AN205495

F²MC-16FX, All Series, Reload Timer

This application note describes the functionality of the Reload Timer and gives some examples. The Reload Timer is a down counter which reloads its 16-bit timer value again on counter underflow (if the reload functionality is enabled).

Contents

1 Introduction....1
1.1 Key Features....1
2 The Reload Timer....2
2.1 Block Diagram....2
2.2 Registers....3
2.3 Reload Timer Peripheral Clock 1....5
2.4 Operation Modes and Usage....6
2.5 Frequency Examples for TOTn Pin....13
3 Reload Timer Examples....14
3.1 Interval Timer with Output Function and without Interrupts....14
3.2 Interval Timer with Output Function and Interrupts....14
3.3 Timer with Trigger Input and Output Function....15
3.4 Timer with Gate Input....16
3.5 Event Counter....17
3.6 Cascading of RLT0 and RLT1....18
4 Additional Information....18
Document History....19

1 Introduction

This application note describes the functionality of the Reload Timer and gives some examples.

The Reload Timer is a down counter which reloads its 16-bit timer value again on counter underflow (if the reload functionality is enabled).

1.1 Key Features

- 16-bit Counter Value
- Output Frequency Range from 1.9 Hz to 4 MHz at 16 MHz Peripheral Clock
- Timer Interval Time from 125 ns to 263 ms at 16 MHz Peripheral Clock
- Pre-scalar Settings for Clock Divisions
- External clock and trigger selectable
- One-shot or Reload Counter Mode
- Interrupt Generation on Counter Underflow
- Timer Output
- Trigger Output for ADC from Reload Timer 1
- Reload Timer 6 underflow as clock source for PPG

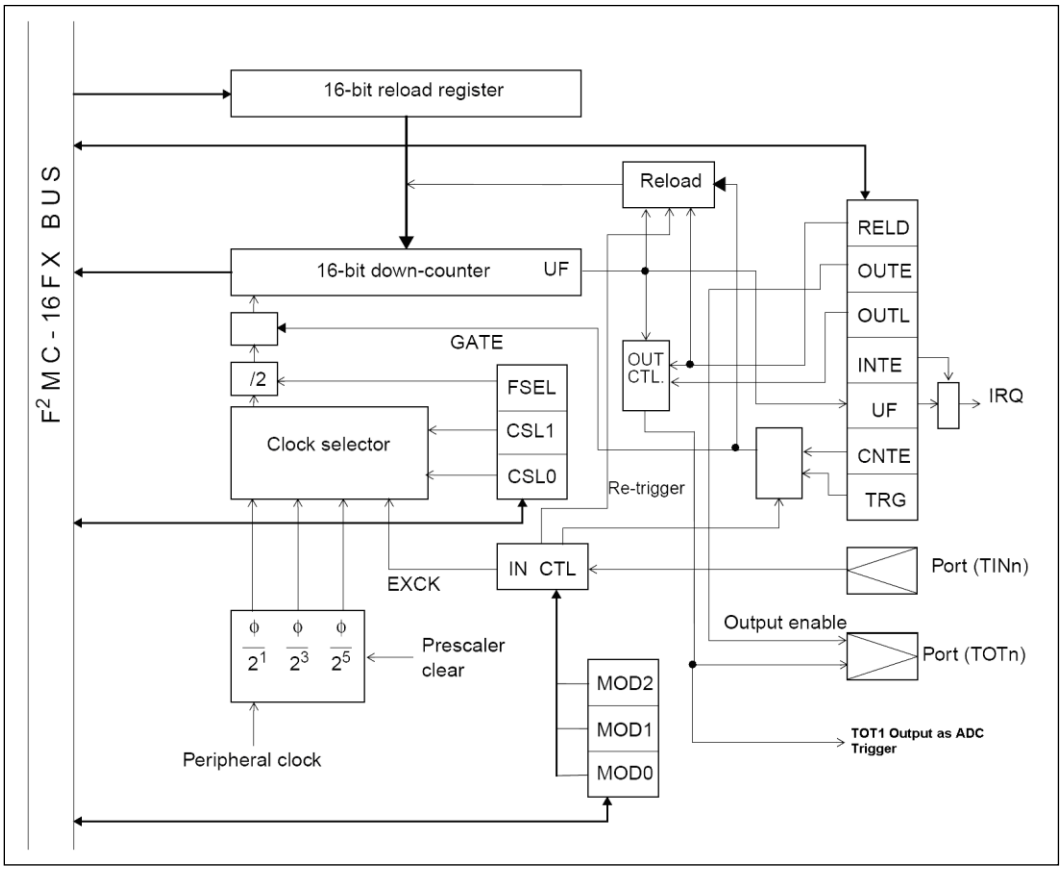
2 The Reload Timer

The basic functionality of the reload timer

2.1 Block Diagram

Figure 1 shows the internal block diagram of a Reload Timer channel.

Figure 1. Reload Timer block diagram



2.2 Registers

2.2.1 Timer Control Status Register (TMCSR)

This 16-Bit register controls the operation mode and interrupts for the reload timer.

Table 1. TMCSR

Bit No.	Name	Explanation	Value	Operation
15	-	*Undefined*	-	-
14	-	*Undefined*	-	-
13	-	*Undefined*	-	-
12, 11, 10	FSEL, CSL1, CSL0	Clock Division Control and Select	0, 0, 0	Clock = CLKP1 / 2^2
			0, 0, 1	Clock = CLKP1 / 2^4
			0, 1, 0	Clock = CLKP1 / 2^6
			0, 1, 1	External Event Count / 2
			1, 0, 0	Clock = CLKP1 / 2^1
			1, 0, 1	Clock = CLKP1 / 2^3
			1, 1, 0	Clock = CLKP1 / 2^5
			1, 1, 1	External Event Count
9, 8, 7	MOD2, MOD1, MOD0	Mode Setting	*See tables in the next page*	*See tables in the next page*
6	OUTE	Output Enable	0	Output Disabled
			1	Output Enabled
5	OUTL	Output Level	0	Normal Output
			1	Inverted Output
4	RELD	Reload	0	One-shot Mode
			1	Reload Mode
3	INTE	Interrupt Enable	0	Interrupt Disabled
			1	Interrupt Enabled
2	UF	Underflow	0	Read: No underflow Write: Clear Underflow
			1	Underflow occurred
1	CNTE	Count Enable	0	Counter Disabled
			1	Counter Enabled
0	TRG	Trigger	0	No Effect
			1	Write: Trigger Counter

Mode Settings:

CSL0/1 = “00”, “01”, “10”

Table 2. TMCSR – Mode Settings

MOD2	MOD1	MOD0	Input Pin Function	Active Edge or Level
0	0	0	Trigger Disabled	-
0	0	1	Trigger Input	Rising Edge
0	1	0		Falling Edge
0	1	1		Both Edges
1	x	0	Gate Input	“L” Level
1	x	1		“H” Level

CSL0/1 = “11”

Table 3. TMCSR – Mode Settings, CSL0/1 = “11”

MOD2	MOD1	MOD0	Input Pin Function	Active Edge or Level
x	0	0	-	-
	0	1	Event Input	Rising Edge
	1	0		Falling Edge
	1	1		Both Edges

2.2.2 16-Bit Reload Register (write) (TMRLR)

This 16-Bit register contains the reload value, which is set after counting underflow and reload function.

2.2.3 16-Bit Reload Register (read) (TMR)

This 16-Bit register contains the actual reload count.

2.2.4 Reload Timer Input Select Register (TMISR)

This 8-Bit register allows the user to create its own n * 16-bit-Reload Timer by cascading more than 1 reload timer.

Table 4. TMISR

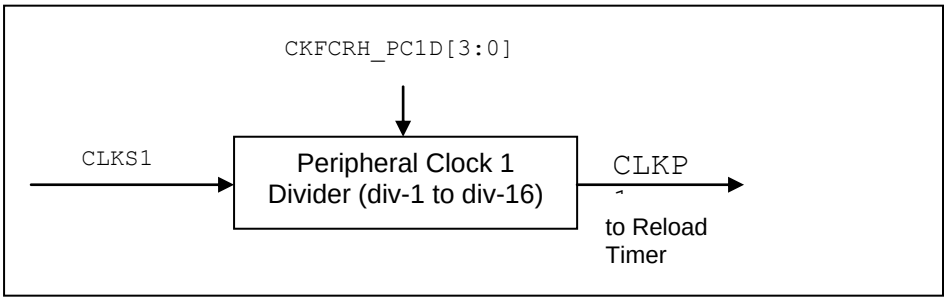
Bit No.	Name	Explanation	Value	Operation
7	-	*Undefined*	-	-
6	-	*Undefined*	-	-
5	TMIS5	Reload Timer 5 Input Select	0	Use TIN5 as trigger input
			1	Use Underflow event of Reload Timer 4 as trigger input
4	TMIS4	Reload Timer 4 Input Select	0	Use TIN4 as trigger input
			1	Use Underflow event of Reload Timer 3 as trigger input
3	TMIS3	Reload Timer 3 Input Select	0	Use TIN3 as trigger input
			1	Use Underflow event of Reload Timer 2 as trigger input
2	TMIS2	Reload Timer 2 Input Select	0	Use TIN2 as trigger input
			1	Use Underflow event of Reload Timer 1 as trigger input
1	TMIS1	Reload Timer 1 Input Select	0	Use TIN1 as trigger input
			1	Use Underflow event of Reload Timer 0 as trigger input
0	TMIS0	Reload Timer 0 Input Select	0	Use TIN0 as trigger input
			1	Use Underflow event of Reload Timer 5 as trigger input

The number of reload timers that can be cascaded is dependent on the total number of reload times available on the actual device.

2.3 Reload Timer Peripheral Clock 1

Please consider that the source clock frequency of the Reload Timer (CLKP1) depends on the settings of the Peripheral Clock 1 Divider.

Figure 2. Reload Timer Clock



2.4 Operation Modes and Usage

This section describes various modes in which the 16-Bit Reload Timer (RLT) can operate.

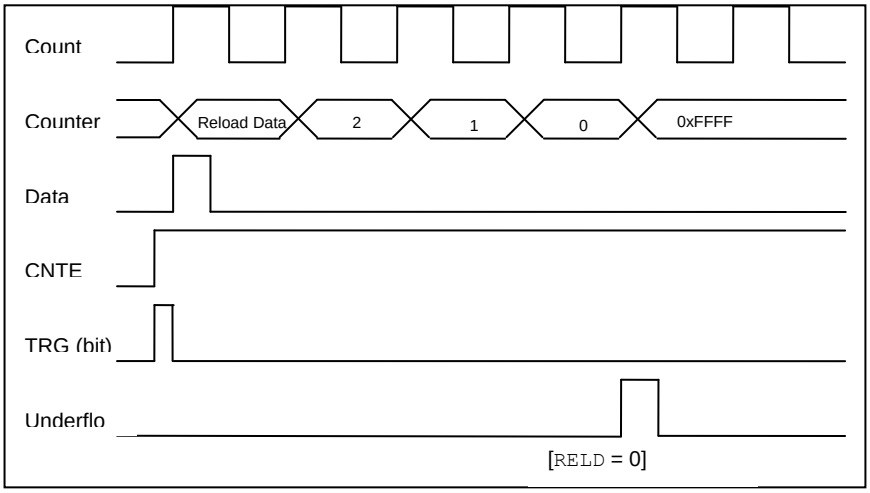
2.4.1 Interval Timer

The RLT can be used as an interval timer in one-shot mode or reload mode depending upon the setting of RELD bit of TMCSRn register. After the counter is enabled (TMCSRn:CNTE = 1) and triggered (TMCSRn:TRG = 1), the reload value is loaded from reload register to the counter and it starts counting. The counter decrements after every clock cycle of counter clock.

The counter decrements from 0 to 0xFFFF, if RELD bit of TMCSRn register is cleared to “0” or it decrements from 0 to the reload value, if RELD bit of TMCSRn register is set to “1”. In both the cases the underflow flag (TMCSRn:UF) gets set. If the INTE bit is "1" at this time, an interrupt request is generated.

The following figure describes the behavior of RLT in one-shot mode:

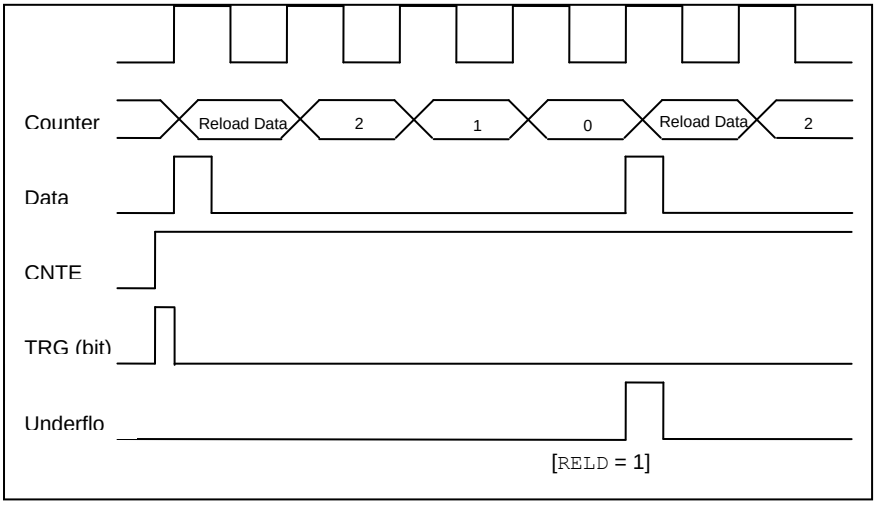
Figure 3. RLT as Interval Timer with RELD = 0



This mode can be used if an interrupt after specific interval is required. The corresponding software examples are discussed in sections 3.1 and 3.2.

The following figure describes the behavior of RLT in reload mode:

Figure 4. RLT as Interval Timer with RELD = 1

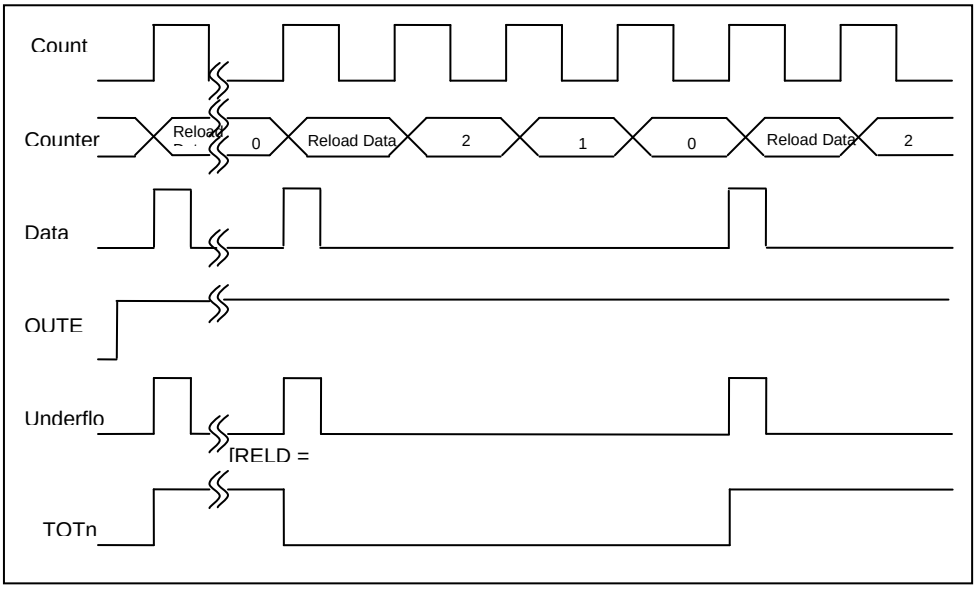


This mode can be used if periodic interrupts after specific interval is required.

2.4.2 Interval Timer with Output Function

The interval timer described in section 2.4.1 can also be used with TOTn output. TOTn output toggles after every underflow. Hence the frequency of this output equals half the frequency of the underflow interrupt.

Figure 5. RLT with TOTn Output in Reload Mode



The polarity of the TOTn output depends on OUTL bit of TMCSRn register. The following table describes the behavior:

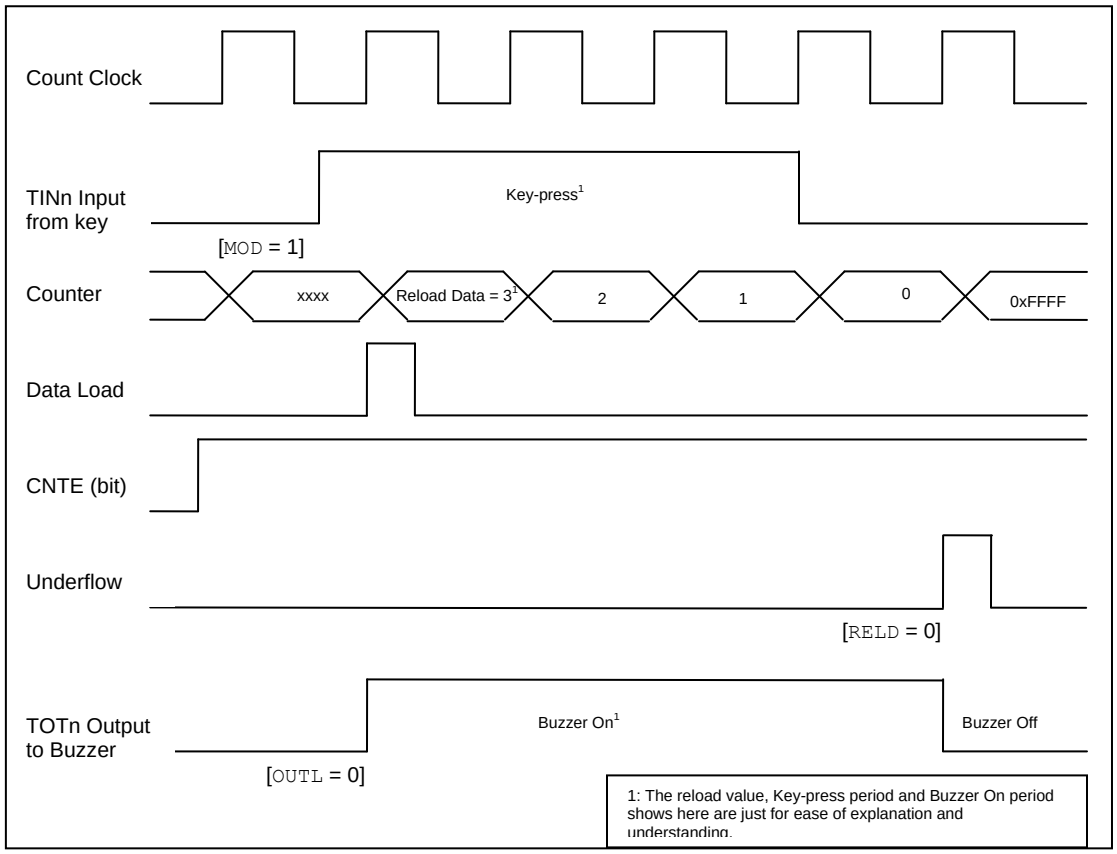
Sr. No.	OUTL	Condition	TOTn Output Status	
			One-shot Mode [RELD = 0]	Reload Mode [RELD = 1]
1.	0	Counter Enabled and triggered	1	0
2.		After First Underflow	0	1
3.		After Second Underflow	Not Applicable	0
4.	1	Counter Enabled and triggered	0	1
5.		After First Underflow	1	0
6.		After Second Underflow	Not Applicable	1

2.4.3 Timer with Trigger Input and Output Function

This mode operates exactly same as modes described in sections 2.4.1 and 2.4.2 except for the fact that the RLT is triggered by the signal on TINn input pin. In this case it is considered that trigger input edge is “Rising”. After the rising edge on TINn pin the reload value (3) is loaded in the counter and the counter starts counting.

This mode may be used for generating “Key-stroke Beeps”. If an application requires that each key-stroke / key-press on a key matrix is required to be acknowledged by a beep of the buzzer, then the key input can be fed to the TINn input (debounce of the key needs to be taken care separately) and the TOTn output can be fed to buzzer (amplifier circuitry may be required to drive the buzzer).

Figure 6. RLT with TINn as Trigger Input



Here the RLT is configured in trigger input mode with rising edge (TMCSRn:MOD[2:0] = 1). Hence if the key is pressed, positive edge would appear at the TINn input of the reload timer. This would load the reload value into the counter and the TOTn output is set to high (since the OUTL bit of TMCSRn register is configured as “0”) and in turn the buzzer would be turned on.

The reload value is dependent on the time of the beep. Once the required delay is expired the underflow flag would be set and the TOTn output would be cleared and hence the buzzer would stop beeping.

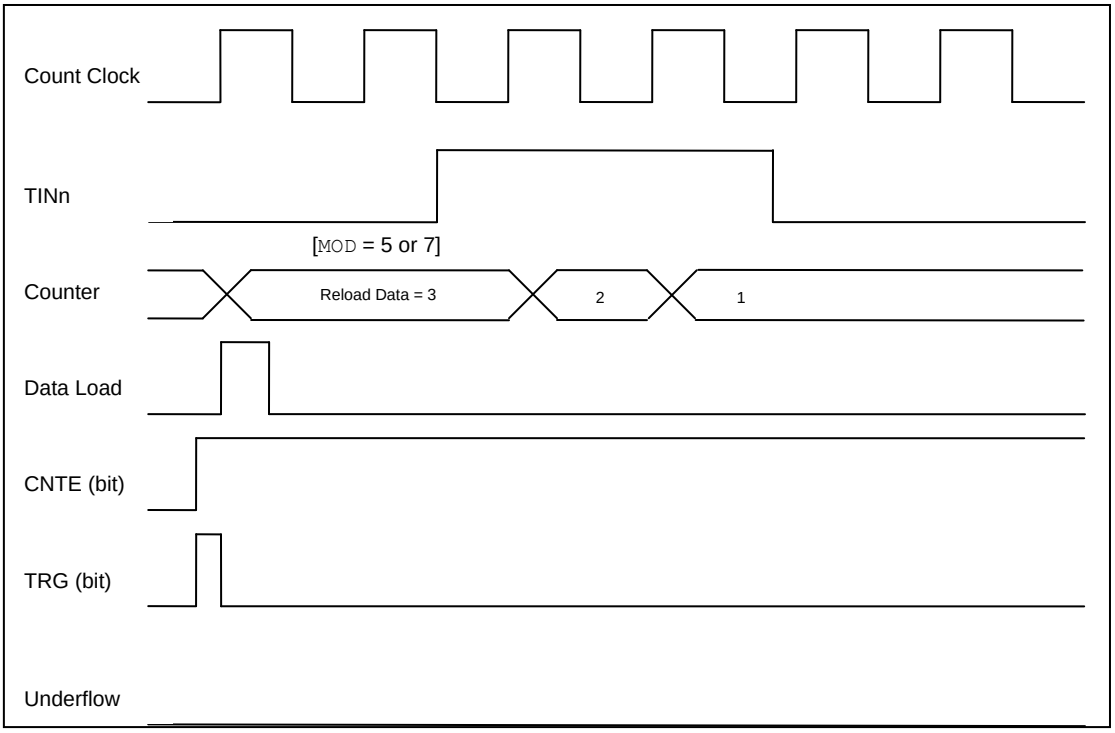
The corresponding software example is discussed in section 3.3.

Note: In order to use the TINn pin as an input, the corresponding port input must be enabled by setting the appropriate IE bit in the required PIERx register.

2.4.4 Timer with Gate Input

In this mode, TINn input is used a gate input and the counter only counts when the TINn input is at the level specified by MOD0 bit of TMCSRn register. The underflow and the reload operation are exactly same as discussed for the preceding modes above.

Figure 7. RLT with TINn as Gate Input



This mode can be used for the pulse width detection.

The software example which uses RLT in gate input mode is discussed in section 3.4.

Note: In order to use the TINn pin as an input, the corresponding port input must be enabled by setting the appropriate IE bit in the required PIERx register.

2.4.5 Event Counter

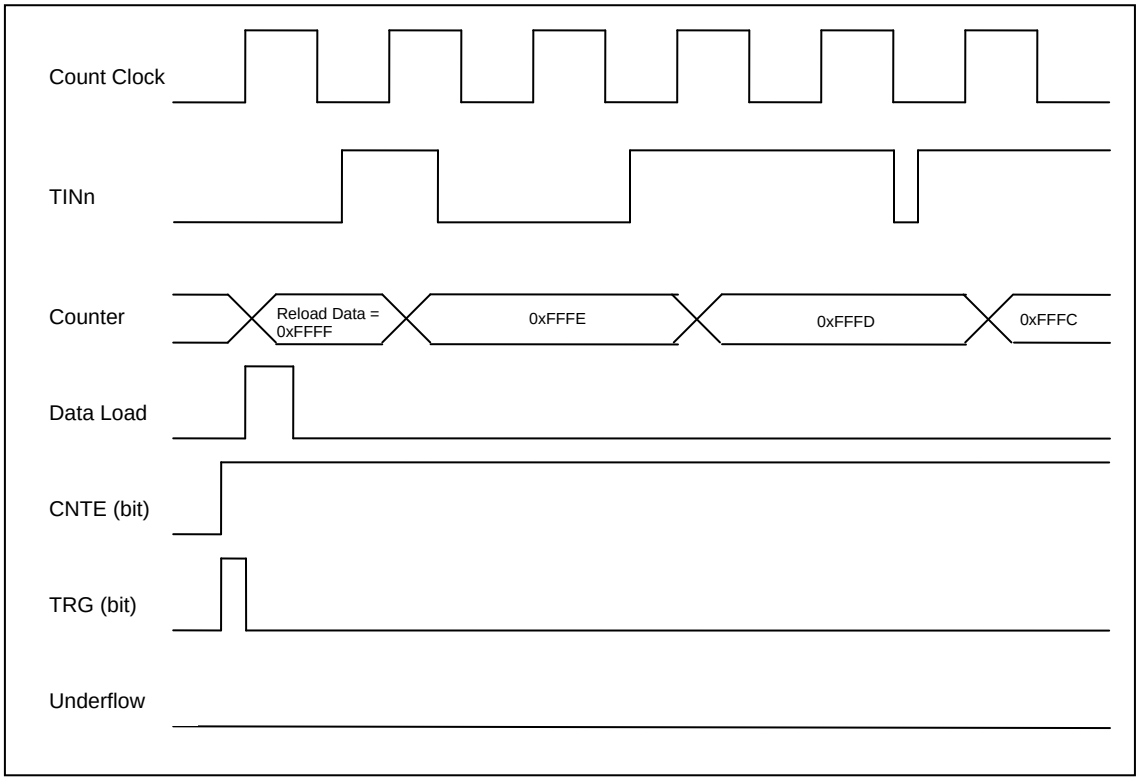
In this mode, TINn input is used as an external event input. The reload value is loaded in to the counter after the counter is enabled (TMCSRn:CNTE = 1) and triggered (TMCSRn:TRG = 1). However, the counter starts decrementing only when the active edge appears at TINn input pin. This edge is configured by `MOD[1:0]` bits of TMCSRn register.

The pulse width of the input signal should be at least 4T (T: 1 cycle of peripheral clock CLKP1) to the TINn pin.

This mode may be used for “Frequency Measurement”. The frequency of the signal is essentially number of cycles of the signal in the period of 1 second. Here two reload timers are used: one for generating 1 second delay (RLT0) and the other for counting the number of edges in this 1 second delay (RLT1).

The below diagram shows the operation of RLT1 only, which essentially counts the number of edges in the period of 1 second.

Figure 8. RLT with TINn as External Event Input



The RLT0 and RLT1 are triggered simultaneously. RLT0 is configured to generate a delay of 1 second. It is used as an interval timer in one-shot mode with interrupt. RLT1 is configured as event counter counting rising edges. RLT0 starts counting as soon as it is enabled and triggered. However RLT1 counter would decrement only after a rising edge appears at TINn input pin. The number of edges would be equal to the number of decrements of RLT1 counter.

After 1 second delay RLT0 underflow flag is set and interrupt is generated. In the RLT0 interrupt service routine, the both RLT0 and RLT1 are disabled. The frequency of the signal would be calculated as 0xFFFF – value of TMR1. It should be noted that the maximum measurable frequency here is 65535 Hz.

The corresponding software example is discussed in section 3.5.

Note: In order to use the TINn pin as an input, the corresponding port input must be enabled by setting the appropriate IE bit in the required PIERx register.

2.4.6 Cascading of Reload Timers

Cascading two or mode reload timers is required if a bigger time delay is required to be generated out of a relatively faster peripheral clock 1 (CLKP1).

- The first timer in the chain can operate with an internal clock or an external event as counter clock.
- If configured, the underflow signal of the first timer would act as a trigger input for the second timer in the chain. After every underflow of the first timer, the second timer would decrement by 1. The second timer would underflow depending upon its reload value.
- If configured, the underflow signal of the second timer would act as a trigger input for the third timer in the chain. After every underflow of second timer, the third timer would decrement by 1, so on and so forth.

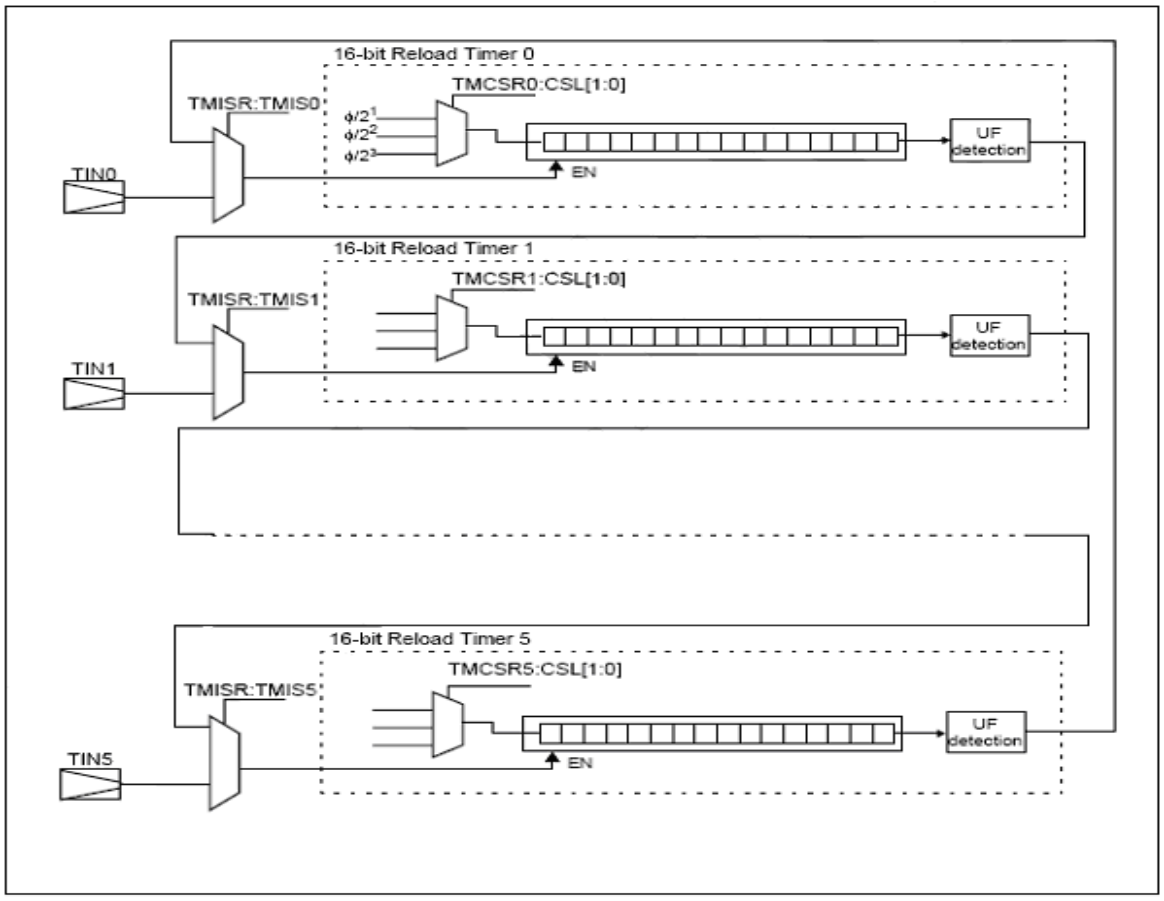
All the timers except first timer in the chain must be set as follows:

Configuration	Meaning
TMCSRn:MOD[2:0] = "101"	Gate count mode "H" level
TMCSRn:CSL[1:0] = "00" and TMCSRn:FSEL = "1"	No clock division
TMISR:TMISx = "1"	Use underflow signal of preceding timer as trigger input

It should also be noted that, all the reload timers except first timer in the chain must be enabled and triggered before the first reload timer starts counting.

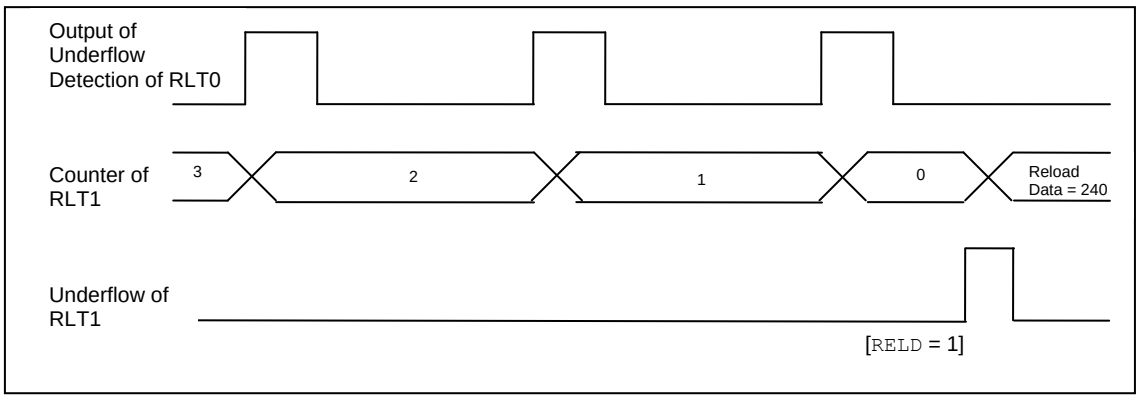
The below figure shows the cascading of reload timers

Figure 9. Cascading of RLTs



In the below figure it is considered that the RLT0 and RLT1 are cascaded and both are used in reload counter. RLT0 is operating with CLKP1 and the underflow signal of the RLT0 act as a trigger input for RLT1. After every underflow of RLT0, RLT1 would decrement by 1.

Figure 10. Cascading of RLT0 and RLT1



The corresponding software example is discussed in section 3.6.

2.5 Frequency Examples for TOTn Pin

The following table shows some frequency settings for the Reload Timer clocked by CLKP1.

CLKP1	FSEL, CSL1, 0 Division	Reload Value TMRLR	Period Time	Period Frequency
16 MHz	2	0xFFFF	16.38 ms	61.1 Hz
		0x0000	250 ns	4 MHz
	4	0xFFFF	32.77 ms	30.5 Hz
		0x0000	500 ns	2 MHz
	8	0xFFFF	65.54 ms	15.3 Hz
		0x0000	1 µs	1 MHz
	16	0xFFFF	131.1 ms	7.63 Hz
		0x0000	2 µs	500 KHz
	32	0xFFFF	262.1 ms	3.81 Hz
		0x0000	4 µs	250 KHz
	64	0xFFFF	524.3 ms	1.91 Hz
		0x0000	8 µs	125 KHz

The formula for the Reload Timer frequency f_{RLT} using f_{CLKP1} (`CLKP1`) as clock source is:

$$f_{RLT} = \frac{f_{CLKP1}}{2 \cdot div_{FSEL} \cdot (R+1)}$$, where div_{FSEL} is the `FSEL, CSL1, 0` division factor and R the Period Value

Important Note:

The frequencies and period times above are related to the Reload Timer *output* pins. If you use interrupts, the period time is as half as long and the frequency doubles.

3 Reload Timer Examples

Examples for reload timer operation

3.1 Interval Timer with Output Function and without Interrupts

The following example shows how to set up the Reload Timer 0 for operation without interrupts.

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

void InitReloadTimer0(void)
{
    TMRLR0 = 0x2000;           // set reload value
    TMCSR0 = 0x1053;           // pre-scalar 1:2
}
```

The example outputs a frequency of about 488 Hz at the TOTn pin using 16 MHz Peripheral Clock (CLKP1).

3.2 Interval Timer with Output Function and Interrupts

Setting the `INTE`-Bit enables interrupts for the Reload Timer. An interrupt occurs, if an underflow of the timer is detected. The interrupt is cleared by setting the `UF`-Bit to “0”.

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

void InitReloadTimer0(void)
{
    TMRLR0 = 0x2000;           // set reload value
    TMCSR0 = 0x105B;           // pre-scalar 1:2, interrupt enable
}

__interrupt void ReloadTimer0 (void)
{
    TMCSR0_UF = 0;             // reset underflow interrupt request flag
}
```

The example outputs a frequency of about 488 Hz at the TOTn pin using 16 MHz Peripheral Clock (CLKP1). Therefore interrupts are generated in periodic interval of 1.024 ms.

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

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

void InitIrqLevels(void)
{
  ICR = ((51 & 0xFF) << 8) | 6;     // Reload Timer 0 of MB9634x Series
}
    . . .
__interrupt void ReloadTimer0(void);// Prototype

    . . .

#pragma intvect ReloadTimer0      51// RLT0 of MB9634x Series
```

3.3 Timer with Trigger Input and Output Function

The following software example demonstrates to configure the RLT0 on MB96340 Series with Trigger Input and Output Function in order to generate the “Key-stroke Beeps” as explained in the section 2.4.3. After every key press the buzzer would beep for 100 ms.

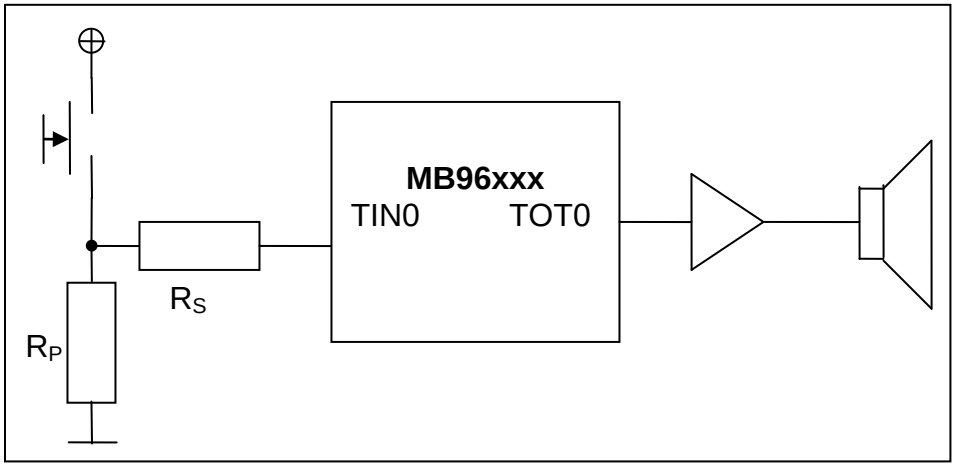
The CLKP1 is considered as 4 MHz. After the divider of 64 the RLT0 clock would become 62.5 kHz. The reload counter of RLT0 is set to a value equal to `0x1869`. Hence the period for the TOTn output would stay high after the rising edge at TIN0 for 100 ms (1/62.5 kHz * (`0x1689` + 1)). Here RLT0 is configured in one-shot mode and TOT0 output would stay high while the count is in progress (i.e. till underflow). After the underflow the output would be cleared and the buzzer stops beeping.

```
/*                    SAMPLE CODE                                              */
/*-------------------------------------------------------------------------------*/
  void InitReloadTimer0(void)
{
    TMRLR0 = 0x1869;     // set reload value = 6249
    PIER08_IE0 = 1;      // enable TIN0 input pin
    TMCSR0 = 0x083C;     // pre-scalar 1:64, trigger input-rising edge, output
                         // enable, count enable, trigger
       }
         void main(void)
{
  InitReloadTimer0();

  while(1);
}
```

The below figure depicts the connection diagram corresponding to the above code:

Figure 11. Connection Diagram for Keystroke Beeps



In the above figure RP is the pull-down resistor to limit the current once the key is pressed where as RS is the series resistor to limit the input current and also to filter the noise. The TOTn output can be fed to the buzzer via an amplifier as shown above.

3.4 Timer with Gate Input

The following software example demonstrates to configure the RLT0 on MB96340 Series with gate input of low level on TINn pin. This gate also works even if the MCU is in sleep mode. This functionality can also be used for power saving modes which do not affect gated timer outputs.

The signal appearing at TIN0 pin is at high level when the key is not pressed. Once the key is pressed a low pulse would appear at TIN0 pin. This would start decrementing the counter and it would continue decrementing as long as the key is pressed. If the key is continued to be pressed till 0x2000 counter clock cycles then the counter would underflow and TOT0 output would become high. If the key continued to be pressed long enough, the TOT0 output keeps on toggling after every 0x2000 counter clock cycles.

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

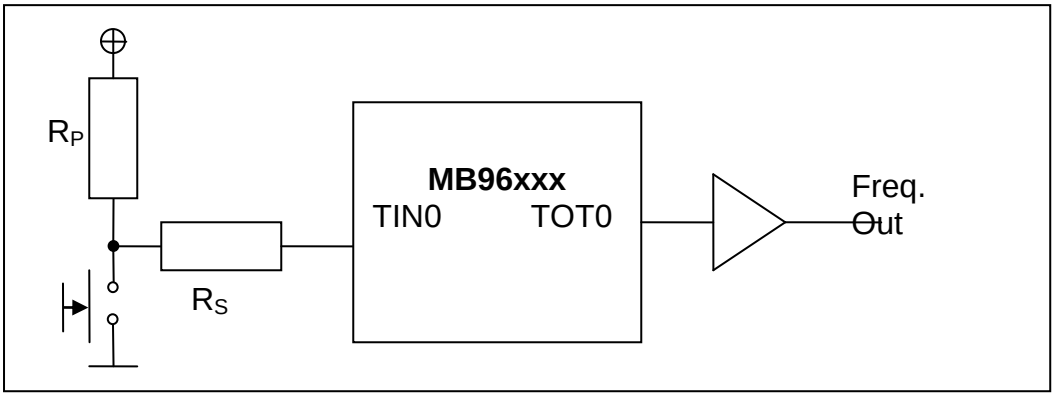
void InitReloadTimer0(void)
{
    TMRLR0 = 0x2000;  // set reload value = 8192
    PIER08_IE0 = 1;   // enable TIN0 input pin
    TMCSR0 = 0x1253;  // pre-scalar 1:2, reload, output enable, gated "L" level
}

void main(void)
{
 InitReloadTimer0();

 SMCR = 0x01;         // Sleep Mode Request
 while(1);
}
```

The below figure depicts the connection diagram corresponding to the above code:

Figure 12. Connection Diagram for RLT with Gate Input



In the above figure R_P is the pull-up resistor to limit the current once the key is pressed where as R_S is the series resistor to limit the input current and also to filter the noise. The TOT0 output can be fed via an amplifier as shown above. This may be used to manage the turn indicator lights in a vehicle.

3.5 Event Counter

The following software example demonstrates to configure the RLT1 as an event counter and RLT0 as an interval timer on MB96340 Series in order to “measure frequency” as explained in the section 2.4.5.

TIN1 is used as external event input. The counter decrements at every edge of the signal appearing on TIN1 input (of which the frequency is to be measured).

The CLKP1 is considered as 4 MHz. After the divider of 64, the RLT0 clock would become 62.5 kHz. The reload counter of RLT0 is set to a value equal to 0xF423. Hence the underflow interrupt of RLT0 would occur after 1 second (1/62.5 kHz * (0xF423 + 1)). By that time the counter of RLT1 would have decremented by the number of edges at TIN1 input. In the RLT0 interrupt service routine the frequency of the signal is calculated as 0xFFFF – TMR1.

```
/*                            SAMPLE CODE                                    */
/*---------------------------------------------------------------------------*/
void InitReloadTimer0_1(void)
{
    TMRLR0 = 0xF423;    // set reload value = 62499
    TMRLR1 = 0xFFFF;    // set event counter value

    PIER01_IE0 = 1;     // enable TIN1 input pin

    TMCSR0 = 0x080B;    // prescaler 1:64, count enable, interrupt enable,
                        // trigger
    TMCSR1 = 0x1C83;    // External count mode, event input, rising edge,
                        // count enable, trigger
}
unsigned int Frequency;

void main(void)
{
   InitReloadTimer0_1();

   while (1);
}
__interrupt void ReloadTimer0 (void)
{
    TMCSR0_UF = 0;             // reset underflow interrupt request flag
    Frequency = 0xFFFF - TMR1;// calculate frequency
    TMCSR0_INTE = 0;          // disable interrupts
}
```

Also note that the pulse width of signal appearing on TIN1 should be at least 4 times the CLKP1 (i.e. 1 µs at 4 MHz CLKP1).

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

```
/*                           SAMPLE CODE                                   */
/*-------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
  ICR = ((51 & 0xFF) << 8) | 6;     // Reload Timer 0 of MB9634x Series
}
 . . .

__interrupt void ReloadTimer0(void);// Prototype
 . . .
#pragma intvect ReloadTimer0      51// RLT0 of MB9634x Series
```

3.6 Cascading of RLT0 and RLT1

The following software example demonstrates to cascade RLT0 and RLT1 on MB96340 Series in order to generate a delay of 60 seconds as explained in the section 2.4.6 with the CLKP1 of 16 MHz.

Here both RLT0 and RLT1 are used in interval timer mode. The CLKP1 is considered as 16 MHz. After the divider of 64, the RLT0 clock would become 250 kHz. The reload counter of RLT0 is set to a value equal to `0xF423`. Hence the underflow of RLT0 would occur after 250 ms (1/250 kHz * (`0xF423` + 1)). After every underflow of RLT0, RLT1 counter would decrement by 1. Since reload counter of RLT0 is set to a value equal to `0x00EF`, the underflow of RLT would happen after 60 seconds (250 ms * (`0x00EF` + 1)). Since the RLT1 interrupt is enabled, ISR for RLT1 would be executed after every 60 seconds.

```
/*                              SAMPLE CODE                                   */
/*----------------------------------------------------------------------------*/
void CascadeRLT0_1(void)
{
    TMRLR0 = 0xF423;    // set reload value = 62499
    TMRLR1 = 0x00EF;    // set reload value = 239
    TMISR = 0x02;       // use underflow event of RLT0 as gate input
    TMCSR0 = 0x0813;    // pre-scalar 1:64, reload, count enable, trigger
    TMCSR1 = 0x129B;    // no clock division, gate input – high level, reload,
                        // interrupt enable, count enable, trigger
}
__interrupt void ReloadTimer1 (void)
{
    TMCSR1_UF = 0;      // reset underflow interrupt request flag
}
```

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

```
/*                              SAMPLE CODE                                   */
/*----------------------------------------------------------------------------*/
void InitIrqLevels(void)
{
  ICR = ((52 & 0xFF) << 8) | 6;       // Reload Timer 1 of MB9634x Series
}
 . . .
__interrupt void ReloadTimer1(void);  // Prototype
 . . .
#pragma intvect ReloadTimer1      52 // RLT1 of MB9634x Series
```

4 Additional Information

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

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

The software examples related to this application note is:

96340_rlt_irq

96340_adc_rlt

96340_ppg_rlt_adc_dma

Document History

Document Title: AN205495 - F²MC-16FX, All Series, Reload Timer

Document Number:002-05495

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	04/25/2006	First Version; MWi
			12/12/2006	V1.1, Reviewed the document and updated with review findings, MPi
			02/21/2007	V1.2, Updated with re-review findings, MPi
			08/01/2007	V1.3, Add more examples; add cascaded timer, MPi, PHu
*A	5076469	MKEA	05/18/2016	Converted Spansion Application Note “MCU-AN-300202-E-V13” to Cypress format
*B	5870122	AESATP12	09/01/2017	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, 2006-2017. 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.