AN204823

F²MC-16FX Family with RTC, Sub Clock And Clock Calibration Unit

This application note describes a software mechanism how to use the Clock Calibration Unit, so that the accuracy of the RTC nearly only depends on the accuracy of the Main Clock oscillation.

Contents

1 Introduction

When using the Real Time Clock (RTC) with the 32 kHz Sub Clock, its accuracy is about ±125 ppm, which means a worse deviation of about ±10.54 s per day.

This application note describes a software mechanism how to use the Clock Calibration Unit, so that the accuracy of the RTC nearly only depends on the accuracy of the Main Clock oscillation.

Note: In this application note the Sub Clock oscillation is assumed to 32768 Hz regardless of the naming “32 kHz”.

2 Accuracy of Real Time Clock

Overview of the Accuracy of the RTC at Different Clock Sources

2.1 Sub Second Register (WTBR)

The Real Time Clock can be trimmed by the sub second register (`WTBR`), which is a prescaler for the (usual) 4 MHz clock for the RTC.

The following values should be used for ideal clock sources:

Clock Source	WTBR Divisor
4 MHz Main Clock	1000000
32 kHz Sub Clock	8192
100 kHz RC Clock	25000
2 MHz RC Clock	500000

2.2 Source Clock Accuracy

Because the sub second register can be changed by a value of 1 the following minimum frequency “deltas” can be set:

Clock Source	WTBR Divisor	Resulting RTC Clock in Hz	Accuracy in %	Accuracy in ppm
32 kHz Sub Clock	8191	4.0004883409	+0.0122	+122
	8192	4.0000000000	-	-
	8193	3.9995117783	-0.0122	-122
100 kHz RC Clock	24999	4.0001600064	+0.004	+40
	25000	4.0000000000	-	-
	25001	3.9998400064	-0.004	-40
2 MHz RC Clock	499999	4.0000080000	+0.0002	+2
	500000	4.0000000000	-	-
	500001	3.9999920000	-0.0002	-2
4 MHz Main Clock	999999	4.0000040000	+0.0001	+1
	1000000	4.0000000000	-	-
	1000001	3.9999960000	-0.0001	-1

Note: Usual crystals for watches have an accuracy of about 20 ppm. Usual Automotive crystals have an accuracy of about 200 ppm.

According to the 86400 seconds of a whole day and 365 days per year, the accuracies for the different frequencies are:

Clock Source	Accuracy in %	Difference per day in seconds	Difference per year
32 kHz Sub Clock	±0.0122	±10.54	±64.1 min
100 kHz RC Clock	±0.004	±3.456	±21.0 min
2 MHz RC Clock	±0.0002	±0.1728	±63.0 s
4 MHz Main Clock	±0.0001	±0.0864	±31.5 s

This means that the RTC clocked by 32 kHz Sub Clock oscillation can be trimmed by a granularity of about 10.54 s per day.

2.3 Time Deviation Formula for given Accuracy

With the following formula the time deviation per day for a given clock source accuracy acc in *ppm* can be calculated:

$$t_{day-dev} = acc * (\pm 0.0864s)$$

The deviation per year can be calculated as:

$$t_{year-dev} = acc * (\pm 31.536s)$$

3 Using Clock Calibration Unit

Trimming the Sub Clock with the Clock Calibration Unit

3.1 Clock Measurement and Calibration

The Clock Calibration Unit uses two different clock sources. One is always the Main Clock and the other the Sub or RC Clock. In our case we are using the Sub Clock.

With the CUTD register a duration is being set for the Sub Clock gating the Main Clock. Assume we use the value 0x4000 (8192). This represents a gate duration of 0.5 s. In this case a value about 0x1E8480 (2000000) is expected for CUTR result.

The following formula shows how to calibrate the sub second register (WTBR) of the RTC with the measurement data of CUTD.

$$WTBR_{CALIBRATED} = \frac{f_{SC} \cdot f_{MAIN} \cdot t_{INTERVAL}}{f_{RTC} \cdot CUTR}$$

f_{SC}:	Clock source of Real Time Clock (CLKSC)
f_{MAIN}:	Main Clock (CLKMC)
$t_{INTERVAL}$:	Interval time given by CUTD
t_{RTCL}:	RTC clock frequency (4 Hz)

3.1.1 Example

The SC clock at 32 kHz should be measured against the (ideal) Main Clock of 4 MHz. For this a duration interval of 0.5s is used. If the RC clock is also ideal, 4 MHz • 0.5 s = 2000000 main clock ticks should be measured.

Assume that 1999888 Main Clock ticks were measured. This means that the SC Clock is a bit faster than 32 kHz (less Main Clock ticks than expected). For the RTC sub second register (WTBR) a new value can be calculated for calibration:

$$WTBR_{CALIBRATED} = \frac{32768s^{-1} \cdot 4000000s^{-1} \cdot 0.5s}{4s^{-1} \cdot 1999888} = 8192.4588$$

3.2 Accuracy

The example above shows an accuracy problem. Assuming the Main Clock oscillation exact to be 4 MHz, the Clock Calibration Unit measured for the Sub Clock a frequency of:

$$f_{RC(REAL)} = f_{RC(IDEAL)} * \frac{CUTR_{IDEAL}}{CUTR} = 32768Hz * \frac{2000000}{1999888} = 32769.84Hz$$

If we round down the measured WTBR value to 8192, we have a deviation to the real frequency of:

$$dev = \frac{f_{RC(REAL)}}{f_{RTC} * WTBR} = \frac{32769.84Hz}{4Hz * 8192} = 1.00005615$$

This means our RTC is clocked too fast by 0.0056%. This means a systematical error of about -4s per day – every day – if the temperature is constant and thus the frequency of the Sub Clock is stable.

3.3 Example Scenario

The following table shows a scenario which uses the formula from above and accepts systematic errors. The RTC is recalibrated each 60 seconds. A new WTBR is calculated and set. An RTC-independent timer (e.g. Sub Clock Timer) triggers the calculation process.

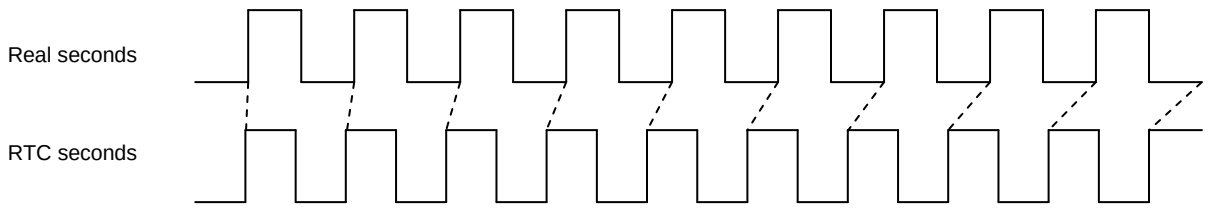
The basic parameters are:

f_{MAIN}	= 4 MHz
$t_{INTERVAL}$	= 0.5 s
$t_{CALIBRATION}$	= 60 s

Time [s] (ideal)	CUTR	Sub Clock [Hz] (real)	WTBR (ideal)	WTBR	Time [s] (real)	Time diff. [s]
60	2000189	32764.90	8191.2259	8191	60.0017	0.0017
120	2000189	32764.90	8191.2259	8191	120.0033	0.0033
180	2000189	32764.90	8191.2259	8191	180.0050	0.0050
240	2000189	32764.90	8191.2259	8191	240.0066	0.0066
300	2000189	32764.90	8191.2259	8191	300.0083	0.0083
360	2000189	32764.90	8191.2259	8191	360.0099	0.0099
420	2000189	32764.90	8191.2259	8191	420.0116	0.0116
480	2000189	32764.90	8191.2259	8191	480.0132	0.0132
540	2000189	32764.90	8191.2259	8191	540.0149	0.0149
600	2000189	32764.90	8191.2259	8191	600.0165	0.0165
660	2000189	32764.90	8191.2259	8191	660.0182	0.0182
720	2000189	32764.90	8191.2259	8191	720.0199	0.0199
780	2000189	32764.90	8191.2259	8191	780.0215	0.0215
840	2000189	32764.90	8191.2259	8191	840.0232	0.0232
900	2000189	32764.90	8191.2259	8191	900.0248	0.0248
960	2000189	32764.90	8191.2259	8191	960.0265	0.0265
1020	2000189	32764.90	8191.2259	8191	1020.0281	0.0281
1080	2000189	32764.90	8191.2259	8191	1080.0298	0.0298
1140	2000189	32764.90	8191.2259	8191	1140.0314	0.0314
1200	2000189	32764.90	8191.2259	8191	1200.0376	0.0376

It can clearly be seen, that the deviation between ideal `WTBR` and real `WTBR` leads to a systematic error in time. In this case the subsecond divider (`WTBR`) is too small, resulting in a higher clock frequency for the RTC. Therefore the time difference increases with the run time.

The following graphic shows the problem as a qualitative timing diagram.:



4 Using Clock Calibration Unit with Error Compensation

Trimming the Sub Clock with the CCU and Error Compensation

4.1 Clock Measurement, Calibration and Error Compensation

Calculation of WTBR with a finally rounding to a whole-number value leads to a systematic error. It is a good solution to calculate also this error for each interval between the calibrations and integrate the error deviation to the new calculated WTBR value.

The new formula for error compensation is:

$$WTBR_{n,CALIBRATED} = \left\lfloor 0.5 + \frac{f_{SC} \cdot f_{MAIN} \cdot t_{INTERVAL}}{f_{RTC} \cdot CUTR} \right\rfloor + w * \Delta t$$

$\lfloor \ \rfloor$ is the floor function

where w is a “weight factor”, which controls how heavy should be the compensation influence and

$$\Delta t = t_{REAL,n-1} - t_{IDEAL,n-1}$$

Where $t_{REAL,n-1}$ is the time, which passed due to $WBTR_{n-1}$ value and $t_{IDEAL,n-1}$ the time which should passed ideally.

Note: From experience a reasonable value for w is between 50 and 100. It can be understood as a feedback amplitude like in a closed loop.

Note:

4.2 Example Scenario

The following table shows a scenario with the same measured CUTR values shown in 3.3. The scenario uses the formula from above with the error compensation. The RTC is recalibrated each 60 seconds. The “weight factor” was chosen at 100.

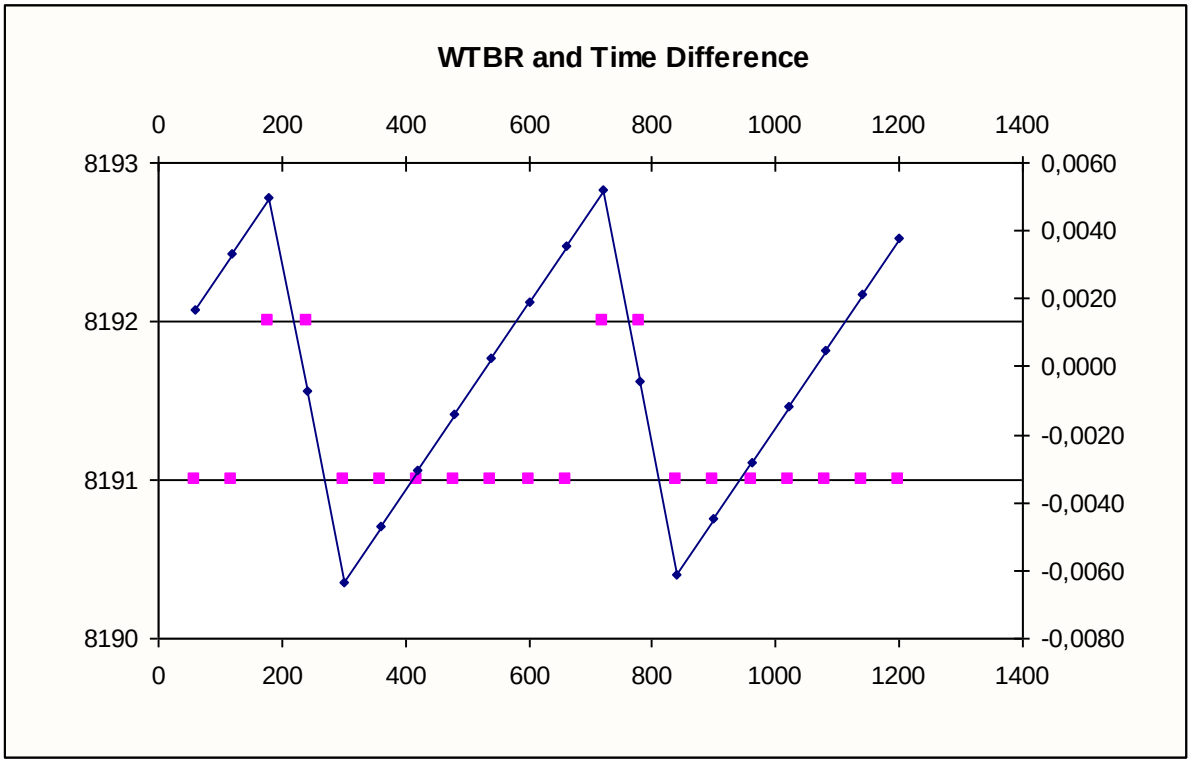
The basic parameters are:

f_{MAIN} = 4 MHz
$t_{INTERVAL}$ = 0.5 s
$t_{CALIBRATION}$ = 60 s
w = 100

Time [s] (ideal)	CUTR	Sub Clock [Hz] (real)	WTBR (ideal)	WTBR	Time [s] (real)	Time diff. [s]
60	2000189	32764.90	8191.2259	8191	60.0017	0.0017
120	2000189	32764.90	8191.3914	8191	120.0033	0.0033
180	2000189	32764.90	8191.5569	8192	180.0050	0.0050
240	2000189	32764.90	8191.7224	8192	239.9993	-0.0007
300	2000189	32764.90	8191.1555	8191	299.9936	-0.0064
360	2000189	32764.90	8190.5885	8191	359.9953	-0.0047
420	2000189	32764.90	8190.7540	8191	419.9986	-0.0031
480	2000189	32764.90	8190.9195	8191	479.9986	-0.0014
540	2000189	32764.90	8191.0850	8191	540.0002	0.0002
600	2000189	32764.90	8191.2505	8191	600.0036	0.0036
660	2000189	32764.90	8191.4160	8191	660.0036	0.0036
720	2000189	32764.90	8191.5815	8192	720.0052	0.0052
780	2000189	32764.90	8191.7470	8192	779.9995	-0.0005

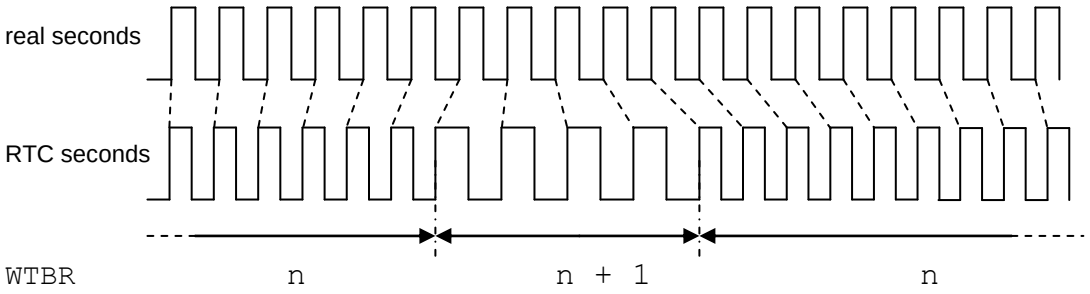
Time [s] (ideal)	CUTR	Sub Clock [Hz] (real)	WTBR (ideal)	WTBR	Time [s] (real)	Time diff. [s]
840	2000189	32764.90	8191.1800	8191	839.9939	-0.0061
900	2000189	32764.90	8191.6131	8191	899.9955	-0.0045
960	2000189	32764.90	8191.7786	8191	959.9972	-0.0028
1020	2000189	32764.90	8191.9441	8191	1019.9988	-0.0012
1080	2000189	32764.90	8191.1096	8191	1080.0005	0.0005
1140	2000189	32764.90	8191.2751	8191	1140.0021	0.0021
1200	2000189	32764.90	8191.4406	8191	1200.0038	0.0038

At bold entries it can be seen, that in certain lines the `WTBR` value is 2 higher than usual. This compensates the error continuously. The following graphic illustrates this.



Like in the example in chapter 3.3 the time error (*blue*) increases with the `WTBR` value (*pink*) of 8191, but with the error correction a value of 8182 for two calibration cycles compensates this error. This compensation still produces an error, but over run time the average error is about 0.

The following graphic shows the compensation as a qualitative timing diagram:



5 Example Program for Self Calibration and Error Correction

The Following Chapter Shows How to Implement an Self Calibration

5.1 Source Codes

The following code shows an example, how to implement a self-calibration of the RTC, which is clocked by the Sub Clock with an external crystal of 32768 Hz. The Main Clock is 4 Mhz. The calibration interval is set to 60 seconds, which is controlled by the Sub Clock Timer interval of 4 seconds (each 15th interrupt a calibration is done). The calibration duration is set to 0.5 seconds.

5.1.1 Global definitions

The following code shows a global definition file (e.g. *global.h*).

```
/* 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.                                              */
/*----------------------------------------------------------------------------*/

#define MAIN_CLOCK 4000000   // Main Clock in Hz
#define SUB_CLOCK 32768      // Sub clock in Hz
#define INTERVAL 0.5         // Measurement duration in s

#define CUTD_VALUE (INTERVAL * SUB_CLOCK)
#define CUTR_IDEAL (unsigned long) (INTERVAL * MAIN_CLOCK)

#define SCT 0x47             // irq all 4 s

#define UPDATE_INTERVAL 15   // 4 s * 15 = 60 s
#define WEIGHT (float)100    // 50 ... 100
#define T_INTERVAL 60        // [s]
```

5.1.2 Initialization of the Sub Clock Timer

The following code can be used for initialization of the Sub Clock Timer. Please note that the initial value (`SCT`) is defined above in the global definition. Please also not, that the corresponding interrupt vector and level have to be initialized too.

```
/* 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.                                              */
/*----------------------------------------------------------------------------*/

//---------------------------- INIT SC CLOCK TIMER ---------------------------

void InitSCT(void)
{
    SCTCR = SCT;
}
```

5.1.3 Initialization of the Real Time Clock

The following code shows how to initialize the RTC.

```
/* 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.                                              */
/*----------------------------------------------------------------------------*/

//--------------------------------- INIT RTC ---------------------------------

void InitRTC(unsigned char hour, unsigned char minute, unsigned long subsecond)
{
       WTCKSR = 0x01;                    // Sub clock for RTC
       WTCR_ST = 0;                      // Stop RTC
   WTCR_INTE0 = 0;               // No Interrupts
   WTCR_INTE1 = 0;
   WTCR_INTE2 = 0;
   WTCR_INTE3 = 0;
   WTCER_INTE4 = 0;
   WTCR_OE = 1;                  // Output for monitoring (optional)

   WTBR0 = (0xFFFF & subsecond); // Set calibrated prescaler value
   WTBR1 = (subsecond >> 16);
   WTHR = hour;
   WTMR = minute;
   WTSR = 0;
   WTCR_UPDT = 1;                // Update RTC

   // maybe a start condition can be programmed here

   WTCR_ST = 1;                  // Start RTC
}
```

5.1.4 Clock Measurement Routine

The following routine calculates a new WTBR value corresponding to error times (dt) and weight factor (WEIGHT).

```
/* 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.                                              */
/*----------------------------------------------------------------------------*/

//--------------------------- SC CLOCK MEASUREMENT ---------------------------

unsigned long ClockMeasure(unsigned int duration)
{
   double calibrate, correction;
   unsigned long wtbr_cal2;

   CUTD = duration;
   CUCR_INT = 0;    // clear flag prophylacticly

   CUCR_STRT = 1;   // start calibration
   while (!CUCR_INT);      // wait for end (irq handling possible here, beacuse
                           //   of 0.5 s latency time)

   CUCR_INT = 0;    // clear flag
                                                              ▼
```

```
                                                              ▲
   // calibration
   calibrate = 0.5 + (((float) SUB_CLOCK * (float) MAIN_CLOCK
(float) INTERVAL) / ((float) 4 * (float) CUTR));

   wtbr_cal2 = (unsigned long) calibrate;

   return wtbr_cal2;
}
```

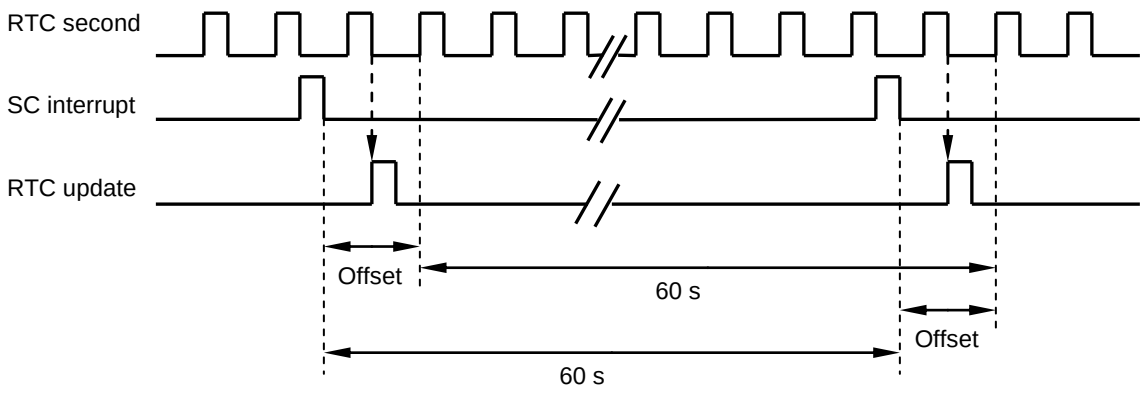
5.1.5 Interrupt Handler of Sub Clock Timer and Re-Calibration

The following code shows the ISR for the Sub Clock Timer. Please note that only in times defined by `UPDATE_INTERVAL` the calibration process is started.

```
/* 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.                                              */
/*----------------------------------------------------------------------------*/
//---------------------------------- ISR -------------------------------------
//
// Recalibration of RTC

__interrupt void ISR_SC(void)
{
    unsigned long wtbr_cal_old;

    SCTCR = SCT;                    // clear irq

    if (update_count >= UPDATE_INTERVAL)
    {
        update_count = 0;

        // calculate time deviation
        dt += (((double)CUTR / (MAIN_CLOCK / 2)) * SUB_CLOCK / cal
               / 4 * (double)T_INTERVAL) - (double)T_INTERVAL;

        cal = ClockMeasure(CUTD_VALUE);
        cal += dt * WEIGHT;

        WTCR_INT0 = 0;
        while(!WTCR_INT0);          // wait for second

        WTBR0 = (0xFFFF & cal);     // Set calibrated prescaler value
        WTBR1 = (cal >> 16);
    }
    update_count++;
}
```

Please note, that `WTBR` should only be set after internal reload of the RTC. Therefore it is waited for a new second before update.

Don't worry about this delay, because it is a fix time offset for all calibration intervals. The following graphic illustrates this.



The `ISR_RC` must not be interrupted by a higher priority interrupt of course. Otherwise the RTC update is performed too late and the calibration interval is longer than 60 seconds then!

5.1.6 Interrupt Definition

If the FME template project is used as a base for this kind of application, the `ISR_RC` has to be declared in the *vectors.c* file.

```
/* 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 = (15 << 8) | 2;
}

       . . .

__interrupt void ISR_SC (void);

       . . .

#pragma intvect ISR_SC            15   /* Sub Clock Timer              */

       . . .
```

5.1.7 Main Routine and global Variable Definition

The following code shows all needed global variable definitions and the main function itself.

```
/* 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.                                             */
/*---------------------------------------------------------------------------*/

#include "mb96xxxxx.h"        // depends on used device
#include "global.h"           // Global definitions

char update_count;            // update interval counter for SC clock timer irq
unsigned long wtbr_cal;       // calibrated WTBR value
double dt;                    // time deviation due to real SC frequency
double fSCreal;               // calculated real SC frequency

//---------------------------------- MAIN ----------------------------------

void main(void)
{
    unsigned char hour, minute;

    InitIrqLevels();
    __set_il(7);              // allow all levels
    __EI();                   // globally enable interrupts

    update_count = 1;

    CUCR_CKSEL = 0;           // Sub clock for calibration unit

    dt = 0;          // begin with time deviation of 0
    wtbr_cal = ClockMeasure(CUTD_VALUE);

    hour = 0;        // Set some reasonable values here
    minute = 0;

    InitRTC(hour, minute, wtbr_cal);
    InitSCT();

    while(1)
    {
        // main application loop
    }
}
```

6 Outlook

How to Improve Accuracy More

6.1 Accuracy Dependency

The accuracy of the RTC mostly depends on the accuracy of the Main Clock frequency.

When using a (standard) crystal, a quality depending frequency drift over the temperature can be assumed.

6.2 Typical Crystal Accuracy against Temperature

The following diagram shows a typical “tuning fork” crystal accuracy against the temperature with a parabolic shaped curve.

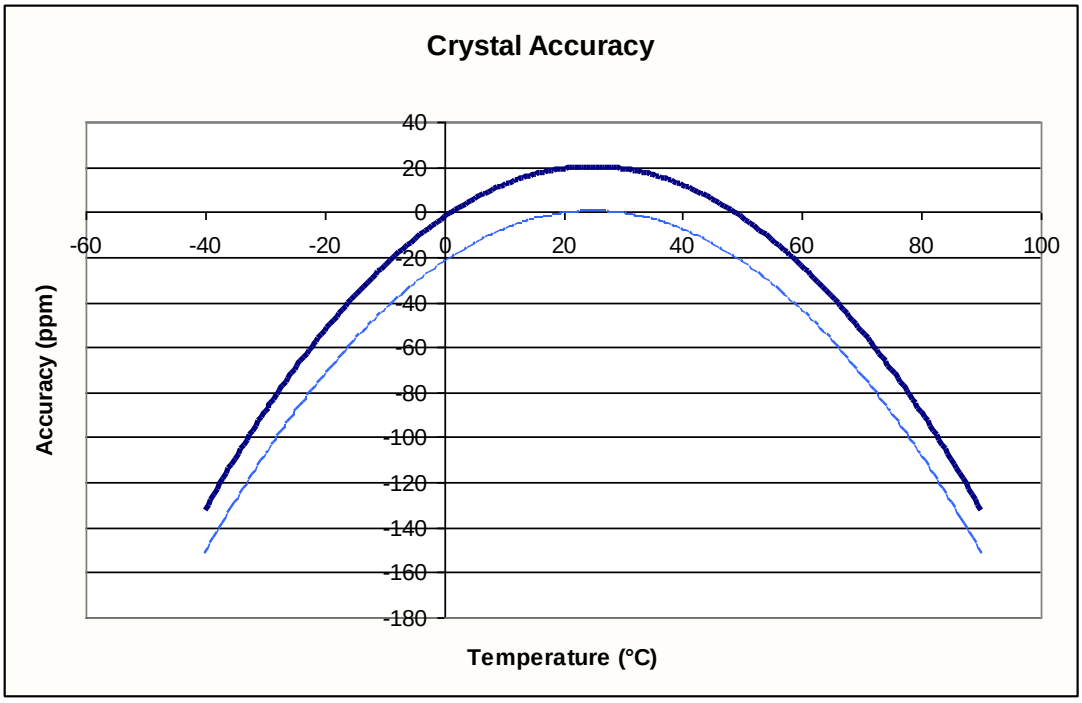
The accuracy can be calculated as:

$$acc = K(T - T_0)^2$$

acc	Accuracy
K	Curvature characteristic [ppm/°C]
T	Working temperature [°C]
T_0	Turnover (room) temperature [°C]

Please note, that this example diagram is for an ideal crystal with 0 ppm at +25°C.

If the crystal has for example +20 ppm at +25 °C the new curve will look like the following diagram:



If the parameters of a crystal are known, the working frequency can be calculated, if the temperature is also known.

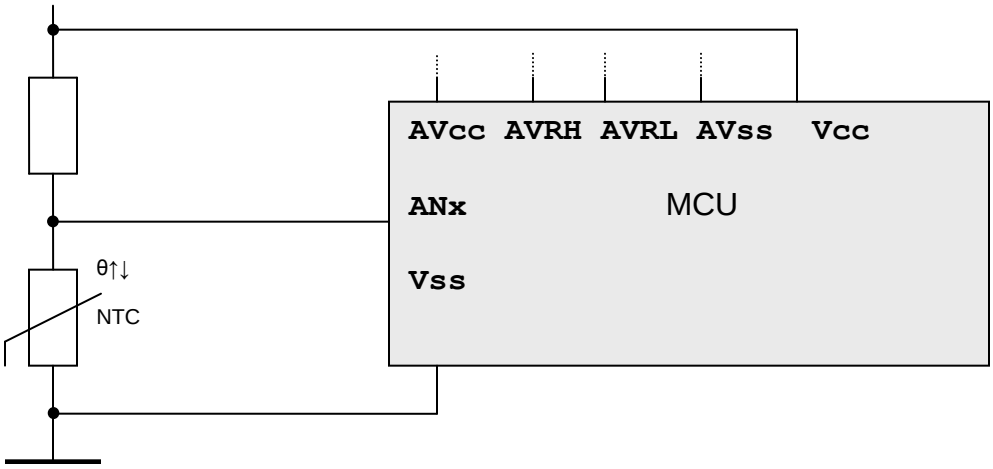
$$f_{work} = f_0\left(K\left[T_{work} - T_0\right]^2 + a_0 + 1\right)$$

f_{work}	Working frequency
f_0	Frequency at room temperature
K	Curvature characteristic [ppm/°C]
T_{work}	Working temperature [°C]
T_0	Turnover (room) temperature [°C]
a_0	Turnover (RT) accuracy

6.3 Temperature Measurement

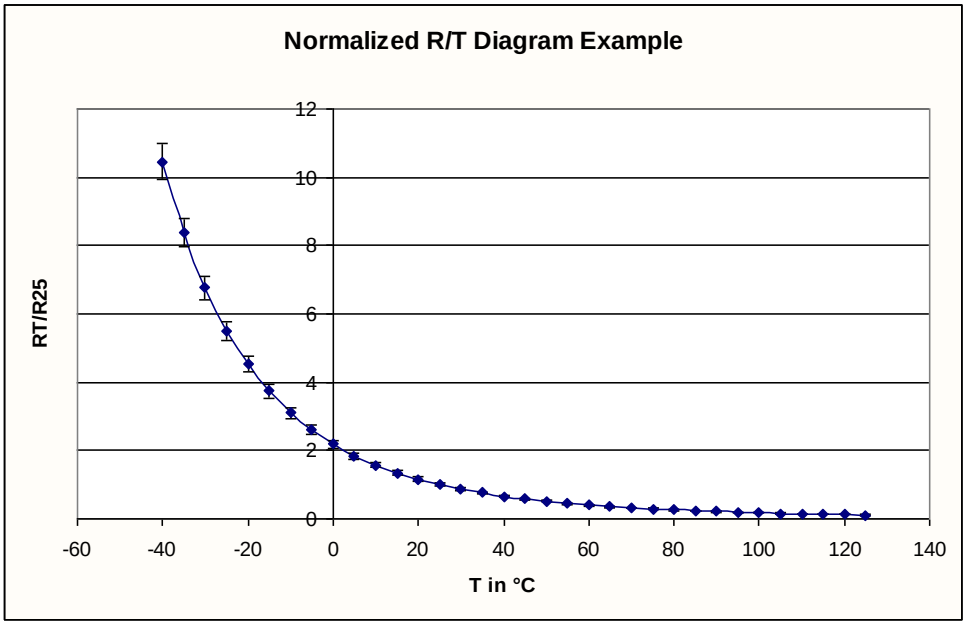
There exist many possibilities to measure the temperature with a Microcontroller. Mostly a temperature depending resistor together with a reference voltage and an ADC input is used. Well-known temperature depending resistors are PTCs (e.g. Pt100) and NTCs.

The following schematic shows a possible solution for temperature measurement.

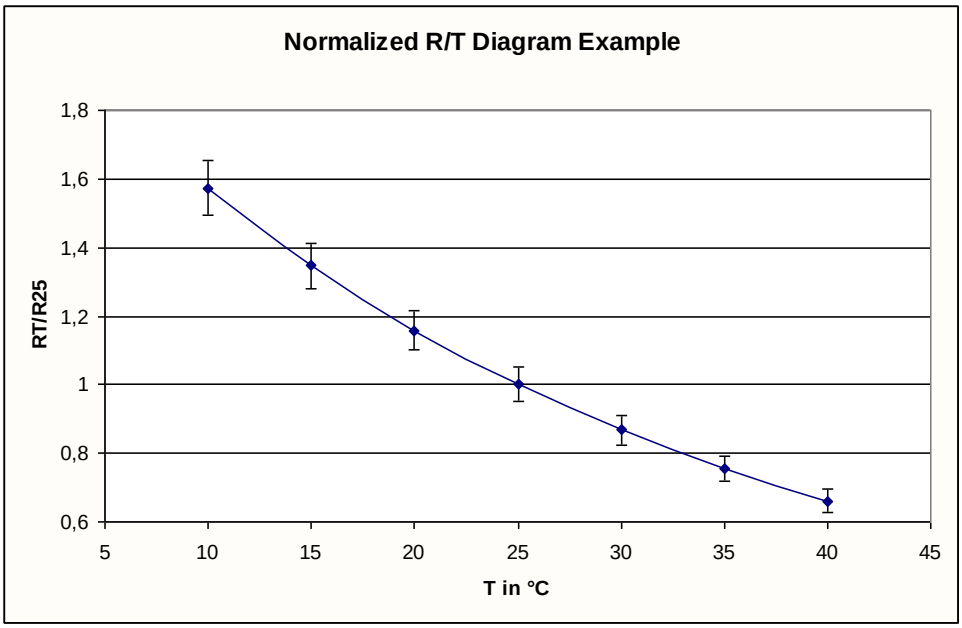


Note: Is has to be considered, that the accuracy of the used ADC and thermal resistor is meaningful due to the crystal accuracy and the purpose of increasing the RTC accuracy.

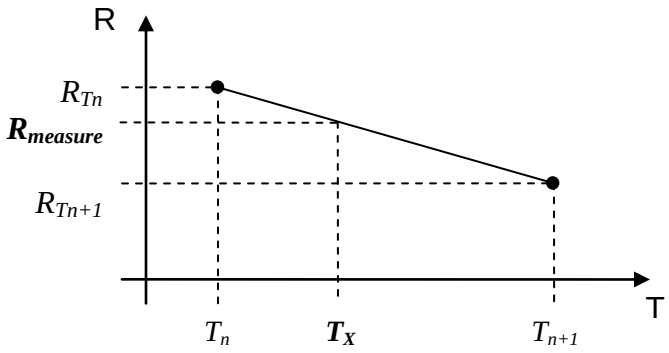
The following graphic shows a typical normalized characteristic line of an NTC taken from a specification table. Note that an example tolerance of 5% is shown with upper and lower bars.



In the interesting temperature area (about 25°C) the following graphic shows a zoom of the above one.



Because the 5% tolerances of the points nearly overlap, a linear interpolation can be used:



$$T_X = \frac{R_{measure}}{R_{Tn} + \frac{R_{Tn+1} - R_{Tn}}{T_{n+1} - T_n}} + T_n$$

$R_{measure}$	measured NTC resistor value
T_n	Temperature entry left of measurement
T_{n+1}	Temperature entry right of measurement
R_{Tn}	Corresponding resistor entry to T_n
R_{Tn+1}	Corresponding resistor entry to T_{n+1}

For a more precise interpolation the temperature coefficient α_n has to be known too. For temperature calculation the following formula can be used:

$$R_T = R_{Tn} \cdot e^{\left[\frac{\alpha_n}{100} T_n^2 \left(\frac{1}{T} - \frac{1}{T_n}\right)\right]}$$

R_T	NTC resistor value at measurement temperature
R_{Tn}	resistor lower table interval value
α_n	coefficient lower table interval value
T	interesting temperature in Kelvin
T_n	temperature lower table interval value in Kelvin

Transformed to acquire the interesting temperature:

$$T = \left(\frac{100 \cdot \ln\left[\frac{R_T}{Rn} \right]}{\alpha_n - T_n^2} + \frac{1}{T_n} \right)^{-1}$$

7 Related Application Notes and Contact

Overview of Related Application Notes and Contact to Cypress

7.1 Manuals and Application Notes

- **mcu-an-300207-e-16fx_rtclk**
  Usage of the Real Time Clock
- **mcu-an-300229-e-16fx_clk_cal**
  Usage of the Clock Calibration Unit

7.2 Software

Example software to this application note:

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

`96340_rtc_clkcal_sc_autocorr`

8 Document History

Document Title: AN204823 - F²MC-16FX Family with RTC, Sub Clock and Clock Calibration Unit

Document Number: 002-04823

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MKEA	05/08/2007	Initial Release
*A	5061298	MKEA	12/24/2015	Migrated Spansion Application Note from MCU-AN-300234-E-V10 to Cypress format
*B	5835299	AESATMP9	07/27/2017	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