The following document contains information on Cypress products. The document has the series name, product name, and ordering part numbering with the prefix “MB”. However, Cypress will offer these products to new and existing customers with the series name, product name, and ordering part number with the prefix “CY”.

**How to Check the Ordering Part Number**

1. Go to www.cypress.com/pcn.
2. Enter the keyword (for example, ordering part number) in the **SEARCH PCNS** field and click **Apply**.
3. Click the corresponding title from the search results.
4. Download the Affected Parts List file, which has details of all changes

**For More Information**

Please contact your local sales office for additional information about Cypress products and solutions.

**About Cypress**

Cypress is the leader in advanced embedded system solutions for the world's most innovative automotive, industrial, smart home appliances, consumer electronics and medical products. Cypress' microcontrollers, analog ICs, wireless and USB-based connectivity solutions and reliable, high-performance memories help engineers design differentiated products and get them to market first. Cypress is committed to providing customers with the best support and development resources on the planet enabling them to disrupt markets by creating new product categories in record time. To learn more, go to www.cypress.com.

AN205275

# F²MC - 8FX Family, MB95200H/210H Series, 8/16-Bit Composite Timer

This application note describes how to use the 8/16-bit composite timer in various modes. The application note gives the features and some examples on the 8/16-bit composite timer.

## Contents

1 Introduction....1
2 The 8/16-Bit Composite Timer....2
2.1 Key Features....2
2.2 Block Diagram....2
2.3 Registers....3
2.4 Operating Modes....4
3 Operations and Examples....5
3.1 Operation of Interval Timer Function (One-shot Mode)....5
3.2 Operation of Interval Timer Function (Continuous Mode)....7
3.3 Operation Description of Interval Timer Function (Free-run Mode)....9
3.4 Operation of PWM Timer Function (Fixed-cycle Mode)....12
3.5 Operation of PWM Timer Function (Variable-cycle Mode)....14
3.6 Operation of PWC Timer Function....16
3.7 Operation of the Input Capture Function....18
4 Notes on Using 8/16-bit Composite Timer....20
5 Appendix....20
5.1 Sample Code....20
6 Additional Information....31
Document History....32

## 1 Introduction

This application note describes how to use the 8/16-bit composite timer in various modes.

The application note gives the features and some examples on the 8/16-bit composite timer.

# 2 The 8/16-Bit Composite Timer

This chapter introduces the basic functions of the 8/16-bit composite timer

## 2.1 Key Features

The 8/16-bit composite timer consists of two 8-bit counters, which can be used as two 8-bit timers, or as one 16-bit timer if they are connected in cascade.

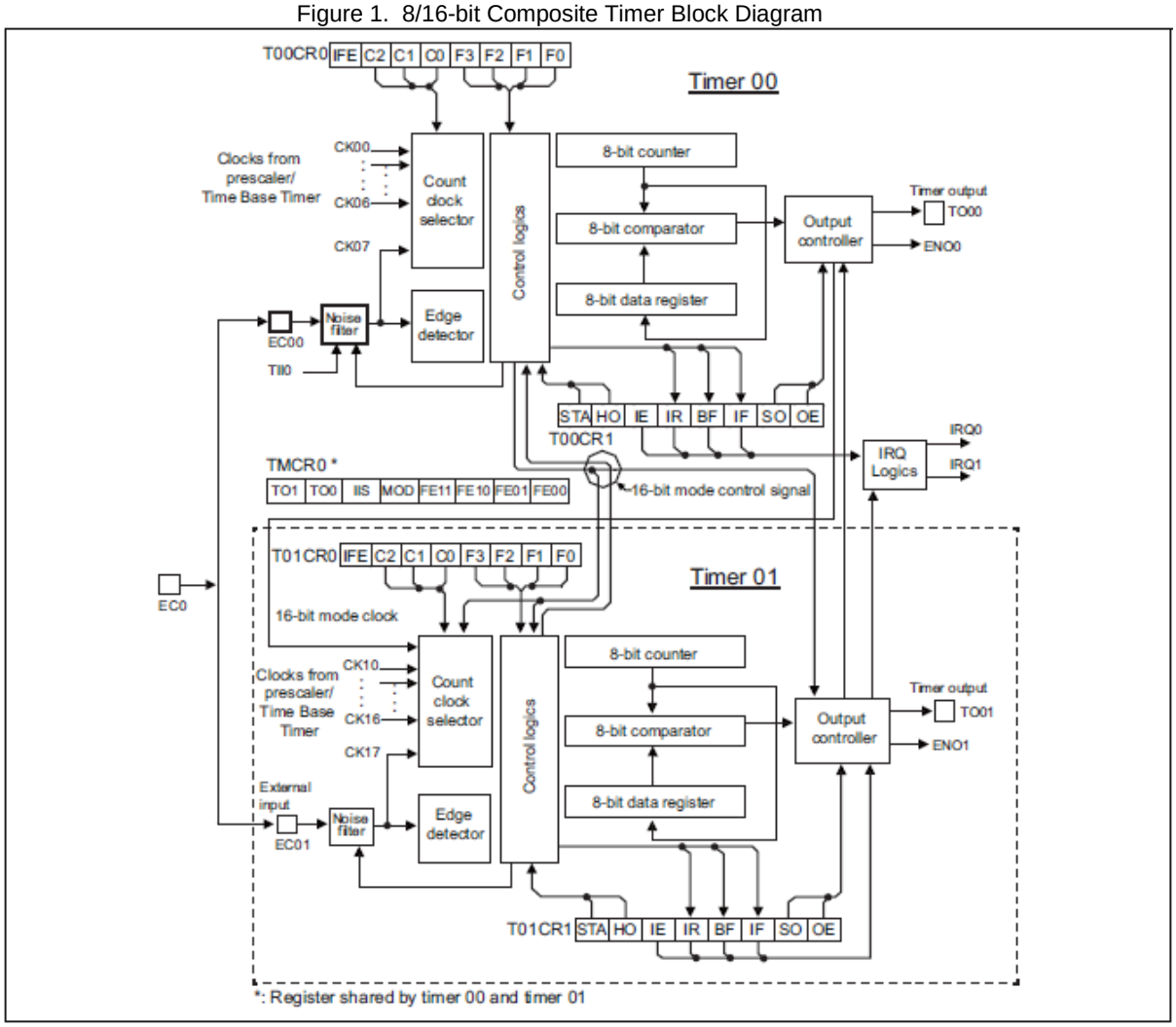
The 8/16-bit composite timer has the following functions:

- Interval timer function
- PWM timer function
- PWC timer function (pulse width measurement)
- Input capture function

## 2.2 Block Diagram

Figure 1 shows the internal block diagram of the 8/16-bit composite timer.

Figure 1. 8/16-bit Composite Timer Block Diagram



## 2.3 Registers

Please refer to Chapter 14 of the MB95200H/210H Series Hardware Manual for detailed register setting.

### 2.3.1 8/16-bit Composite Timer 00/01 Control Status Register 0 (T00CR0/T01CR0)

This register is used to select the timer operating mode, select the count clock, and enable or disable IF flag interrupts. Register T00CR0 corresponds to timer 00 and register T01CR0 timer 01.

Figure 2. T00CR0/T01CR0

| Address | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| 0F92H T01CR0 | IFE | C2 | C1 | C0 | F3 | F2 | F1 | F0 | $00000000_B$ |
| 0F93H T00CR0 | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

### 2.3.2 8/16-bit Composite Timer 00/01 Control Status Register 1 (T00CR1/T01CR1)

This register is used to control the interrupt flag, timer output and timer operations. Register T00CR1 corresponds to timer 00 and T01CR1 timer 01.

Figure 3. T00CR1/T01CR1

| Address | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| 0036H T01CR1 | STA | HO | IE | IR | BF | IF | SO | OE | $00000000_B$ |
| 0037H T00CR1 | R/W | R/W | R/W | R(RM1),W | R/WX | R(RM1),W | R/W | R/W | |

### 2.3.3 8/16-bit Composite Timer 00/01 Timer Mode Control Register (TMCR0)

This register is used to select noise filter function, 8-bit or 16-bit operating mode, signal input to timer 00 and the timer output value. This register serves both timer 00 and timer 01.

Figure 4. TMCR0

| Address | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| 0F96H TMCR0 | TO1 | TO0 | IIS | MOD | FE11 | FE10 | FE01 | FE00 | $00000000_B$ |
| | R/WX | R/WX | R/W | R/W | R/W | R/W | R/W | R/W | |

### 2.3.4 8/16-bit Composite Timer 00/01 Data Register ch.0 (T00DR/T01DR)

This register is used to write the maximum value counted during an interval timer or PWM timer operation and to read the count value during a PWC timer or an input capture operation. Register T00DR corresponds to timer 00 and T01DR timer 01.

Figure 5. T00DR/T01DR

| Address | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | Initial value |
|---|---|---|---|---|---|---|---|---|---|
| 0F94H T01DR | TDR7 | TDR6 | TDR5 | TDR4 | TDR3 | TDR2 | TDR1 | TDR0 | $00000000_B$ |
| 0F95H T00DR | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

R/W: Readable/writeable (Read value is the same as write value)

R/WX: Read only (Readable, writing has no effect on operation)

R (RM1), W: Readable/writable (Read value is different from write value, "1" is read by read-modify-write instruction)

## 2.4 Operating Modes

The 8/16-bit Composite timer functions in seven operating modes are shown as below:

- Interval Timer Function (One-shot Mode)
- Interval Timer Function (Continuous Mode)
- Interval Timer Function (Free-run Mode)
- PWM Timer Function (Fixed-cycle Mode)
- PWM Timer Function (Variable-cycle Mode)
- PWC Timer Function
- Input Capture Function

# 3 Operations and Examples

This chapter provides examples for 8/16-bit composite timer

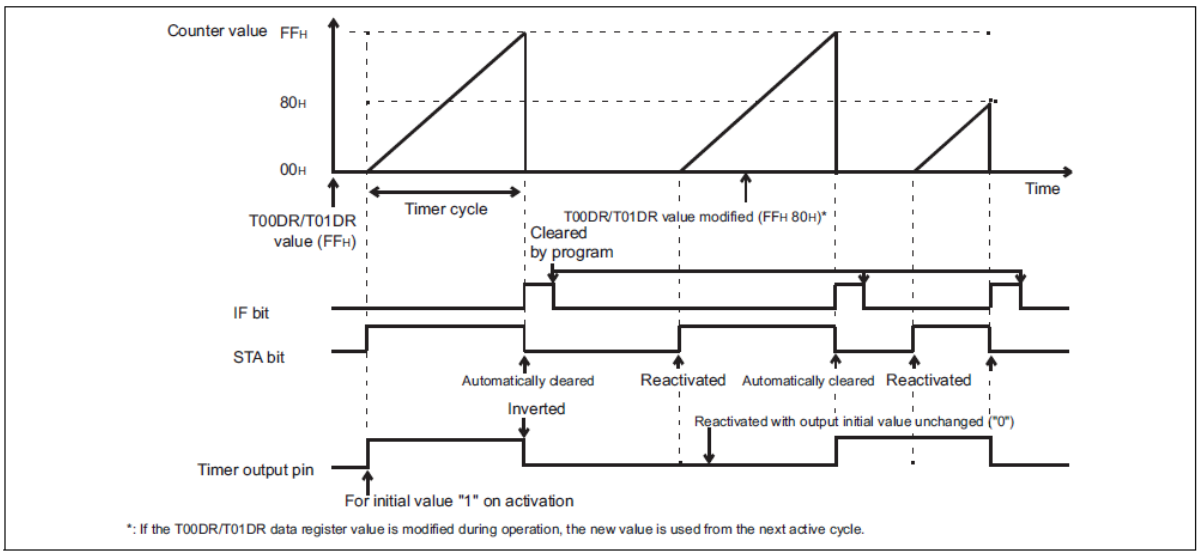
This section describes various modes and provides some examples for the 8/16-bit composite timer.

## 3.1 Operation of Interval Timer Function (One-shot Mode)

When the interval timer function (one-shot mode) is selected, the counter starts counting from "00H" as the timer is activated. When the counter value matches the register setting value, the timer output is inverted. An interrupt request is made, and the counter stops counting.

Figure 6 shows the operation of the interval timer function in 8-bit mode.

Figure 6. Operation of Interval Timer Function in the 8-bit Mode (Timer 0)




Setting Procedure Example

The procedure below shows how to set the 8/16-bit composite timer operating in an interval timer function (one-shot mode, 8 bit).

1. Set the mode (T00CR0: [F3..F0] = 0 0 0 0 )
2. Set the counter clock (T00CR0: [C2..C0] )
3. Set the counter value (T00DR)
4. Set the interrupt level if necessary (ILR1)
5. Enable the timer output (T00CR1: OE = 1)
6. Start the timer (T00CR1: STA = 1)

If the overflow interrupt is enabled, the flag in ISR shall be cleared. (T00CR1: IF = 0)

The following example shows how to set the 8/16-bit composite timer 0 for operation in one-shot mode.

Refer to *Appendix Sample Code* for project “One-shot”

```
/*                                SAMPLE CODE                                */
/*---------------------------------------------------------------------------*/

/* initial timer for interval timer (One-shot) function */
void InitCompTimer (void)
{
       T00DR = 0xFF;       // set the count value
       TMCR0 = 0x00;       // 8-bit, no filtering
       T00CR0 = 0x00;      // interval timer in one-shot mode
       T00CR1 = 0x01;      // disable interrupt, enable output
}

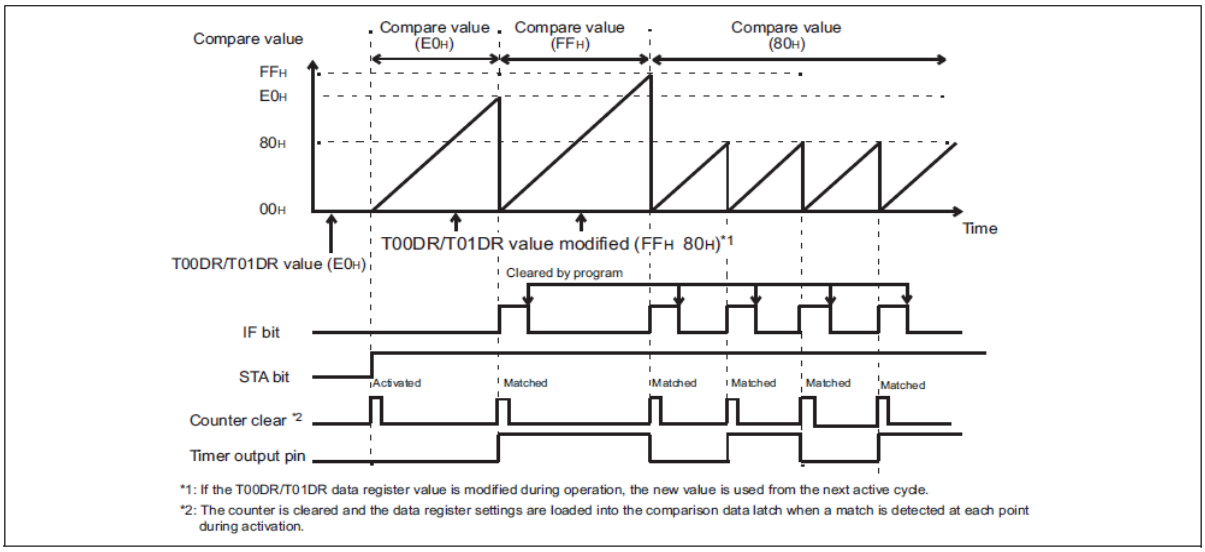

/* main routine */
void main (void)
{
       InitComTimer();
       ...
       T00CR1_STA = 1;     // start the timer
}
```

## 3.2 Operation of Interval Timer Function (Continuous Mode)

When the interval timer function (continuous mode) is selected, the counter starts counting from "00H" as the timer is activated. When the counter value matches the register setting value, the timer output is inverted. An interrupt request is made and the counter returns to "00H" and restarts counting. The timer outputs a square wave as a result of this repeated operation.

Figure 7 shows the operation of the interval timer function in 8-bit mode.

Figure 7. Operating Diagram of Interval Timer Function (Continuous Mode)




Setting Procedure Example

The procedure below shows how to set 8/16-bit composite timer working in interval timer function (Continuous mode).

1. Set the mode (T00CR0: [F3..F0] = 0 0 0 1 )
2. Set the counter clock (T00CR0 : [C2..C0] )
3. Set counter value (T00DR)
4. Set the interrupt level if necessary (ILR1)
5. Enable timer output (T00CR1: OE = 1)
6. Start timer (T00CR1: STA = 1)

If the overflow interrupt is enabled, the flag in ISR shall be cleared. (T00CR1: IF = 0)

The following example shows how to set the 8/16-bit composite timer 0 for operation with interrupt (Continuous mode, 16 bit).

```
/*                            SAMPLE CODE                                  */
/*-------------------------------------------------------------------------*/
/* initial timer for interval timer (continuous) function */
void InitCompTimer (void)
{
       T01DR = 0x01;       // set the count value (upper 8 bits)
       T00DR = 0xFF;       // set the count value (lower 8 bits)
       TMCR0 = 0x10;       // 16-bit, no filtering
       T00CR0 = 0x81;      // interval timer in the continuous mode
                           // enable IF flag interrupt
       T00CR1 = 0xA1;      // enable the interrupt, enable the output
                           // start the timer
}
/* enter ISR while the counter value is matches the pre-set value */
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;      // disable the interrupt
       T00CR1_IF = 0;      // clear the flag

       ...                 // interrupt service routine

       T00CR1_IE = 1;      // enable the interrupt
}

/* main routine */
void main (void)
{
       ...
       InitCompTimer();
       ...
       while (1);
}
```

Please note that the corresponding interrupt vector and level shall be defined in the vector.c module of Fujitsu's standard template project.

Refer to *Appendix Sample Code* for project "Continuous".

```
/*                            SAMPLE CODE                                  */
/*-------------------------------------------------------------------------*/

/* interrupt level setting */
void InitIrqLevels (void)
{
       ...

       ILR1 = 0xF3;        // IRQ5: 8/16-bit timer ch.0 (lower)
       ...
}

__interrupt void CompTimer (void);      // Prototype

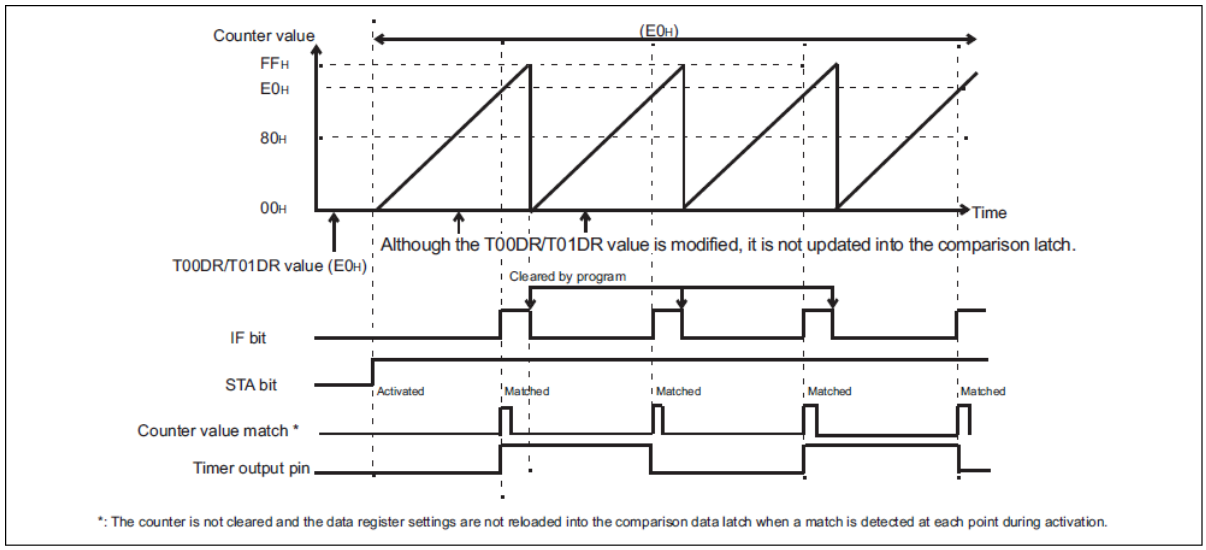
       ...

#pragma intvect CompTimer  5      // 8/16-bit timer ch.0 (lower)
```

## 3.3 Operation Description of Interval Timer Function (Free-run Mode)

When the interval timer function (free-run mode) is selected, the counter starts counting from "00H". When the counter value matches the register setting value, the timer output is inverted and an interrupt request is made. If the counter continues to count and reaches "FFH", it returns to "00H" and restarts counting. The timer outputs square wave as a result of this repeated operation.

Figure 8. Operating Diagram of Interval Timer Function (Free-run Mode)




Setting Procedure Example

The procedure below shows how to set 8/16-bit composite timer working in interval timer function (Free run mode).

1. Set the mode (T00CR0: [F3..F0] = 0 0 1 0 )
2. Set the counter clock (T00CR0: [C2..C0] )
3. Set the counter value (T00DR)
4. Set the interrupt level if necessary (ILR1)
5. Enable the timer output (T00CR1: OE = 1)
6. Start the timer (T00CR1: STA = 1)

If the overflow interrupt is enabled, the flag in ISR shall be cleared. (T00CR1: IF = 0)

The following example shows how to set the 8/16-bit composite timer 0 for operation in free-run mode.

```
/*                              SAMPLE CODE                                  */
/*---------------------------------------------------------------------------*/

/* initial timer for interval timer (continuous) function */
void InitCompTimer (void)
{
       T01DR = 0x01;       // set the count value (upper 8 bits)
       T00DR = 0xFF;       // set the count value (lower 8 bits)
       TMCR0 = 0x10;       // 16-bit, no filtering
       T00CR0 = 0x82;      // interval timer in free run mode
                           // enable IF flag interrupt
       T00CR1 = 0xA1;      // enable the interrupt, enable output
                           // start the timer
}

/* enter ISR while counter value matches the pre-set value */
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;      // disable the interrupt
       T00CR1_IF = 0;      // clear the flag

       ...                 // interrupt service routine

       T00CR1_IE = 1;      // enable the interrupt
}

/* main routine */
void main (void)
{
       ...
       InitCompTimer();

       ...
       while (1);
}
```

Please note that the corresponding interrupt vector and level shall be defined in the vector.c module of Cypress standard template project.

```
/*                          SAMPLE CODE                                      */
/*-----------------------------------------------------------------------*/

/* interrupt level setting */
void InitIrqLevels (void)
{
       ...

       ILR1 = 0xF3;        // IRQ5: 8/16-bit timer ch.0 (lower)
       ...
}

__interrupt void CompTimer (void);      // Prototype

       ...

#pragma intvect CompTimer  5      // 8/16-bit timer ch.0 (lower)

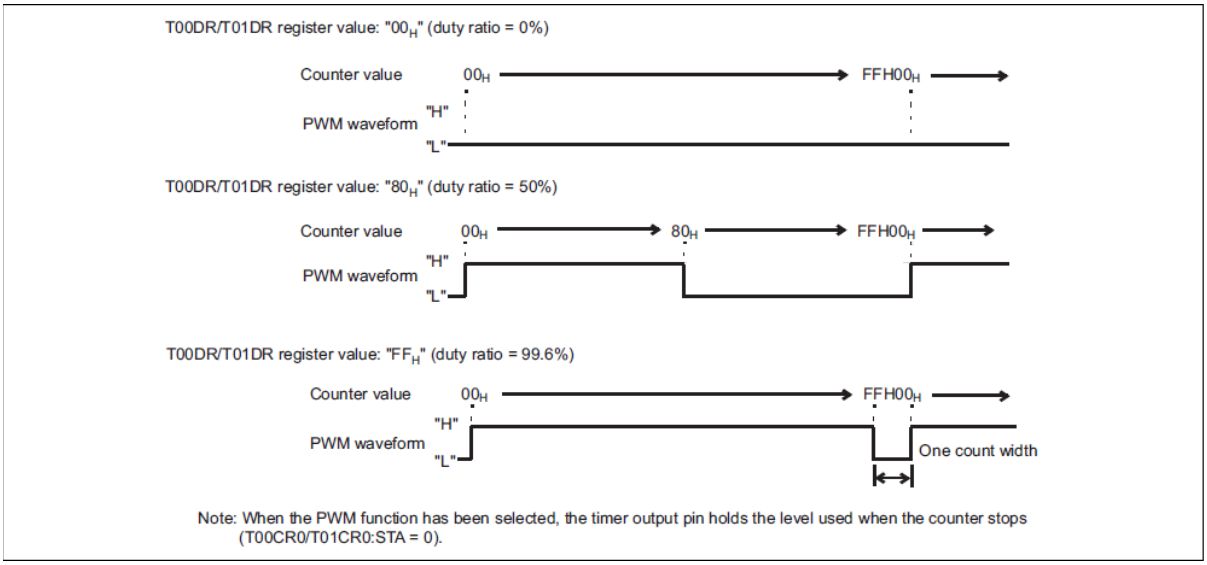
       ...
```

Refer to *Appendix Sample Code* for project “Free-run”.

## 3.4 Operation of PWM Timer Function (Fixed-cycle Mode)

When the PWM timer function (fixed-cycle mode) is selected, a PWM signal with a variable "H" pulse width is generated in fixed cycle. The cycle is fixed to "FFH" in 8-bit operation or "FFFFH" in 16-bit operation. The time is determined by the count clock selected. The “H” pulse width is specified by setting a corresponding register

Figure 9. Operating Diagram of PWM Timer Function (Fixed-cycle Mode)




Setting Procedure Example

The procedure below shows how to set 8/16-bit composite timer working in the PWM timer function (Fixed-cycle Mode).

1. Set the mode (T00CR0: [F3..F0] = 0 0 1 1 )
2. Set the counter clock (T00CR0: [C2..C0])
3. Set the duty by setting T00DR
4. Start the timer (T00CR1: STA = 1)

The following example shows how to set 8/16-bit composite timer 0 for operation in the PWM timer function (fixed-cycle mode).

```
/*                              SAMPLE CODE                                */
/*-------------------------------------------------------------------------*/

/* initial timer for the PWM function (fixed cycle) */
void InitCompTimer (void)
{
       T00DR = 0x40;       // set the PWM timer duty
       TMCR0 = 0x00;       // 8-bit, no filtering

       T00CR0 = 0x03;      // PWM timer function (fixed cycle mode)
       T00CR1 = 0x01;      // enable the output
}
       ...

/* main routine */
void main (void)
{
       ...

       InitCompTimer ();

       ...

       T00CR1_STA = 1;     //start the timer

       While(1);
}
```

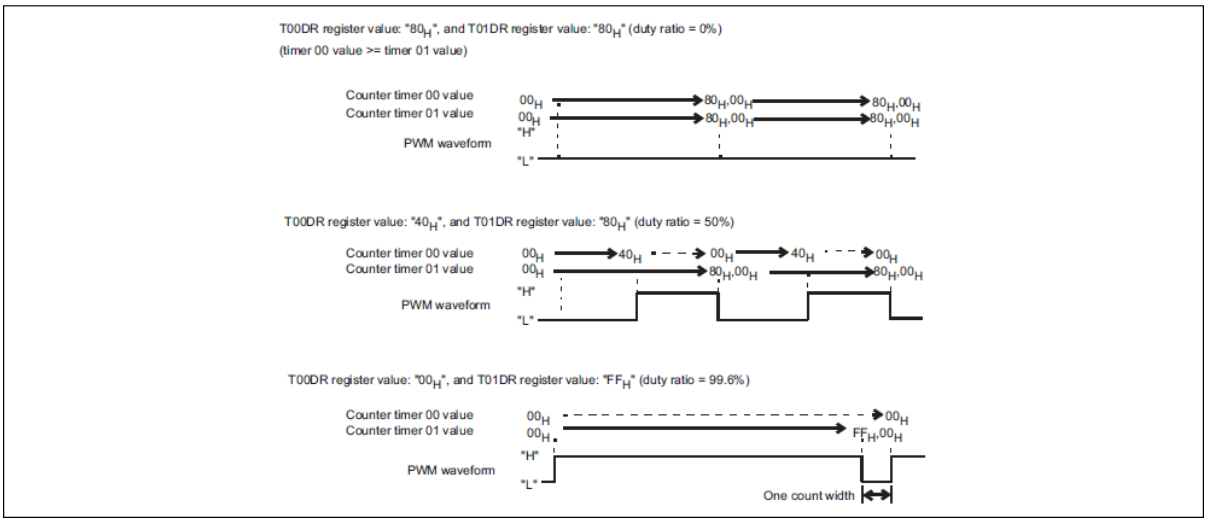
Refer to *Appendix Sample Code* for project “PWM_Fixed”.

## 3.5 Operation of PWM Timer Function (Variable-cycle Mode)

When the PWM timer function (variable-cycle mode) is selected, two 8-bit counters are used to generate an 8-bit PWM signal in any cycles and duty depending on the cycle and "L" pulse width specified by corresponding registers.

In this operating mode, the composite timer cannot form a 16-bit counter, as two 8-bit counters are used.

Figure 10.Operating Diagram of PWM Timer Function (Variable-cycle Mode)




Setup Procedure Example

The procedure below shows how to set 8/16-bit composite timer working in PWM timer function (variable-cycle mode).

1. Set the mode (T00CR0: [F3..F0] = 0 1 0 0 )
2. Set the counter clock (T00CR0: [C2..C0])
3. Set the counter clock (T01CR0: [C2..C0])
4. Set the duty by setting T00DR
5. Set the cycle by setting T01DR
6. Enable the timer output (T00CR1: OE = 1)
7. Start the timer (T00CR1: STA = 1)

The following example shows how to set the 8/16-bit composite timer for operation with PWM timer function (variable-cycle mode).

```
/*                            SAMPLE CODE                                    */
/*---------------------------------------------------------------------------*/

/* initial timer for PWM function (variable cycle) */
void InitCompTimer (void)
{
       T01DR = 0x80;       // counter timer 01 value for duty
       T00DR = 0x40;       // counter timer 00 value for cycle
                           // duty ratio 50%
       TMCR0 = 0x00;       // 8-bit, no filtering

       T00CR0 = 0x04;      // PWM timer function (variable-cycle mode)
       T01CR0 = 0x04;      // choose the same count clock as Timer 00
       T00CR1 = 0x01;      // enable the output
}
       ...

/* main routine */
void main (void)
{
       ...
       InitCompTimer ();
       ...

       T00CR1_STA = 1;     //start the timer

       while(1);
}
```

Refer to *Appendix Sample Code* for project “PWM_Vari”.

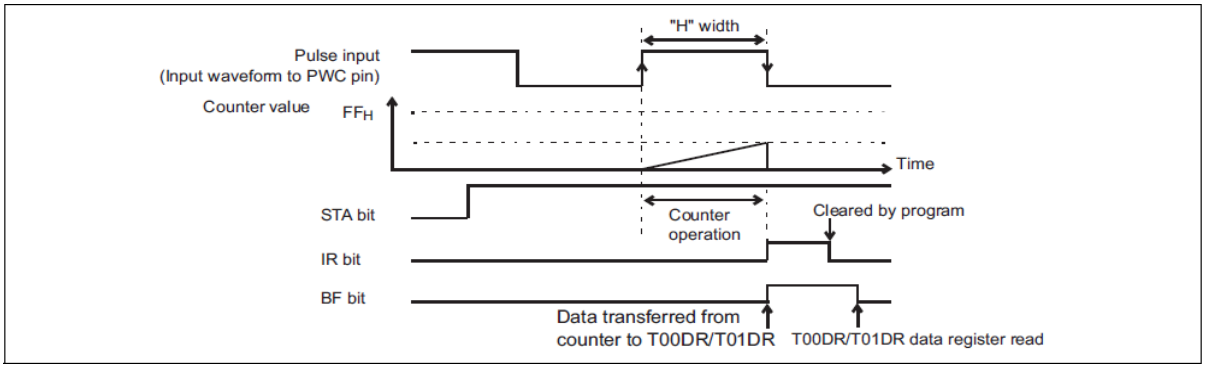
Please note that timer 00 and timer 01 shall use the same counter clock when using this function.

## 3.6 Operation of PWC Timer Function

When the PWC timer function is selected, the width and the cycle of an external input pulse can be measured.

In this operating mode, the counter starts counting from "00H" upon the detection of a count start edge of an external input signal and transfers the count value to a register to generate an interrupt upon the detection of a count end edge.

Figure 11. Operating Diagram of PWC Timer (Example of H-pulse Width Measurement)




Setting Procedure Example

The procedure below shows how to set up an 8/16-bit composite timer working in the PWC timer function (“H” pulse).

1. Set input for timer (DDR0_P04 = 0; AIDRL_P04 = 1;)
2. Select the input port ( SYSC_EC0SL = 1)
3. Set the mode (T00CR0: [F3..F0] = 0 1 0 1 )
4. Set the counter clock (T00CR0 : [C2..C0] )
5. Set the interrupt level if need (ILR1)
6. Enable timer output (T00CR1: OE = 1)
7. Start timer (T00CR1: STA = 1)
8. Read the value while upon completion of the measurement

If the interrupt is enabled, the flag in ISR shall be cleared. (T00CR1: IR = 0).

The PWC timer has five measurement modes as shown below:

Table 1. Relationship between T00CR0/T01CR_F3..F0 and PWC Timer Function

| T00CR0/T01CR0 [F3..F0] | Description |
|---|---|
| 0 1 0 1 | "H" pulse = rising to falling |
| 0 1 1 0 | “L" pulse = falling to rising |
| 0 1 1 1 | cycle = rising to rising |
| 1 0 0 0 | cycle = falling to falling |
| 1 0 0 1 | “H" pulse = rising to falling; Cycle = rising to rising |

The following example shows how to set the 8/16-bit composite timer 0 for operation in the PWC Timer Function.

```
/*                                SAMPLE CODE                                */
/*---------------------------------------------------------------------------*/
/* initial timer for the PWC function */
void InitCompTimer (void)
{
       SYSC_EC0SL = 1;   // P04 as EC0
       DDR0_P04 = 0;     // P04 input
       AIDRL |= 0x10;    // disable AN04 input
       TMCR0 = 0x00;       // 8-bit, no filtering
       T00CR0 = 0x05;      // PWC timer ("H" pulse = from rising to falling)
       T00CR1 = 0xA0;      // enable an interrupt, disable the output
                           // start the timer
}
/* enter ISR while measurement completes or overflows */
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;      // disable an interrupt

       if (T00CR1_BF)      // measurement data present in data register ?
       {
             ucHCycle = T00DR;   // read value
       }
       T00CR1_IR = 0;            // clear the flag
       T00CR1_IE = 1;            // enable the interrupt
}
/* main routine */
void main (void)
{
       ...

       InitCompTimer ();

       ...                 // input signal to be measured

       while(1);
}
```

Please note that the corresponding interrupt vector and level shall be defined in the vector.c module of Fujitsu's standard template project.

```
/*                           SAMPLE CODE                                     */
/*---------------------------------------------------------------------------*/

/* interrupt level setting */
void InitIrqLevels (void)
{
       ...

       ILR1 = 0xF3;        // IRQ5: 8/16-bit timer ch.0 (lower)
       ...
}
__interrupt void CompTimer (void);      // Prototype
       ...

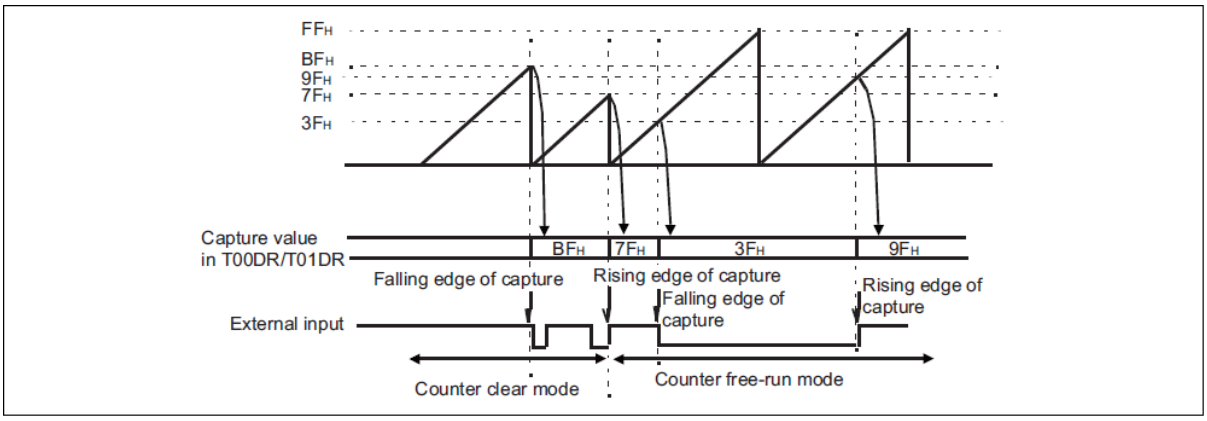
  #pragma intvect CompTimer      5      // 8/16-bit timer ch.0 (lower)
       ...
```

Refer to *Appendix Sample Code* for project "PWC".

## 3.7 Operation of the Input Capture Function

When the input capture function is selected, the counter value is stored in a register upon the detection of an edge for an external input signal.

Figure 12. Operating Diagram of the Input Capture Function




Setup Procedure Example

The procedure below shows how to set an 8/16-bit composite timer working in the Input Capture Function (rising, counter clear).

1. Set the input for timer (DDR0_P04 = 0; AIDRL_P04 = 1; )
2. Select the input port ( SYSC_EC0SL = 1)
3. Set the mode (T00CR0: [F3..F0] = 1 1 0 1 )
4. Set the counter clock (T00CR0: [C2..C0] )
5. Set the interrupt level if need (ILR1)
6. Enable the timer output (T00CR1: OE = 1)
7. Start the timer (T00CR1: STA = 1)
8. Read the value upon completion of the measurement

If the interrupt is enabled, the flag in ISR shall be cleared. (T00CR1: IR = 0).

The Input capture has six capture modes as shown below.

Table 2. Relationship between T00CR0/T01CR_F3..F0 and Input Capture Function

| T00CR0/T01CR0 [F3..F0] | Description |
|---|---|
| 1 0 1 0 | rising, free-run counter |
| 1 0 1 1 | falling, free-run counter |
| 1 1 0 0 | both edges, free-run counter |
| 1 1 0 1 | rising, counter clear |
| 1 1 1 0 | falling, counter clear |
| 1 1 1 1 | both edges, counter clear |

The following example shows how to set an 8/16-bit composite timer 0 for operation in the Input Capture Function.

```
/*                                SAMPLE CODE                                */
/*---------------------------------------------------------------------------*/
/* initial timer for the input capture function */
void InitCompTimer (void)
{SYSC_EC0SL = 1;   // P04 as EC0
       DDR0_P04 = 0;     // P04 input
       AIDRL |= 0x10;    // disable AN04 input

       TMCR0 = 0x00;       // 8-bit, no filtering

       T00CR0 = 0x0D;      // Input capture (rising, counter clear)

       T00CR1 = 0xA0;      // enable the interrupt, disable the output
                           // start the timer
}
/* Enter ISR while capturing the input */
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;      // disable the interrupt

       ucHCycle = T00DR;   // read value

       T00CR1_IR = 0;      // clear the flag
       T00CR1_IE = 1;      // enable the interrupt
}
/* main routine */
void main (void)
{
       ...

       InitCompTimer ();

       ...                 // input signal to be captured

       while(1);
}
```

Please note that the corresponding interrupt vector and level shall be defined in the vector.c module of Fujitsu's standard template project.

```
/*                              SAMPLE CODE                                  */
/*---------------------------------------------------------------------------*/
/* interrupt level setting */
void InitIrqLevels (void)
{
       ...

       ILR1 = 0xF3;        // IRQ5: 8/16-bit timer ch.0 (lower)
       ...
}
__interrupt void CompTimer (void);      // Prototype
       ...
#pragma intvect CompTimer  5      // 8/16-bit timer ch.0 (lower)
       ...
```

Refer to *Appendix Sample Code* for project "Capture".

# 4 Notes on Using 8/16-bit Composite Timer

Chapter four is devoted to the notes on using 8/16-bit composite timer.

- To switch the timer function with the timer operation mode select bits (T00CR0/T01CR0:F3, F2, F1, F0), stop the timer operation first (T00CR1/T01CR1: STA = 0), then clear the interrupt flag (T00CR1/T01CR1: IF, IR), the interrupt enable bits (T00CR1/T01CR1: IE, T00CR0/T01CR0: IFE) and the buffer full flag (T00CR1/T01CR1: BF).
- In the operation of the PWC function or the input capture function, an interrupt may occur even before the timer is activated (STA = 0). Therefore, invalidate the value of the 8/16-bit composite timer 00/01 data register (T00DR/T01DR) existing before the start of the timer.

# 5 Appendix

## 5.1 Sample Code

### 5.1.1 Project Name: One-shot

**Interval Timer Function (One-shot Mode)**

**main.c**

```
#include "mb95200.h"
/*-------------------------------------------------------------------------
   name: InitCompTimer();
   function: initial timer for the interval timer (One-shot) function
-------------------------------------------------------------------------*/
void InitCompTimer (void)
{
      T00DR = 0xFF;       // set the count value
      TMCR0 = 0x00;       // 8-bit, no filtering
      T00CR0 = 0x00;            // interval timer in one-shot mode
      T00CR1 = 0x01;            // disable the interrupt, enable the output
}
/*-------------------------------------------------------------------------
   name: main();
   function: main loop
-------------------------------------------------------------------------*/
void main(void)
{
      InitCompTimer();
      T00CR1_STA = 1;           // start the timer
}
```

**Project Name: Continuous**

**Interval Timer Function (continuous mode)**

**main.c**

```
#include "mb95200.h"
/*-----------------------------------------------------------------------------
   name: InitCompTimer();
   function: initial timer for the interval timer (continuous) function
-----------------------------------------------------------------------------*/
void InitCompTimer (void)
{
       T01DR = 0x01; // set the count value (upper 8 bits)
       T00DR = 0xFF; // set the count value (lower 8 bits)
       TMCR0 = 0x10; // 16-bit, no filtering
       T00CR0 = 0x81;      // interval timer in continuous mode
                           // enable the IF flag interrupt
       T00CR1 = 0xA1;      // enable the interrupt, enable the output
                           // start the timer
}

/*-----------------------------------------------------------------------------
   name: CompTimer ();
   function: enter ISR while the counter value matches the pre-set value
-----------------------------------------------------------------------------*/
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;      // disable the interrupt
       T00CR1_IF = 0;      // clear the flag
       //...               // interrupt service routine
       T00CR1_IE = 1;      // enable the interrupt
}
/*-----------------------------------------------------------------------------
   name: main ();
   function: main loop
-----------------------------------------------------------------------------*/
void main (void)
{
       InitCompTimer();
       InitIrqLevels();
       __EI();
       while (1);
}
```

**vector.c**

```
#include "mb95200.h"
/*-----------------------------------------------------------------------
   name: InitIrqLevels ();
   function: Interrupt level (priority) setting
-----------------------------------------------------------------------*/
void InitIrqLevels(void)
{
      ILR1 = 0xF3;  // IRQ4:  UART/SIO ch0
                    // IRQ5:  8/16-bit timer ch.0 (lower)
                    // IRQ6:  8/16-bit timer ch.0 (upper)
                    // IRQ7:  LIN-UART (reception)
}

/*-----------------------------------------------------------------------
   Prototypes
-----------------------------------------------------------------------*/
__interrupt void CompTimer (void);

/*-----------------------------------------------------------------------
   Vector definition
-----------------------------------------------------------------------*/
#pragma intvect CompTimer 5   // IRQ5:  8/16-bit timer ch.0 (lower)
```

### 5.1.2 Project Name: Free-run Interval Timer Function (Free-run mode)

**main.c**

```
#include "mb95200.h"
/*-----------------------------------------------------------------------------
   name: InitCompTimer();
   function: initial timer for the interval timer (Free-run) function
-----------------------------------------------------------------------------*/
void InitCompTimer (void)
{
       T01DR = 0x01; // set count value (upper 8 bits)
       T00DR = 0xFF; // set count value (lower 8 bits)
       TMCR0 = 0x10; // 16-bit, no filtering
       T00CR0 = 0x82;      // interval timer in a free run mode
                           // enable the IF flag interrupt
       T00CR1 = 0xA1;      // enable the interrupt, enable the output
                           // start the timer
}
/*-----------------------------------------------------------------------------
   name: CompTimer ();
   function: enter ISR while the counter value matches the pre-set value
-----------------------------------------------------------------------------*/
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;      // disable the interrupt
       T00CR1_IF = 0;      // clear the flag
       //...               // interrupt service routine
       T00CR1_IE = 1;      // enable the interrupt
}
/*-----------------------------------------------------------------------------
   name: main ();
   function: main loop
-----------------------------------------------------------------------------*/
void main (void)
{
       InitCompTimer();
       InitIrqLevels();
       __EI();
       while (1);
}
```

**vector.c**

```
#include "mb95200.h"
/*-----------------------------------------------------------------------------
   name: InitIrqLevels ();
   function: Interrupt level (priority) setting
-----------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
       ILR1 = 0xF3;  // IRQ4:  UART/SIO ch.0
                     // IRQ5:  8/16-bit timer ch.0 (lower)
                     // IRQ6:  8/16-bit timer ch.0 (upper)
                     // IRQ7:  LIN-UART (reception)
}
/*-----------------------------------------------------------------------------
   Prototypes
-----------------------------------------------------------------------------*/
__interrupt void CompTimer (void);

/*-----------------------------------------------------------------------------
   Vector definition
-----------------------------------------------------------------------------*/
#pragma intvect CompTimer 5   // IRQ5:  8/16-bit timer ch0 (lower)
```

### 5.1.3 Project Name: PWM_Fixed

**PWM Timer Function (Fixed Cycle Mode)**

**main.c**

```
#include “mb95200.h”
/*-------------------------------------------------------------------------
   name: InitCompTimer();
   function: initial timer for the PWM (fixed-cycle) function
-------------------------------------------------------------------------*/
void InitCompTimer (void)
{
       T00DR = 0x40;// set the PWM timer duty
       TMCR0 = 0x00;// 8-bit, no filtering
       T00CR0 = 0x03;      // PWM timer function (fixed cycle mode)
       T00CR1 = 0x01;      // enable the output
}
/*-------------------------------------------------------------------------
   name: main ();
   function: main loop
-------------------------------------------------------------------------*/
void main (void)
{
       InitCompTimer();
       T00CR1_STA = 1;     // start the timer
       while(1);
}
```

### 5.1.4 Project Name: PWM_Vari

**PWM Timer Function (Variable-cycle Mode)**

**main .c**

```
#include "mb95200.h"
/*-----------------------------------------------------------------------------
   name: InitCompTimer();
   function: initial timer for PWM (variable cycle) function
-----------------------------------------------------------------------------*/
void InitCompTimer (void)
{
       T01DR = 0x80;// set the PWM timer cycle
       T00DR = 0x40;// set the PWM timer duty
                        // duty ratio 50%
       TMCR0 = 0x00;// 8-bit, no filtering
       T00CR0 = 0x04;      // PWM timer function (variable cycle mode)
       T01CR0 = 0x04;      // choose the same count clock as Timer 00
       T00CR1 = 0x01;      // enable the output
}
/*-----------------------------------------------------------------------------
   name: main ();
   function: main loop
-----------------------------------------------------------------------------*/
void main (void)
{
       InitCompTimer();
       T00CR1_STA = 1;     // start the timer
       while(1);
}
```

### 5.1.5 Project Name: PWC

**PWC Timer Function**

**main.c**

```
#include "mb95200.h"
unsigned char ucHCycle = 0xFF;
/*-----------------------------------------------------------------------------
   name: InitCompTimer();
   function: initial timer for the PWC function
-----------------------------------------------------------------------------*/
void InitCompTimer (void)
{
       SYSC_EC0SL = 1;     // P04 as EC0
       DDR0_P04 = 0;// P04 input
       AIDRL |= 0x10;      // disable AN04 input
       TMCR0 = 0x00;// 8-bit, no filtering
       T00CR0 = 0x05;      // PWC timer ("H" pulse = from rising to falling)
       T00CR1 = 0xA0;      // enable the interrupt, disable the output
                           // start the timer
}
/*-----------------------------------------------------------------------------
   name: CompTimer ();
   function: enter ISR while the counter value matches the pre-set value
-----------------------------------------------------------------------------*/
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;              // disable the interrupt
       if (T00CR1_BF)              // measurement data present in a data register ?
       {
              ucHCycle = T00DR;    // read the value
       }
       T00CR1_IR = 0;              // clear the flag
       T00CR1_IE = 1;              // enable the interrupt
}
/*-----------------------------------------------------------------------------
   name: main ();
   function: main loop
-----------------------------------------------------------------------------*/
void main (void)
{
       InitCompTimer();
       InitIrqLevels();
       __EI();
                    // input signal to be measured
       while (1);
}
```

**vector.c**

```
#include "mb95200.h"
/*-----------------------------------------------------------------------
   name: InitIrqLevels ();
   function: Interrupt level (priority) setting
-----------------------------------------------------------------------*/
void InitIrqLevels(void)
{
      ILR1 = 0xF3;  // IRQ4:  UART/SIO ch.0
                    // IRQ5:  8/16-bit timer ch.0 (lower)
                    // IRQ6:  8/16-bit timer ch.0 (upper)
                    // IRQ7:  LIN-UART (reception)
}

/*-----------------------------------------------------------------------
   Prototypes
-----------------------------------------------------------------------*/
__interrupt void CompTimer (void);

/*-----------------------------------------------------------------------
   Vector definition
-----------------------------------------------------------------------*/
#pragma intvect CompTimer 5   // IRQ5:  8/16-bit timer ch.0 (lower)
```

### 5.1.6 Project Name: Capture Input Capture Function

**main.c**

```
#include "mb95200.h"
unsigned char ucHCycle = 0xFF;
/*-----------------------------------------------------------------------------
   name: InitCompTimer();
   function: initial timer for the capture input function
-----------------------------------------------------------------------------*/

void InitCompTimer (void)
{
       SYSC_EC0SL = 1;     // P04 as EC0
       DDR0_P04 = 0;       // P04 input
       AIDRL |= 0x10;      // disable AN04 input
       TMCR0 = 0x00;       // 8-bit, no filtering
       T00CR0 = 0x0D;      // Input capture (rising, counter clear)
       T00CR1 = 0xA0;      // enable the interrupt, disable the output
                           // start the timer
}
/*-----------------------------------------------------------------------------
   name: CompTimer ();
   function: enter ISR while capturing the input
-----------------------------------------------------------------------------*/
__interrupt void CompTimer (void)
{
       T00CR1_IE = 0;      // disable the interrupt
       ucHCycle = T00DR;   // read the value
       T00CR1_IR = 0;      // clear the flag
       T00CR1_IE = 1;      // enable the interrupt
}
/*-----------------------------------------------------------------------------
   name: main ();
   function: main loop
-----------------------------------------------------------------------------*/
void main (void)
{
       InitCompTimer();
       InitIrqLevels();
       __EI();
                    // input signal to be captured
       while (1);
}
```

**vector.c**

```
#include "mb95200.h"
/*-----------------------------------------------------------------------------
   name: InitIrqLevels ();
   function: Interrupt level (priority) setting
-----------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
      ILR1 = 0xF3; //  IRQ4:  UART/SIO ch0
                   //  IRQ5:  8/16-bit timer ch0 (lower)
                   //  IRQ6:  8/16-bit timer ch0 (upper)
                   //  IRQ7:  LIN-UART (reception)
}

/*-----------------------------------------------------------------------------
   Prototypes
-----------------------------------------------------------------------------*/
__interrupt void CompTimer (void);

/*-----------------------------------------------------------------------------
   Vector definition
-----------------------------------------------------------------------------*/
#pragma intvect CompTimer 5   //  IRQ5:  8/16-bit timer ch0 (lower)
```

# 6 Additional Information

For more information on Cypress Microcontrollers Products, please visit the following websites:

http://www.cypress.com/cypress-microcontrollers

http://www.cypress.com/cypress-mcu-product-softwareexamples

## Document History

Document Title: AN205275 – F²MC - 8FX Family, MB95200H/210H Series, 8/16-Bit Composite Timer

Document Number: 002-05275

| Revision | ECN | Orig. of Change | Submission Date | Description of Change |
|---|---|---|---|---|
| ** | - | Levi, Zhang | 20/03/2008 | V1.0, First Draft |
| | | | 18/07/2008 | V1.1, Add URL in chapter 6 Additional Information; modify some sample code. |
| *A | 5260372 | HUAL | 05/06/2016 | Migrated Spansion Application Note “MCU-AN-500004-E-11” 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,2008-2016. 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.