AN205497

F²MC-16FX Family Series With I/O Timer

This application note describes the functionality of the I/O Timer and its corresponding resources and gives some examples.

Contents

1 Introduction....1
1.1 Key Features....1
2 The I/O Timer....2
2.1 Block Diagrams....2
2.2 Registers....4
3 I/O Timer Example....10
3.1 Basic Functionality of the Free Running Timer....10
3.2 Basic Functionality of the Output Compare Unit....11
3.3 OCUs as PWMs....12
3.4 Basic Functionality of the Input Capture Unit....13
3.5 Frequency counter based on ICU and FRT....15
4 Additional Information....17
5 Document History....18
Worldwide Sales and Design Support....19
Products....19
PSoC® Solutions....19
Cypress Developer Community....19
Technical Support....19

1 Introduction

This application note describes the functionality of the I/O Timer and its corresponding resources and gives some examples.

The I/O Timer consists of a 16-bit Free Running Timer, Output Compare, and Input Capture Units.

1.1 Key Features

1.1.1 16-Bit Free Running Timer

- Clock selectable for Free Running Timer from 62.5 ns to 8 µs at 16 MHz Peripheral Clock 1
- External clocking possible
- Interrupt Generation at counter Overflow or Counter Match with Compare Register 0 and 4
- Up-counter

1.1.2 Output Compare Unit

- Two Output Compare (Match) Registers for Compare Output Toggle
- Complex Combinations with up to 4 Compare Channels possible
- Output pin initial value selectable
- Interrupt Generation on Compare Match

1.1.3 Input Capture Unit

- Two independent Input Capture Units per Module
- Rising, falling or both Edge selectable for Input Event
- Interrupt Generation on Input Event
- Input Capture unit's input source selectable as the LIN-USART Sync Field output

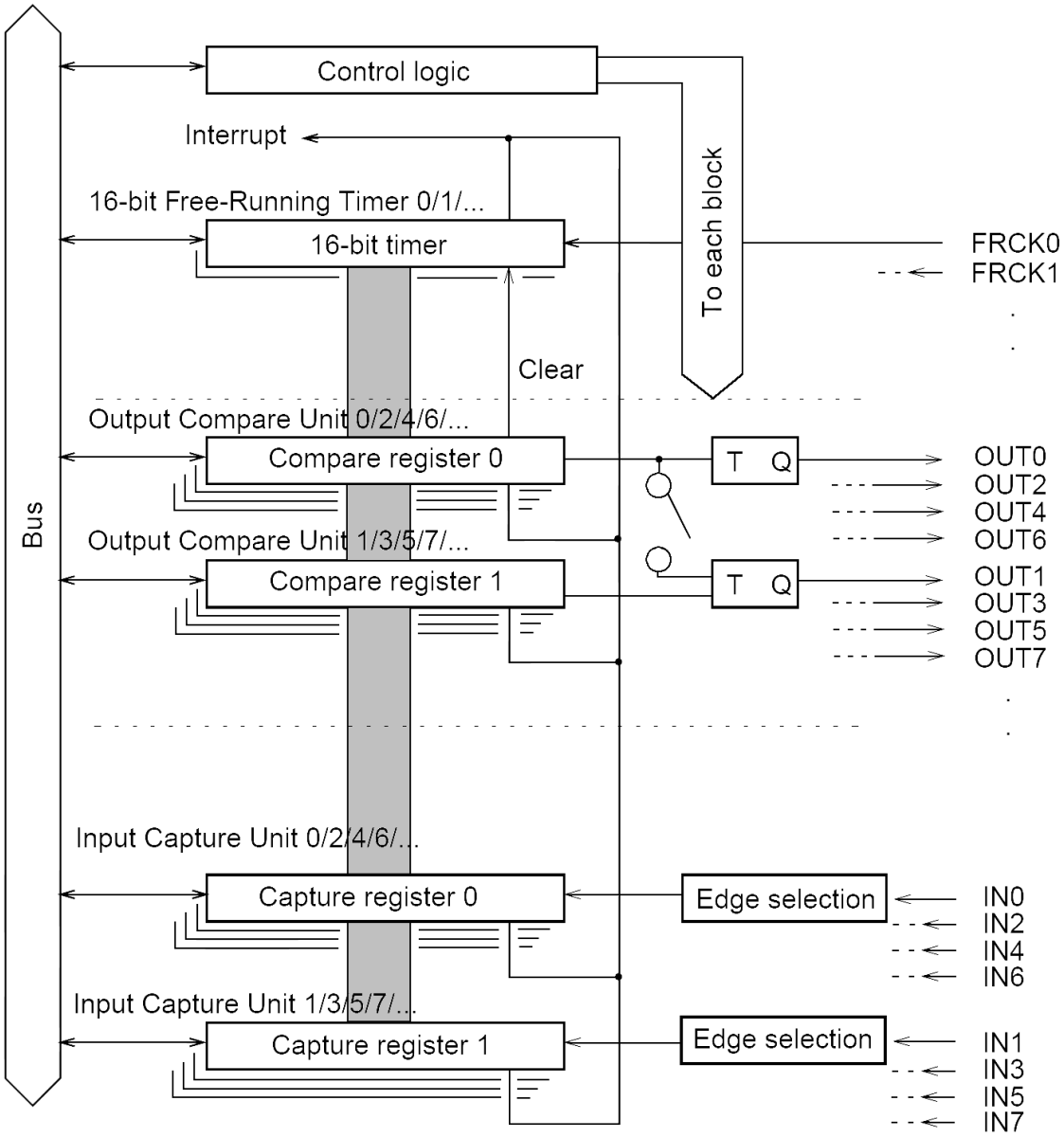
2 The I/O Timer

The basic functionality of the I/O timer module

2.1 Block Diagrams

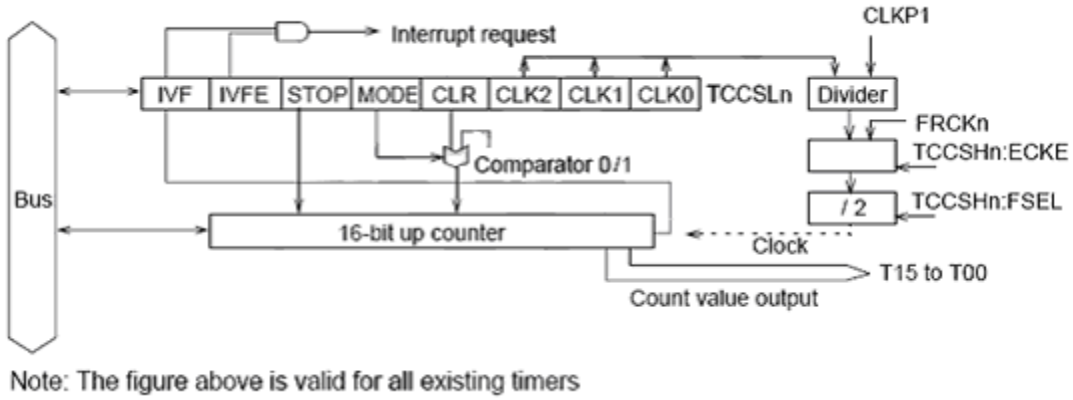
Figure 1 shows the internal block diagram of an I/O Timer.

Figure 1. I/O Timer Block Diagram



The I/O-Timer consists of a 16-Bit free running timer and the Output Compare and Input Capture Unit.

Figure 2. 16-Bit Free Running Timer Block Diagram



Note: The figure above is valid for all existing timers

Figure 3. Output Compare Block Diagram

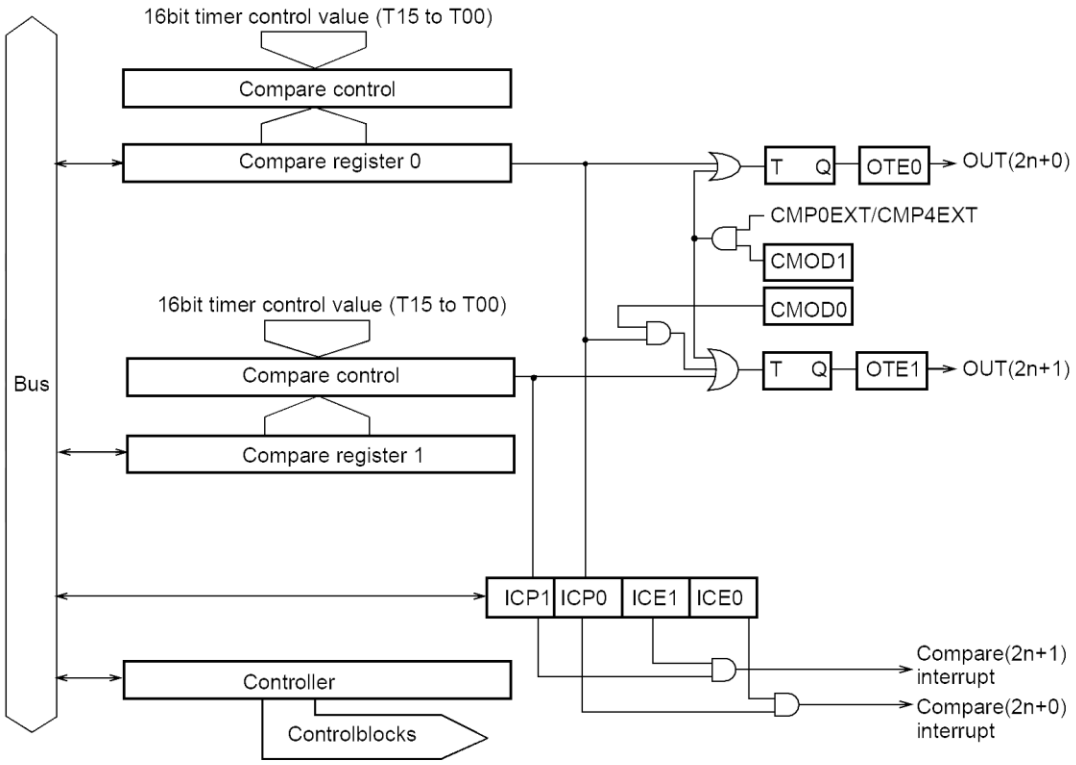
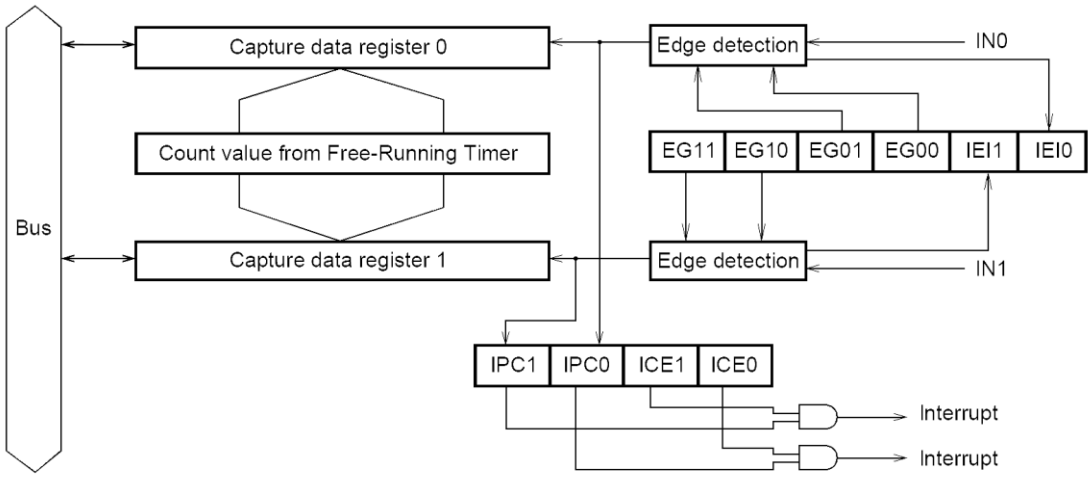


Figure 4. Input Capture Block Diagram



2.2 Registers

2.2.1 16-Bit Free Running Timer

The 16-Bit Free Running Timer is an up-counter.

Control Status Register (Lower Byte) (TCCSL)

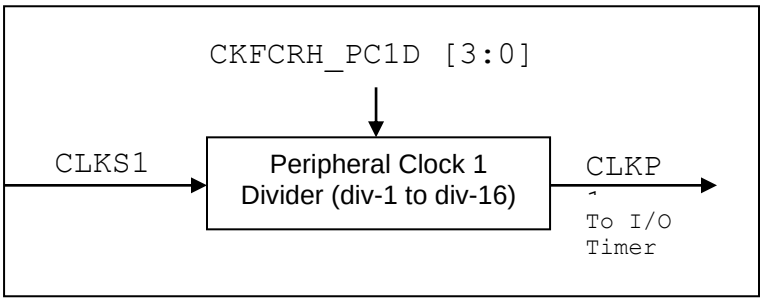
This Byte contains several status and control bits.

Table 1. TCCSL

Bit No.	Name	Explanation	Value	Operation	
7	IVF	Interrupt Request	0	Read: No interrupt, Write: Clear Request	
			1	Read: Interrupt request; Write: No effect	
6	IVFE	Interrupt Request Enable	0	No Interrupts	
			1	Interrupt Enabled	
5	STOP	Stop the Counter	0	Counter Enabled	
			1	Counter Disabled	
4	MODE	Set Reset Condition of Timer	0	Initialization by Reset or Clear Bit	
			1	Initialization by Reset, Clear Bit, or Compare Register 0 and 4	
3	CLR	Clear Timer	0	Write: no effect	
			1	Write: Clear to “0000”	
2, 1, 0	CLK2, CLK1, CLK0[1]	Count Clock Selection		TCCSH:FSEL = 1	TCCSH:FSEL = 0
			0, 0, 0	CLKP1	CLKP1 / 2
			0, 0, 1	CLKP1 / 2	CLKP1 / 4
			0, 1, 0	CLKP1 / 4	CLKP1 / 8
			0, 1, 1	CLKP1 / 8	CLKP1 / 16
			1, 0, 0	CLKP1 / 16	CLKP1 / 32
			1, 0, 1	CLKP1 / 32	CLKP1 / 64
			1, 1, 0	CLKP1 / 64	CLKP1 / 128
			1, 1, 1	CLKP1 / 128	CLKP1 / 256

Please consider that the clock frequency of CLKP1 depends on the settings in the Peripheral Clock 1 Divider.

Figure 5. I/O Timer Clock



[1] Count clock selection setting is only valid if internal clock CLKP1 is selected as clock source for I/O Timer by clearing ECKE bit of TCCSH register.

Control Status Register (Upper Byte) (TCCSH)

This Byte contains control bits for external clocking and prescaler.

Table 2. TCCSH

Bit No.	Name	Explanation	Value	Operation
15	ECKE	External Clock Enable	0	Select Internal Clock
			1	Select External Clock (FRCK)
14	FSEL	Frequency Selection	0	Divide Input Clock by 2
			1	Bypass Input Clock
13	–	*Undefined*	-	-
12	–	*Undefined*	-	-
11	–	*Undefined*	-	-
10	–	*Undefined*	-	-
9	–	*Undefined*	-	-
8	–	*Undefined*	-	-

Data Register of the Free-Running Timer (TCDT)

This 16-Bit register reads out the current counter value. Writing sets a new value to it if the timer is stopped. This register must be accessed by word access instructions.

2.2.2 Output Compare Unit

The Output Compare Unit toggles an output pin whenever the free running timer value matches a pre-set compare value. This allows to output square and PWM wave signals. It offers two main modes:

- Output square waves width same frequency, 50% duty cycle, and phase shift (3.2 Basic Functionality of the Output Compare Unit);
- Output base frequency with 50% duty cycle and additional PWM signals with twice the base frequency and selectable duty cycle (3.3 OCUs as PWMs).

It should be noted that the free running timer is started with every reset. Especially in the second mode, one must take care to enable free running timer and Output Compare Unit such that the polarity of the output signal is always same.

Control Status Registers of Output Compare (OCS)

This 16-Bit register contains several control and status bits.

Table 3. OCS

Bit No.	Name	Explanation	Value	Operation for Channel (2n+0) or (2n+1)
15, 12	CMOD1,CMOD0	Comparison Mode	*x, y*	*See table below*
14	–	*Undefined*	-	Always write 0
13	–	*Undefined*	-	Always write 0
11	OTE*(2n+1)*	OCU Output Enable	0	Output disabled
			1	Output enabled
10	OTE*(2n+0)*	OCU Output Enable	0	Output disabled
			1	Output enabled
9	OTD*(2n+1)*	Output Pin Level Select	0	Sets “0” for Compare Pin Output
			1	Sets “1” for Compare Pin Output
8	OTD*(2n+0)*	Output Pin Level Select	0	Sets “0” for Compare Pin Output
			1	Sets “1” for Compare Pin Output
7	ICP*(2n+1)*	Compare Match	0	No Compare Match
			1	Compare Match occurred
6	ICP*(2n+0)*	Compare Match	0	No Compare Match
			1	Compare Match occurred
5	ICE*(2n+1)*	Compare Interrupt Enable	0	Interrupt disabled
			1	Interrupt enabled
4	ICE*(2n+0)*	Compare Interrupt Enable	0	Interrupt disabled
			1	Interrupt enabled
3	–	*Undefined*	-	Always write 0
2	–	*Undefined*	-	Always write 0
1	CST*(2n+1)*	Comparison with Timer	0	Compare Operation disabled
			1	Compare Operation enabled
0	CST*(2n+0)*	Comparison with Timer	0	Compare Operation disabled
			1	Compare Operation enabled

CMOD Settings

Table 4. CMOD Settings

CMOD1	CMOD0	Functionality
0	0	The Output is toggled on every match with the corresponding Compare Register Value. Each Output is controlled by only one Compare Register. E.g. Output OUT0 is toggled on every match with the Compare Register OCCP0; Output OUT1 is toggled on every match with the Compare Register OCCP1, so on and so forth.
0	1	The EVEN Output (2n+0) is toggled on every Match with the corresponding Compare Register (2n+0) Value. The ODD Output (2n+1) is toggled on every Match with either the Compare Register (2n+0) Value or (2n+1) Value. E.g. Output OUT0 is toggled on every match with the Compare Register OCCP0; Output OUT1 is toggled on every match with the Compare Registers OCCP0 or OCCP1. Output OUT2 is toggled on every match with the Compare Register OCCP2, Output OUT3 is toggled on every match with the Compare Registers OCCP2 or OCCP3, so on and so forth.
1	0	This mode is only applicable for OCS3 and OCS7, hence for outputs OUT2, OUT3, OUT6 and OUT7. The EVEN Output *(2n+2)* is toggled on every Match with either the Compare Register *(2n+0)* Value or *(2n+2) Value*. The ODD Output *(2n+3)* is toggled on every Match with either the Compare Register *(2n+0)* Value or *(2n+3) Value*. E.g. Output OUT2 is toggled on every match with the Compare Registers OCCP0 or OCCP2, Output OUT3 is toggled on every match with the Compare Registers OCCP0 or OCCP3. Output OUT6 is toggled on every match with the Compare Registers OCCP4 or OCCP6; Output OUT7 is toggled on every match with the Compare Registers OCCP4 or OCCP7.
1	1	This mode is only applicable for OCS3 and OCS7, hence for outputs OUT2, OUT3, OUT6 and OUT7. The EVEN Output *(2n+2)* is toggled on every Match with either the Compare Register *(2n+0)* Value or *(2n+2) Value*. The ODD Output *(2n+3)* is toggled on every Match with either the Compare Register *(2n+0)* Value or *(2n+2) Value* or *(2n+3) Value*. E.g. Output OUT2 is toggled on every match with the Compare Registers OCCP0 or OCCP2, Output OUT3 is toggled on every match with the Compare Registers OCCP0 or OCCP2 or OCCP3. Output OUT6 is toggled on every match with the Compare Registers OCCP4 or OCCP6, Output OUT7 is toggled on every match with the Compare Registers OCCP4 or OCCP6 or OCCP7.

Output Compare Register (OCCP)

For each channel, there are two 16-bit wide Output Compare Registers. The 16-bit Output Compare registers are compared with the 16-bit Free-Running Timer. When the value of the register matches that of the 16-bit Free-Running Timer, a compare signal is generated and the output compare interrupt flag is set. If output is enabled, the output level corresponding to the compare register is reversed.

2.2.3 Input Capture Unit

The input capture unit stores the current counter value of free running timer when a defined input event occurs. This allows to measure pulse widths or frequencies.

Note: The corresponding input pin must be enabled using the appropriate Port Input Enable Register (PIER).

Input Capture Control Status Register (ICS)

This 8-Bit register contains several control and status bits.

Table 5. ICS

Bit No.	Name	Explanation	Value	Operation for Channel (2n+0) or (2n+1)
7	ICP*(2n+1)*	Interrupt Request Flag/Clear Bit	0	Read: No edge detected, Write: Clear Flag
			1	Edge detected
6	ICP*(2n+0)*	Interrupt Request Flag/Clear Bit	0	Read: No edge detected, Write: Clear Flag
			1	Edge detected
5	ICE*(2n+1)*	Interrupt Enable Bit	0	Interrupt disabled
			1	Interrupt enabled
4	ICE*(2n+0)*	Interrupt Enable Bit	0	Interrupt disabled
			1	Interrupt enabled
3, 2	EG*(2n+1)*1, EG*(2n+1)*0	Edge Selection Bit	0, 0	No Edge Detection
			0, 1	Rising Edge Detection
			1, 0	Falling Edge Detection
			1, 1	Both Edge Detection
1, 0	EG*(2n+0)*1, EG*(2n+0)*0	Edge Selection Bit	0, 0	No Edge Detection
			0, 1	Rising Edge Detection
			1, 0	Falling Edge Detection
			1, 1	Both Edge Detection

Input Capture Edge Register (ICE)

Table 6. ICE

Bit No.	Name	Explanation	Value	Operation for Channel (2n+0) or (2n+1)
15, 14, 13, 12, 11, 10	–	*Undefined* *(Some bits contain settings for LIN-UART. Please see Hardware Manual for Details)*	-	-
9	IEI*(2n+1)*	Edge Detection Indication	0	Falling Edge detected
			1	Rising Edge detected
8	IEI*(2n+0)*	Edge Detection Indication	0	Falling Edge detected
			1	Rising Edge detected

Input Capture Data Register (IPCP)

This 16-Bit register stores the timer value if an edge event occurs.

Input Capture Unit and LIN-USART

Some ICUs can be connected internally to LIN-USARTs. This feature is for measurement of baud rates for LIN-USART as LIN slave.

Hence while designing a LIN-slave application, ICUs for LIN-USART and ICUs for external events has to be identified exclusively.

3 I/O Timer Example

Example for I/O timer

3.1 Basic Functionality of the Free Running Timer

The following example shows how to set up the Free Running Timer.

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.         */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/

void InitFRTimer0(void)
{
    TCCS0 = 0x4048;           // no clock division, interrupt enabled, clear
        // timer
}

    . . .

__interrupt void FRTimer0(void)
{
    TCCSL0_IVF = 0;           // clear interrupt request

    . . .
}
```

At 16 MHz Peripheral Clock, the above example generates interrupts at an interval of 4.096 ms.

Please note, that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project.

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                              */
/*----------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
        . . .

        ICR = (72 << 8) | 6;          // Free Running Timer 0 of MB96340 Series

        . . .
}

__interrupt void FRTimer0 (void);            // prototype

    . . .

#pragma intvect FRTimer0          72  // FRT0 of MB96340 Series

    . . .
```

3.2 Basic Functionality of the Output Compare Unit

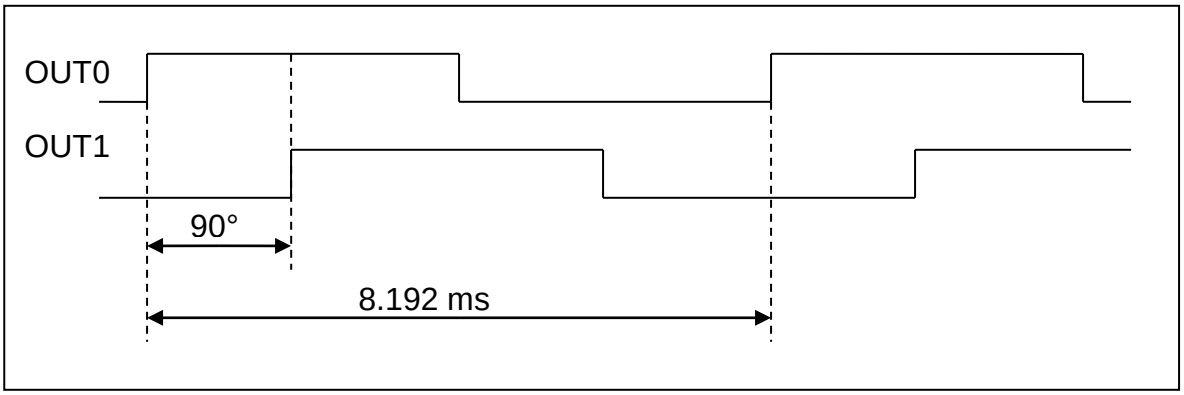
The following example shows how to set up the Output Compare. OCU0 and OCU1 are used with the Free Running 0. OCU1 is phase-shifted by 90° (Counter Value = 0x8000 = ½ • 0xFFFF + 1).

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                              */
/*----------------------------------------------------------------------------*/

void InitFRTimer0(void)
{
    TCCS0 = 0x4008;  // no clock division, interrupt disabled, clear
       // timer
}

void InitOCU01(void)
{
    OCS0 = 0x03;     // OCU0 and OCU1 enable
    OCS1 = 0x0C;     // Mode 00, Output 0 and 1 enable
    OCCP0 = 0x0000;  // Frequency = Free Running Timer
    OCCP1 = 0x8000;  // Phase = 90 deg.
}
```

The above configuration generates a PWM signal with 50% duty cycle. The cycle length depends solely on the CLKP1 frequency and the selected prescaler setting of the I/O Timer. Assuming that InitOCU01() is called right after InitFRTimer0(), there is no need to check for FRT counter being too close to compare value. Hence, it is not necessary to check the current timer value.

Figure 6 shows the output waveform:

Figure 6. Output Compare – Waveform



3.3 OCUs as PWMs

It is possible to generate 3 synchronous PWM signals with 4 OCU channels. The base frequency can be adjusted by the OCU0 channel, which restarts the Free Running Timer at match. Choosing Mode `CMOD [1:0] ="01"` for OCU0/1 and `CMOD [1:0] ="10"` for OCU2/3, OCU1/2/3 outputs are also reset at OCU0 match. Their outputs are set at their own matches.

The following code shows a possible setting:

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.         */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/
void InitFRTimer0(void)
{
// no clock division, interrupt disabled, reset by  // OCU0, clear timer, stop timer
// reset timer value just in case timer was already // stopped
}

void InitOCU0123(void)
{
    OCS0 = 0x03;       // OCU0 and OCU1 enable
    OCS1 = 0x1C;       // Mode 01, Output 0 and 1 enable
    OCS2 = 0x03;       // OCU2 and OCU3 enable
    OCS3 = 0x8C;       // Mode 10, Output 2 and 3 enable
    OCCP0 = 0xC000;    // Period 3.07 ms @ 16 MHz CLKP1
    OCCP1 = 0x4000;    // Match @ 1.02 ms
    OCCP2 = 0x8000;    // Match @ 2.04 ms
    OCCP3 = 0xA000;    // Match @ 2.56 ms
}

void StartFRT0(void)
{
    TCCS0_STOP = 0;    // start FRT again
}
```

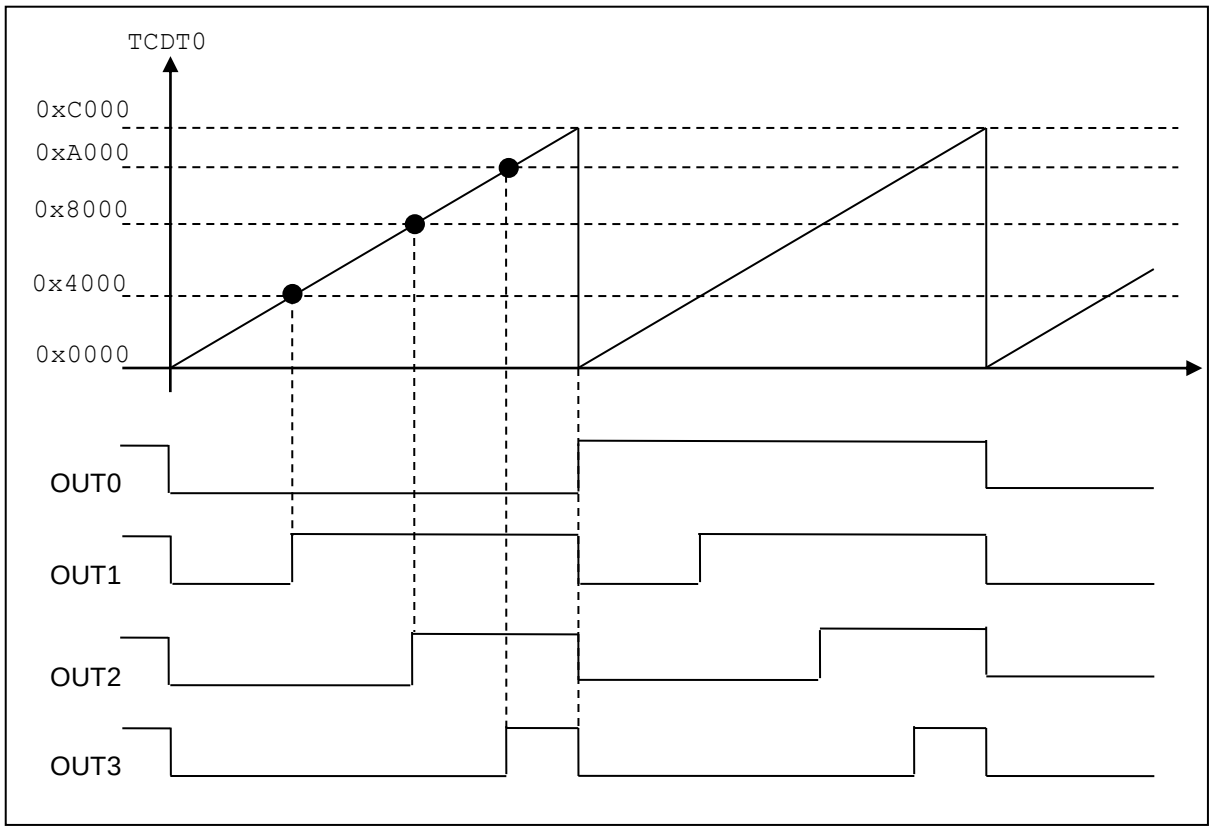
Function *InitFRTimer0()* disables free running timer 0 and resets the counter. This makes sure that all OCUs can be initialized without suffering from unintentional output toggles within the configuration. Note that the TCCSn_CLR bit only works while counter is running. Hence, it is always a good idea to clear the timer by accessing TCDT, just in case the timer was stopped already by some other routine.

Finally, the timer must be started again.

InitFRTimer0 () should be called first, followed by *InitOCU123 ()* and finally *StartFRT0 ().*

The following graphic illustrates the working and output waveforms:

Figure 7. Output Compare as PWM- Waveform



3.4 Basic Functionality of the Input Capture Unit

Assume you want to use Input Capture Unit 0 with both edge detection. The following code shows a solution. Please note, that for all Inputs the corresponding Port Input Enable Register has to be set first.

The main difference to External Interrupts is that the actual counter value of the Free Running Timer is stored in the Input Capture Data Register (IPCPn) at interrupt occurrence.

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/

void InitFRTimer0(void)
{
    TCCS0 = 0x4008;         // no clock division, interrupt disabled, clear
       // timer,
}

void InitICU0(void)
{
    PIER02_IE4 = 1;         // Enable ICU port input of MB9634x Series
       ICS01 = 0x13;              // Interrupt enable, both edge detection
}
    . . .

__interrupt void InterruptICU0(void)
{
    ICS01_ICP0 = 0;         // clear request
    if (ICE01_IEI0 == 1)    // rising edge?

    . . .           // IPCP0 can be read here for external
                // time/frequency measurement
}
```

Please also note that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project.

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.          */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                              */
/*----------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
       . . .

       ICR = (65 << 8) | 6;              // Input Capture Unit 0 of MB9634x
       // Series

       . . .
}

__interrupt void InterruptICU0 (void);  // prototype

    . . .

#pragma intvect InterruptICU0     65    // ICU0

    . . .
```

3.5 Frequency counter based on ICU and FRT

As the ICU captures the current value of the FRT it is easy to measure the time between two events and to calculate the resulting period respectively frequency:

Period[1] = ABS(ICU_new – ICU_old);

Frequency[1] = 1 / Period;

[1] The peripheral clock and prescaler have to be taken into account in order to calculate the right time values.

By same mechanism but different selections for the start and stop edges instead of frequency the pulse width can be calculated:

Time measurement	Start edge	Stop edge
Frequency	Rising edge	Rising edge
Frequency	Falling edge	Falling edge
Low pulse width	Falling edge	Rising edge
High pulse width	Rising edge	Falling edge

The following software extract is taken from the software example '96340_icu_frt_uart.zip':

```
/* THIS SAMPLE CODE IS PROVIDED AS IS AND IS SUBJECT TO ALTERATIONS.         */
/* MICROELECTRONICS ACCEPTS NO RESPONSIBILITY OR LIABILITY FOR ANY ERRORS OR */
/* ELIGIBILITY FOR ANY PURPOSES.                                             */
/*---------------------------------------------------------------------------*/
//-----------------------------------------------------------------------------
// FRT0 (IRQ)
//-----------------------------------------------------------------------------
__interrupt void FRT0_IRQ (void)
{
  FRT0_ovl_cnt++;                       // Count the overflows
  TCCSL0_IVF = 0;                       // Clear FRT Irq-flag
}
//-----------------------------------------------------------------------------
// ICU2 (IRQ)
//-----------------------------------------------------------------------------
__interrupt void ICU2_IRQ (void)
{
  ICU2_new = IPCP2;                     // Save current ICU value
  FRT0_ovl_cnt_new = FRT0_ovl_cnt;      // Save current FRT value
  if (TCCSL0_IVF && (IPCP2 < 0x8000))   // Check for FRT/ICU race condition
  {                                     // In case that FRT was not yet handled
    FRT0_ovl_cnt_new++;                 // then increase temp FRT counter
  }
  // Calculation of time period         // Define the ending edge:
  if (ICS23_EG2 == 1)                   // 1:Rising  2:Falling  3:Both Edges
  {
    value =   abs(FRT0_ovl_cnt_new - FRT0_ovl_cnt_old) * 0x10000UL
            + ICU2_new - ICU2_old;
    flag_value_valid = 1;               // Flag to be used by main application.
  }                                     // UART output here might be too long
  ICU2_old = ICU2_new;                  // Save current ICU value
  FRT0_ovl_cnt_old = FRT0_ovl_cnt_new; // Save current FRT value

  if (ICS23_EG2 == 1)                   // Define next edge
    ICS23_EG2  = 1;                     // 1:Rising  2:Falling  3:Both Edges
  else
    ICS23_EG2  = 1;

  ICS23_ICP2 = 0;                       // Clear ICU Irq-flag
}
```

The ICU captures a 16-bit value. That means a maximum signal length of 65536 clock ticks can be measured.

T_{max} = 65536 * ticks e.g.: T_{max} = 65536 * 1/16MHz = 4.096ms

Although the tick time can be influenced by changing the frequency setting of the peripheral clock CLKP1 (see `start.asm`) and by the IO-Timer Count Clock Selection `TCCSL_CLK[2:0]` the 16-bit range might not sufficient.

In order to allow the measurement of longer respectively slower signals also the overruns of the FRT has to be counted and must be included to the time calculation:

`Period`[1] = ABS(ICU_new−ICU_old) + 0x10000 * ABS(FRT_new[2] − FRT_old[2]);
[2] FRT_new and FRT_old are the number of FRT interrupts from the current and from last event.

It might happen that the input signal occurs just in that moment when the FRT overruns. Therefore please take care that for ICU and FRT the same interrupt level has to be defined by software. Nevertheless a race condition might occur within a dedicated time slot, while the FRT-overflow occurs internally and simultaneously the external signal is asserted. For this reason the FRT-overflow flag is also observed within the ICU Interrupt Service Routine (ISR). In case that the overflow flag was not handled yet, the overflow counter is increased temporarily. The global overflow counter then will be incremented by the FTR ISR after the ICU service is finished.

The captured ICU value and the FRT overrun counter value are saved and will be used as reference value while next calculation.

As already discussed the related edge detection has to be adjusted in order to measure a whole signal period or just a pulse.

The maximum signal length that can be measured by symbiosis of ICU and FRT depends on the type of overrun counter.

4 Additional Information

Information about CYPRESS Microcontrollers can be found on the following Internet page:

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

The software examples related to this application note is:

96340_icu_frt_uart (Simple frequency counter)

96340_frt_irq

96340_frt_icu_frt_ocu

96340_icu0

It can be found on the following Internet page:

http://www.cypress.com/products/16FX

5 Document History

Document Title: AN205497 - F²MC-16FX Family Series with I/O Timer

Document Number: 002-05497

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	04/27/2006	Initial release
			12/15/2006	Reviewed the document and updated with review findings
			02/21/2007	Updated with re-review findings
			08/21/2007	Added list of tables and figures, modified figures, corrected code
			07/07/2008	Add information on PIER; add information on how to start OCU correctly
			05/12/2009	Example of 'Frequency counter based on ICU/FRT added
*A	5066301	MKEA	12/28/2015	Migrated Spansion Application Note MCU-AN-300204-E-V15 to Cypress format
*B	5835216	AESATMP9	07/27/2017	Updated logo and copyright.
*C	6038727	NOFL	01/19/2018	Updated logo. Updated links. Updated Sales page and Copyright year.

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 MCU

Cypress Developer Community

Community | 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, 2006-2018. 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. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. 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.