# AN205058

# F²MC-8FX Family MB95310/370 Series Event Counter API

**Associated Part Family: MB95310/370 Series**

This Application Note introduces API for event counter of MB95F310 series MCU.

## Contents


1 Introduction....1
1.1 Background....1
2 Description of Event Counter Theory....2
3 MB95F310 Event Counter Register....3
4 Event Counter Library Function List....6
5 Event Counter Function Detail....6
5.1 ECounter_Init Function....6
5.2 __interrupt void INTER_ReloadTimer(void) Function....6
5.3 __interrupt void INTER_CompositeTimer(void) Function....7
6 Usage Demo....7
6.1 Hardware Design....7
6.2 Steps for Software Design....8
6.3 Steps for Adding Event Counter files....11
6.4 Range of External Frequency....12
7 Debug....13
8 Additional Information....13
Document History....14
Worldwide Sales and Design Support....15
Products....15
PSoC® Solutions....15
Cypress Developer Community....15
Technical Support....15


# 1 Introduction

This document introduces API for event counter of MB95F310 series MCU.

The event counter is mainly used to measure the frequency of external clock with configurable measure period. 16-bit reload timer and 8/16-bit composite timer ch1 are configured to provide an event counter operation mode in the event counter. In following chapter, the event counter theory and how to capture external clock will be described.

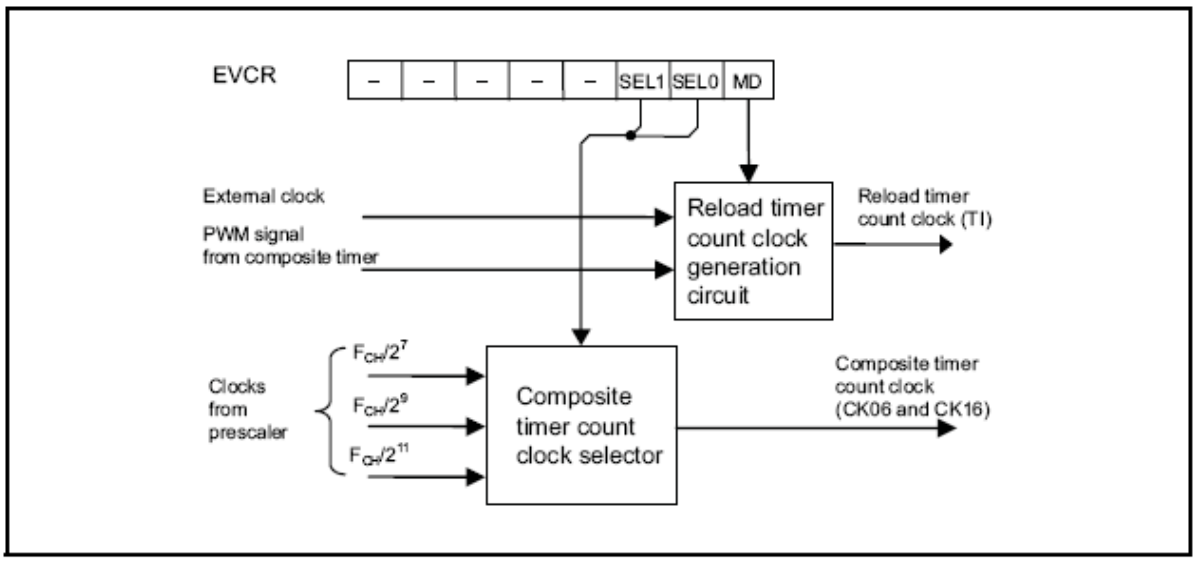
## 1.1 Background

This chapter introduces the background of event counter.

In event counter mode, 8/16-bit composite timer ch1 is used to generate a PWM signal. Then external clock will be gated by this PWM signal, and then input to the 16-bit reload timer as count clock. 16-bit reload timer operates in external clock mode (reload mode). The frequency of external clock could be calculated with configured measure period in the interrupt service subroutine of 8/16-bit composite timer ch1.

Following is event counter block diagram.

Figure 1-1. Event Counter Block Diagram




## 2 Description of Event Counter Theory

This chapter describes theory of event counter.

When external clock input by TI0, the composite timer generates PWM. And 16-bit reload timer begins to down count when PWM is turn to ‘H’ from ‘L’. When PWM ‘H’ level finished, an interrupt will be generated by system, and user can calculate external clock by 16-bit reload counter and PWM.

Figure 2-1 describes the operation of event counter.

Figure 2-1. Event Counter Theory

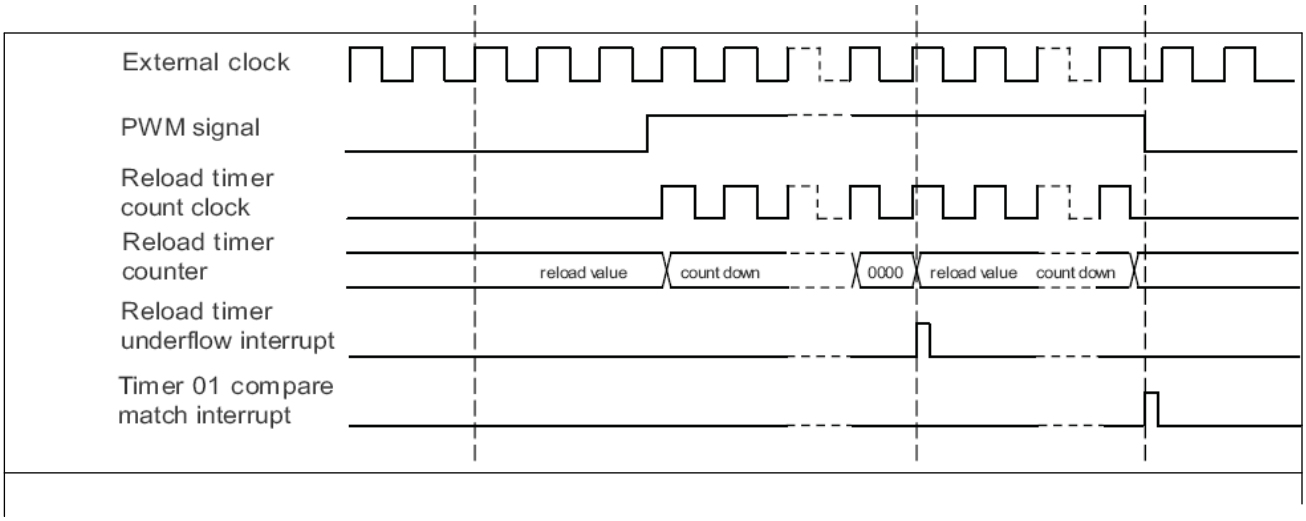


Figure 2-2 describes the arithmetic of external clock.

Figure 2-2. Arithmetic of External Clock

$$\text{Frequency of external clock} = \frac{\text{Count value of reload timer}}{\text{H Pulse width of PWM signal}}$$

In the above expression:

(count value of reload timer) = (TMRLRH0/TMRLRL0 set value) * (underflow times)
+ (TMRLRH0/TMRLRL0 set value) - (read value of TMRH0/TMRL0)

$$\text{(H pulse width of PWM signal)} = \frac{\text{(T11DR set value - T10DR set value)}}{\text{(frequency of composite timer count clock source)}}$$

# 3 MB95F310 Event Counter Register

This chapter describes MB95F310 event counter register.

In MB95F310 series MCU, event counter function is realized by 8/16-BIT composite timer and 16-bit reload timer. Following table describes related registers in composite timer, 16-bit reload timer and event counter.

Table 3-1. Event Counter Registers List

| Register | | Description |
|---|---|---|
| 8/16-bit composite timer | T10CR0 | Low 8bits of status control register0 |
| | T11CR0 | High 8bits of status control register0 |
| | T10CR1 | Low 8bits of status control register1 |
| | T11CR1 | High 8bits of status control register1 |
| | T10DR | PWM “L” define register |
| | T11DR | PWM cycle register |
| | TMCR1 | Timer mode control register |
| 16-bit reload timer | TMR0 | 16-bit reload data register |
| | TMCSRH0 | 16-bit control status high 8 bits register |
| | TMCSRL0 | 16-bit control status low 8 bits register |
| Event counter | EVCR | Event counter enable and clock register |
| I/O port | LCDCE1 | I/O port setting register |

For detailed register please refer to MB95F310 hardware chapter 18 19 and 20.

Figure 3-1 describes 8/16-bit composite timer registers work condition in detail.

Figure 3-1. 8/16-bit Composite Timer Work Condition

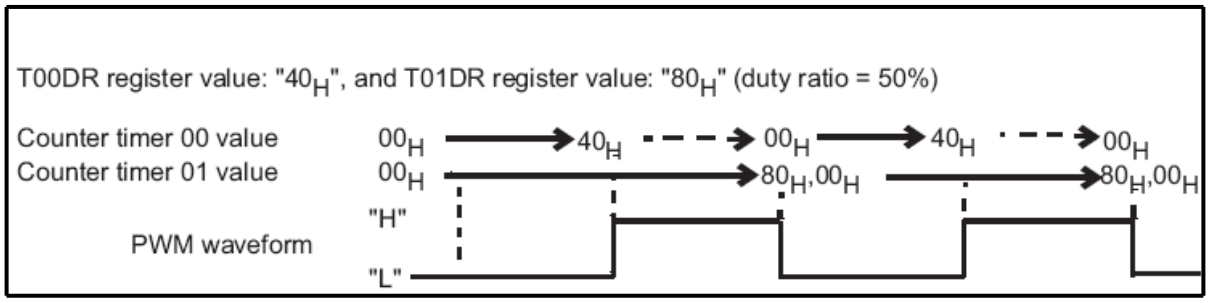
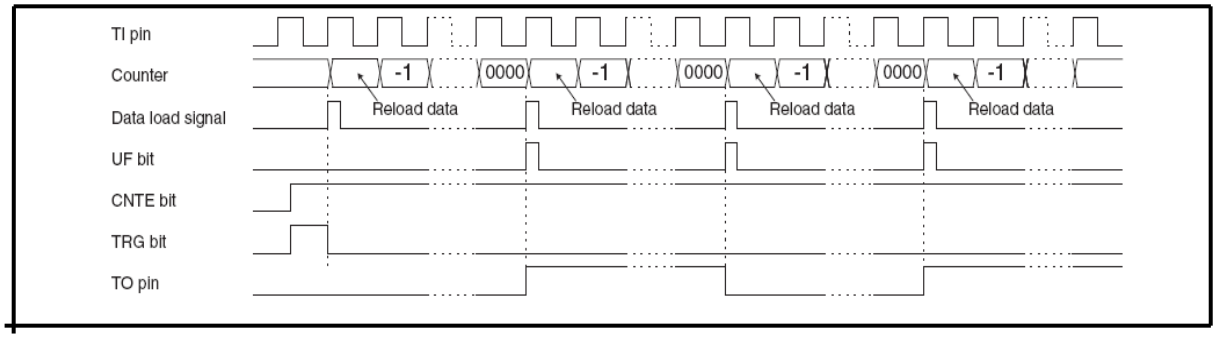


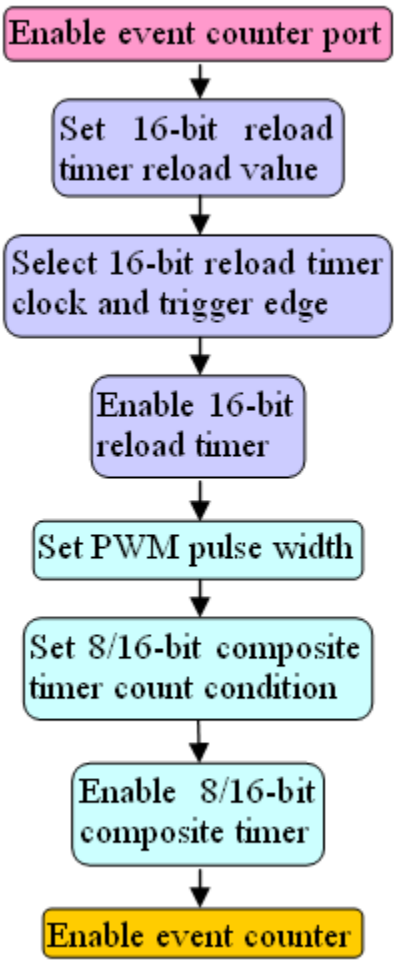

Figure 3-2 describes 16-bit reload timer registers work condition in detail

Figure 3-2. 16-bit Reload Timer Work Condition



Following picture describes the general setting procedure.

Figure 3-3. General Setting Procedure



# 4 Event Counter Library Function List

This chapter introduces all functions in event counter *sample code EventCounter.prj*.

Table 4-1 lists the event counter functions.

Table 4-1. Event Counter Functions

| Function name | Description |
|---|---|
| void ECounter_Init(void) | Initializes event counter registers |
| __interrupt void INTER_ReloadTimer(void) | Interrupt to record timer underflow times |
| __interrupt void INTER_CompositeTimer(void) | Interrupt to calculate external clock |

# 5 Event Counter Function Detail

This chapter introduces the detail of event counter functions.

## 5.1 ECounter_Init Function

Table 5-1 describes ECounter_Init function.

Table 5-1. ECounter_Init Function

| Function name | initial_I2C |
|---|---|
| Function prototype | Void ECounter_Init (void) |
| Behavior description | Initializes event counter timers |
| Input parameter | None |
| Return value | None |
| Example | The library function sets reload timer value to 0x200, PWM “L” to 0xa0 an PWM cycle to 0xe0:<br>ECounter_Init (); |

## 5.2 __interrupt void INTER_ReloadTimer(void) Function

Table 5-2 describes __interrupt void INTER_ReloadTimer(void).

Table 5-2. Describes Reload Timer Interrupt Function

| | |
|---|---|
| Function prototype | __interrupt void INTER_ReloadTimer(void) |
| Behavior description | When reload timer underflow the interrupt will record the underflow times |
| Input parameter | None |
| Return value | None |
| Interrupt level | IRQ11: 16-bit reload timer ch0 |
| Example | __interrupt void INTER_ReloadTimer(); |

### 5.3 __interrupt void INTER_CompositeTimer(void) Function

Table 5-3 describes __interrupt void INTER_CompositeTimer(void) function.

Table 5-3. Describes Composite Timer Interrupt Function

| Function name | AD_Read |
|---|---|
| Function prototype | __interrupt void INTER_CompositeTimer(void) |
| Behavior description | When interrupt occur calculate the external clock |
| Input parameter | None |
| Return value | None |
| Interrupt level | IRQ14: 8/16-bit timer ch1 (upper) |
| Example | __interrupt void INTER_ CompositeTimer (); |

**Note:** Global parameter "`Ext_FreqVat`" records the external clock frequency, user can read it in function *INTER_CompositeTimer*, for details please refer to Chapter 7.

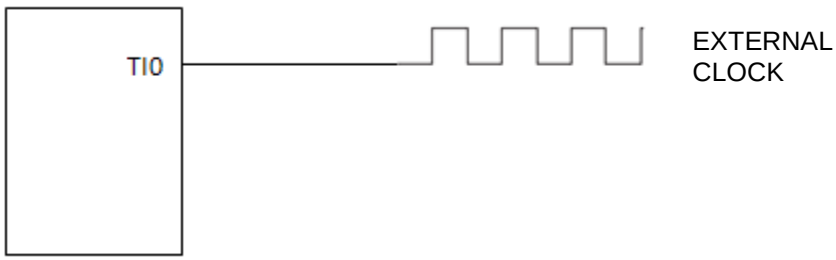
## 6 Usage Demo

This chapter describes something we must pay attention to when we use.

### 6.1 Hardware Design

Refer to the following figure for hardware design.

Figure 6-1. Event Counter Hardware Design



## 6.2 Steps for Software Design

- For event counter software design, initialization is the first step, which opens the interrupt, sets the interval time and PWM width.

  Following setting is used for initial function.

Figure 6-2. Initialization Design

```
      LCDCE1_PICTL = 1;
      PUL5_P52 = 0;                  //pull up disabled
      DDR5_P52 = 0;
      DDR5_P53 = 1;

      TMR0 = 0x200;                  //reload time to 0x200
      TMCSRH0 = 0x39;                //rising edge, event count
      TMCSRL0 = 0x58;                //enable reload timer,
                                     //enable underflow interrupt,
                                     //clear interruption flag,
                                     //enable counter, count after
                                     reload
      TMCSRL0 |= 0x03;
      T10DR = 0xa0;
      T11DR = 0xe0;                  //PWM IS 64
      T11CR0 = 0xe4;                 //enable interrupt
      T10CR0 = 0xe4;
      T11CR1 = 0x23;
      T10CR1 = 0x03;;                //enable "L" counter
      TMCR1 = 0x80;                  //output value of timer11
      T10CR1_STA = 1;
      T11CR1_STA = 1;

      EVCR = 0x01;
```

Following picture describes how to use the initial function

```
void main(void)
{
      InitIrqLevels(); // initialise Interrupt level register and
IRQ vector table
      __EI ();
      ECounter_Init();
      While (1)
      { }
}
```

- When 16-bit reload timer underflows, record the underflow times.

For 16-bit reload timer interrupt, please refer to following setting

```
      TMCSRL0_UF = 0;                 //clear flag
      TMCSRL0_INTE = 0;               //disable underflow interrupt
      TimeUnderFlow++;
      TMCSRL0_INTE = 1;               //enable underflow interrupt
```

- When 8/16-bit composite timer interrupt occurs, calculate the external clock by 16-bit reload timer and PWM “H” width.

  For the 8/16-bit composite timer setting, please refer to following picture.

```
unsigned int USVa;
T11CR1_IF = 0;                          //clear interrupt flag
T10CR1_IF = 0;
T11CR1_IE = 0;                          //disable interrupt
T10CR1_IE = 0;
TMCSRL0 &= 0xfc;
USVa = TMR0;
Ext_FreqVat = RecordUse * TimeUnderFlow;
Ext_FreqVat = Ext_FreqVat + RecordUse - USVa;
if(Ext_FreqVat< 600)
{
      Ext_FreqVat = Ext_FreqVat * 100;
      Ext_FreqVat = Ext_FreqVat/128;
      Ext_FreqVat = Ext_FreqVat*60;
      Ext_FreqVat = Ext_FreqVat/64;
      //Ext_FreqVat = Ext_FreqVat*6;
}
else
{
      Ext_FreqVat = Ext_FreqVat*30;
      Ext_FreqVat = Ext_FreqVat/128;
      Ext_FreqVat = Ext_FreqVat*100;
      Ext_FreqVat = Ext_FreqVat/64;
      Ext_FreqVat = Ext_FreqVat * 2;
}
TimeUnderFlow = 0;

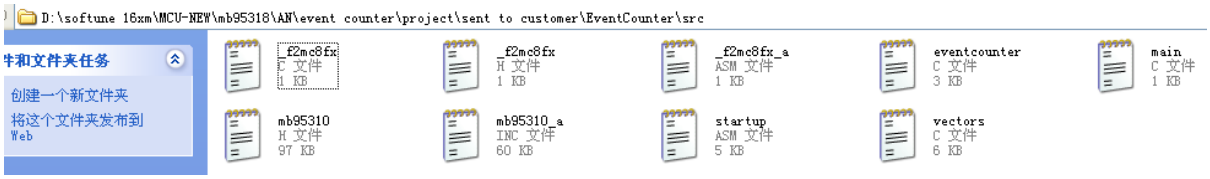
TMCSRL0 |= 0x03;
T11CR1_IE = 1;
T10CR1_IE = 1;
while(1)
{
      //read frequency; unit is KHz
}
```

## 6.3 Steps for Adding Event Counter files

When use this project please refer to following steps

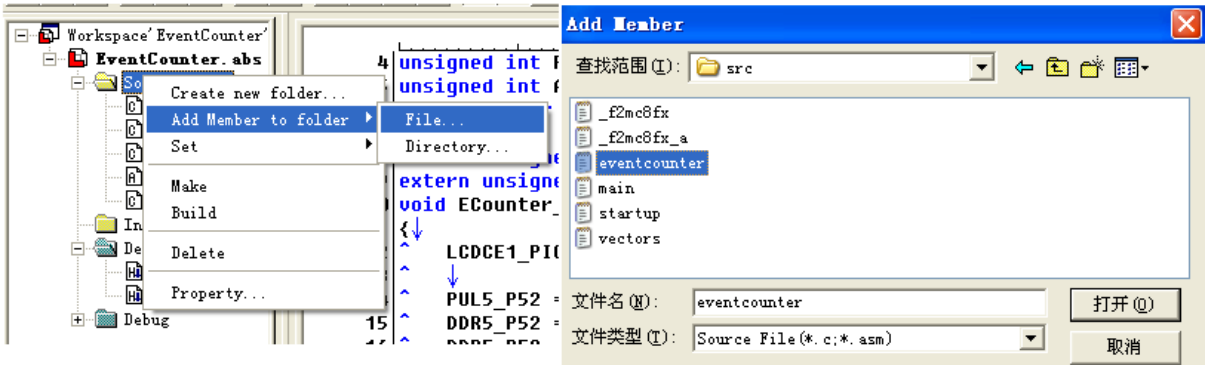
- First step is to add files to document. Figure 6-3 describes this step.

Figure 6-3. Files Use First Step




- Second step is to add function to project. Figure 6-4 describes this step.

Figure 6-4. File Use Second Step




- Third step is to add initial function to *main.c*. Figure 6-5 describes this step.

Figure 6-5. Library Use Third Step

```
11
12 void main(void)
13 {
14     InitIrqLevels();
15
16     __EI();
17     ECounter_Init();
18
19     while(1)
20     {
21
22     }
23 }
24 [EOF]
```

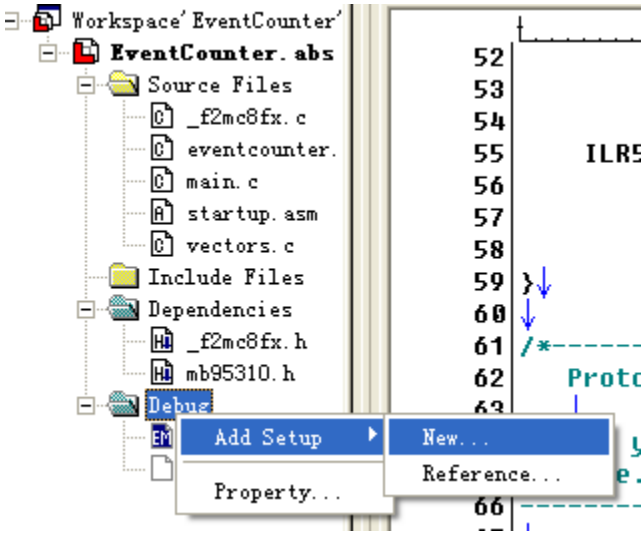
- Fourth step is to add interrupt function to *vector.c* and set interrupt level. Figure 6-6 describes this step.

Figure 6-6. Library Use Fourth Step

```
------------------------------------------------------
__interrupt void DefaultIRQHandler (void);
__interrupt void INTER_ReloadTimer (void);
__interrupt void INTER_CompositeTimer (void);
/*------------------------------------------------------
```

- The fifth step is debugging, set an environment before debugging. Figure 6-7 describes this step.

Figure 6-7. Library Use Fifth Step




## 6.4 Range of External Frequency

The maximum measurable frequency is limited by peripheral resource clock. When peripheral resource clock frequency is FPCLK, the maximum frequency measurable is FPCLK/4.The minimum measurable frequency is limited by the measure period, in order to ensure the frequency measurement precision.

The main oscillator clock is 6MHz, so the max external frequency is 1.5MHz.

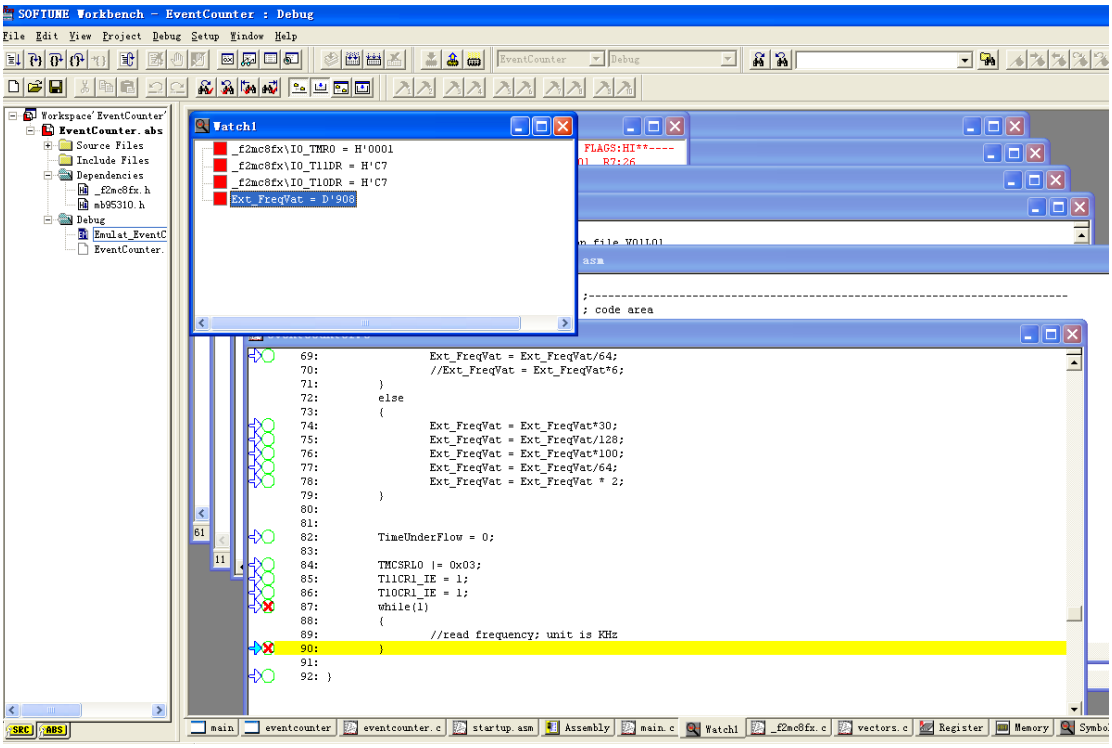
# 7 Debug

This chapter describes how to debug the sample code on EV-board and what will happen when the code is running.

This project is based on MCU MB95F310/370. When debugging, please refer to Figure 6-1 for the hardware connection.

Run project, the external clock value will be read by global parameter "`Ext_FreqVat`". Please refer to Figure 7-1 for details.

Figure 7-1. Debugging Description




# 8 Additional Information

Please contact your local support team for any technical question.

## Document History

Document Title: AN205058 - F²MC-8FX Family MB95310/370 Series Event Counter API

Document Number: 002-05058

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | — | HUAL | 04/07/2010 | Initial release |
| *A | 5260613 | HUAL | 06/23/2016 | Migrated Spansion Application Note “MCU-AN- 500082-E-10” to Cypress format |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| ARM® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Lighting & Power Control | cypress.com/powerpsoc |
| Memory | cypress.com/memory |
| PSoC | cypress.com/psoc |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless/RF | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

## Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corporation. All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com

© Cypress Semiconductor Corporation, 2010-2016. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.