Oscillator handling

XMC1000

About this document

Scope and purpose

This application note describes the calibration of the frequency of the on-chip oscillator in the XMC1000 microcontroller family. Two methods for calibrating the internal clock are covered:

- Using an external reference signal
- Using the on-chip temperature sensor

Intended audience

This document is intended for engineers or developers who would like to improve the accuracy of the XMC1000 internal oscillator.

Revision 1.6
2016-04-22

Table of contents

1 System clock calibration with an external clock reference

1.1 Overview

The XMC1000 family of devices provides an internal oscillator for the generation of the system clock. This clock is available to the CPU and the on-chip peripherals. In the XMC1400, an external crystal is also supported for the generation of the system clock.

The frequency of the internal clock varies when the supply voltage or temperature changes. In the XMC1100, XMC1200 or XMC1300 applications where the accuracy of the system clock is crucial, the internal clock can be calibrated at runtime. This calibration is performed by adjusting the MCLK using the fractional divider in the clock control unit. However, this calibration method is not recommended for XMC1400. Instead, a more accurate clock can be achieved with an external crystal.

In this application note, we introduce an example for calibrating the internal clock with an external reference clock. The external reference clock (such as a crystal or stable clock source) is fed into the microcontroller and the frequency is captured by the Capture Compare Unit 4 (CCU4). After capturing the frequency, the calibration software determines whether the system clock is still running within the correct range. If the system clock is out of range, the fractional divider is adjusted accordingly.

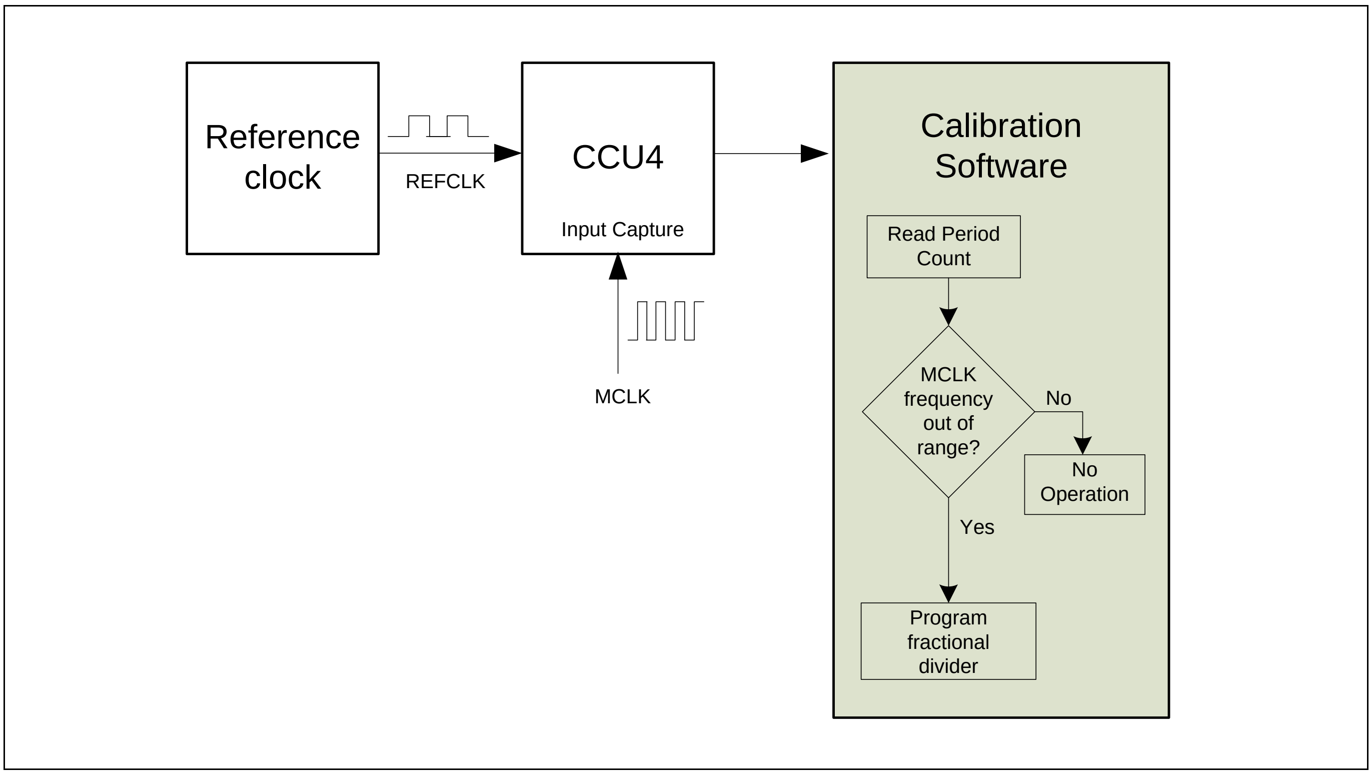


Figure 1 **An example of calibrating internal clock with an external reference clock**

The following figure shows an example of the reference clock circuit that could be used to provide the reference clock for calibration. An external crystal is used in this example.

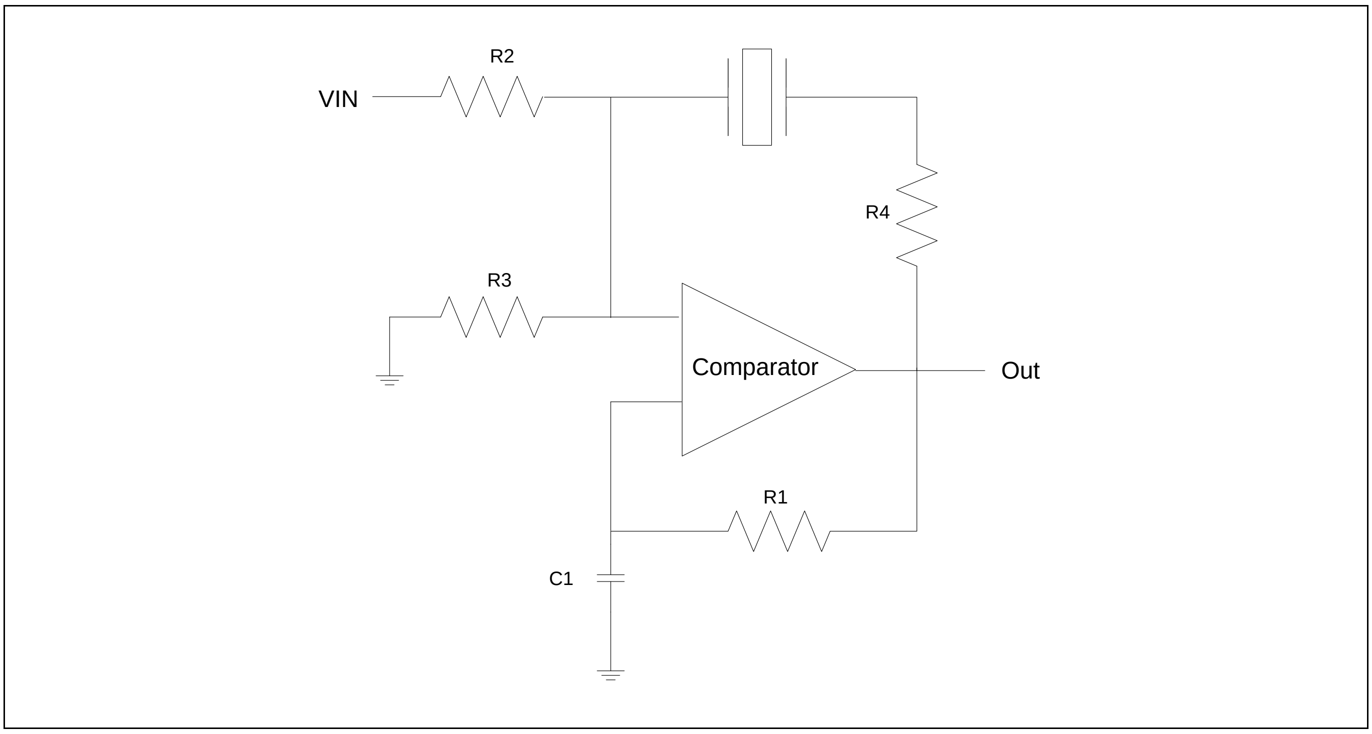


Figure 2 An example of the reference clock circuit

Instead of using a crystal as the reference clock, any stable clock source can also be used. For example, in a system which uses the XMC1000 as a slave controller and the master is a higher performance microcontroller (such as the XMC4000), an accurate clock output from the master can be used as a reference clock to the CCU4 input of the XMC1000.

Note: The circuit provided and the method of calibrating the MCLK described in this document is given only as an example implementation. If you choose to follow this implementation, then it is your responsibility to validate and fine tune your actual board accordingly, especially for the oscillator startup time and stability because it depends on the type of crystal used.

1.2 Implementation

The following code examples are based on the XMC1200.

1.2.1 Oscillator circuit to generate reference clock

An example of the external oscillator circuit as shown in Figure 2 is required to ensure a stable reference clock signal. The following sub-chapters show the matching component values for each type of reference clock circuit used in the oscillator circuit. In these examples, we use an oscillator circuit that generates a 32.768 kHz clock signal. If available, the on-chip analog comparator can be used as part of the oscillator circuit.

In this example:

- ACMP0 is used to emulate an oscillator circuit
- Pin P0.10 is used as ACMP0.OUT, to output the stable clock source
- P2.8 is used as ACMP0.INN, as negative input of the oscillator
- P2.9 is used as ACMP0.INP, as positive input of the oscillator

The code examples to initialize the ACMP and GPIO are given below.

```
void CMPCU_Init(void){


 COMPARATOR->ANACMP0 =
   ((0x0 << COMPARATOR_ANACMP0_CMP_OUT_Pos) & COMPARATOR_ANACMP0_CMP_OUT_Msk)
|
   ((0x0 << COMPARATOR_ANACMP0_CMP_LPWR_Pos) &
COMPARATOR_ANACMP0_CMP_LPWR_Msk) |
   ((0x0 << COMPARATOR_ANACMP0_ACMP0_SEL_Pos) &
COMPARATOR_ANACMP0_ACMP0_SEL_Msk) |
   ((0x0 << COMPARATOR_ANACMP0_CMP_HYST_ADJ_Pos)
   & COMPARATOR_ANACMP0_CMP_HYST_ADJ_Msk)|
   ((0x0 << COMPARATOR_ANACMP0_CMP_INV_OUT_Pos)
   & COMPARATOR_ANACMP0_CMP_INV_OUT_Msk) |
   ((0x1 << COMPARATOR_ANACMP0_CMP_FLT_OFF_Pos)
   & COMPARATOR_ANACMP0_CMP_FLT_OFF_Msk) |
   ((0x1 << COMPARATOR_ANACMP0_CMP_EN_Pos) & COMPARATOR_ANACMP0_CMP_EN_Msk) ;
}
void PORTS_Init(void)
{
  PORT0->IOCR8 =
     ((0x0 << PORT0_IOCR8_PC11_Pos) & PORT0_IOCR8_PC11_Msk) |
     ((0x14 << PORT0_IOCR8_PC10_Pos) & PORT0_IOCR8_PC10_Msk) |
     ((0x0 << PORT0_IOCR8_PC9_Pos) & PORT0_IOCR8_PC9_Msk) |
     ((0x0 << PORT0_IOCR8_PC8_Pos) & PORT0_IOCR8_PC8_Msk) ;
}
```

Three oscillator circuit examples are included for consideration in implementation.

1.2.1.1 Oscillator circuit example A

The following figure shows the component values for the Seiko FC-13A 32.768 kHz quartz crystal.

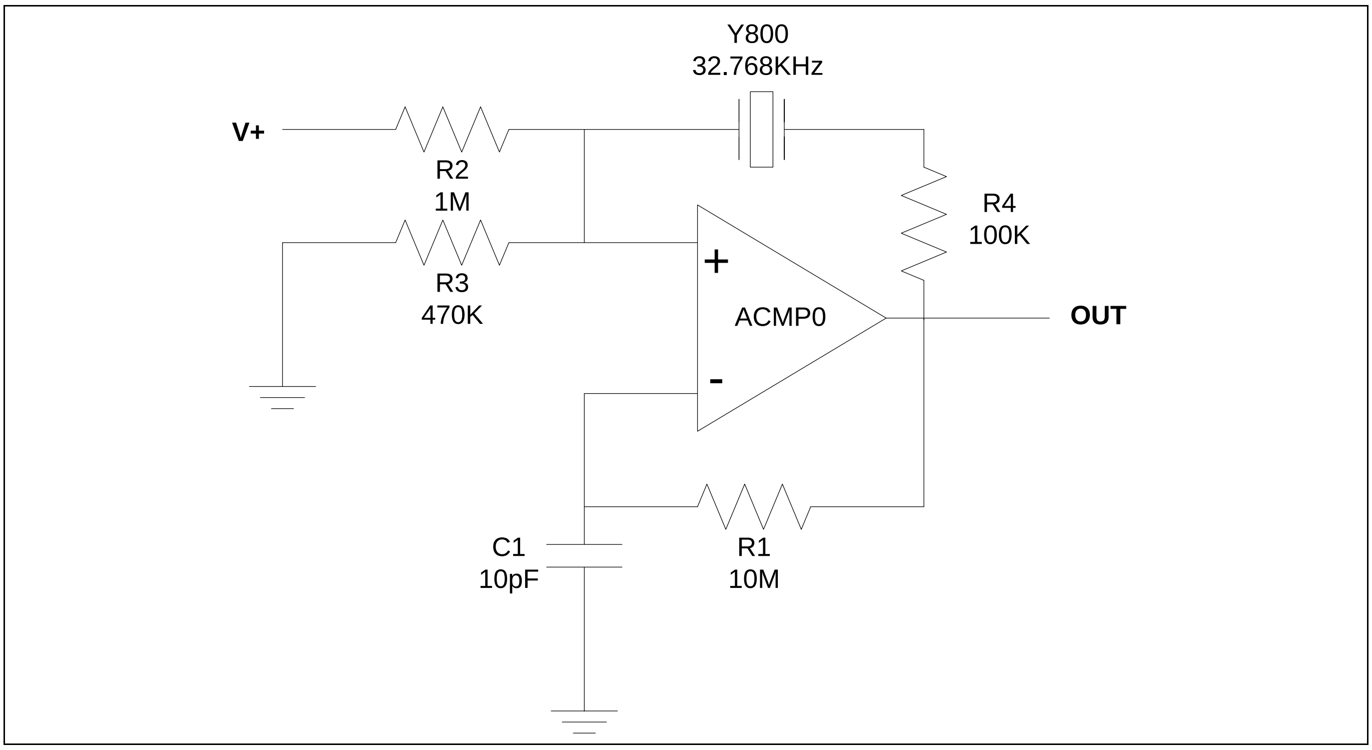


Figure 3 32kHz oscillator circuit with Seiko crystal

This figure shows the oscillator circuit design using the on-chip comparator.

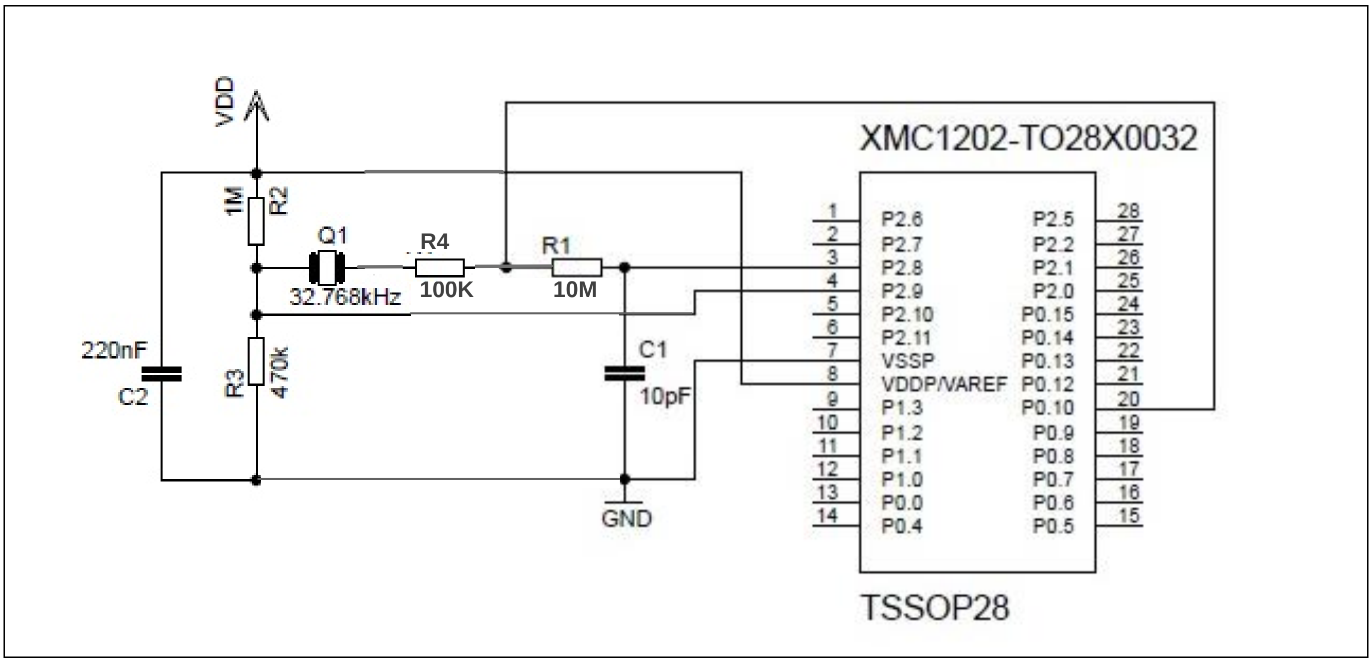


Figure 4 Oscillator circuit with XMC1200 on-chip analog comparator

1.2.1.2 Oscillator circuit example B

The following figure shows the component values for the Hong Kong X’tals Limited 2060K3276IB00 and 2060K3276IB01 crystals.

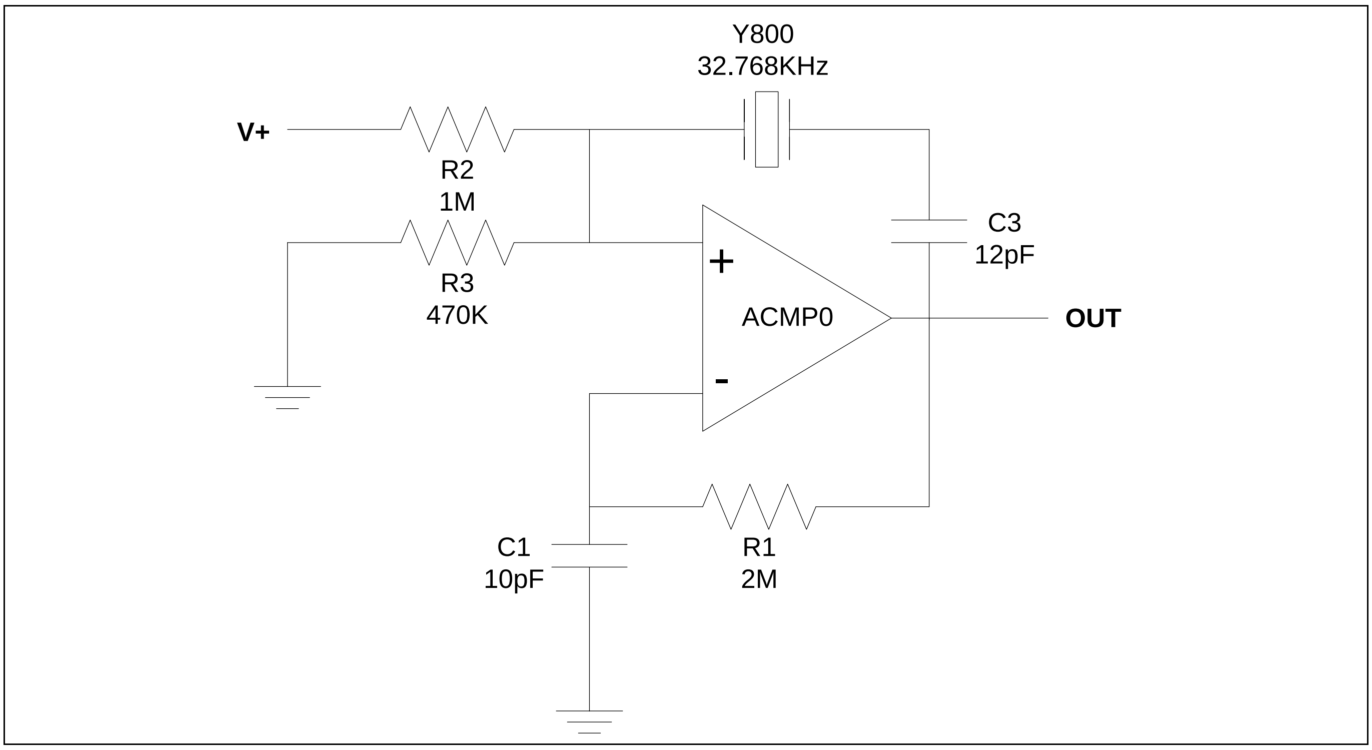


Figure 5 32 kHz oscillator circuit with Hong Kong X’tals Limited crystal

This figure shows the oscillator circuit design using the on-chip comparator.

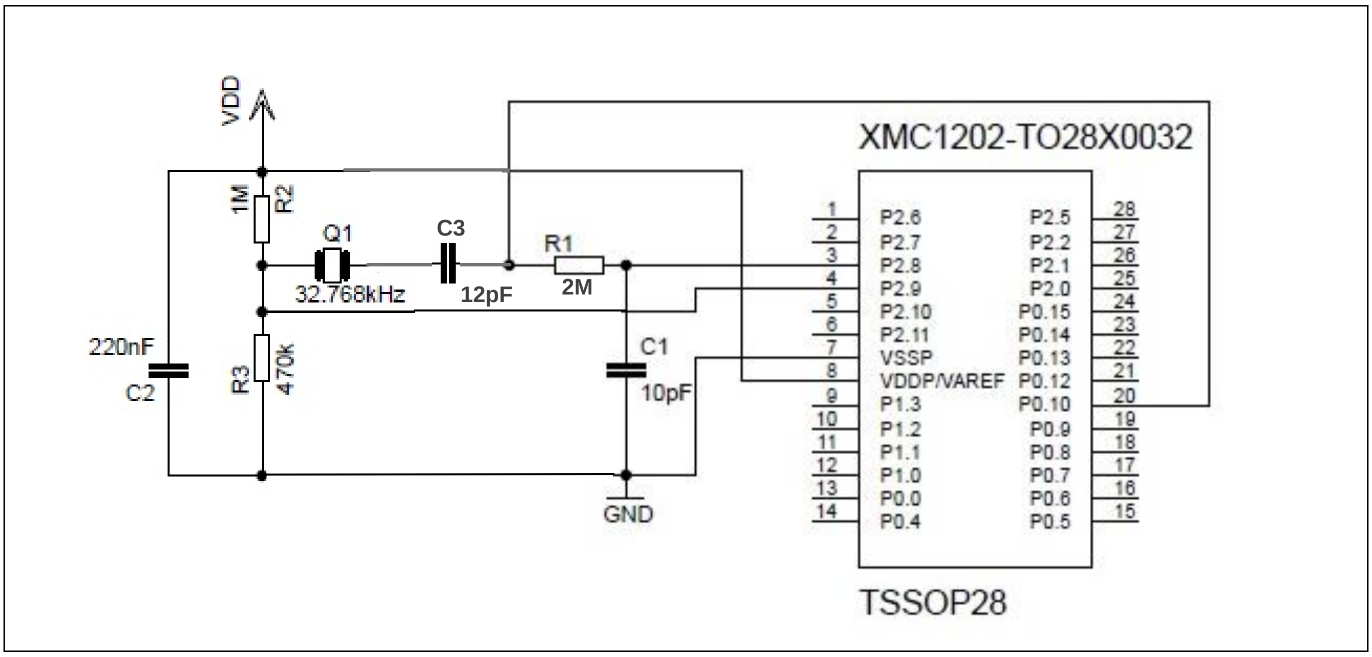


Figure 6 Oscillator circuit with XMC1200 on-chip analog comparator

1.2.1.3 Oscillator circuit example C

The following figure shows the component values for the Seiko Crystal Seiko FC-13A 32.768 kHz quartz crystal.

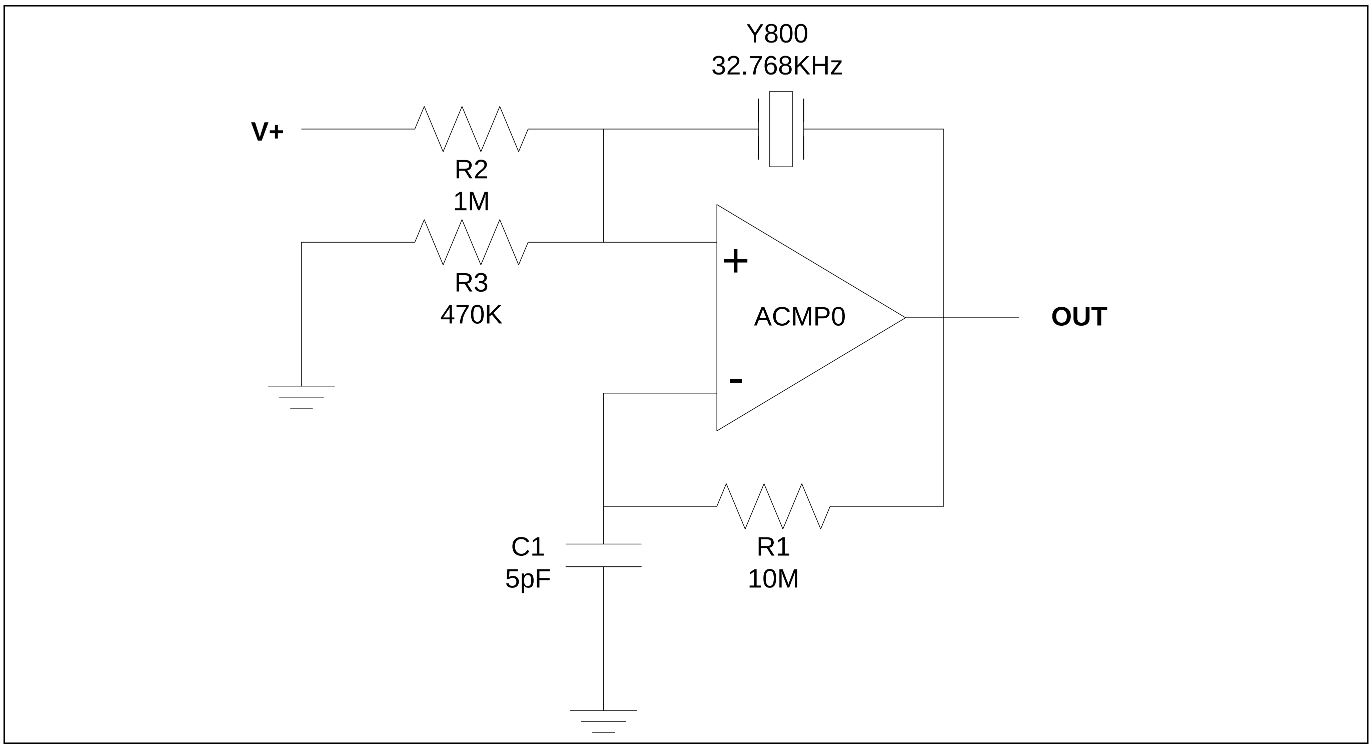


Figure 7 32kHz oscillator circuit with Seiko crystal

This figure shows the oscillator circuit design using the on-chip comparator.

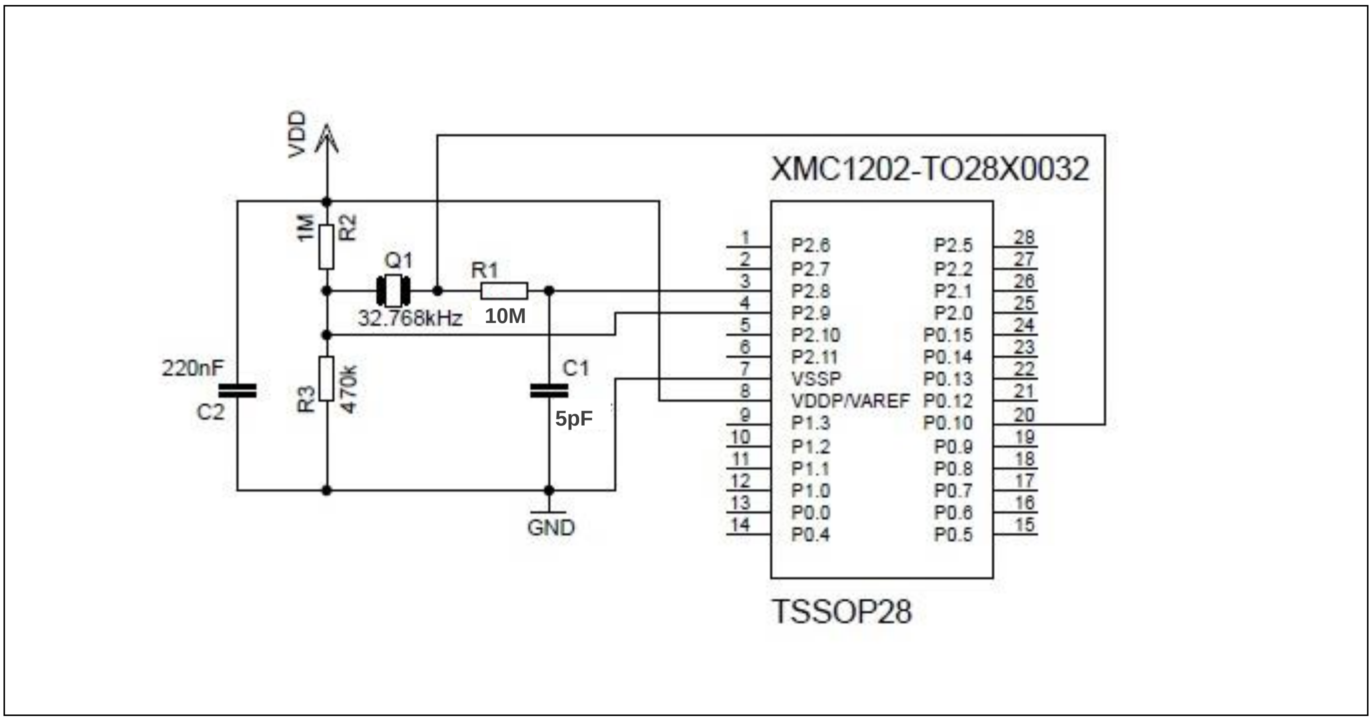


Figure 8 Oscillator circuit with XMC1200 on-chip analog comparator

1.2.2 Clock configuration

The CPU clock should be configured to a smaller value than the maximum MCLK value that can be set. This is required so that when the clock is running slower than expected, there is room for adjustment.

In this example, the CPU clock is set to run at 16 MHz. The peripheral clock is set to run at 32 MHz, which is twice the frequency of the CPU clock.

The code example below includes the code for disabling the CCU4 clock gating.

```
typedef union CLKCR_TAG
{
        struct
        {
                uint32_t FDIV      :8;
                uint32_t IDIV      :8;
                uint32_t PCLKSEL   :1;
                uint32_t RTCCLKSEL :3;
                uint32_t CNTADJ    :10;
                uint32_t VDDC2LOW  :1;
                uint32_t VDDC2HIGH :1;
        }B;     /* Bit-addressing */

        uint32_t ALL;
}CLKCR_T;
CLKCR_T config_CLKCR;

void SCU_Clock_Init (void)
{
        config_CLKCR.B.FDIV             = 0;
        config_CLKCR.B.IDIV             = 2;   /* MCLK = 16MHz */
        config_CLKCR.B.PCLKSEL          = 1;
        config_CLKCR.B.RTCCLKSEL        = 0;
        config_CLKCR.B.CNTADJ           = 0x3FF;
        config_CLKCR.B.VDDC2LOW         = 0;
        config_CLKCR.B.VDDC2HIGH        = 0;

        SCU_GENERAL->PASSWD = 0x000000C0UL;
        SCU_CLK->CLKCR = config_CLKCR.ALL;     /* Config SCU Clock */
        while((SCU_CLK->CLKCR)&0x40000000UL);  /* wait for VDDC to stabilize */
        SCU_GENERAL->PASSWD = 0x000000C3UL;

        SCU_GENERAL->PASSWD = 0x000000C0UL;
        SCU_CLK->CGATCLR0 |= 0x04;              /* Disable CCU4 Gating */
        while((SCU_CLK->CLKCR)&0x40000000UL);  /* wait for VDDC to stabilize */
        SCU_GENERAL->PASSWD = 0x000000C3UL;
}
```

1.2.3 Capture Compare Unit 4 (CCU4)

CCU4 is used to capture the frequency of the reference clock. The reference clock is fed into the input pin of the CCU4 timer slice and the period of the reference clock is captured.

In this example, the CCU4 timer slice is configured to capture on every rising edge of the signal and the capture is performed continuously. We need an interrupt for every period captured, so an interrupt is enabled for the capture event. The following code example shows the CCU4 timer slice initialization code.

```
void CCU40_CC41_init(void)
{
    /* Prescaler Run Bit Set */
    CCU40->GIDLC    |= CCU4_GIDLC_SPRB_Msk;
    CCU40_CC41->INS |= 0x03;

    /* Configure Event 0 to be active on rising edges */
    CCU40_CC41->INS |= 0x90000;

    /* External Capture on Event 0 */
    CCU40_CC41->CMC |= 0x0090;

    /* Timer is cleared on a capture event */
    CCU40_CC41->TC = 0x1048;

    /* Event 0 interrupt */
    CCU40_CC41->INTE = 0x0100;

    /* Interrupt is forward to CC4ySR0 */
    CCU40_CC41->SRS = 0x0100;

    /* Configure NVIC for interrupt */
    NVIC_SetPriority(CCU40_1_IRQn, 20);
    NVIC_EnableIRQ(CCU40_1_IRQn);
}
```

With this setting, the period of the clock signal is captured and an interrupt is generated when the capture is made. In this example, the timer resolution is set to 32 MHz (peripheral clock), and the captured value for the period of a 32.768 kHz reference clock is expected to be 977 (32 MHz / 32.768 kHz).

In the Interrupt Service Routine (ISR), the captured value is read out and the adjustment of the fractional divider of the MCLK is based on the conditions stated below. Changing the fractional divider can be performed at any time without causing any spikes or glitches to MCLK.

- If the captured value is below 975 (MCLK lower than expected), increase MCLK by decrementing FDIV
- If the captured value is above 980 (MCLK higher than expected), decrease MCLK by incrementing FDIV

The following code example shows the capture event Interrupt Service Routine:

```
#define PERIOD_HI          980
#define PERIOD_LO          975
#define FDIV_MAX           20
#define FDIV_MIN           0
uint32_t Period_cnt;
void CCU40_1_IRQHandler (void)
{
  Period_cnt = CCU40_CC41->CV[1];
  Period_cnt &= 0x000FFFF;
  config_CLKCR.ALL = SCU_CLK->CLKCR;
    if(Period_cnt > PERIOD_HI)
    {
    /* If the MCLK frequency is too high, reduce MCLK frequency by FDIV++ */
    /* Set boundary limits to prevent over adjustment of MCLK frequency */
      if(config_CLKCR.B.FDIV < FDIV_MAX)
      {
        config_CLKCR.B.FDIV++;
        SCU_GENERAL->PASSWD = 0x000000C0UL;
        SCU_CLK->CLKCR = config_CLKCR.ALL;
        while((SCU_CLK->CLKCR)&0x40000000UL); /*wait for VDDC to stabilize*/
        SCU_GENERAL->PASSWD = 0x000000C3UL;
       }
     }
     else if(Period_cnt < PERIOD_LO)
     {
     /* If the MCLK frequency is too low, increase MCLK frequency by FDIV--
*/
     if(config_CLKCR.B.FDIV > FDIV_MIN)
       {
         config_CLKCR.B.FDIV--;
         SCU_GENERAL->PASSWD = 0x000000C0UL;
         SCU_CLK->CLKCR = config_CLKCR.ALL;
         while((SCU_CLK->CLKCR)&0x40000000UL);/*wait for VDDC to stabilize*/
         SCU_GENERAL->PASSWD = 0x000000C3UL;
```

```
            }
        }
}
```

The code above is an example for generic applications targeting a CPU clock configured to a smaller value than the maximum MCLK value that can be set (as highlighted in section 1.2.2). Specific implementations may require adaptations to handle boundary conditions, such as the situation of a broken wire or a missing external clock.

In such cases the application should implement additional steps and provide handling at a higher application level, according to the system requirements.

The following figure shows the addition of a lower and upper limit for the FDIV value. Such limits should be implemented in the application software to improve the robustness of operation in multiple application scenarios.

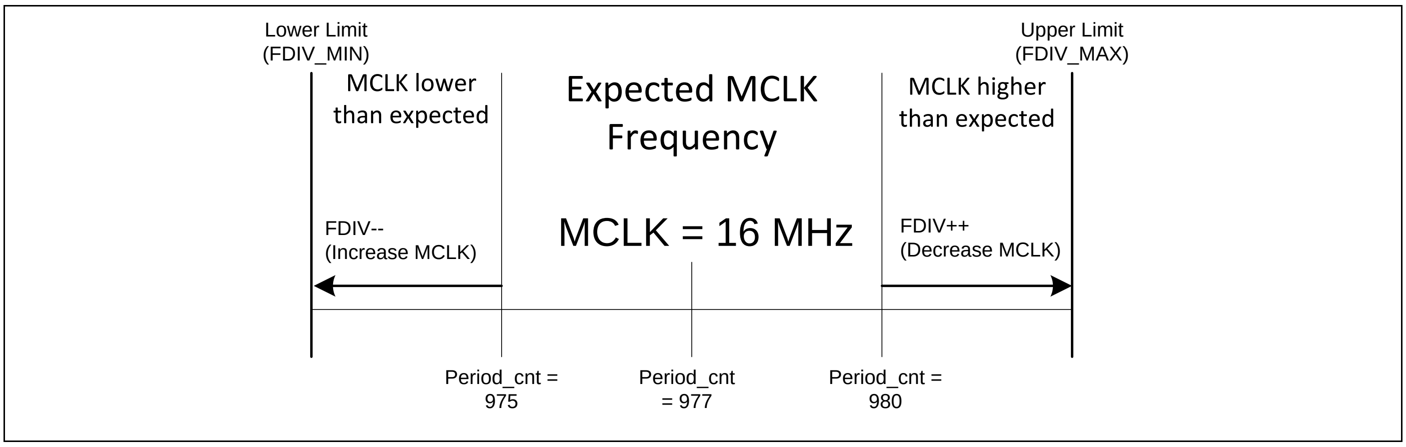


Figure 9 An example on the lower/upper limit adjustment of FDIV for 16 MHz MCLK

1.2.4 External Request Unit (ERU)

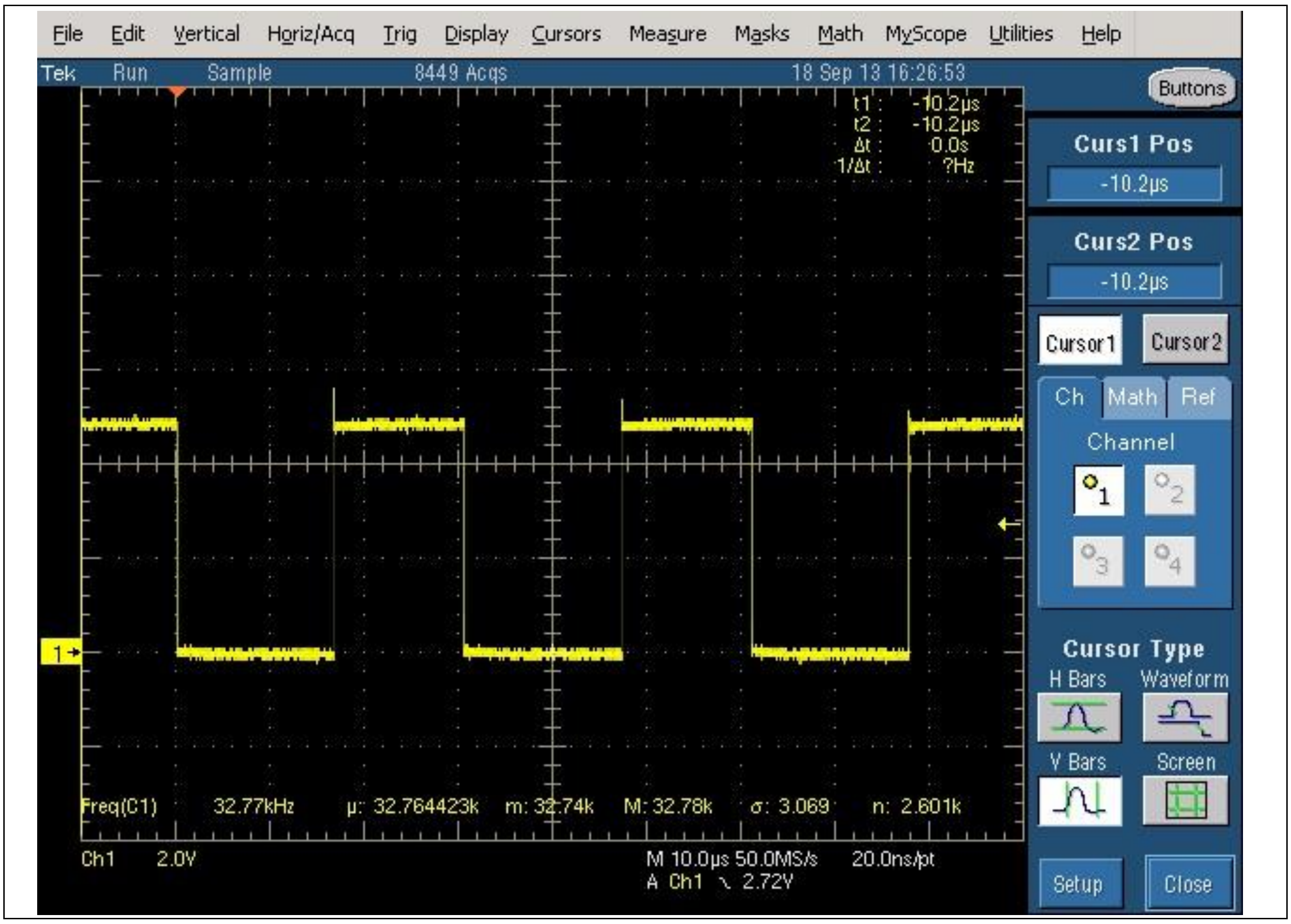
To reduce the use of the CCU4 input pin, the output from the analog comparator (ACMP0.OUT) can be connected internally to the CCU4 input via the ERU module. In our example, ACMP0.OUT is connected to ERU0.0A0 input, and this signal is routed to ERU output ERU0.PDOUT0. This signal is then fed into CCU4 input.

```
void ERU0_Init(void)
{
        ERU0->EXISEL &= 0xFFFC;        /* Event S0A from input 0A0 */
        ERU0->EXICON[0] |= 0x0007;     /* Rising edge trigger, Output to OGU0 */
        ERU0->EXOCON[0] |= 0x1024;     /* Enable event detection */

}
```

1.3 Reference clock output

The reference clock (generated from the oscillator circuit) takes about 5 mS to stabilize after power on. Once the reference clock is stable, there is only minimal fluctuation in frequency or noise.



Tek Run Sample 8449 Acqs 18 Sep 13 16:26:53

t1 : -10.2µs
t2 : -10.2µs
Δt : 0.0s
1/Δt : ?Hz

Curs1 Pos -10.2µs
Curs2 Pos -10.2µs

Freq(C1) 32.77kHz µ: 32.764423k m: 32.74k M: 32.78k σ: 3.069 n: 2.601k

Ch1 2.0V M 10.0µs 50.0MS/s 20.0ns/pt
A Ch1 ↘ 2.72V

Figure 10 32.768 kHz reference clock

1.4 Temperature test

A temperature test has been conducted to measure the accuracy of the oscillator over the range of -40°C to 105°C with the clock calibration methodology shown in this application note.

1.4.1 Test setup

The XMC1200 bootkit is used in the test. The device is configured to run the calibration continuously and output a 100 kHz PWM signal at the same time.

The bootkit is placed in the temperature chamber and the PWM output connected to an oscilloscope outside the temperature chamber.

The test starts at room temperature (25°C), and slowly ramped up to 105°C, and was then kept at 105°C for 15 minutes. The temperature was then adjusted to slowly ramp down to -40°C and to remain at that temperature for 15 minutes.

The fluctuation on the PWM signal was observed.

1.4.2 Test result

The maximum frequency observed during the temperature test on the 100 kHz PWM signal was 100.4 kHz and the minimum frequency observed was 99.9 kHz. Therefore, we can conclude that the fluctuation of the oscillator for temperature -40°C to 105°C is within 0.5%.

Attention: ***The test was performed on 3 device units under laboratory conditions. The test results shared in this documentation are for reference purposes only.***

1.5 Conclusion

Because the reference clock is periodically captured, any changes in the system clock can be adjusted back to the required value. With this implementation, the accuracy of the system clock can improve upon internal oscillator characteristics.

1.6 Extension for applications requiring maximum frequency of 32 MHz

As mentioned in section 1.2.2, the fractional divider approach is primarily using a divider and can only be used when targeting frequencies lower than the maximum MCLK value that can be set. For cases where for example, 32 MHz needs to be reached, alternative method that make use of the ANAOFFSET register can be used.

The ANAOFFSET register provides the means to adjust the DCO with an additional offset in both positive and negative directions. The detailed description, range and settings can be found in the ANAOFFSET register definition, in the SCU chapter of the Reference Manual.

The basic principle is outlined in the following figure:

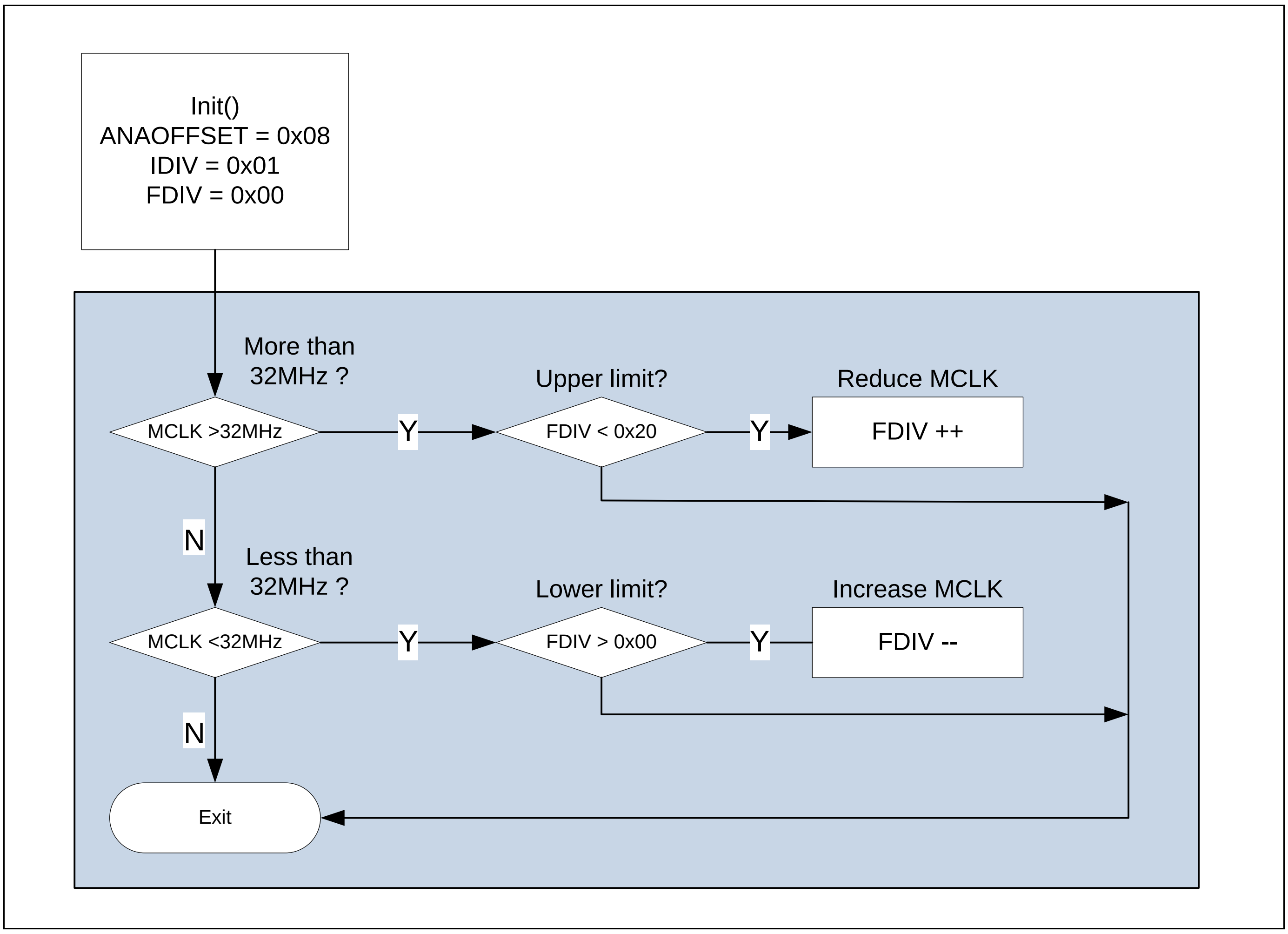


Figure 11 Basic principle of using ANAOFFSET for MCLK=32 MHz

Note: The same application specific checks on boundary conditions may be required here as, as described in section 1.2.3.

2 DCO1 calibration with respect to temperature

The clock frequency drifts when the temperature changes. The data sheet states that the internal oscillator (DCO1) accuracy for the XMC1000 family production device could drift in the range of -3.9% to 4% with respect to the whole temperature range (-40°C to 105°C). The DCO1 clock frequency falls at cold temperature and rises with higher temperatures. The XMC1000 family devices therefore provide a solution to calibrate the internal oscillator during runtime with respect to temperature changes. For a more accurate clock, external crystal support is available in XMC1400.

The die temperature can be measured with the XMC1000 on-chip temperature sensor.

Based on the measured temperature, an offset value can be obtained for the DCO1 calibration. The offset adjustment for DCO1 is made via the ANAOFFSET register.

The following code examples are based on XMC1300.

2.1 Calibration parameters

Parameters are defined in the flash configuration sector for the DCO1 calibration, located in the flash address range 0x10000F30 to 0x10000F33. These parameters must be defined in the project in order to use the calibration code.

```
#define CONFIG_BASE 0x10000000
#define ANA_TSE_T1        (CONFIG_BASE + 0x0F30)
#define ANA_TSE_T2        (CONFIG_BASE + 0x0F31)
#define DCO_ADJLO_T1      (CONFIG_BASE + 0x0F32)
#define DCO_ADJLO_T2      (CONFIG_BASE + 0x0F33)
```

Attention: *Only production devices (No 'EE' or 'ES' within the device marking) have these parameters defined. DCO calibration can only be applied on production devices. Applying calibration software to ES or EE devices is undefined.*

2.2 Initialization code

Initialize the calibration code by calling the software function TSE_Calibration_Init(). This function enables the on-chip temperature sensor and assigns the parameters.

```
int32_t var_ANA_TSE_T1;
int32_t var_ANA_TSE_T2;

void TSE_Calibration_Init (void)
{
        int32_t TSE1, TSE2;

        /* Enable Temperature sensor */
        SCU_ANALOG->ANATSECTRL = 0x01;

        /* Get parameters from flash config sector */
        TSE1 = *((uint8_t*)ANA_TSE_T1);
        TSE2 = *((uint8_t*)ANA_TSE_T2);

        /* Check for device parameters */
        if((TSE1 == 0) && (TSE2 == 0))
        {
                var_ANA_TSE_T1 = 25;
                var_ANA_TSE_T2 = 115;
        }
        else
        {
                var_ANA_TSE_T1 = TSE1;
                var_ANA_TSE_T2 = TSE2;
        }
}
```

2.3 Calibration routine

The calibration routine is provided in the software function TSE_DCO_Calibration().

TSE_DCO_Calibration() is called during run time to calibrate the internal oscillator. One method of running it is to periodically call it with a system timer interrupt.

The device temperature is measured when the routine runs. A temperature sensor library file is required in order to obtain the temperature measurement from the on-chip temperature sensor. Once the current temperature has been measured, it is used to calculate the new offset value. The offset calculation described in this document includes rounding the calculated value to the nearest integer.

This new offset value must be programmed into the ANAOFFSET register if it is different from the previous offset value.

- If the calculated offset value is less than 0, then use 0.
- If the calculated offset value is greater than 8, then use 8.

The internal oscillator frequency is adjusted automatically if the new offset value is different from the previous offset value.

```
#define PASSWD_OPEN_ACCESS          (0x000000C0)

void TSE_DCO_Calibration (void)
{
        int32_t a,b,c,d,e;
        int32_t ADJL_OFFSET;

        a = *((uint8_t*)DCO_ADJLO_T2);
        b = *((uint8_t*)DCO_ADJLO_T1);
        d = var_ANA_TSE_T1;
        e = var_ANA_TSE_T2;

        /* Get current temperature and convert to celcius */
        c = XMC1000_CalcTemperature();
        c = c - 273;

        /* Calculate the offset value */
        ADJL_OFFSET = ((a-b)*(c-d) + ((e-d)>>1) + b*(e-d))/(e-d);

        /* Configure the offset value into ANAOFFSET register */
        if(ADJL_OFFSET >= 0 && ADJL_OFFSET <= 8)
        {
          SCU_GENERAL->PASSWD = PASSWD_OPEN_ACCESS;
  WR_REG( SCU_ANALOG->ANAOFFSET,
                        SCU_ANALOG_ANAOFFSET_ADJL_OFFSET_Msk,
                        SCU_ANALOG_ANAOFFSET_ADJL_OFFSET_Pos, ADJL_OFFSET);
        }
}
```

2.4 Temperature test for calibration code

A temperature test has been conducted on a microcontroller with and without the calibration code. The test covered the temperature range from -40°C to 105°C. The device outputs a 100 kHz PWM signal and the frequency variation on this PWM signal was observed.

2.4.1 Test setup

The XMC1302-T038X0200 microcontroller was tested in the temperature chamber. First, the microcontroller was tested without the calibration code and then tested again with the calibration code added.

The test started at room temperature (25°C), before slowly ramping up to 105°C and being kept there for 15 minutes. The temperature was then slowly ramped down to -40°C and remained at that temperature for 15 minutes.

The frequency variation on the PWM signal throughout the test was observed.

2.4.2 Test result

The graph shows the frequency variations of the 100 kHz PWM signal over the temperature range.

Without calibration code

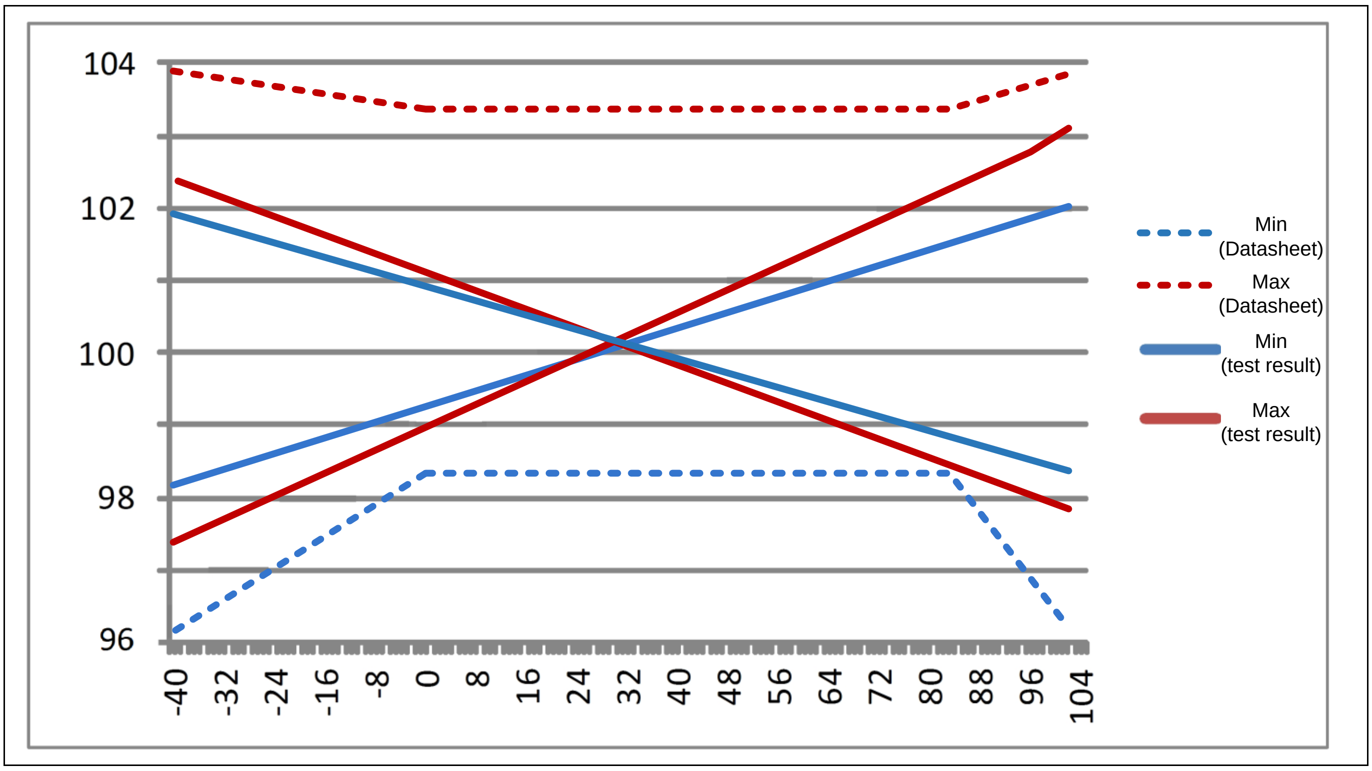


Figure 12 100 kHz PWM output without DCO1 calibration

With calibration

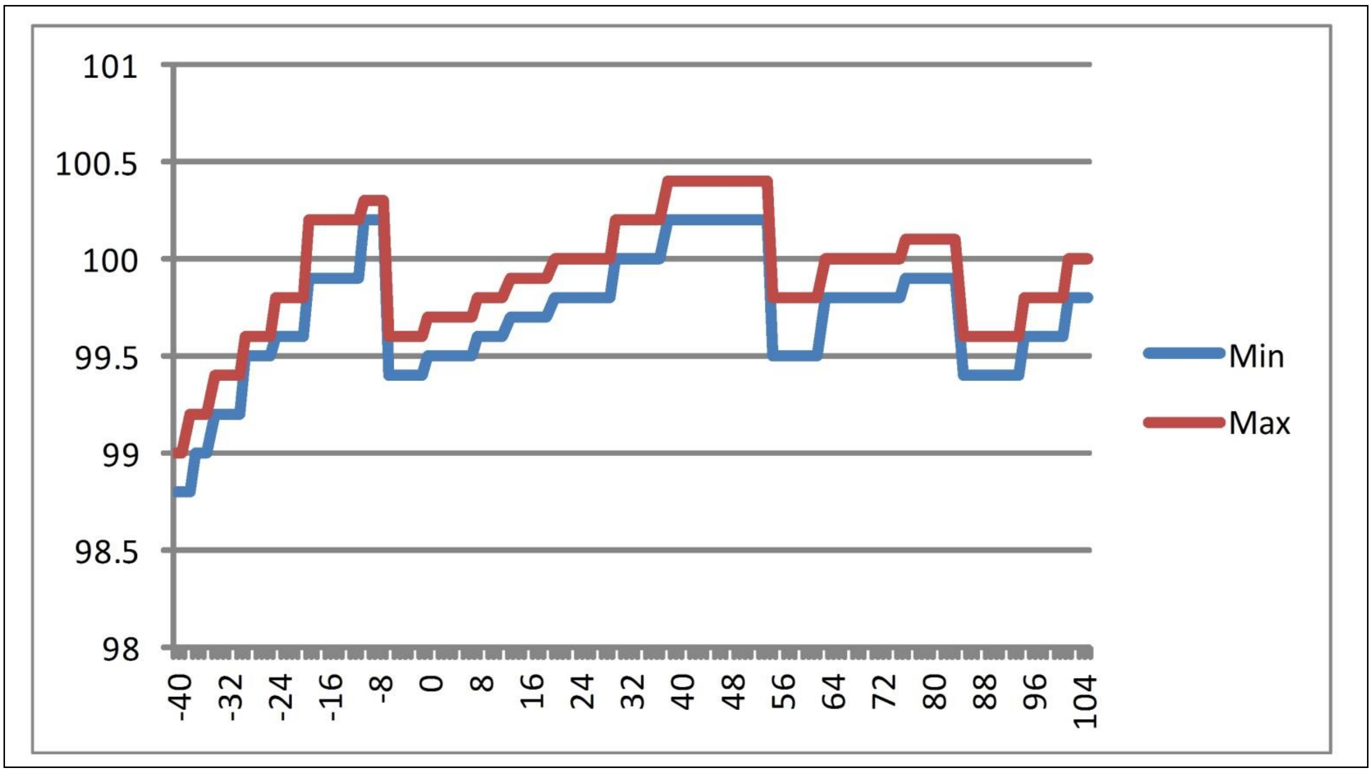


Figure 13 100 kHz PWM output with DCO1 calibration

From the observation, the frequency of the 100 kHz PWM can vary as high as 101.4 kHz at 105°C and as low as 98.1 kHz at -40°C without the calibration code.

After adding the calibration code, the highest frequency observed was 100.4 kHz at 38°C – 53°C and the lowest frequency observed was 98.8 kHz at -40°C.

The conclusion is that frequency variation is significantly reduced after adding the calibration code.

Typically the clock accuracy with respect to the nominal frequency is expected to be improved by a factor of 2 for temperatures above 0°C.

References

Infineon: XMC1000 Reference Manual, http://www.infineon.com/XMC1000 Tab: Documents
Infineon: XMC1000 Data sheet, http://www.infineon.com/XMC1000 Tab: Documents
Infineon: DAVE™, http://www.infineon.com/DAVE

Revision history

Major changes since the last revision

Page or reference	Description of change
All	Changed in Application Note template.
Chapters 1.1 & 2	Added availability of external crystal support for a more accurate clock in XMC1400
Chapter 1.1	Enhanced the description of the example
Chapter 1.2	Included values for the example of reference clock circuit based on the crystal used
Chapter 1.2.1	Enhanced description on functions of resources used for MCLK adjustment in code examples
Chapter 1.2.2	Enhanced code example
Chapter 1.2.3	Enhanced description of conditions for captured value for fractional divider adjustment, code example and Figure 9
Figure 12	Enhanced the figure with XMC1100/XMC1200/XMC1300 Datasheet values

Edition 2016-04-22

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32321