AN205545

F²MC-16FX Family, Real Time Clock

This application note describes the functionality of the Real Time Clock and gives some examples.

Contents

1 Introduction

This application note describes the functionality of the Real Time Clock and gives some examples.

1.1 Key Features

- Clock selectable as Main Clock, Sub Clock and RC Clock
- RTC is not deactivated during Timer Mode
- Interrupts selectable for: ½ second, 1 second, 1 minute, 1 hour, and 1 day

2 The Real Time Clock

The Basic Functionality of the Real Time Clock

2.1 Block Diagrams

Figure 1 shows the internal block diagram of the Real Time Clock.

Figure 1. Real Time Clock Block Diagram

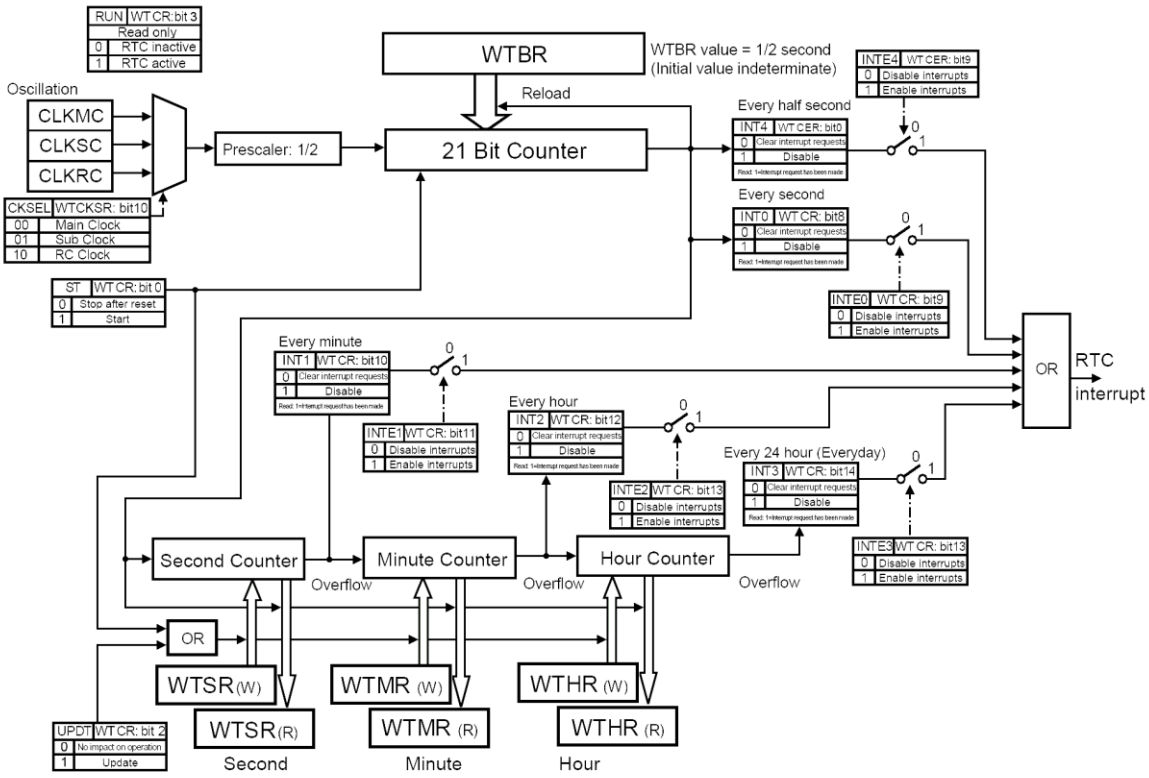
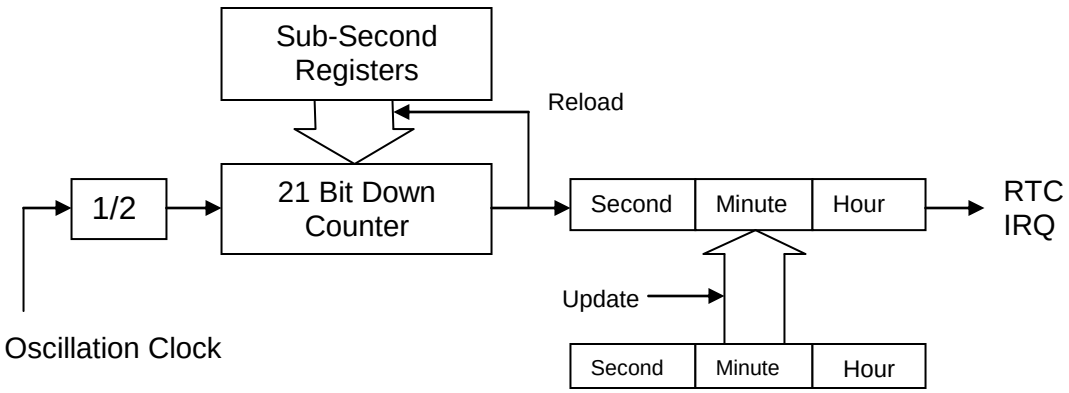


Figure 2 shows the simplified block diagram of the Real Time Clock.

Figure 2. Simplified Real Time Clock Block Diagram



2.2 Registers

Please write always “0” to undefined bits when accessing a register via word or byte, if not stated otherwise.

2.2.1 Timer Control Register (WTCR)

Table 1. WTCR

Bit No.	Name	Explanation	Value	Operation
15	INTE3	Interrupt Request each 24 Hour	0	Interrupt disabled
			1	Interrupt, if 24 Hour (1-Day)Counter overflow
14	INT3	24-Hour Interrupt Flag/Clear	0	Write: Clear Request
			1	Interrupt Request
13	INTE2	Interrupt Request each Hour	0	Interrupt disabled
			1	Interrupt, if 1 Hour Counter overflow
12	INT2	Hour Interrupt Flag/Clear	0	Write: Clear Request
			1	Interrupt Request
11	INTE1	Interrupt Request each Minute	0	Interrupt disabled
			1	Interrupt, if Minute Counter overflow
10	INT1	Minute Interrupt Flag/Clear	0	Write: Clear Request
			1	Interrupt Request
9	INTE0	Interrupt Request each Second	0	Interrupt disabled
			1	Interrupt, if Second Counter overflow
8	INT0	Second Interrupt Flag/Clear	0	Write: Clear Request
			1	Interrupt Request
7	–	*Undefined*	-	-
6	–	*Undefined*	-	-
5	–	*Undefined*	-	-
4	–	*Undefined*	-	-
3	RUN	Operation Status	0	RTC inactive
			1	RTC is active
2	UPDT	Update Counter with written Values into WTSR, WTHR, and WTMR	0	Write: No effect
			1	Write: Update
1	OE	Output Enable of WOT Pin	0	WOT Pin is general Purpose I/O Pin
			1	WOT is Output for Watch Timer (21-bit down counter)
0	ST	Start[1]	0	Clear Clock and Stop it
			1	Load Values and Start clock

[1] It is recommended to set the ST-Bit to “0”, when changing the clock time of the RTC.

2.2.2 Timer Control Extended Register (WTCER)

Table 2. WTCER

Bit No.	Name	Explanation	Value	Operation
7	–	Undefined	-	-
6	–	Undefined	-	-
5	–	Undefined	-	-
4	–	Undefined	-	-
3	–	Undefined	-	-
2	–	Undefined	-	-
1	INTE4	Enable Interrupt Request each half Second	0	Interrupt disabled
			1	Interrupt, if Half-Second Counter overflow
0	INT4	Half-Second Interrupt Flag/Clear	0	Write: Clear Request
			1	Interrupt Request

2.2.3 Clock Select Register (WTCKSR)

Table 3. WTCKSR

Bit No.	Name	Explanation	Value	Operation
7	–	Undefined	-	-
6	–	Undefined	-	-
5	–	Undefined	-	-
4	–	Undefined	-	-
3	–	Undefined	-	-
2	–	Undefined	-	-
1, 0	CKSEL1,0	Clock Select	0, 0	Main Clock (CLKMC)
			0, 1	Sub Clock (CLKSC)
			1, 0	RC Clock (CLKRC)
			1, 1	*prohibited*

2.2.4 Sub-Second Register (WTBR)

This register contains the 21-Bit reload value, which divides the clock source. This value should be *clock-frequency /4* (in order to get half a second interrupt).

The lower 16 Bits of this value is stored in WTBR0 and the remaining upper 5 Bits of the value in the lower 5 Bits of WTBR1.

The following table shows example values for different clock sources.

Table 4. Clock Source and WTBR values

Clock Source	WTBR decimal	WTBR hexadecimal
Main Clock 4 MHz	1000000	0x0F4240
RC Clock 100 kHz	25000	0x0061A8
Sub Clock 32.768 kHz	8192	0x002000

2.2.5 Second/Minute/Hour Registers (WTSR, WTMR, WTHR)

The lower 6 Bits of the WTSR contains the actual second counter value. Writing a value to it memorizes the value. The Second-Counter is updated with this value by writing “1” to WTCR_UPDT.

The lower 6 Bits of the WTMR contains the actual minute counter value. Writing and reading has the same behaviour like for WTSR.

The lower 5 Bits of the WTHR contains the actual hour counter value. Writing and reading has the same behaviour like for WTSR.

Please store only reasonable values to these registers. If values, that do not present a clock time, are used, the behaviour of the RTC will be undefined.

Note: Only byte access is allowed to these registers.

3 Real Time Clock Examples

Examples for the Real Time Clock

3.1 RTC with main clock source and without interrupts

```
#define DividerMC 1000000

void InitRTCAfterReset (void)
{
    WTBR0 = (0xFFFF & DividerMC);  // Set Sub-Second Prescaler
    WTBR1 = (DividerMC >> 16);

    WTCR_INTE0 = 0;         // No Interrupts
    WTCR_INTE1 = 0;
    WTCR_INTE2 = 0;
    WTCR_INTE3 = 0;
    WTCER_INTE4 = 0;
    WTCR_OE = 0;            // No Output
    WTCKSR = 0;             // Main Clock Source

    WTSR = 56;              // Seconds: 56
    WTMR = 34;              // Minutes: 34
    WTHR = 12;              // Hours:   12

    WTCR_ST = 1;            // ... and go!
}

void main(void)
{
    . . .

    InitRTCAfterReset ();  // Init and start the RTC

    . . .
}
```

The above sample code example demonstrates how to initialize RTC immediately after a Reset. Here it is considered that the Main Clock is 4 MHz.

3.2 Re-Initialize the RTC with Sub clock

```
#define DividerSC 8192

void InitRTC (void)
{
    WTBR0 = (0xFFFF & DividerSC); // Set Sub-Second Prescaler
    WTBR1 = (DividerSC >> 16);

    WTSR = 28;                    // Seconds: 28
    WTMR = 59;                    // Minutes: 59
    WTHR = 18;                    // Hours:   18

    WTCR_ST = 0;                  // Stop the RTC
    while (WTCR_RUN != 0);        // Wait till the RTC stops

    WTCKSR = 1;                  // Sub Clock Source
    WTCR_ST = 1;                 // ... and go!
}
```

The above sample code examples demonstrates how to initialize the Sub-Second, Second, Minute & Hour registers if the RTC is already running. Here it is considered that the Sub Clock is 32.768 kHz.

3.3 Initialize the Sub-Second Register in ISR

```
#define DividerRC 1000000/2

__interrupt void RealTimeClock (void)
{
    WTCR_INT0 = 0;                  // Clear the interrupt flag
    WTBR0 = (0xFFFF & DividerRC);  // Set Sub-Second Prescaler
    WTBR1 = (DividerRC >> 16);

}
```

The above sample code examples demonstrates how to initialize the Sub_Second register within the RTC interrupt service routine (if RTC is already running). Here the RTC does not need to be stopped since there is enough time to securely modify the registers until the next reload operation (next second interrupt). Here the RC Clock is considered to be running at 2 MHz. It should be noted that the RTC Second interrupt needs to be enabled.

Please note that the corresponding interrupt vector and level has to be defined in the *vectors.c* module of our standard template project as shown in the below sample code example.

```
void InitIrqLevels(void)
{
       . . .

       ICR = ((95 & 0xFF) << 8) | 6;     // Real Time Clock of MB9634x Series
       . . .
}

__interrupt void RealTimeClock (void);  // Prototype

       . . .

#pragma intvect RealTimeClock      95  // RTC of MB9634x Series
```

3.4 Read the Time inside ISR

```
extern unsigned char second, minute, hour;

__interrupt void RealTimeClock (void)
{
       WTCR_INT0 = 0;               // Clear the interrupt flag
       second = WTSR;
       minute = WTMR;
       hour = WTHR;

}
```

It is recommended that the interrupts (INT0-4) should be used to read the time information, as this would eliminate the possibility of reading incorrect values from time (hour/minute/second) registers (in case of carry while reading). In the above example the Time information is read in the RTC Second (INT0) interrupt service routine.

Please note that the corresponding interrupt vector and level has to be defined in the *vectors.c* module as shown in the above sample code example.

3.5 Read the Time inside application

```
#define TRUE          1
#define FALSE         0

unsigned char second, minute, hour;
unsigned char second1, minute1, hour1;
unsigned long int time, time1;

void ReadTime (void)
{
        unsigned char result = FALSE;

        /* Normally this loop would exit at 1st iteration (e.g. 02:59:59 ->
         03:00:00). In some cases only this loop will have 2 iterations at the max
        (e.g. 02:59:59 -> 03:59:59 (1st Iteration), 03:00:00 -> 03:00:00 (2nd  Iteration)).*/
        while (result != TRUE)
        {
                /* First Set of Time */
                second = WTSR;
                minute = WTMR;
                hour = WTHR;

                /* Second Set of Time */
                second1 = WTSR;
                minute1 = WTMR;
                hour1 = WTHR;

                /* Calculating absolute seconds for first set of time */
                time = hour*3600 + minute*60 + second;

                /* Calculating absolute seconds for second set of time */
                time1 = hour1*3600 + minute1*60 + second1;

                /* If the difference in the first & second set is 0 or 1 then the second
                set contains the latest accurate time information */
                if (((time1 - time) == 1) || ((time1 - time) == 0)))
                {
                        result = TRUE;
                }
                /* The following condition takes care of day change situation 23:59:59 ->
                00:00:00, and the second set contains the latest accurate time information */
                else if (time == 86399 && time1 == 0)
                {
                        result = TRUE; // 23:59:59 -> 00:00:00
                }
        }
}
```

The above sample code example demonstrates how the time can be read inside an application (without using interrupts). This also takes care of reading the time (hour/minute/second) registers at the very timing of changing over the hour or minute boundary.

Here the time registers are read twice. Then it is converted into absolute seconds value and if difference between the old and the new value is 0/1 or the old value is 86399 & the new value is 0 (i.e. 23:59:59 -> 00:00:00), then the second set (second1/minute1/hour1) is considered to contain the correct time information.

4 Additional Information

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

www.cypress.com/cypress-microcontrollers

The software example related to this application note is:

96340_rtc_init_read

It can be found on the following Internet page:

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

5 Document History

Document Title: AN205545 - F²MC-16FX Family, Real Time Clock

Document Number: 002-05545

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	05/05/2006	Initial release
			12/22/2006	Reviewed the document and updated with review findings
			02/20/2007	Updated with re-review findings
			08/10/2007	Typos fixed
*A	5071195	MKEA	04/19/2016	Migrated Spansion Application Note from MCU-AN-300207-E-V13 to Cypress format
*B	5876443	AESATMP9	09/07/2017	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

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