AN205398

F²MC-8FX Family MB95200 Series 8-Bit Microcontroller Counter and Timer API

Associated Part Family: MB95200 Series

This document introduces API for Timebase Timer, Watch dog timer, Watch and Composite timer.

Contents

1 Introduction....1
2 Timebase Timer....1
2.1 Timebase Timer Library Function List....1
2.2 Timebase Timer Function Detail....2
2.3 Usage Demo....2
3 Hardware/Software Watchdog Timer....3
3.1 Watchdog Timer Library Function List....3
3.2 Watchdog Timer Function Detail....3
3.3 Usage Demo....5
4 Watch Prescaler....6
4.1 Watch Prescaler Library Function List....6
4.2 Watch Prescaler Function Detail....6
4.3 Usage Demo....7
5 Composite Timer....7
5.1 Composite Timer Library Function List....7
5.2 Composite Timer Function Detail....7
5.3 Usage Demo....15
6 Additional Information....18

1 Introduction

This document introduces API for Timebase Timer, Watch dog timer, Watch and Composite timer.

2 Timebase Timer

This section introduces all functions of Timebase Timer library.

2.1 Timebase Timer Library Function List

Table 1 lists the Timebase Timer library functions.

Table 1. Timebase Timer Functions

Function name	Description
void Timebase_Timer_Initialize (uchar Interval_Size)	Initialize Timebase Timer

2.2 Timebase Timer Function Detail

2.2.1 Timebase_Timer_Initialize Function

Table 2 describes Timebase_Timer_Initialize function.

Table 2. Timebase_Timer_Initialize Function

Function name	Timebase_Timer_Initialize
Function prototype	void Timebase_Timer_Initialize (uchar Interval_Size)
Behavior description	initialize the Timebase Timer
Input parameter	Interval_Size
Return value	None

Table 3 describes the Interval_Size parameter values.

Table 3. Interval_Size Definition

Interval_Size	Description set the size of interval time
Interval_Time_2_9	set the size of interval time: 29 × 2/FCH or 29 × 1/FCRH
Interval_Time_2_10	set the size of interval time: 210 × 2/FCH or 210 × 1/FCRH
Interval_Time_2_11	set the size of interval time: 211 × 2/FCH or 211 × 1/FCRH
Interval_Time_2_12	set the size of interval time: 212 × 2/FCH or 212 × 1/FCRH
Interval_Time_2_13	set the size of interval time: 213 × 2/FCH or 213 × 1/FCRH
Interval_Time_2_14	set the size of interval time: 214 × 2/FCH or 214 × 1/FCRH
Interval_Time_2_15	set the size of interval time: 215 × 2/FCH or 215 × 1/FCRH
Interval_Time_2_16	set the size of interval time: 216 × 2/FCH or 216 × 1/FCRH
Interval_Time_2_17	set the size of interval time: 217 × 2/FCH or 217 × 1/FCRH
Interval_Time_2_18	set the size of interval time: 218 × 2/FCH or 218 × 1/FCRH
Interval_Time_2_19	set the size of interval time: 219 × 2/FCH or 219 × 1/FCRH
Interval_Time_2_20	set the size of interval time: 220 × 2/FCH or 220 × 1/FCRH
Interval_Time_2_21	set the size of interval time: 221 × 2/FCH or 221 × 1/FCRH
Interval_Time_2_22	set the size of interval time: 222 × 2/FCH or 222 × 1/FCRH
Interval_Time_2_23	set the size of interval time: 223 × 2/FCH or 223 × 1/FCRH
Interval_Time_2_24	set the size of interval time: 224 × 2/FCH or 224 × 1/FCRH

2.3 Usage Demo

2.3.1 Set Interval Time of Timebase Timer

Initialize the Timebase Timer, input the interval time value.

For example:

Timebase_Timer_Init (Interval_Time_2_10);

3 Hardware/Software Watchdog Timer

This section introduces all functions of Watchdog library.

3.1 Watchdog Timer Library Function List

Table 4 lists the Watchdog Timer library functions.

Table 4. Watchdog Timer Functions

Function name	Description
void WatchDog_Soft_Count_Clock (uchar COUNT_CLOCK)	Set watchdog count clock
void WatchDog_Soft_Start(void)	Start Software Watchdog
void WatchDog_Clear(void)	Clear watchdog
uchar WatchDog_Hard_Get_Status (void)	Get Hardware watchdog status

In routine, the library has four functions to realize control watchdog, it including set software watchdog count clock, start watchdog, clear watchdog and get hardware watchdog status.

3.2 Watchdog Timer Function Detail

3.2.1 WatchDog_Soft_Count_Clock Function

Table 5 describes WatchDog_Soft_Count_Clock function.

Table 5. WatchDog_Soft_Count_Clock Function

Function name	WatchDog_Soft_Count_Clock
Function prototype	void WatchDog_Soft_Count_Clock (uchar COUNT_CLOCK)
Behavior description	Set watchdog count clock
Input parameter	COUNT_CLOCK
Return value	None

The function can set the watchdog count clock. Select parameter ‘Count_Clock’ to correct value and it will set watchdog count clock.

Table 6 describes the COUNT_CLOCK parameter values.

Table 6. COUNT_CLOCK Definition

COUNT_CLOCK	Description
FCH_2_21	Watchdog count clock
FCH_2_20	Watchdog count clock
FCL_2_14	Watchdog count clock
FCL_2_13	Watchdog count clock

3.2.2 WatchDog_Start Function

Table 7 describes WatchDog_Start function.

Table 7. WatchDog_Start Function

Function name	WatchDog_Start
Function prototype	void WatchDog_Soft_Start(void)
Behavior description	Start Software Watchdog
Input parameter	None
Return value	None

The function can start the watchdog. If you run this function first, it can realize start the watchdog and stop watchdog until reset.

3.2.3 WatchDog_Clear Function

Table 8 describes WatchDog_Clear function.

Table 8. WatchDog_Clear Function

Function name	WatchDog_Clear
Function prototype	void WatchDog_Clear(void)
Behavior description	Clear watchdog
Input parameter	None
Return value	None

The function can clear watchdog including software watchdog and hardware watchdog.

3.2.4 WatchDog_Hard_Get_Status Function

Table 9 describes WatchDog_Hard_Get_Status function.

Table 9. WatchDog_Hard_Get_Status Function

Function name	WatchDog_Hard_Get_Status
Function prototype	uchar WatchDog_Hard_Get_Status (void)
Behavior description	get the hardware watchdog status
Input parameter	None
Return value	‘1’ hardware watchdog run ‘0’ hardware watchdog stop

The function can get the hardware watchdog status. If you run this function and can realize get the hardware watchdog run or stop.

3.3 Usage Demo

3.3.1 Hardware Watchdog Usage

If you want to use the hardware watchdog, enable the hardware watchdog and start it.

The sample as:

```
#define HWD_ENABLE              ; if define this, Hard Watchdog will enable.
Start hardware watchdog:
{
        WatchDog_Start();       //start hardware watchdog
}
```

3.3.2 Get Status of Hardware Watchdog

If you want to know the status of hardware watchdog, use the WatchDog_Hard_Get_Status() function, it will return the status of hardware watchdog.

The sample as:

```
Void watchdog_hardware(void)
   {
Uchar status;
Status=WatchDog_Hard_Get_Status();
}
```

3.3.3 Software Watchdog Usage

If you want to use the software watchdog, set the watchdog count clock and start software watchdog.

The sample as:

```
Void software_watchdog(void)
{
   WatchDog_Soft_Count_Clock(FCH_2_21);
   WatchDog_Start();
   Watchdog_Clear();   //clear software watchdog
}
```

4 Watch Prescaler

This section introduces all functions of Watch Prescaler library.

4.1 Watch Prescaler Library Function List

Table 10 lists the watch prescaler library functions.

Table 10. Watch Prescaler Functions

Function name	Description
void Watch_Prescaler_Init(uchar Interval_Time)	Interval time by the system clock ensure

4.2 Watch Prescaler Function Detail

4.2.1 Watch_Prescaler_Init Function

Table 11 describes Watch_Prescaler_Init function.

Table 11. Watch_Prescaler_Init Function

Function name	Watch_Prescaler_Init
Function prototype	void Watch_Prescalar_Initial(uchar Interval_Time)
Behavior description	Interval time by the system clock ensure
Input parameter	Interval_Time
Return value	None

The function set interval time when the system clock works on sub clock.

Table 12 describes the Interval_Time parameter values.

Table 12. Interval_Time Definition

COUNT_CLOCK	Description	
	Interval time (Sub clock FCL=32.768kHz)	Interval time (Sub CR clock FCRL=100kHz)
Interval_time_2_10	210 × 2/FCL (62.5ms)	210 × 2/FCRL (20.48ms)
Interval_time_2_11	211 × 2/FCL (125.ms)	211 × 2/FCRL (40.96ms)
Interval_time_2_12	212 × 2/FCL (250.ms)	212 × 2/FCRL (81.92ms)
Interval_time_2_13	213 × 2/FCL (500.ms)	213 × 2/FCRL (163.84ms)
Interval_time_2_14	214 × 2/FCL (1.s)	214 × 2/FCRL (327.68ms)
Interval_time_2_15	215 × 2/FCL (2.s)	215 × 2/FCRL (655.36ms)
Interval_time_2_16	216 × 2/FCL (4.s)	216 × 2/FCRL (1.311s)
Interval_time_2_17	217 × 2/FCL (8.s)	217 × 2/FCRL (2.621s)

4.3 Usage Demo

Use Watch_Prescaler_Init function.

The sample as:

```
void initialize (void)
{
    Watch_Prescaler_Init (Interval_time_2_10);          // interval time 2^10x2/FCL
}
```

5 Composite Timer

This section introduces all functions of Composite Timer library.

5.1 Composite Timer Library Function List

Table 13 lists the Composite Timer library functions.

Table 13. Composite Timer Functions

Function name	Description
Composite_Timer_Time	Initialize one-shot mode or continuo-us mode or free-run mode
Composite_Timer_PWMF	Initialize PWM(fixed-cycle mode)
Composite_Timer_PWMV	Initialize PWM(variable-cycle mode)
Composite_Timer_PWC	Initialize PWC mode
Composite_Timer_capture	Initialize capture mode
Composite_Timer_Output	Initialize channel output states
Composite_Timer_Operate	start, stop and suspended
Composite_Timer_Read_Counter	Read the value from counter

5.2 Composite Timer Function Detail

5.2.1 Composite_Timer_Time Function

Table 14 describes Composite_Timer_Time function.

Table 14. Composite_Timer_Time Function

Function name	Composite_Timer_Time
Function prototype	void Composite_Timer_Time (uchar Mod, uchar Channel, uchar Mclk, uchar Select_Bits, uint Counter_Init);
Behavior description	Initialize one-shot mode or continuo-us mode or free-run mode
Input parameter1	Mod, Set the mode of using
Input parameter2	Channel, set the channel
Input parameter3	Mclk, Set which mclk of using
Input parameter4	Select_Bits, set the counter working at 8-bit or 16-bit
Input parameter5	Counter_Init, set initial value of counter, 0~255 or 0~65535
Return value	None

Note: Input parameter Counter_Init values be relate to Select_Bits parameter.

For example: when Select_Bits is Counter_8_bit, Counter_Init values range: 0~255. When Select_Bits is Counter_16_bit, Counter_Init values range: 0~65535.

This function to set the work states (one-shot or continuous or free-run) of Composite Timer, and initialize the state, if use interrupts of Composite Timer, please refers to Interrupts chapter.

Table 15 describes the Mod parameter values.

Table 15. Mod Definition

Mod	Description
One_shot	One-shot Mode
Continuous	Continuous Mode
Free_run	Free-run Mode

Table 16 describes the Channel parameter values.

Table 16. Channel Definition

Channel	Description
T00	set the channel TO00
T01	set the channel TO01
T10	set the channel TO10
T11	set the channel TO11

Table 17 describes the Mclk parameter values.

Table 17. Mclk Definition

Mclk	Description
MCLK_Div_1	1 × MCLK (machine clock)
MCLK_Div_2	1/2 × MCLK (machine clock)
MCLK_Div_4	1/4 × MCLK (machine clock)
MCLK_Div_8	1/8 × MCLK (machine clock)
MCLK_Div_16	1/16 × MCLK (machine clock)
MCLK_Div_32	1/32 × MCLK (machine clock)
MCLK_EX_Div_128	1/27 × FCH
External_clock	External clock

Table 18 describes the Select_bits parameter values.

Table 18. Select_bits Definition

Select_Bits	Description
Counter_8_bit	8-bit
Counter_16_bit	16-bit

5.2.2 Composite_Timer_PWMF Function

Table 19 describes Composite_Timer_PWMF function.

Table 19. Composite_Timer_PWMF Function

Function name	Composite_Timer_PWMF
Function prototype	void Composite_Timer_PWMF (uchar Mod, uchar Channel, uchar Mclk, uchar Select_Bits, uint Counter_init);
Behavior description	
Input parameter1	Mod, Set the mode of using. Refer to table5-3 for more details on the allowed values for this parameter.
Input parameter2	Channel, set the channel. Refer to table5-4 for more details on the allowed values for this parameter.
Input parameter3	Mclk, Set which mclk of using. Refer to table5-5 for more details on the allowed values for this parameter.
Input parameter4	Select_Bits, set the counter working at 8-bit or 16-bit Refer to table5-6 for more details on the allowed values for this parameter.
Input parameter5	Counter_init, set initial value of counter(T00/T10), 0~65535
Return value	None

This function to set the work states (PWM Timer Function (Fixed-cycle Mode)) of Composite Timer, and initialize the state, if use interrupts of Composite Timer, please refers to Interrupts chapter.

5.2.3 Composite_Timer_PWMV Function

Table 20 describes Composite_Timer_PWMV function.

Table 20. Composite_Timer_PWMV Function

Function name	Composite_Timer_PWMV
Function prototype	void Composite_Timer_PWMV (uchar Channel,uchar Mclk, uchar Duty_Cycle, uchar Cycle)
Behavior description	Initialize PWM(variable-cycle mode)
Input parameter1	Channel, set the channel Refer to table5-4 for more details on the allowed values for this parameter.
Input parameter2	Mclk, Set which mclk of using Refer to table5-5 for more details on the allowed values for this parameter.
Input parameter3	Duty_Cycle, set initial value of counter(T00/T10), 0~255
Input parameter4	Cycle, set initial value of counter(T01/T11), 0~255
Return value	None

This function can set the work states (PWM Timer Function (variable-cycle Mode)) of Composite Timer and initialize the state. If use interrupts of Composite Timer, please refer to Interrupts chapter.

5.2.4 Composite_Timer_PWC Function

Table 21 describes Composite_Timer_PWC function.

Table 21. Composite_Timer_PWC Function

Function name	Composite_Timer_PWC
Function prototype	void Composite_Timer_PWC (uchar Mod, uchar Channel, uchar Mclk, uchar Select_Bits, uchar Select_In, uchar Filter)
Behavior description	Initialize PWC mode
Input parameter1	Mod, Set the mode of PWC
Input parameter2	Channel, set the channel Refer to table5-4 for more details on the allowed values for this parameter.
Input parameter3	Mclk, Set which mclk of using Refer to table5-5 for more details on the allowed values for this parameter.
Input parameter4	Select_Bits, set the counter working at 8-bit or 16-bit Refer to table5-6 for more details on the allowed values for this parameter.
Input parameter5	Select_In, Select external signal or internal signal to input
Input parameter6	Filter, Set the input signal to Filter or not, and select which mode to filter
Return value	None

This function can set the work states (PWC) of Composite Timer, initialize the state and set the work state of PWC. If use interrupts of Composite Timer, please refer to Interrupts chapter.

Table 22 describes the Mod parameter values.

Table 22. Mod Definition

Mod	Description
Pulse_H	PWC timer ("H" pulse = rising to falling)
Pulse_L	PWC timer ("L" pulse = falling to rising)
Cycle_H-H	PWC timer (cycle = rising to rising)
Cycle_L-L	PWC timer (cycle = falling to falling)
Mod_P_H_C_H-H	PWC timer ("H" pulse = rising to falling; Cycle = rising to rising)

Table 23 describes the Select_In parameter values.

Table 23. Select_In Definition

Select_In	Description
In_EC00	Selecting external signal (EC00) as timer 00 input
In_TII0	Selecting internal signal (TII0) as timer 00 input

Table 24 describes the Filter parameter values.

Table 24. Filter Definition

Filter	Description
Filter_NO	No filtering
Filter_H	Removing "H" pulse noise
Filter_L	Removing "L" pulse noise
Filter_Both	Removing "H"/"L" pulse noise

5.2.5 Composite_Timer_Capture Function

Table 25 describes Composite_Timer_Capture function.

Table 25. Composite_Timer_Capture Function

Function name	Composite_Timer_Capture
Function prototype	void Composite_Timer_Capture(uchar Mod, uchar Channel, uchar Mclk, uchar Select_Bits, uchar Select_In, uchar Filter)
Behavior description	Initialize capture mode
Input parameter1	Mod, Set the mode of capture
Input parameter2	Channel, set the channel Refer to table5-4 for more details on the allowed values for this parameter.
Input parameter3	Mclk, Set which Mclk of using Refer to table5-5 for more details on the allowed values for this parameter.
Input parameter4	Select_Bits, set the counter working at 8-bit or 16-bit Refer to table5-6 for more details on the allowed values for this parameter.
Input parameter5	Select_In, Select external signal or internal signal to input Refer to table5-12 for more details on the allowed values for this parameter.
Input parameter6	Filter, Set the input signal to Filter or not, and select which mode to filter. Refer to table5-13 for more details on the allowed values for this parameter.
Return value	None

This function to set the work states (capture) of Composite Timer, initialize the state and set the work state of PWC. If use interrupts of Composite Timer, please refer to Interrupts chapter.

Table 26 describes the Mod parameter values.

Table 26. Mod Definition

Mod	Description
FreeRun_Rising	Input capture (rising, free-run counter)
FreeRun_Falling	Input capture (falling, free-run counter)
FreeRun_Both	Input capture (both edges, free-run counter)
Counter_Clear_Rising	Input capture (rising, counter clear)
Counter_Clear_Falling	Input capture (falling, counter clear)
Counter_Clear_Both	Input capture (both edges, counter clear)

5.2.6 Composite_Timer_Output Function

Table 27 describes Composite_Timer_Output function.

Table 27. Composite_Timer_Output Function

Function name	Composite_Timer_Output
Function prototype	void Composite_Timer_Output(uchar Channel, uchar Output, uchar Output_init)
Behavior description	Initialize channel output states
Input parameter1	Channel, select the channel to operated Refer to table5-4 for more details on the allowed values for this parameter.
Input parameter2	Output, enable or disable output
Input parameter3	Output_Init, Initialize channel output value
Return value	None

This function use to enable the channel to output and set the initialization.

Table 28 describes the Output parameter values.

Table 28. Output Definition

Output	Description
Out_Disable	disable output
Out_Enable	enable output

Table 29 describes the Output_Init parameter values.

Table 29. Output_Init Definition

Output_Init	Description
Out_initial_0	Initialization value is “0”
Out_initial_1	Initialization value is “1”

5.2.7 Composite_Timer_Operate Function

Table 30 describes Composite_Timer_Operate function.

Table 30. Composite_Timer_Operate Function

Function name	Composite_Timer_Operate
Function prototype	void Composite_Timer_Operate (uchar Mod, uchar Channel)
Behavior description	start, stop and suspended
Input parameter1	Mod: start, stop, suspend or restore
Input parameter2	Channel, select the channel to operated Refer to table5-4 for more details on the allowed values for this parameter.
Return value	None

This function use to start, stop, suspend or restore Composite Timer.

Table 31 describes the Mod parameter values.

Table 31. Mod Definition

Mod	Description
Counter_Diable	stop
Counter_Enable	start
Counter_Suspend	suspended
Counter_Restore	restore

5.2.8 Composite_Timer_Read_Counter Function

Table 32 describes Composite_Timer_Read_Counter function.

Table 32. Composite_Timer_Read_Counter Function

Function name	Composite_Timer_Read_Counter
Function prototype	uchar Composite_Timer_Read_Counter (uchar Channel)
Behavior description	Read the value from counter
Input parameter	Channel, set the channel
Return value	value of counter

This function use to read the value of data register.

Table 33 describes the Channel parameter values.

Table 33. Channel Definition

Channel	Description
T00	select counter to read, T00DR
T01	select counter to read, T01DR
T10	select counter to read, T10DR
T11	select counter to read, T11DR

5.3 Usage Demo

5.3.1 Start

1. One-shot

   When the interval timer function (one-shot mode) is selected, the counter starts counting from "00H" as the timer is started. When the counter value matches the value of the 8/16-bit composite timer 00/01 data register, the timer output is inverted, an interrupt request occurs, and the counter stops counting.

   Initialize Composite_Timer_Time (), use Composite_Timer_Output () and use Composite_Timer_Operate () to startup the timer.

   For example:

   Composite_Timer_Time (one_shot, T00, MCLK_Div_1, Counter_8_bit, 130);

   Composite_Timer_Output (T00, Out_Enable, Out_initial_1);

   Composite_Timer_Operate (Counter_Enable, T00);

2. Continuous

   When the interval timer function (continuous mode) is selected, the counter starts counting from "00H" as the timer is started. When the counter value matches the value of the 8/16-bit composite timer 00/01 data register, the timer output is inverted, an interrupt request occurs, and the counter counts from "00H" again. The timer outputs square wave as a result of this repeated operation.

   Initialize Composite_Timer_Time (), use Composite_Timer_Output () and use Composite_Timer_Operate () to startup the timer.

   For example:

   Composite_Timer_Time (continuous, T01, MCLK_Div_1, Counter_8_bit, 130);

   Composite_Timer_Output (T01, Out_Enable, Out_initial_1);

   Composite_Timer_Operate (Counter_Enable, T01);

3. Free-run

   When the interval timer function (free-run mode) is selected, the counter starts counting from "00H". When the counter value matches the value of the 8/16-bit composite timer 00/01 data register, the timer output is inverted and an interrupt request occurs. Under these conditions, if the counter continues to count and reaches "FFH", it restarts counting from "00H". The timer outputs square wave as a result of this repeated operation.

   Initialize Composite_Timer_Time (), use Composite_Timer_Output () and use Composite_Timer_Operate () to startup the timer.

   For example:

   Composite_Timer_Time (free_run, T10, MCLK_Div_1, Counter_8_bit, 130);

   Composite_Timer_Output (T10, Out_Enable, Out_initial_1);

   Composite_Timer_Operate (Counter_Enable, T10);

4. PWM (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 cycles. The cycle is fixed at "FFH" in 8-bit operation or at "FFFFH" in 16-bit operation. The time is determined by the count clock selected. The "H" pulse width is specified by setting a specific register.

   Initialize Composite_Timer_PWMF (), use Composite_Timer_Output () and use Composite_Timer_Operate () to startup the timer.

   For example:

   Composite_Timer_PWMF (T10, MCLK_Div_1, Counter_8_bit, 100);

   Composite_Timer_Output (T10, Out_Enable, Out_initial_1);

   Composite_Timer_Operate (Counter_Enable, T10);

5. PWM (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 of variable cycle and duty depending on the cycle and "L" pulse width specified by registers.

   In this operating mode, since the two 8-bit counters have to be used separately, the composite timer cannot operate as a 16-bit counter.

   Initialize Composite_Timer_PWMV (), use Composite_Timer_Output () and use Composite_Timer_Operate () to startup the timer.

   For example:

   Composite_Timer_PWMV (T10, MCLK_Div_1, 100, 200);

   Composite_Timer_Output (T10, Out_Enable, Out_initial_1);

   Composite_Timer_Operate (Counter_Enable, T10);

6. PWC

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

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

   Initialize Composite_Timer_PWMV (), and use Composite_Timer_Operate () to startup the timer.

   For example:

   Composite_Timer_PWC (Pulse_H, T10, MCLK_Div_1, Counter_8_bit, In_EC00, Out_Init_H, Filter_H);

   Composite_Timer_Operate (Counter_Enable, T10);

7. Capture

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

   This function is available in either free-run mode or clear mode for count operation.

   In clear mode, the counter starts counting from "00H" and transfers its value to a register to generate an interrupt upon detection of an edge. Afterward, the counter restarts counting from "00H".

   In free-run mode, the counter transfers its value to a register to generate an interrupt upon detection of an edge. Afterward, unlike in clear mode, the counter continues to count without being cleared to "00H".

   Initialize Composite_Timer_PWMV (), and use Composite_Timer_Operate () to startup the timer.

   For example:

   Composite_Timer_Capture (FreeRun_Rising, T00, MCLK_Div_1, Counter_8_bit, In_EC00, Filter_H);

   Composite_Timer_Operate (Counter_Enable, T10);

5.3.2 Suspend

You can suspend the channel of timer, use Composite_Timer_Operate () to suspend the channel timer.

For example:

Composite_Timer_Operate (Counter_Suspend, T00);

5.3.3 Restore

When the timer suspend, you can restore the timer, Composite_Timer_Operate () to restore the channel timer.

For example:

Composite_Timer_Operate (Counter_Restore, T00);

5.3.4 Stop

You can stop the channel of timer, use Composite_Timer_Operate () to stop the channel timer.

For example:

Composite_Timer_Operate (Counter_Enable, T00);

5.3.5 Get the Value of Counter

Use Composite_Timer_Read_Counter () to read the value of counter, it return the value.

For example:

Composite_Timer_Read_Counter (T00);

5.3.6 Use Interrupt of Composite Timer

Please refer to interrupt API.

6 Additional Information

For more information about how to use MB95200H EV-board, BGM Adaptor and SOFTUNE, please refer to SKT MB2146-410-01-E User Manual, or visit website:

http://www.cypress.com/documentation/software-and-drivers/f2mc-8fx-mb95200h210h-series-starter-kit-mb2146-410a-01-e-setup

Document History

Document Title: AN205398 – F²MC-8FX Family MB95200 Series 8-Bit Microcontroller Counter and Timer API

Document Number: 002-05398

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	HUAL	03/13/2009	Initial release
*A	5276201	HUAL	05/18/2016	Migrated Spansion Application Note MCU-AN- 500023-E-10 to Cypress format.
*B	5848930	MALI	08/09/2017	Updated logo and copyright.

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
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

© Cypress Semiconductor Corporation, 2009-2017. 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, WICED, 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.