred.



<Fig.2.3> Timer/Counter0, Mode2: 8-Bit Timer/Counter with Auto-Reload

Fig.2.4 shows the structure of Timer0/Counter0 Mode3: Two 8-Bit Timer/Counter. For this mode, Timer1 and Timer0 is operated using both TH0 register and TL0 register simultaneously. The interrupts can respectively occur depending on Timer0 and Timer1. TCON.6 is the START bit of Timer1.



<Fig.2.4> Timer/Counter0, Mode3: Two 8-Bit Timers/Counters

The next section will explain the example codes for Timer/Counter0 above.

## 2.1 Timer0 8bit Timer, Software Gated

```
void main(void)
{
    TMOD = 0x00;           //Timer0 mode0 8bit Timer, Software Gated
    TH0 = 0; TL0 = 0;      //TH0, TL0 setting
    ET0 = 1;               //Enable Timer0 Interrupt
    EA = 1;                // Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;               //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
}
```

```
EA = 1;
}
```

Set Timer0 to Mode0 8bit timer, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer0 interrupt using TH0 and TL0. After that, start Timer0 by setting TR0 bit. If an interrupt occurs, reset the Timer0 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

## 2.2 Timer0 8bit Timer, Hardware Gated

```
void main(void)
{
    TMOD = 0x08;           //Timer0 mode0 8bit Timer, Hardware Gated
    TH0 = 0; TL0 = 0;      //TH0, TL0 setting
    ET0 = 1;               // Enable Timer0 Interrupt
    EA = 1;                // Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;               //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}
```

All the steps are the same as section 2.1 except the part for Hardware Gating Control of TMOD register. In this mode, Timer0 operates only when the GATE0 pin is Set.

## 2.3 Timer0 8bit Counter, Software Gated

```
void main(void)
{
    TMOD = 0x04;           //Timer0 mode0 8bit Counter, Software Gated
    TH0 = 0; TL0 = 0;      //TH0, TL0 setting
    ET0 = 1;               //Enable Timer0 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
}
```

```

while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;                //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;           //Toggling P0_3
    EA = 1;
}
    
```

Set Timer0 to Mode0 8bit Counter, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer0 using TH0 and TL0. After that, start Counter0 by setting TR0 bit. Start counting after confirming the input in T0 pin. If an interrupt occurs, reset Timer0 the interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

## 2.4 Timer0 8bit Counter, Hardware Gated

```

void main(void)
{
    TMOD = 0x0C;            //Timer0 mode0 8bit Counter, Hardware Gated
    TH0 = 0; TL0 = 0;       //TH0, TL0 setting
    ET0 = 1;                // Enable Timer0 Interrupt
    EA = 1;                 // Enable Global Interrupt
    TR0 = 1;                //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    TF0 = 0;                //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;           //Toggling P0_3
}
    
```

All the steps are the same as section 2.3 except the part for Hardware Gating Control of TMOD register. In this mode, Timer0 operates only when the GATE0 pin is Set.



## 2.5 Timer0 16bit Timer, Software Gated

```

void main(void)
{
    TMOD = 0x01;           //Timer0 mode1 16bit Timer, Software Gated
    TH0 = 0; TL0 = 0;      //TH0, TL0 setting
    ET0 = 1;               //Enable Timer0 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;               //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}

```

Set Timer0 to Mode1 16bit Timer, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer0 using TH0 and TL0. After that, start Timer0 by Setting TR0 bit. If an interrupt occurs, reset the Timer0 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

## 2.6 Timer0 16bit Timer, Hardware Gated

```

void main(void)
{
    TMOD = 0x09;           //Timer0 mode1 16bit Timer, Hardware Gated
    TH0 = 0; TL0 = 0;      //TH0, TL0 setting
    ET0 = 1;               //Enable Timer0 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
}

```

```

TF0 = 0;                //Timer0 Interrupt Flag reset
P0_3 = ~P0_3;           //Toggling P0_3
EA = 1;
}

```

All the steps are the same as section 2.5 except the part for Hardware Gating Control of TMOD register. In this mode, Timer0 operates only when the GATE0 pin is Set.

## 2.7 Timer0 16bit Counter, Software Gated

```

void main(void)
{
    TMOD = 0x05;          // Timer0 mode1 16bit Counter, Software Gated
    TH0 = 0; TL0 = 0;     //TH0, TL0 setting
    ET0 = 1;              //Enable Timer0 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR0 = 1;              //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;              //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;         //Toggling P0_3
    EA = 1;
}

```

Set Timer0 to 16bit Counter, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer0 using TH0 and TL0. After that, start Counter0 by Setting TR0 bit. Start Counting after confirming the input in T0 pin. If an interrupt occurs, reset the Timer0 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

## 2.8 Timer0 16bit Counter, Hardware Gated

```

void main(void)
{
    TMOD = 0x0D;          // Timer0 mode1 16bit Counter, Hardware Gated
    TH0 = 0; TL0 = 0;     //TH0, TL0 setting
}

```

```

ET0 = 1;                                //Enable Timer0 Interrupt
EA = 1;                                //Enable Global Interrupt
TR0 = 1;                                //Timer0 Start
while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;                            //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;                        //Toggling P0_3
    EA = 1;
}
    
```

All the steps are the same as section 2.7 except the part for Hardware Gating Control of TMOD register. In this mode, Timer0 operates only when the GATE0 pin is Set.

## 2.9 Timer0 8bit Auto Reload Timer, Software Gated

```

void main(void)
{
    TMOD = 0x02;                        // Timer0 mode2 8bit Auto Reload, Software Gated
    TH0 = Reload Value;                 //Reload value TH0 setting
    TL0 = Initial Value;               //Initial value TL0 setting
    ET0 = 1;                            //Enable Timer0 Interrupt
    EA = 1;                            //Enable Global Interrupt
    TR0 = 1;                            //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;                            //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;                        //Toggling P0_3
    EA = 1;
}
    
```

Set Timer0 to 8bit Auto Reload Timer, Software Gated through the TMOD register, and set the interrupt

occurrence cycle of Timer0 using TH0 and TL0. Set the Reload value to TH0 and initial value to TL0. After that, start Timer0 by Setting TR0 bit. Start Counting after confirming the input in T0 pin. If an interrupt occurs, reset the Timer0 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

## 2.10 Timer0 8bit Auto Reload Timer, Hardware Gated

```
void main(void)
{
    TMOD = 0x0A;           // Timer0 mode2 8bit Auto Reload, Hardware Gated
    TH0 = Reload Value;    //Reload value TH0 setting
    TL0 = Initial Value;   //Initial value TL0 setting
    ET0 = 1;               //Enable Timer0 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;               //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}
```

All the steps are the same as section 2.9 except the part for Hardware Gating Control of TMOD register. In this mode, Timer0 operates only when the GATE0 pin is Set.

## 2.11 Timer0 8bit Auto Reload Counter, Software Gated

```
void main(void)
{
    TMOD = 0x06;           // Timer0 mode2 8bit Auto Reload, Software Gated
    TH0 = Reload Value;    //Reload value TH0 setting
    TL0 = Initial Value;   //Initial value TL0 setting
    ET0 = 1;               //Enable Timer0 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
}
```

```

while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;           //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;      //Toggling P0_3
    EA = 1;
}
    
```

Set Timer0 to 8bit Auto Reload Timer, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Counter0 using TH0 and TL0. Set the Reload value to TH0 and initial value to TL0. After that, start Counter0 by Setting TR0 bit. If an interrupt occurs, reset the Timer0 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

## 2.12 Timer0 8bit Auto Reload Counter, Hardware Gated

```

void main(void)
{
    TMOD = 0x0E;           // Timer0 mode2 8bit Auto Reload, Hardware Gated
    TH0 = Reload Value;    //Reload value TH0 setting
    TL0 = Initial Value;   //Initial value TL0 setting
    ET0 = 1;               //Enable Timer0 Interrupt
    EA = 1;                 //Enable Global Interrupt
    TR0 = 1;               //Timer0 Start
    while(1);
}

void int_test(void) interrupt 1
{
    EA = 0;
    TF0 = 0;           //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;      //Toggling P0_3
    EA = 1;
}
    
```

All the steps are the same as section 2.11 except the part for Hardware Gating Control of TMOD register. In this mode, Timer0 operates only when the GATE0 pin is Set.

## 2.13 Timer0 Two 8-bit Timer, Software Gated

```

void main(void)
{
    TMOD = 0x03;           // Timer0 mode3 Split timer, Software Gated
    TH0 = 0; TL0 = 0;      //TH0, TL0 setting
    ET0 = 1; ET1 = 1;      //Enable Timer0, Timer1 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR0 = 1; TR1 = 1;      //Timer0, Timer1 Start
    while(1);
}

void int_test1(void) interrupt 1
{
    EA = 0;
    TF0 = 0;               //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}

void int_test2(void) interrupt 3
{
    EA = 0;
    TF1 = 0;               //Timer0 Interrupt Flag reset
    P0_4 = ~P0_4;          //Toggling P0_4
    EA = 1;
}

```

Set TMOD register to Mode3 Split timer, Software Gated, and set the interrupt occurrence cycle of Timer0/Timer1 using TH0 and TL0. After that set TR0 and TR1 to operate the Split Timer. If the interrupt of Timer0/Timer1 occurs, reset the Timer0/Timer1 interrupt; then execute the action the user has set. In this document P0\_3 and P0\_4 is each set to toggle.

## 2.14 Timer0 Two 8-bit Timer, Hardware Gated

```

void main(void)
{
    TMOD = 0x0B;           // Timer0 mode3 Split timer, Software Gated
    TH0 = 0; TL0 = 0;      //TH0, TL0 setting

```

```

    ET0 = 1; ET1 = 1;                //Enable Timer0, Timer1 Interrupt
    EA = 1;                          //Enable Global Interrupt
    TR0 = 1; TR1 = 1;                //Timer0, Timer1 Start
    while(1);
}

void int_test1(void) interrupt 1
{
    EA = 0;
    TF0 = 0;                        //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;                    //Toggling P0_3
    EA = 1;
}

void int_test2(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                        //Timer0 Interrupt Flag reset
    P0_4 = ~P0_4;                    //Toggling P0_4
    EA = 1;
}
    
```

All the steps are the same as section 2.13 except the part for Hardware Gating Control of TMOD register. In this mode, Timer0 operates only when the GATE0 pin is Set.

## 2.15 Timer0 Two 8-bit Counter, Software Gated

```

void main(void)
{
    TMOD = 0x07;                    // Timer0 mode3 Split timer, Software Gated
    TH0 = 0; TL0 = 0;                //TH0, TL0 setting
    ET0 = 1; ET1 = 1;                //Enable Timer0, Timer1 Interrupt
    EA = 1;                          //Enable Global Interrupt
    TR0 = 1; TR1 = 1;                //Timer0, Timer1 Start
    while(1);
}

void int_test1(void) interrupt 1
{
    
```

```

EA = 0;
TF0 = 0;                //Timer0 Interrupt Flag reset
P0_3 = ~P0_3;           //Toggling P0_3
EA = 1;
}
void int_test2(void) interrupt 3
{
    EA = 0;
    TF1 = 0;            //Timer0 Interrupt Flag reset
    P0_4 = ~P0_4;       //Toggling P0_4
    EA = 1;
}
    
```

Set TMOD register to Mode3 Split Counter, Software Gated, and set the interrupt occurrence cycle of Counter0/Counter1 using TH0 and TL0. After that set TR0 and TR1 to operate the Split Timer. If an interrupt of Counter0/Counter1 occurs, reset the Counter0/Counter1 interrupt; then execute the action the user has set. In this document P0\_3 and P0\_4 is each set to toggle.

## 2.16 Timer0 Two 8-bit Counter, Hardware Gated

```

void main(void)
{
    TMOD = 0x0F;        // Timer0 mode3 Split timer, Software Gated
    TH0 = 0; TL0 = 0;    //TH0, TL0 setting
    ET0 = 1; ET1 = 1;    //Enable Timer0, Timer1 Interrupt
    EA = 1;              //Enable Global Interrupt
    TR0 = 1; TR1 = 1;    //Timer0, Timer1 Start
    while(1);
}

void int_test1(void) interrupt 1
{
    EA = 0;
    TF0 = 0;            //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;       //Toggling P0_3
    EA = 1;
}

void int_test2(void) interrupt 3
    
```

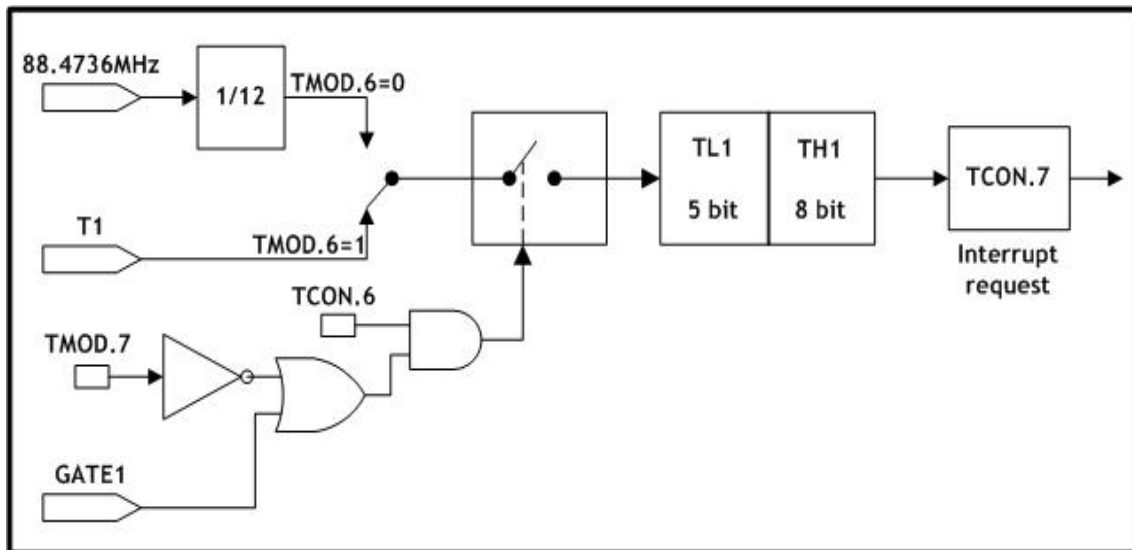


```
{  
    EA = 0;  
    TF1 = 0;                //Timer0 Interrupt Flag reset  
    P0_4 = ~P0_4;          //Toggling P0_4  
    EA = 1;  
}
```

All the steps are the same as section 2.15 except the part for Hardware Gating Control of TMOD register. In this mode, Counter0 operates only when the GATE0 pin is Set.

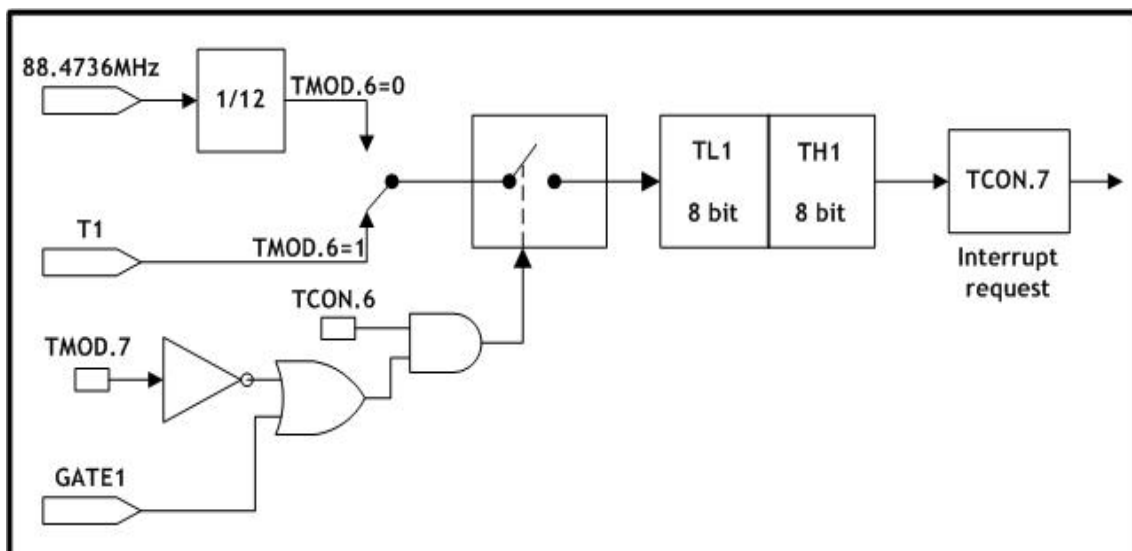
### 3 Timer1/Counter1

Figure 3.1 is the structure of Timer/Counter1 Mode0: 8-Bit Timer/Counter. The structure is very similar to Mode0 of Timer0 except some registers. The 88.4736MHz is the internal clock speed and T1 is the external counter input. TMOD.7 is the Software Gate Control bit, GATE1 is the external Hardware Gate Control pin, and TCON.6 is the Timer1 Start bit.



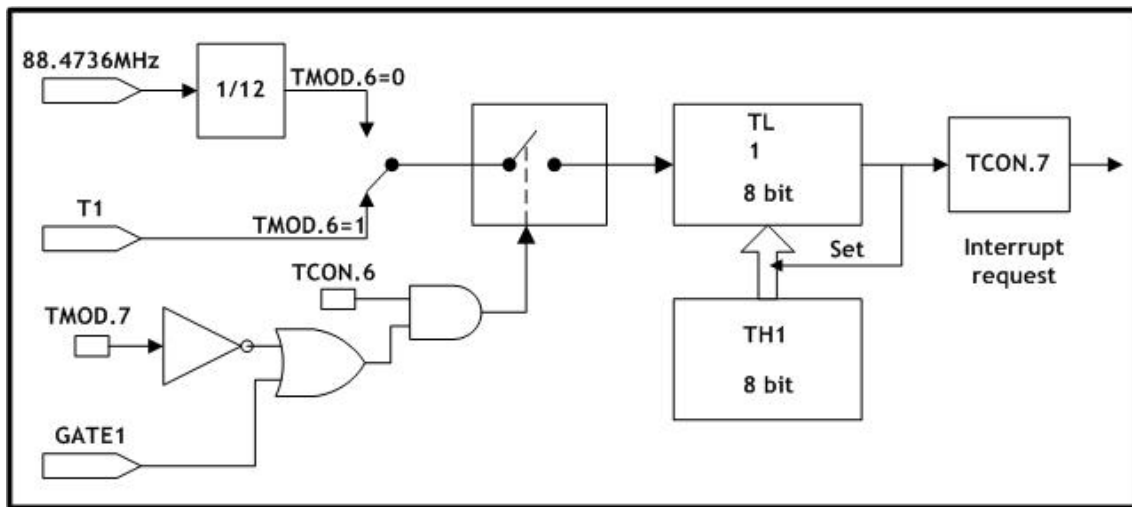
<Fig.3.1> Timer/Counter1, Mode0: 8-Bit Timer/Counter

Figure 3.2 is the structure of Timer/Counter1 Mode1: 16-Bit Timers/Counters. The structure is very similar to Mode 1 of Timer0 except the some registers.



<Fig.3.2> Timer/Counter1, Mode1: 16-Bit Timers/Counters

Fig.3.3 shows the structure of Timer/Counter1 Mode2: 8-Bit Timer/Counter with Auto-Reload. Like the Mode2 of Timer0, Reload the value of TH1 to TL1 if an interrupt occurs.



<Fig.3.3> Timer/Counter1, Mode2: 8-Bit Timer/Counter with Auto-Reload

Please refer to Mode3 of Timer0 for information on Timer/Counter1 Mode3.

### 3.1 Timer1 8bit Timer, Software Gated

```
void main(void)
{
    TMOD = 0x00;                //Timer1 mode0 8bit Timer, Software Gated
    TH1 = 0; TL1 = 0;           //TH1, TL1 setting
    ET1 = 1;                    //Enable Timer1 Interrupt
    EA = 1;                     // Enable Global Interrupt
    TR1 = 1;                     //Timer1 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                    //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;               //Toggling P0_3
    EA = 1;
}
```

Set Timer1 to 8bit timer, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer1 using TH1 and TL1. After that, start Timer1 by Setting TR1 bit. If an interrupt occurs, reset the Timer1 interrupt flag by using interrupt processing function; then execute the action the user

has set. In this document Port 0.3 is set to toggle.

### 3.2 Timer1 8bit Timer, Hardware Gated

```
void main(void)
{
    TMOD = 0x80;                //Timer1 mode0 8bit Timer, Hardware Gated
    TH1 = 0; TL1 = 0;          //TH1, TL1 setting
    ET1 = 1;                    // Enable Timer1 Interrupt
    EA = 1;                     // Enable Global Interrupt
    TR1 = 1;                    //Timer1 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                    //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;               //Toggling P0_3
    EA = 1;
}
```

All the steps are the same as section 3.1 except the part for Hardware Gating Control of TMOD register. In this mode, Timer1 operates only when the GATE1 pin is Set.

### 3.3 Timer1 8bit Counter, Software Gated

```
void main(void)
{
    TMOD = 0x40;                //Timer1 mode0 8bit Counter, Software Gated
    TH1 = 0; TL1 = 0;          //TH1, TL1 setting
    ET1 = 1;                    //Enable Timer1 Interrupt
    EA = 1;                     //Enable Global Interrupt
    TR1 = 1;                    //Timer1 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
}
```

```

TF1 = 0;                //Timer0 Interrupt Flag reset
P0_3 = ~P0_3;           //Toggling P0_3
EA = 1;
}

```

Set Timer1 to 8bit Counter, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer1 using TH1 and TL1. After that, start Counter1 by Setting TR1 bit. Start Counting after confirming the input in T1 pin. If an interrupt occurs, reset the Timer1 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

### 3.4 Timer1 8bit Counter, Hardware Gated

```

void main(void)
{
    TMOD = 0xC0;                //Timer1 mode0 8bit Counter, Hardware Gated
    TH1 = 0; TL1 = 0;           //TH1, TL1 setting
    ET1 = 1;                    // Enable Timer1 Interrupt
    EA = 1;                     // Enable Global Interrupt
    TR1 = 1;                    //Timer0 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                    //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;               //Toggling P0_3
    EA = 1;
}

```

All the steps are the same as section 3.3 except the part for Hardware Gating Control of TMOD register. In this mode, Counter1 operates only when the GATE1 pin is Set.

### 3.5 Timer1 16bit Timer, Software Gated

```

void main(void)
{
    TMOD = 0x10;                //Timer1 mode1 16bit Timer, Software Gated
    TH1 = 0; TL1 = 0;           //TH1, TL1 setting
}

```

```

ET1 = 1;                                //Enable Timer1 Interrupt
EA = 1;                                //Enable Global Interrupt
TR1 = 1;                                //Timer1 Start
while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                            //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;                        //Toggling P0_3
    EA = 1;
}

```

Set Timer1 to 16bit timer, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer1 using TH1 and TL1. After that, start Timer1 by Setting TR1 bit. If an interrupt occurs, reset the Timer1 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

### 3.6 Timer1 16bit Timer, Hardware Gated

```

void main(void)
{
    TMOD = 0x90;                        //Timer1 mode1 16bit Timer, Hardware Gated
    TH1 = 0; TL1 = 0;                   //TH1, TL1 setting
    ET1 = 1;                            //Enable Timer1 Interrupt
    EA = 1;                            //Enable Global Interrupt
    TR1 = 1;                            //Timer1 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                            //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;                        //Toggling P0_3
    EA = 1;
}

```

All the steps are the same as section 3.5 except the part for Hardware Gating Control of TMOD register. In this mode, Timer1 operates only when the GATE1 pin is Set.

### 3.7 Timer1 16bit Counter, Software Gated

```
void main(void)
{
    TMOD = 0x50;                // Timer1 mode1 16bit Counter, Software Gated
    TH1 = 0; TL1 = 0;          //TH1, TL1 setting
    ET1 = 1;                    //Enable Timer1 Interrupt
    EA = 1;                     //Enable Global Interrupt
    TR1 = 1;                    //Timer1 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                    //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;               //Toggling P0_3
    EA = 1;
}
```

Set Timer1 to 16bit Counter, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer1 using TH1 and TL1. After that, start Counter1 by Setting TR1 bit. Start Counting after confirming the input in T1 pin. If an interrupt occurs, reset the Timer1 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

### 3.8 Timer1 16bit Counter, Hardware Gated

```
void main(void)
{
    TMOD = 0xD0;                // Timer1 mode1 16bit Counter, Hardware Gated
    TH1 = 0; TL1 = 0;          //TH1, TL1 setting
    ET1 = 1;                    //Enable Timer1 Interrupt
    EA = 1;                     //Enable Global Interrupt
    TR1 = 1;                    //Timer1 Start
}
```

```

while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}
    
```

All the steps are the same as section 3.7 except the part for Hardware Gating Control of TMOD register. In this mode, Counter1 operates only when the GATE1 pin is Set.

### 3.9 Timer1 8bit Auto Reload Timer, Software Gated

```

void main(void)
{
    TMOD = 0x20;            // Timer1 mode2 8bit Auto Reload, Software Gated
    TH1 = Reload Value;    //Reload value TH1 setting
    TL1 = Initial Value;   //Initial value TL1 setting
    ET1 = 1;               //Enable Timer1 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR1 = 1;               //Timer1 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}
    
```

Set Timer1 to 8bit Auto Reload Timer, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Timer1 using TH1 and TL1. Set the Reload value to TH1 and initial value to TL1. After that, start Timer1 by Setting TR1 bit. If an interrupt occurs, reset the Timer0 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3



is set to toggle.

### 3.10 Timer1 8bit Auto Reload Timer, Hardware Gated

```
void main(void)
{
    TMOD = 0xA0;           // Timer1 mode2 8bit Auto Reload, Hardware Gated
    TH1 = Reload Value;    //Reload value TH1 setting
    TL1 = Initial Value;   //Initial value TL1 setting
    ET1 = 1;               //Enable Timer1 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR1 = 1;               //Timer1 Start
    while(1);
}

void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;               //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}
```

All the steps are the same as section 3.9 except the part for Hardware Gating Control of TMOD register. In this mode, Timer1 operates only when the GATE1 pin is Set.

### 3.11 Timer1 8bit Auto Reload Counter, Software Gated

```
void main(void)
{
    TMOD = 0x60;           // Timer1 mode2 8bit Auto Reload, Software Gated
    TH1 = Reload Value;    //Reload value TH1 setting
    TL1 = Initial Value;   //Initial value TL1 setting
    ET1 = 1;               //Enable Timer1 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR1 = 1;               //Timer1 Start
    while(1);
}
```

```
void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;           //Toggling P0_3
    EA = 1;
}
```

Set Timer1 to 8bit Auto Reload Timer, Software Gated through the TMOD register, and set the interrupt occurrence cycle of Counter0 using TH1 and TL1. Set the Reload value to TH1 and initial value to TL1. After that, start Counter1 by Setting TR1 bit. If an interrupt occurs, reset the Timer0 interrupt flag by using interrupt processing function; then execute the action the user has set. In this document Port 0.3 is set to toggle.

### 3.12 Timer1 8bit Auto Reload Counter, Hardware Gated

```
void main(void)
{
    TMOD = 0xE0;            // Timer1 mode2 8bit Auto Reload, Hardware Gated
    TH1 = Reload Value;     //Reload value TH1 setting
    TL1 = Initial Value;    //Initial value TL1 setting
    ET1 = 1;                //Enable Timer1 Interrupt
    EA = 1;                 //Enable Global Interrupt
    TR1 = 1;                //Timer1 Start
    while(1);
}

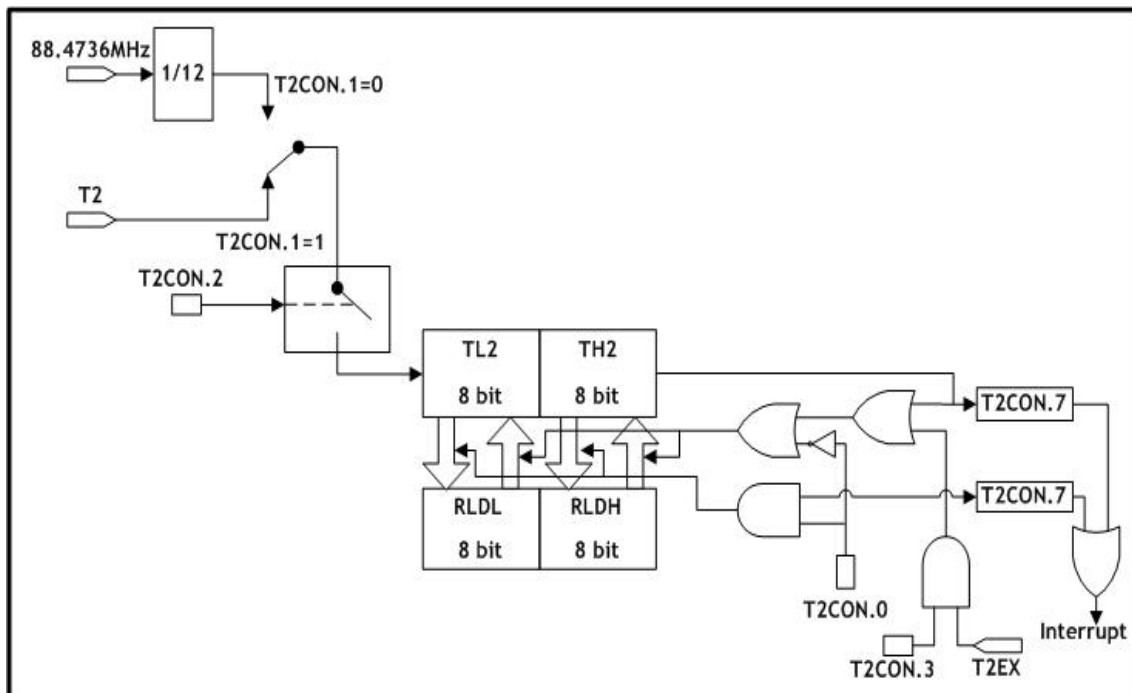
void int_test(void) interrupt 3
{
    EA = 0;
    TF1 = 0;                //Timer0 Interrupt Flag reset
    P0_3 = ~P0_3;           //Toggling P0_3
    EA = 1;
}
```

All the steps are the same as section 3.11 except the part for Hardware Gating Control of TMOD register. In this mode, Counter1 operates only when the GATE1 pin is Set.



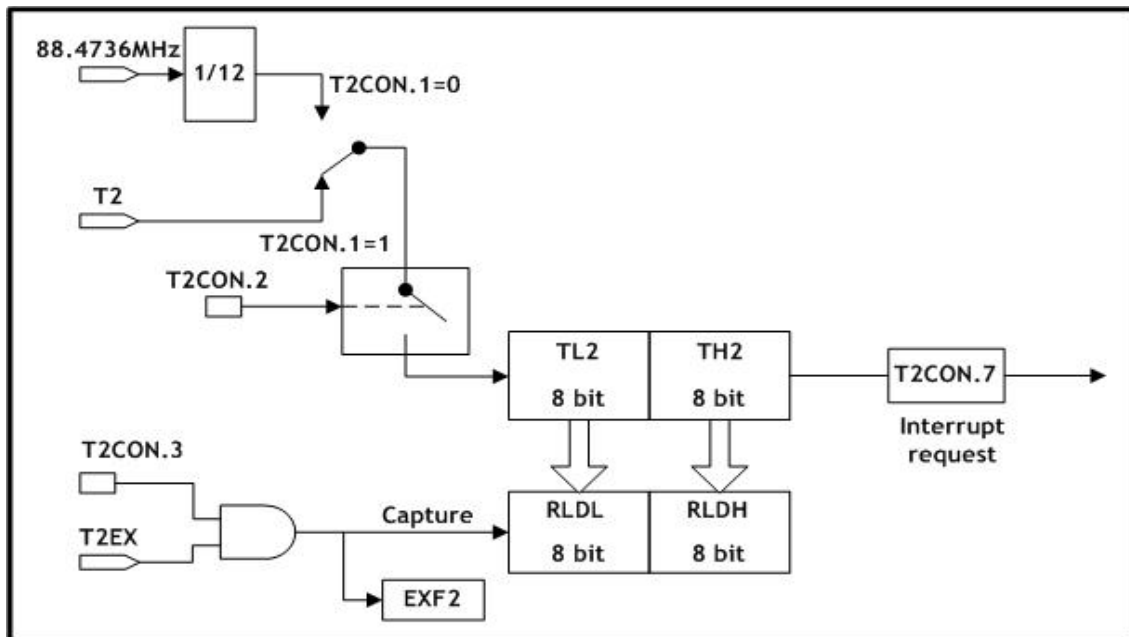
## 4 Timer2/Counter2

Fig.4.1 is the structure of Timer/Counter2 16-Bit Timer/Counter with Auto-Reload. Once the Timer/Counter starts, an Overflow occurs in the TL2 register and TH2 register, and an interrupt occurs. 88.4736MHz is the internal clock speed and T2 is the external counter input. T2CON.1 is the Timer/Counter Select bit and T2CON.2 is the Timer2 START bit. The T2EX pin in the right side of Fig.4.1, is the Timer2 Capture/Reload trigger. It operates at the Falling Edge. The T2CON.3 is the T2EX pin Enable bit, and T2CON.0 is the Capture/Reload Select bit that operates on Capture mode when the bit is 1.



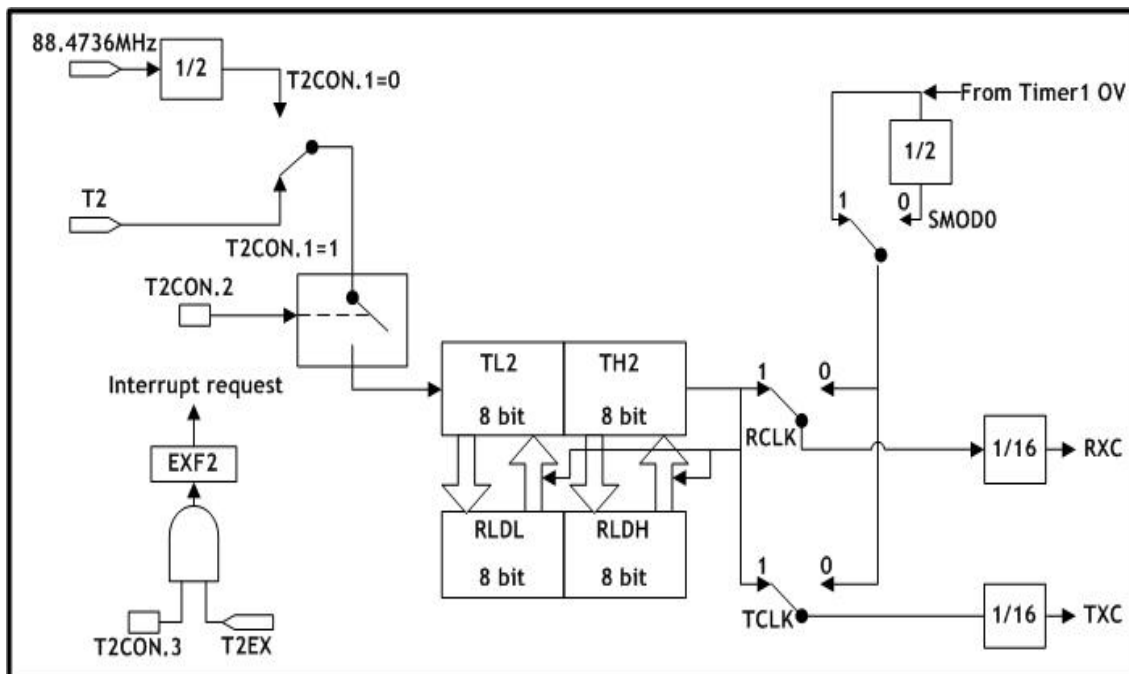
<Fig.4.1> Timer/Counter2, 16-Bit Timer/Counter with Auto-Reload

Fig.4.2 is the structure of Timer/Counter2 16-Bit Timer/Counter with Capture Mode. When turn on the capture sign by using the T2CON.3 and the Falling Edge of T2EX pin, each register value of TL2 and TH2 capture into RLDL and RLDH register. The EXF2 is the External Flag that is set to 1 when an external pin is captured.



<Fig.4.2> Timer/Counter2, 16-Bit Timer/Counter with Capture Mode

Fig.4.3 is the structure of Timer2 Baud Rate Generator Mode. Baud Rate Generator can be driven by both Timer 1 and Timer2. SMOD0 is the Timer1 Baud Rate Speed Control Bit.



<Fig.4.3> Timer2 for Baud Rate Generator Mode

## 4.1 Timer2 16bit Auto Reload Timer

```

void main(void)
{
    T2CON = 0x00;           // Timer2 16bit up/down Auto Reload
    TH2 = MSB Initial Value; //Initial value TH2 setting
    TL2 = LSB Initial Value; //Initial value TL2 setting
    RLDH = MSB Reload Value; //Reload value RLDH setting
    RLDL = LSB Reload Value; //Reload value RLDL setting
    ET2 = 1;               //Enable Timer2 Interrupt
    EA = 1;                //Enable Global Interrupt
    TR2 = 1;               //Timer2 Start
    while(1);
}

void int_test(void) interrupt 5
{
    EA = 0;
    TF2 = 0;               //Timer2 Interrupt Flag reset
    P0_3 = ~P0_3;          //Toggling P0_3
    EA = 1;
}

```

Use T2CON register to set Timer2 to 16bit Auto Reload Timer, and set each MSB and LSB initial value for TH2 and TL2. Also, set the value that will reload in case of an interrupt in RLDH and RLDL. Then, set the TR2 bit to start Timer2. If an interrupt occurs, reset the Timer2 interrupt flag by using interrupt function and execute the action the user has set. In this document Port 0.3 is set to toggle.

## 4.2 Timer2 16bit up/down Auto Reload Counter

```

void main(void)
{
    T2CON = 0x00;           // Timer2 16bit up/down Auto Reload
    CT2 = 1;               // Counter mode ON
    TH2 = MSB Initial Value; //Initial value TH2 setting
    TL2 = LSB Initial Value; //Initial value TL2 setting
    RLDH = MSB Reload Value; //Reload value RLDH setting
    RLDL = LSB Reload Value; //Reload value RLDL setting
    ET2 = 1;               //Enable Timer2 Interrupt
}

```

```

EA = 1;                                //Enable Global Interrupt
TR2 = 1;                                //Timer2 Start
while(1);

}

void int_test(void) interrupt 5
{
    EA = 0;
    TF2 = 0;                            //Timer2 Interrupt Flag reset
    P0_3 = ~P0_3;                        //Toggling P0_3
    EA = 1;
}

```

Use T2CON register to set Timer2 to 16bit Auto Reload Timer, and set CT2bit, which is the 2<sup>nd</sup> bit of T2CON register, to switch to counter mode. An approach classified by bit is possible, but related codes other than W7100.h file should be defined. Then, set each MSB and LSB initial value for TH2 and TL2, and set the value that will Reload in case of an interrupt in RLDH and RLDL. Then, Set TR2 to start Timer2. If an interrupt occurs, reset the Timer2 interrupt flag by using interrupt function and execute the action the user has set. In this document Port 0.3 is set to toggle.

### 4.3 Timer2 16bit Capture Timer

```

void main()
{
    T2CON = 0x00;                        // Timer2 16bit up/down Auto Reload
    EXEN2 = 1;                           //Enable T2EX pin (Active on Falling edge)
    CPRL2 = 1;                            //Capture mode ON
    TH2 = 0; TL2 = 0;                    // TH2, TL2 setting
    RLDH = 0; RLDL = 0;                  // Clear the Capture variable RLDH, RLDL
    EA = 1; ET2 = 1;                     // Interrupt setting
    TR2 = 1;                             // Timer2 Start
    while(1);
}

void int_test2(void) interrupt 5
{
    EA = 0;
    if(EXF2)                             // If EXF2 = 1 (Capture Flag)

```

```

{
    RLDH_tmp = RLDH; // Save the Captured value(RLDH) to temporal variable(RLDH_tmp)
    RLDL_tmp = RLDL; // Save the Captured value(RLDL) to temporal variable(RLDL_tmp)
}
EXF2 = 0;           // Reset the Capture Flag
TF2 = 0;           // Reset the Timer2 Interrupt Flag
P0_3 = ~P0_3;      // Toggling P0_3
}
EA = 1;
}

```

Use T2CON register to set Timer2 to 16bit Auto Reload Timer, and set the EXEN2 bit, which is the 3<sup>rd</sup> bit of T2CON register, to activate the T2EX pin. Note that T2EX pin works at the 1= $\Rightarrow$ 0 Transition. Then set the CPRL2 bit, which is the 0<sup>th</sup> bit of T2CON register, to activate Capture mode. An approach classified by bit is possible for T2CON register, but related codes other than W7100.h file should be defined. Set the value for TH2 and TL2 to set the interrupt time. After that, set TR2 bit to start Timer2.

When Timer2 is running and the Falling sign (1  $\Rightarrow$  0) in the T2EX pin is confirmed, Capture the value of TH2 and TL2, and save them to RLDH and RLDL. When Timer2 interrupt occurs, check whether the Capture motion occurred. If there is a Capture value, save them each to RLDH\_tmp and RLDL\_tmp. Then, Reset the Capture Flag and Timer2 Interrupt Flag. When Timer2 Interrupt occurs, Port 0.3 is set to toggle.

## 4.4 Timer2 16bit Capture Counter

```

void main()
{
    T2CON = 0x00;           // Timer2 16bit up/down Auto Reload
    CT2 = 1;               // Counter mode ON
    EXEN2 = 1;             // Enable T2EX pin (Active on Falling edge)
    CPRL2 = 1;             // Capture mode ON
    TH2 = 0; TL2 = 0;      // TH2, TL2 setting
    RLDH = 0; RLDL = 0;    // Clear the Capture variable RLDH, RLDL
    EA = 1; ET2 = 1;       // Interrupt setting
    TR2 = 1;              // Timer2 Start
    while(1);
}

void int_test2(void) interrupt 5

```



```

{
    EA = 0;
    if(EXF2)                // If EXF2 = 1 (Capture Flag)
    {
        RLDH_tmp = RLDH;    // Save the Captured value(RLDH) to temporal variable(RLDH_tmp)
        RLDL_tmp = RLDL;    // Save the Captured value(RLDL) to temporal variable(RLDL_tmp)
    }
    EXF2 = 0;               // Reset the Capture Flag
    TF2 = 0;                // Reset the Timer2 Interrupt Flag
    P0_3 = ~P0_3;           // Toggling P0_3
}
EA = 1;
}

```

Use T2CON register to set Timer2 to 16bit Auto Reload Timer, and set the CT2 bit, which is the 1<sup>st</sup> bit of T2CON register, to activate the T2EX pin. Note that T2EX pin works at the 1= $\Rightarrow$ 0 Transition. Then set the CPRL2 bit, which is the 0<sup>th</sup> bit of T2CON register, to activate Capture mode. An approach classified by bit is possible for T2CON register, but related codes other than W7100.h file should be defined. Set the value for TH2 and TL2 to set the interrupt time. After that, set TR2 bit to start Timer2.

When Counter2 is running and the Falling sign (1  $\Rightarrow$  0) in the T2EX pin is confirmed, Capture the value of TH2 and TL2, and save them to RLDH and RLDL. When Timer2 interrupt occurs, check whether the Capture motion occurred. If there is a Capture value, save them each to RLDH\_tmp and RLDL\_tmp. Then, Reset the Capture Flag and Timer2 Interrupt Flag. When Timer2 Interrupt occurs, Port 0.3 is set to toggle.

## 4.5 Timer2 Baud Rate Generator

```

void main()
{
    T2CON = 0x00;           // Timer2 16bit up/down Auto Reload
    RCLK = 1;               // UART0 receiver is clocked by Timer2 overflow pulses
    TCLK = 1;               // UART0 transmitter is clocked by Timer2 overflow pulses
    TH2 = 0; TL2 = 0;       // TH2, TL2 setting
    RLDH = 0xFF; RLDL = 0xE8; //Setting for Baud Rate 115200bps
    ET2 = 0;                //Timer2 Interrupt disable
    TR2 = 1;                // Timer2 Start for Baud Rate Generator
}

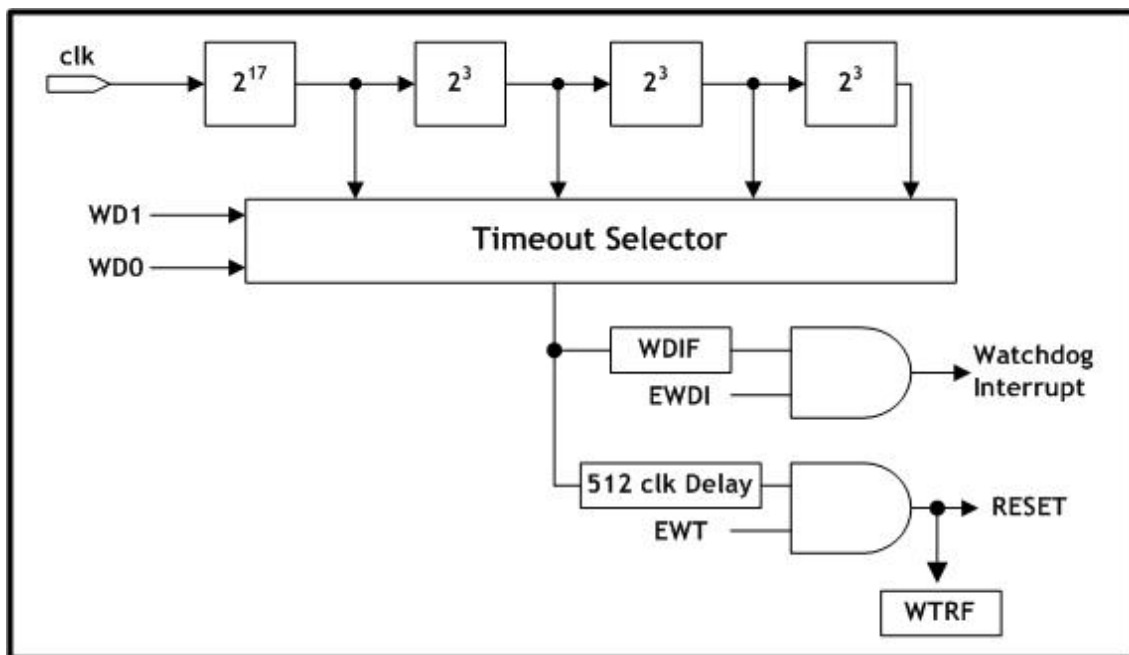
```

This mode is for the Baud Rate Generator of UART. This section explains only about the value for

Timer2. Please refer to W7100 Datasheet for more information on other registers for UART setting. First, initialize T2CON; then set TCLK and RCLK, which are the 4<sup>th</sup> and 5<sup>th</sup> bit of T2CON, to the Baud Rate Generator of UART. Also, set the initial value for TH2 and TL2. For Baud Rate setting, set each value for RLDH and RLDL. In the example, the RLDH and RLDL are set to 0xFF and 0xE8 respectively because the Baud Rate was set to 115200bps. Please refer to W7100 Datasheet section 6.UART for more details on other Baud Rate settings. Then, disable Timer2 Interrupt and set the TR2 bit, which is the 2<sup>nd</sup> bit of T2CON register, to run Baud Rate Generator.

## 5 Watchdog Timer

The Fig.5.1 is the structure of Watchdog Timer. Set the WD1 and WD0 bit of CKCON register to decide the Timeout cycle. The timeout occurs periodically, and interrupt can occur depending on the setting of EWT. WDIF is set once an interrupt occurs. Set EWT so that W7100 can reset. WTRF is set when Reset occurs. If RWT is set periodically before Timeout occurs, then the Watchdog Timer initializes periodically and Timeout does not occur. The CLK is the internal clock of 88.4736MHz. User must done the timed access procedure (TA = 0xAA; TA = 0x55;) before clear or set the watchdog registers.



<Fig.5.1> Watchdog Timer Structure

### 5.1 Watchdog Timer for Interrupt Application

```
void main()
{
    CKCON = 0xC0;    //Watchdog Interval setting
    TA = 0xAA;       //Timed Access Registers setting to prevents the Accidental writes
}
```

```

TA = 0x55;           //For more detail, please refer to the W7100 datasheet
WDCON = 0x00;        //Initialize the WDCON
EA = 1;              //Global Interrupt Enable
TA = 0xAA;
TA = 0x55;
EWDI = 1;            //Enable the Watchdog Interrupt
TA = 0xAA;
TA = 0x55;
EWT = 0;             //Disable the W7100 Reset by Watchdog Timeout
while(1);
}

void int_test3(void) interrupt 12
{
    EA = 0;
    TA = 0xAA;
    TA = 0x55;
    WDIF = 0;         //Reset the Watchdog Timer Interrupt Flag
    P0_3 = ~P0_3;     //Toggling the P0_3
    EA = 1;
}
    
```

This example shows how to use the Watchdog Timer like a regular Timer. Use 6<sup>th</sup> and 7<sup>th</sup> register of CKCON register to set the Watchdog Interval, and set Timed Access Register to prepare for unpredicted situations. Initialize the WDCON register, enable the Global Interrupt, and set EWDI, which is the 4<sup>th</sup> bit of EIE register, to enable the Watchdog Interrupt. Since this example does not use Reset by Watchdog Timer, EWT bit should stay disabled. Also, there are no other settings for running the Timer because the Watchdog Timer is always turned on since the board is turned on or reset. If Watchdog interrupt occurs, P0\_3 toggles by using interrupt clearing function.

## 5.2 Watchdog Timer for Reset Application

```

void main()
{
    CKCON |= 0xC0;    //Watchdog Interval setting
    TA = 0xAA;        //Timed Access Registers setting to prevents the Accidental writes
    TA = 0x55;        //For more detail, please refer to the W7100 datasheet
    WDCON = 0x00;     //Initialize the WDCON
}
    
```

```
TA = 0xAA;
TA = 0x55;
EWDI = 1;           //Enable the W7100 Reset by Watchdog Timeout
TA = 0xAA;
TA = 0x55;
EWT = 1;           //Disable the Watchdog Timer Reset
while(1);
}
```

This example shows how to reset W7100 periodically by using Watchdog Timer. Use 6<sup>th</sup> and 7<sup>th</sup> register of CKCON register to set the Watchdog Interval, and set Timed Access Register to prepare for unpredicted situations. Initialize the WDCON register, and set EWT bit from the WDCON register to enable reset when watchdog timeout occurs. Reset of W7100 can be prevented if the user sets the RWT bit periodically before watchdog timeout occurs. By applying this, the user can reset W7100 when it does not work properly.

## 6 Running Example

This section will explain how to download and run the Timer/Counter example of iMCU7100EVB Application Note. All example codes are based on KEIL compiler and C code. There are two ways of downloading the HEX file, which is created by compiling KEIL project, into iMCU7100EVB. One is to use the WizISP Program, and the other is to use W7100 Debugger. Please refer to 'iMCU7100EVB User's Guide,' 'WizISP Program User Guide,' and 'W7100 Debugger Guide' for more details.

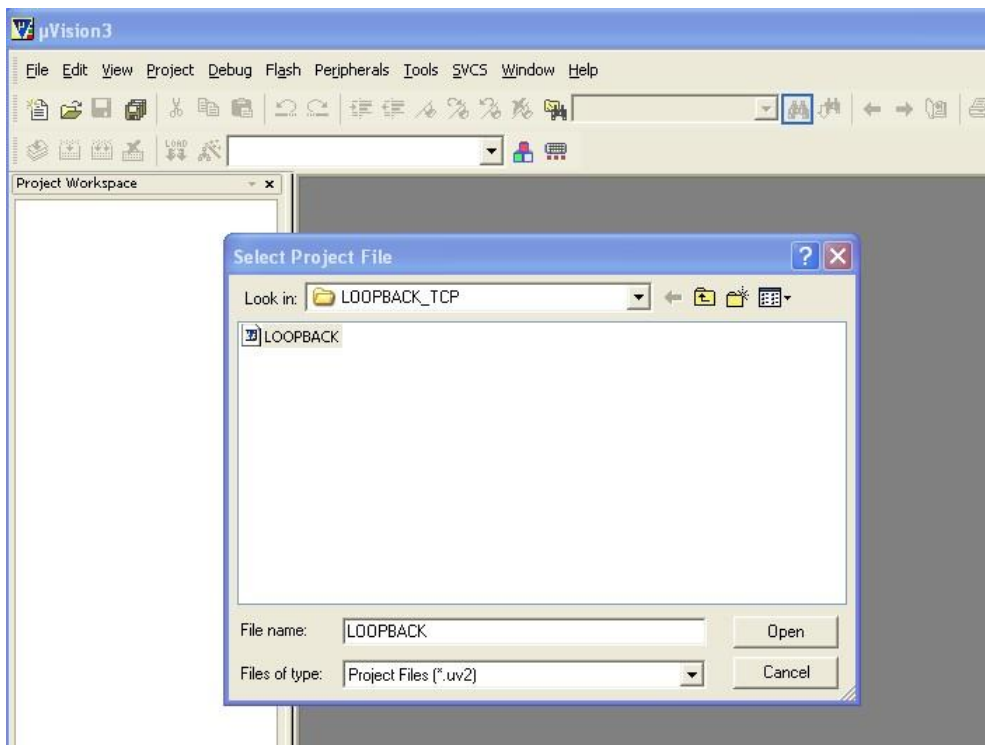
In order to run the example codes of 'iMCU7100EVB Application Note Timer/Counter' in the iMCU7100EVB board, the user must follow the order below.

1. Create KEIL project and Write Timer/Counter example.
2. Compile with KEIL compiler and create HEX file.
3. Download the created HEX file to iMCU7100EVB board by using ISP program or Debugger.
4. Reset the board, run the Timer/Counter example, and check whether LED0 (P0\_3) blinks.

Next section is about the results from each step.

### 6.1 Make a KEIL project

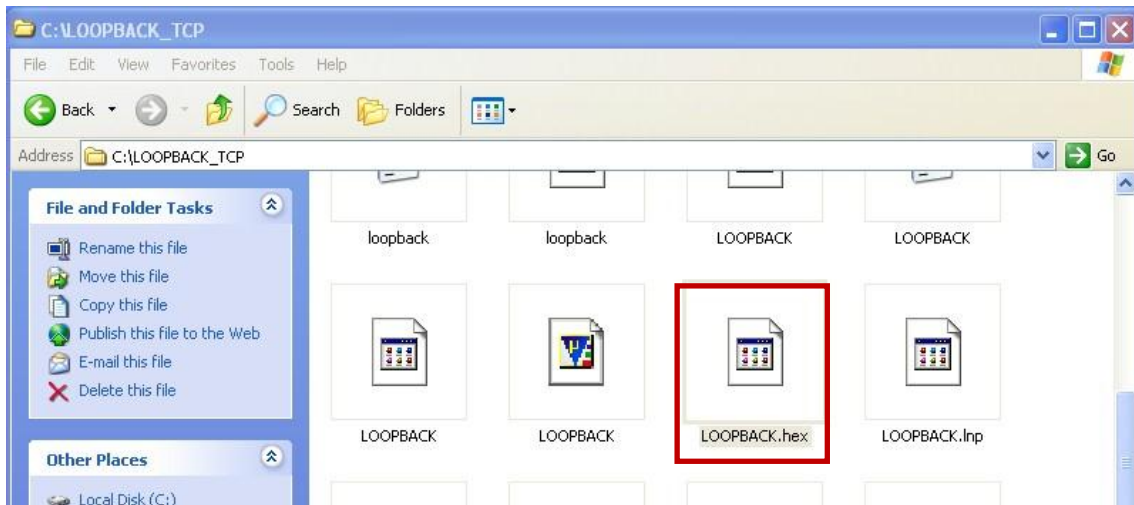
Users can create their own KEIL project, but also can open the attached KEIL project.



<Fig. 6.1> Open the KEIL project of Timer/Counter

### 6.2 Make a HEX file with compile

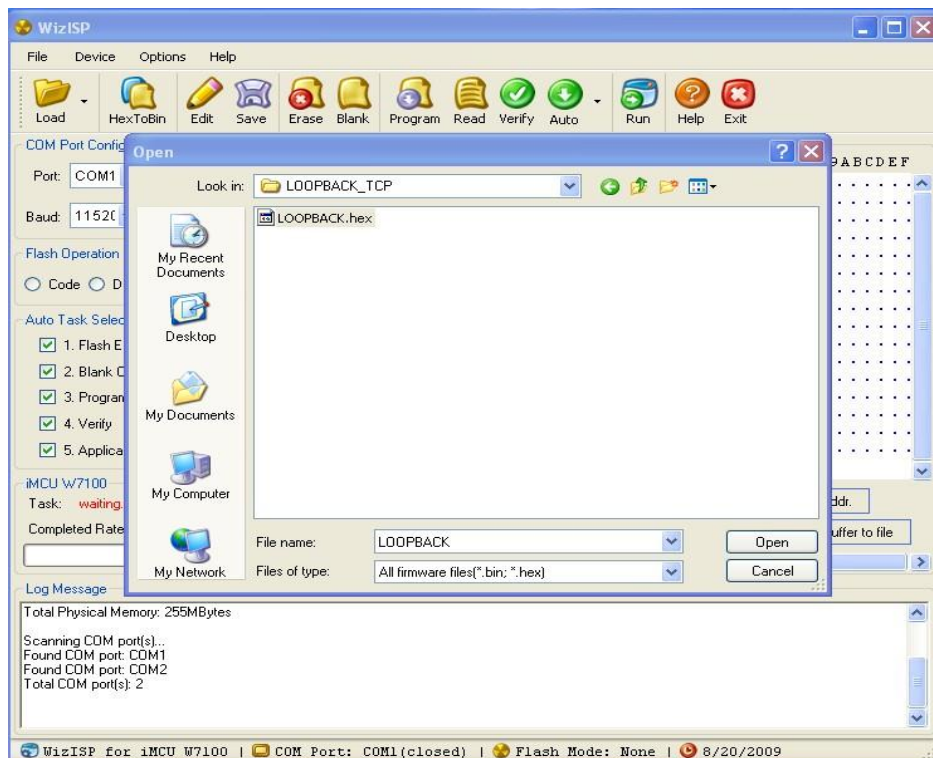
Write the code, compile, and create HEX file.



<Fig.6.2> Make HEX file using KEIL compiler

## 6.3 Download the HEX file to iMCU7100EVb

Download HEX file to iMCU7100EVb board by using ISP program or Debugger. The figure below is the ISP program. Since the ISP program has to load the BIN file, there is a function that changes a HEX file to BIN file. Please refer to WIZISP Program Guide for more details.



<Fig.6.3> Download the HEX file to the iMCU7100EVb

## 6.4 Run the Timer/Counter

Once the HEX file is downloaded into iMCU7100EVb board, reset the board and run the Timer/Counter. If there is no problem with the code, LED1 of the board will blink. If the example codes were used, LED1

will appear to be turned on because of the too fast blinking speed.



<Fig.6.4> Result of the Timer/Counter example code

## Document History Information

Version	Date	Descriptions
Ver. 0.9beta	2009	Release with W7100 launching
Ver. 0.91	2010	Modify the watchdog timer register description and code
Ver. 0.92	2010	Modify the timer1 and watchdog timer code
Ver. 1.0	Mar, 2011	Modify for W7100A QFN 64pin package

## Copyright Notice

Copyright 2011 WIZnet Co.,Ltd. All Rights Reserved.

Technical Support: [support@wiznet.co.kr](mailto:support@wiznet.co.kr)

Sales & Distribution: [sales@wiznet.co.kr](mailto:sales@wiznet.co.kr)

For more information, visit our website at <http://www.wiznet.co.kr>