発振器の取り扱い

XMC1000

本書について

適用範囲と目的

このアプリケーションノートでは、XMC1000 マイクロコントローラファミリのオンチップ発振器の周波数のキャリブレーションについて説明します。内部クロックをキャリブレーションする 2 つの方法について説明します。

- 外部リファレンス信号の使用
- オンチップ温度センサーの使用対象者

対象者

本書は、XMC1000 内部発振器の精度を向上させたいエンジニアまたは開発者を対象とします。

目次

本書について 1
目次 2
1 外部リファレンス クロックによるシステムクロックのキャリブレーション 3
1.1 概要 3
1.2 実装 5
1.2.1 リファレンス クロックを生成する発振回路 5
1.2.1.1 発振回路例 A 6
1.2.1.2 発振回路例 B 7
1.2.1.3 発振回路例 C 8
1.2.2 クロックの設定 9
1.2.3 キャプチャ コンペア ユニット 4 (CCU4) 10
1.2.4 外部リクエストユニット (ERU) 12
1.3 リファレンス クロック出力 13
1.4 温度テスト 14
1.4.1 テストのセットアップ 14
1.4.2 テスト結果 14
1.5 結論 14
1.6 最大周波数 32 MHz を必要とするアプリケーション向けの拡張 14
2 温度に対する DCO1 のキャリブレーション 16
2.1 キャリブレーション パラメータ 16
2.2 初期化コード 17
2.3 キャリブレーション ルーチン 18
2.4 キャリブレーション コードの温度テスト 19
2.4.1 テストのセットアップ 19
2.4.2 テスト結果 19
参考資料 21
改訂履歴 22
免責事項 23

1 外部リファレンス クロックによるシステムクロックのキャリブレーション

1.1 概要

XMC1000 ファミリのデバイスは、システムクロックを生成するための内部発振器を提供しています。このクロックは CPU とオンチップの周辺回路に使用できます。XMC1400 では、システムクロックの生成のために外部水晶振動子もサポートされています。

内部クロックの周波数は、電源電圧または温度が変化すると変化します。システムクロックの精度が重要になる XMC1100, XMC1200, または XMC1300 のアプリケーションでは、実行時に内部クロックをキャリブレーションすることができます。このキャリブレーションは、クロック制御ユニットのフラクショナル分周器を使用して MCLK を調整することによって実行されます。ただし、このキャリブレーション方法は XMC1400 には推奨されません。かわりに、外付けの水晶振動子を使用して、より正確なクロックを実現できます。

このアプリケーションノートでは、外部リファレンスクロックを使用して内部クロックをキャリブレーションする例を紹介します。外部リファレンスクロック (水晶発振器や安定したクロックソースなど) がマイクロコントローラに供給され、その周波数がキャプチャ コンペア ユニット 4 (CCU4) によってキャプチャされます。周波数をキャプチャした後、キャリブレーションソフトウェアはシステムクロックがまだ正しい範囲内で動作しているかどうかを判断します。システムクロックが範囲外の場合、フラクショナル分周器はそれに応じて調整されます。

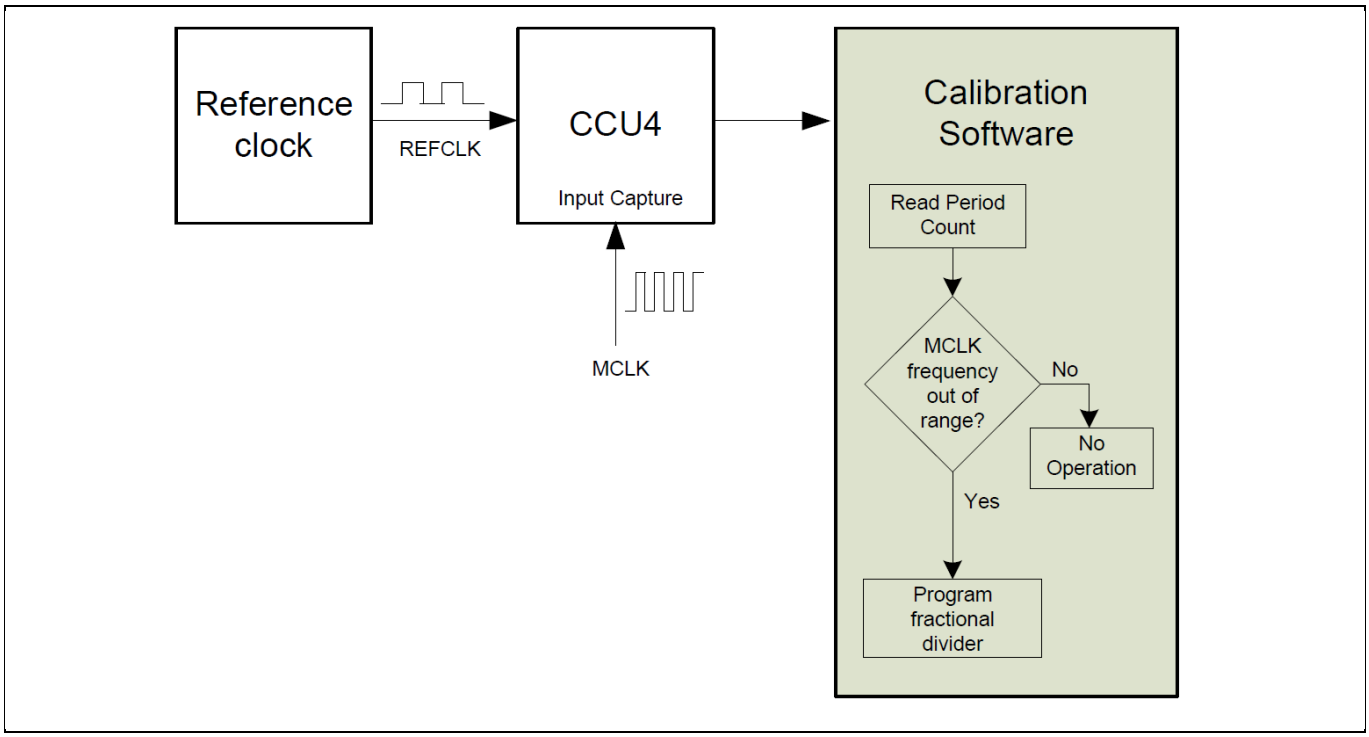


Figure 1 内部クロックを外部リファレンスクロックでキャリブレーションする例

1 外部リファレンス クロックによるシステムクロックのキャリブレーション

次の図に、キャリブレーション用のリファレンスクロックを提供するために使用可能なリファレンスクロック回路の例を示します。この例では、外部水晶振動子を使用しています。

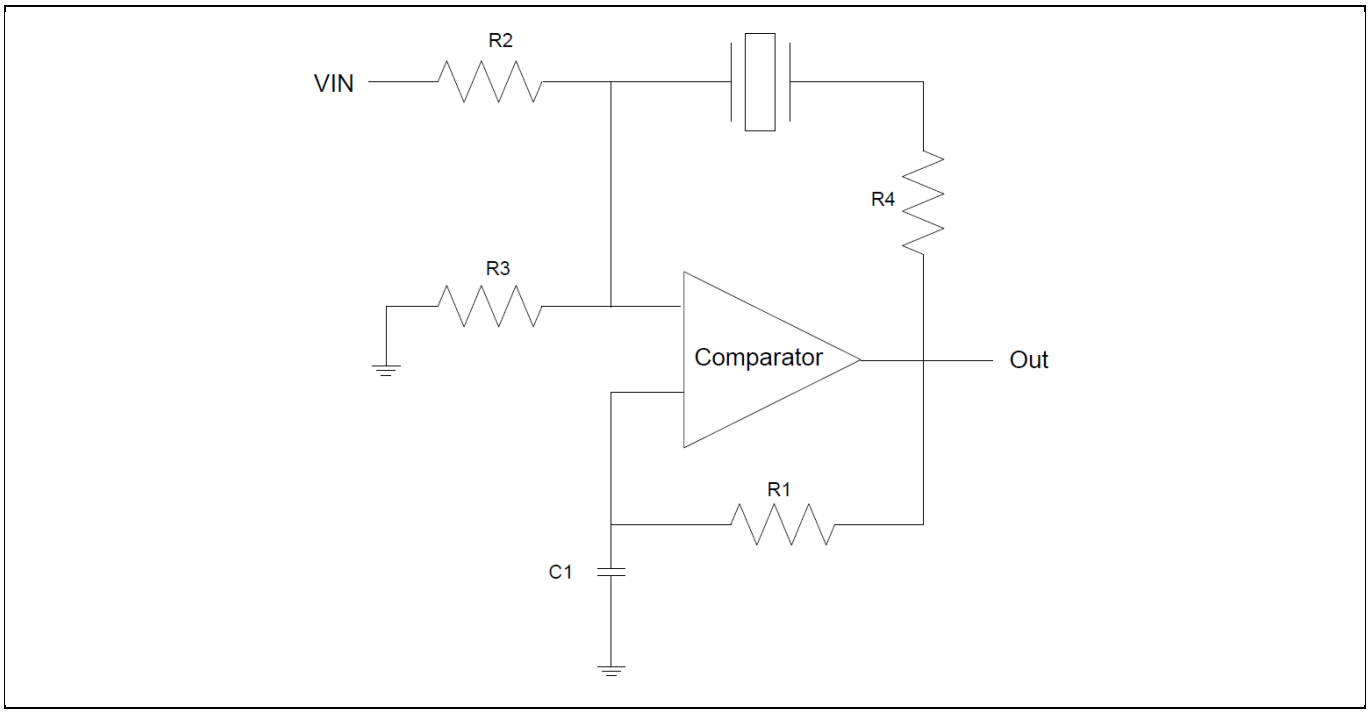


Figure 2　リファレンスクロック回路の例

水晶振動子をリファレンスクロックとして使用する代わりに、任意の安定したクロックソースを使用することもできます。例えば、XMC1000 をスレーブコントローラとして使用し、マスターが高性能マイクロコントローラ (XMC4000 など) であるシステムでは、マスターからの正確なクロック出力を XMC1000 の CCU4 入力へのリファレンスクロックとして使用できます。

Note:　このドキュメントで説明されている回路と MCLK のキャリブレーション方法は、実装例としてのみ提供されています。この実装に従う場合は、使用する水晶振動子の種類によって特性が異なるため、特に発振器の起動時間と安定性について、実際のボードを検証し、微調整する必要があります。

1.2 実装

次のサンプルコードは、XMC1200 に基づいています。

1.2.1 リファレンス クロックを生成する発振回路

Figure 2 に示すような外部発振回路の例は、安定したリファレンスクロック信号を確保するために必要です。以下では、発振回路で使用される各タイプのリファレンスクロック回路に対応する部品のマッチング値を示します。これらの例では、32.768 kHz のクロック信号を生成する発振回路を使用します。利用可能な場合は、オンチップのアナログコンパレータを発振回路の一部として使用できます。

この例では、以下のとおりです。

- ACMP0 は、発振回路をエミュレートするために使用されます。
- ピン P0.10 は、安定したクロックソースを出力する ACMP0.OUT として使用されます。
- P2.8 は、発振器の負入力である ACMP0.INN として使用されます。
- P2.9 は、発振器の正入力である ACMP0.INP として使用されます。

ACMP と GPIO を初期化するサンプルコードを以下に示します。

```
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 DW1_Ch_IntHandler(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) ;
}
```

実装時の検討のために、3 つの発振回路例を示します。

1.2.1.1 発振回路例 A

次の図に、セイコーの FC-13A 32.768 kHz 水晶振動子に対応した部品の値を示します。

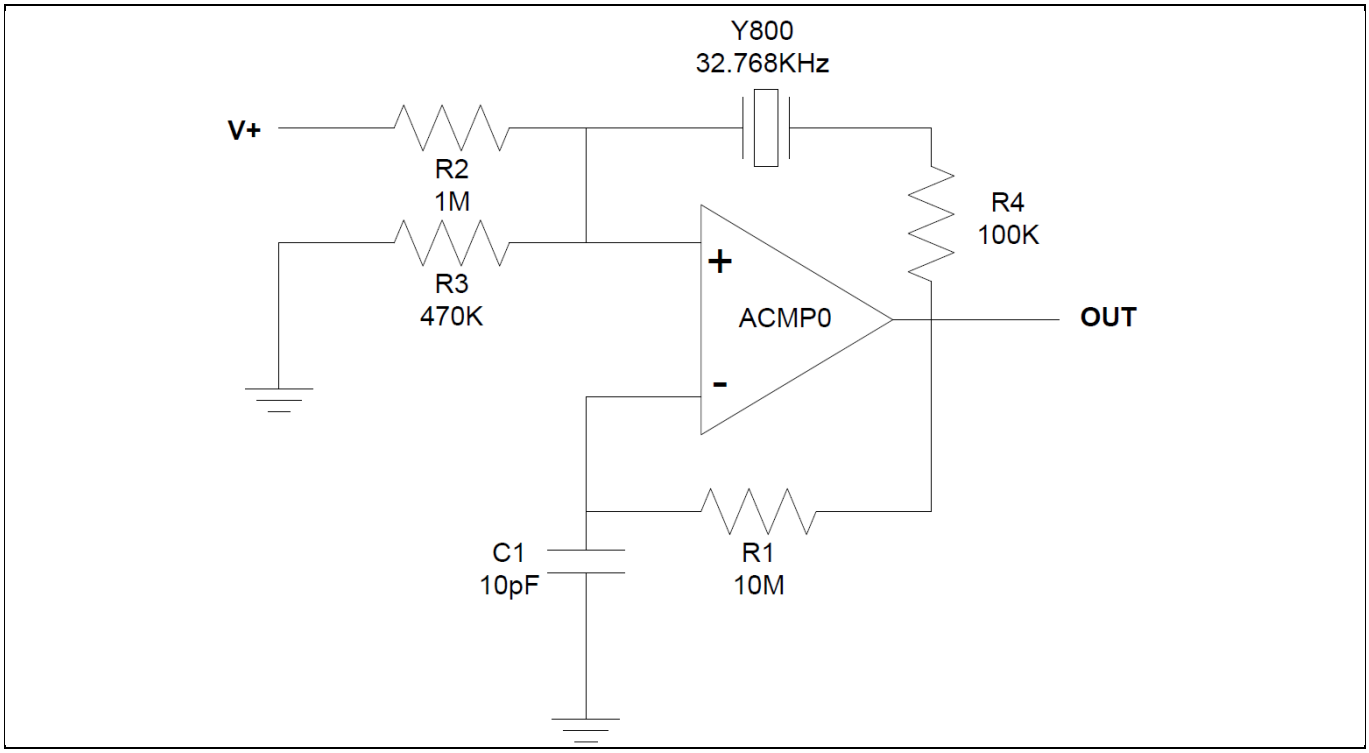


Figure 3 セイコー水晶振動子を使用した 32kHz 発振回路

下図に、オンチップコンパレータを使用した発振回路を示します。

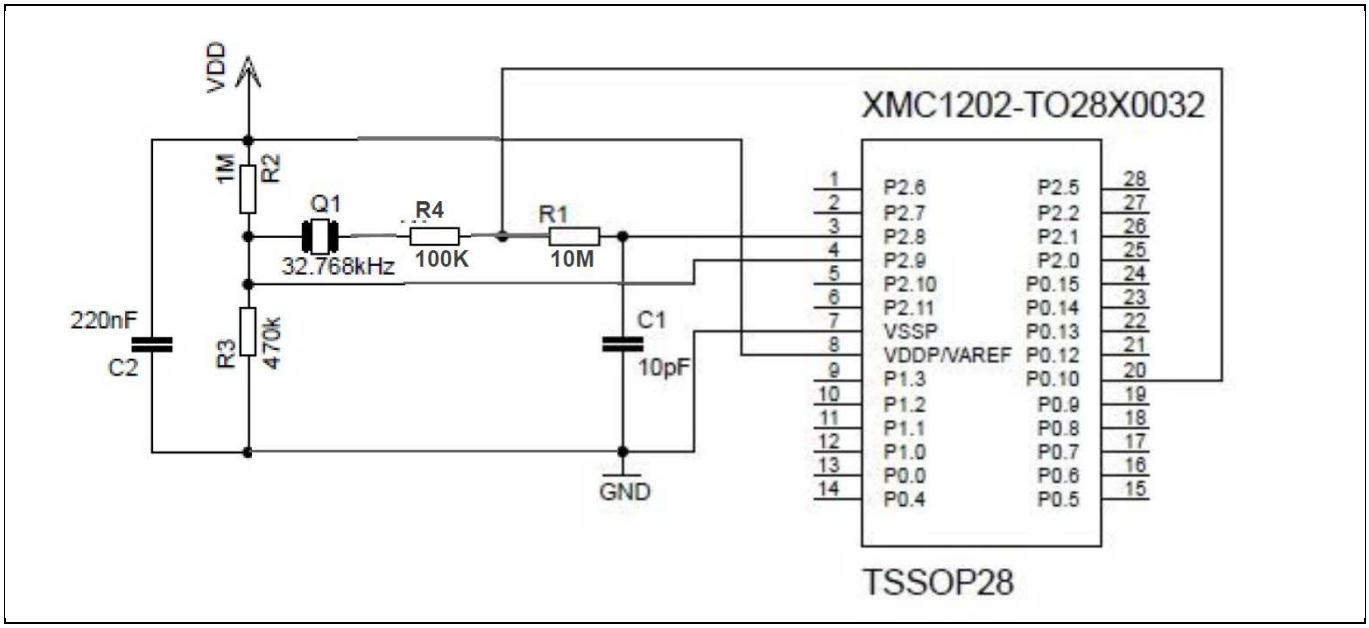


Figure 4 XMC1200 のオンチップアナログコンパレータを使用した発振回路

1.2.1.2 発振回路例 B

次の図に、Hong Kong X'tals Limited の 2060K3276IB00 および 2060K3276IB01 水晶振動子に対応した部品の値を示します。

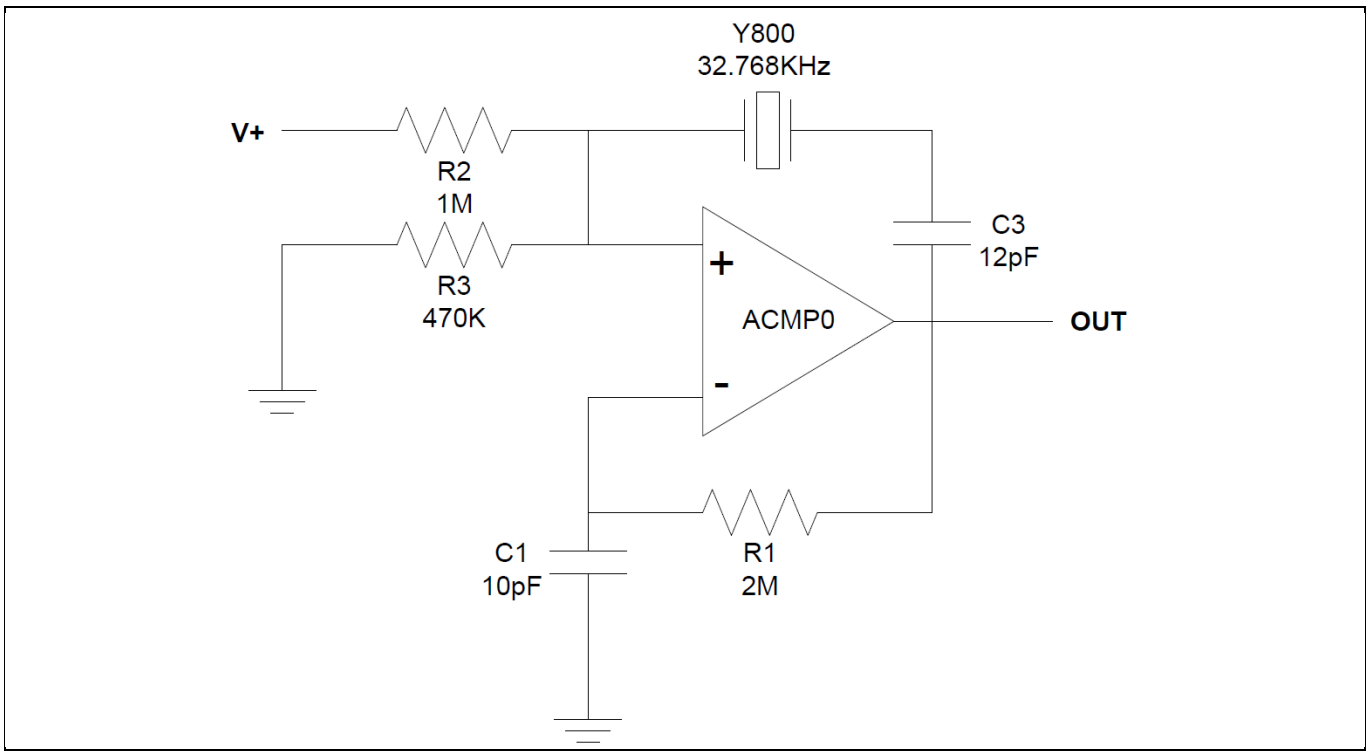


Figure 5 Hong Kong X'tals Limited の水晶振動子を使用した 32 kHz の発振回路

下図に、オンチップ コンパレータを使用した発振回路を示します。

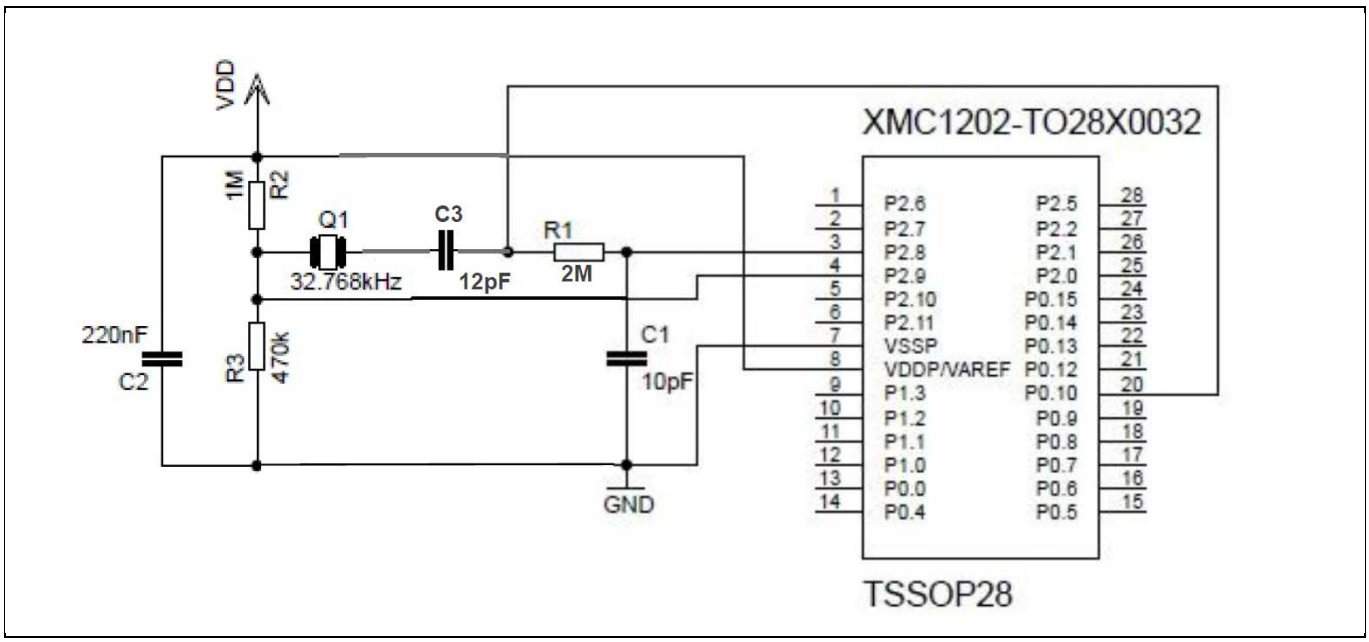


Figure 6 XMC1200 のオンチップアナログコンパレータを使用した発振回路

1.2.1.3 発振回路例 C

次の図に、セイコーの FC-13A 32.768 kHz 水晶振動子に対応した部品の値を示します。

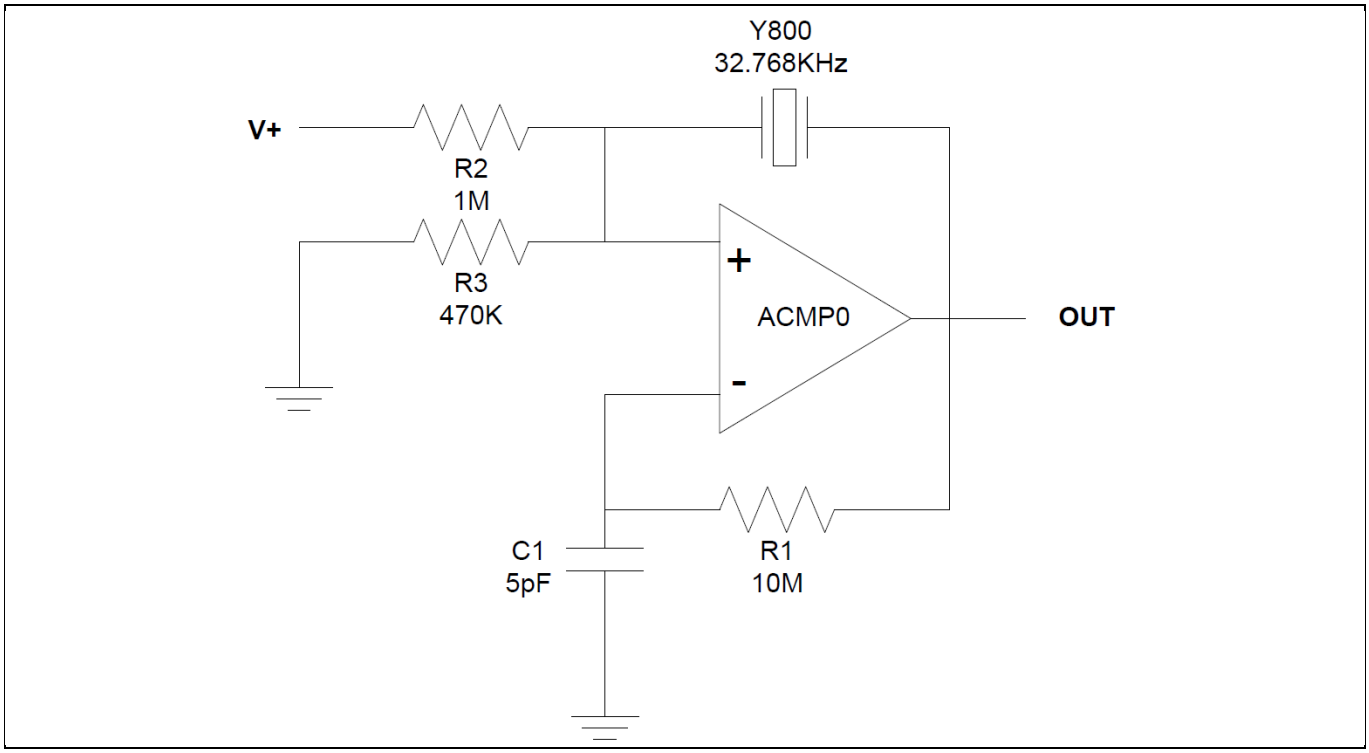


Figure 7 セイコー水晶振動子を使用した 32kHz 発振回路

下図に、オンチップコンパレータを使用した発振回路を示します。

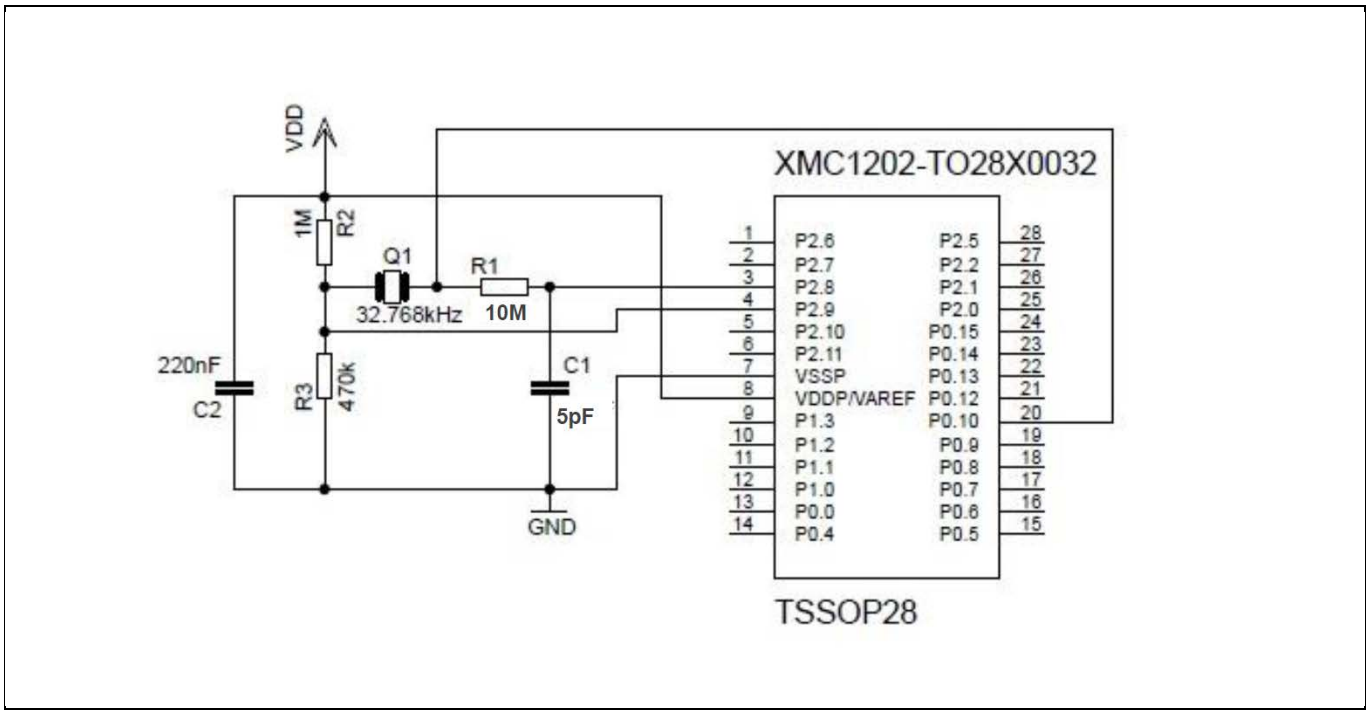


Figure 8 XMC1200 のオンチップアナログコンパレータを使用した発振回路

1.2.2 クロックの設定

CPU クロックは、設定可能な最大 MCLK 値よりも小さい値に設定する必要があります。これは、クロックが予想より遅く動作している場合に、調整の余地を残すために必要です。

この例では、CPU クロックは 16 MHz で動作するように設定されています。周辺クロックは、CPU クロックの 2 倍の周波数である 32 MHz で動作するように設定されています。

以下のサンプルコードには、CCU4 クロックゲーティングを無効にするコードが含まれています。

```
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 キャプチャ コンペア ユニット 4 (CCU4)

CCU4 はリファレンスクロックの周波数をキャプチャするために使用されます。リファレンスクロックは CCU4 タイマーの入力ピンに供給され、リファレンスクロックの周期がキャプチャされます。

この例では、CCU4 タイマーは信号の立ち上りエッジごとにキャプチャするように設定されており、キャプチャは継続的に実行されます。キャプチャされた期間ごとに割込みが必要なため、キャプチャイベントに対して割込みが有効にされます。次のサンプルコードは、CCU4 タイマーの初期化コードを示しています。

```
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);
}
```

1 外部リファレンス クロックによるシステムクロックのキャリブレーション

この設定により、クロック信号の周期がキャプチャされ、キャプチャが行われると割込みが発生します。この例では、タイマー分解能は 32 MHz (周辺クロック) に設定されており、32.768 kHz のリファレンスクロックの周期でキャプチャされた値は 977 (32 MHz / 32.768 kHz) と期待されます。

割込みサービスルーチン (ISR) では、キャプチャされた値が読み出され、MCLK のフラクショナル分周器の調整は、以下に示す条件に基づいて行われます。フラクショナル分周器の変更は、MCLK にスパイクやグリッチを発生させることなく、いつでも実行できます。

- キャプチャされた値が 975 を下回る (MCLK が期待よりも低い) 場合は、FDIV をデクリメントして MCLK を増やします。
- キャプチャされた値が 980 を超える (MCLK が期待よりも高い) 場合は、FDIV をインクリメントして MCLK を減らします。

次のサンプルコードは、キャプチャ イベントの割込みサービスルーチンを示しています。

```
#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;
      }
    }
}
```

上記のコードは、設定可能な最大 MCLK 値 (セクション 1.2.2 でハイライトされた) よりも小さい値に設定された CPU クロックを対象とする一般的なアプリケーションの例です。特定の実装では、断線や外部クロックの欠落などの境界条件を処理するために変更が必要になる場合があります。

このような場合、アプリケーションはシステム要件に応じて追加の手順を実装し、より高いアプリケーションレベルでの処理を提供する必要があります。

次の図に、FDIV 値の下限と上限の追加を示します。このような制限は、複数のアプリケーションシナリオでの動作の堅牢性を向上させるために、アプリケーションソフトウェアに実装する必要があります。

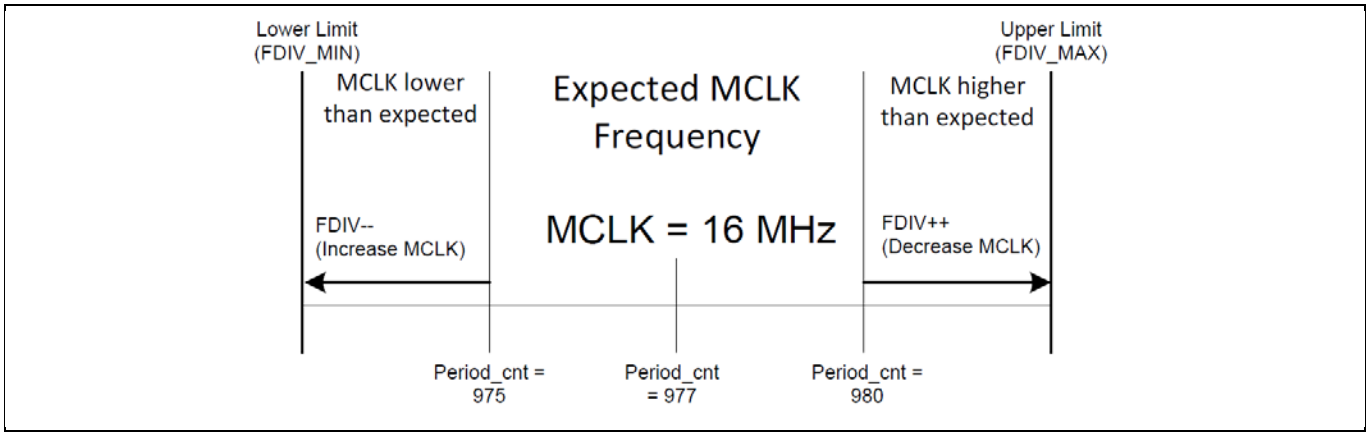


Figure 9　16 MHz MCLK における FDIV の下限/上限調整例

1.2.4　外部リクエストユニット (ERU)

CCU4 入力ピンの使用を減らすために、アナログコンパレータからの出力 (ACMP0.OUT) は、ERU モジュールを介して CCU4 入力に内部的に接続できます。この例では、ACMP0.OUT が ERU0.0A0 入力に接続されることで、この信号は ERU の出力である ERU0.PDOUT0 にルーティングされます。これによりこの信号は CCU4 入力に送られます。

```
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 リファレンス クロック出力

リファレンスクロック (発振回路から生成) は、電源投入後、安定するまでに約 5 ms かかります。リファレンスクロックが安定すると、周波数やノイズの変動は最小限に抑えられます。

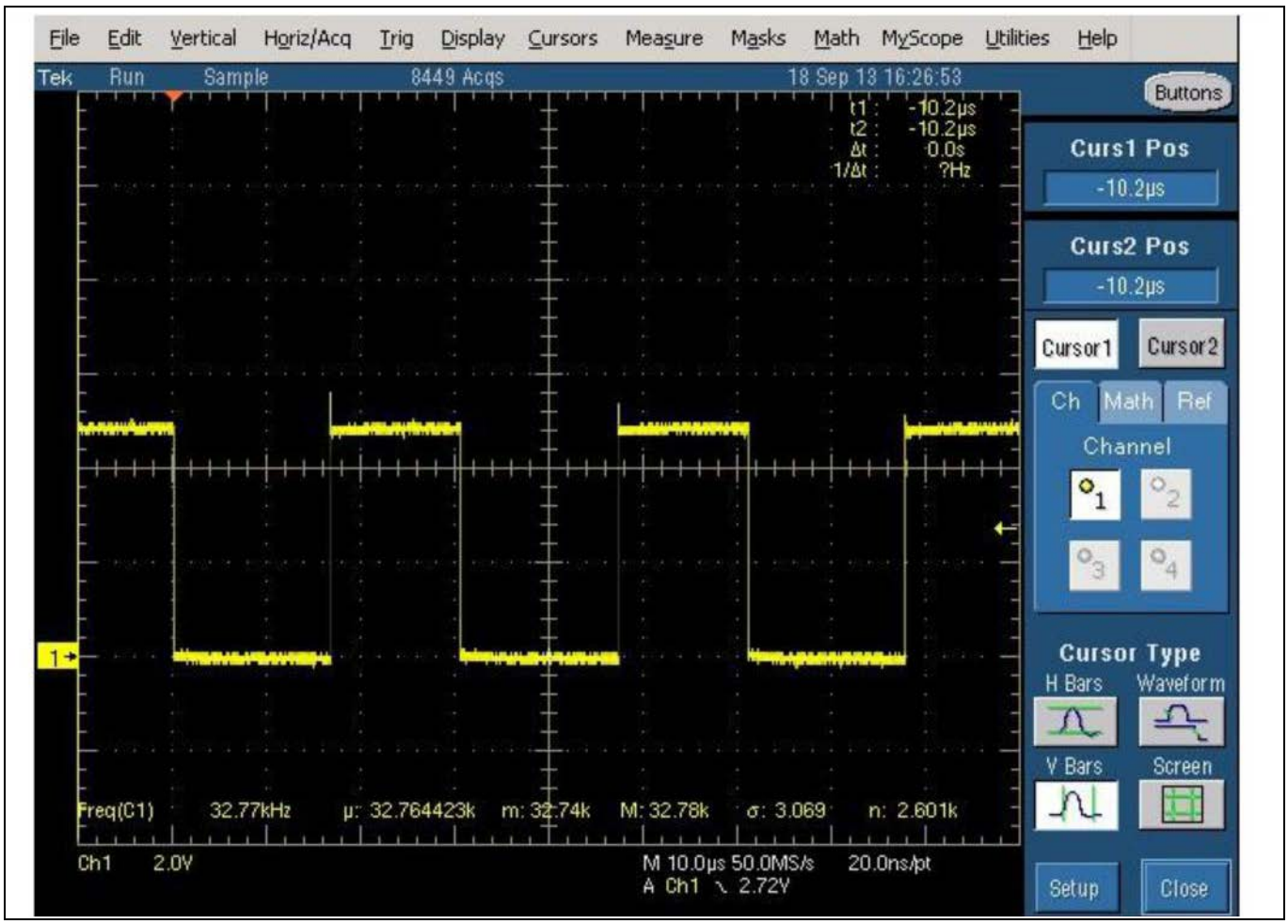


Figure 10 32.768 kHz リファレンスクロック

1.4 温度テスト

-40°C～105°C の範囲で発振器の精度を測定するために、このアプリケーションノートに示されているクロックキャリブレーション方法を使用して温度テストを実施しました。

1.4.1 テストのセットアップ

テストでは、XMC1200 ブートキットが使用されます。このデバイスは、キャリブレーションを連続的に実行し、同時に 100 kHz の PWM 信号を出力するように設定されています。

ブートキットは恒温槽内に置かれ、PWM 出力は恒温槽外のオシロスコープに接続されます。

テストは室温 (25°C) から開始し、ゆっくりと 105°C まで上昇し、その後 105°C で 15 分間維持されます。その後、温度をゆっくりと-40°C まで下げ、その温度で 15 分間維持するように調整しました。

PWM 信号に変動が観測されました。

1.4.2 テスト結果

100 kHz の PWM 信号の温度テスト中に観測された最大周波数は 100.4 kHz で、観測された最小周波数は 99.9 kHz でした。したがって、温度-40°C から 105°C の発振器の変動は 0.5%以内であると結論付けることができます。

Attention: テストは、実験室条件下で 3 つのデバイスユニットで実施されました。このドキュメントで共有されているテスト結果は、参考のみを目的としています。

1.5 結論

リファレンスクロックは定期的にキャプチャされるため、システムクロックの変更を必要な値に調整できます。この実装により、システムクロックの精度は内部発振器の特性に応じて改善できます。

1.6 最大周波数 32 MHz を必要とするアプリケーション向けの拡張

セクション 1.2.2 で説明したように、フラクショナル分周器のアプローチは主に分周器を使用し、設定可能な最大の MCLK 値よりも低い周波数をターゲットにする場合にのみ使用できます。例えば、32 MHz に到達する必要がある場合は、ANAOFFSET レジスタを使用する別の方法を使用できます。

ANAOFFSET レジスタは、正方向と負方向の両方向で追加のオフセットを使用して DCO を調整する手段を提供します。詳細な説明, 範囲, および設定については、リファレンスマニュアルの SCU のセクションにある ANAOFFSET レジスタの定義を参照してください。

基本原理の概要を次の図に示します。

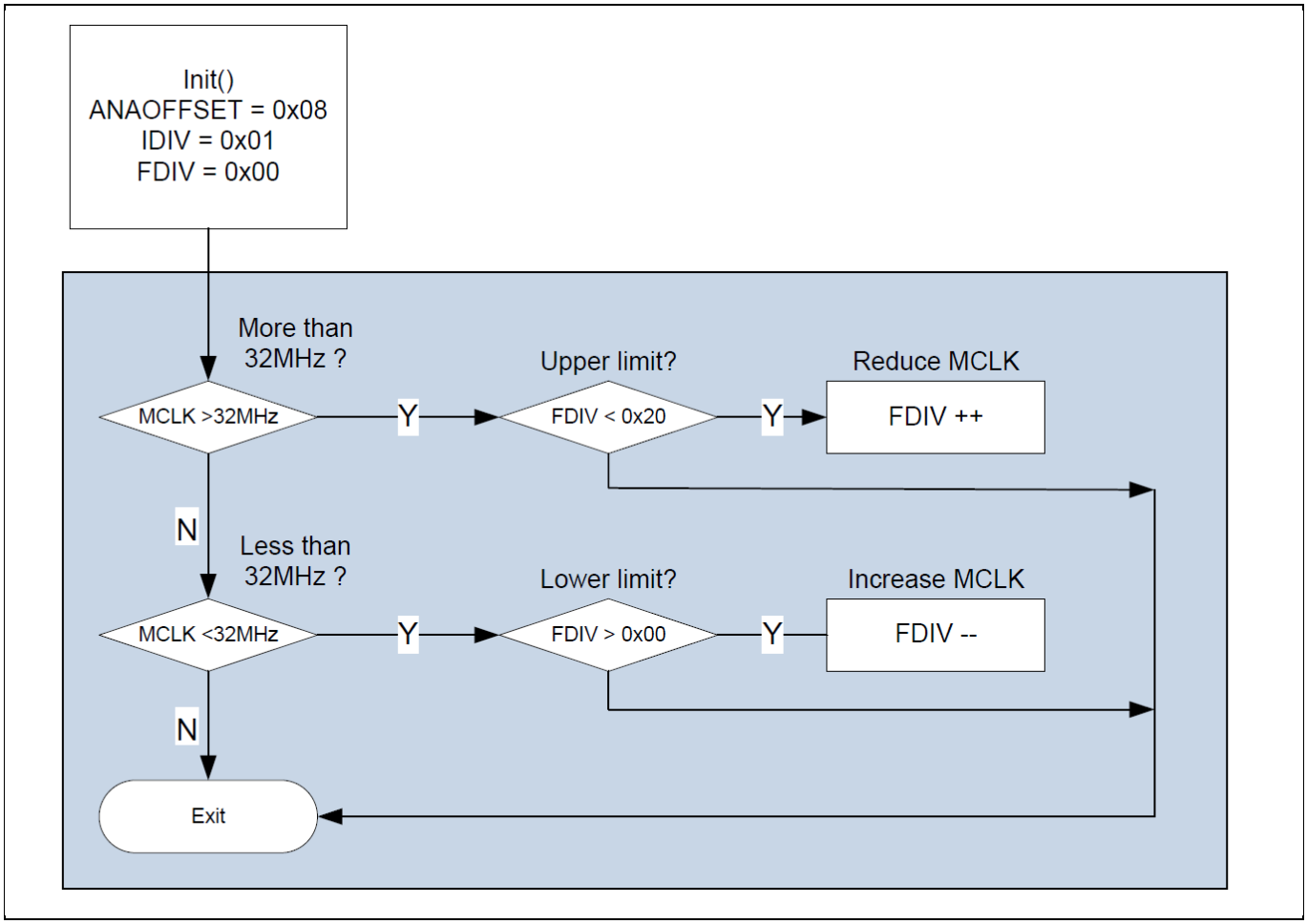


Figure 11 MCLK=32 MHz で ANAOFFSET を使用する基本原理

Note: ここでは、セクション1.2.3 で説明したのと同じアプリケーション固有の境界条件チェックが必要になる場合があります。

2 温度に対する DCO1 のキャリブレーション

温度が変化すると、クロック周波数はドリフトします。データシートには、XMC1000 ファミリの量産デバイスの内部発振器 (DCO1) の精度は、全温度範囲 (-40°C〜105°C) に対して-3.9%〜4%の範囲でドリフトする可能性があると記載されています。DCO1 クロック周波数は、低温では低下し、高温では上昇します。したがって、XMC1000 ファミリのデバイスは、温度変化に対して実行時に内部発振器をキャリブレーションするソリューションを提供します。より正確なクロックを実現するために、XMC1400 では外部水晶振動子がサポートされています。

ダイ温度は、XMC1000 のオンチップ温度センサーで測定できます。

測定された温度に基づいて、DCO1 キャリブレーションのオフセット値を取得できます。DCO1 のオフセット調整は、ANAOFFSET レジスタを介して行います。

次のサンプルコードは、XMC1300 に基づいています。

2.1 キャリブレーション パラメータ

パラメータは DCO1 キャリブレーションのフラッシュコンフィグレーションセクタで定義され、フラッシュアドレスの範囲は 0x10000F30 から 0x10000F33 です。キャリブレーションコードを使用するためには、これらのパラメータをプロジェクトで定義する必要があります。

```
#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: ***これらのパラメータが定義されているのは、量産デバイス (デバイスマーキング内に「EE」または「ES」がない) だけです。DCO キャリブレーションは、量産デバイスにのみ適用できます。ES または EE デバイスへのキャリブレーションソフトウェアの適用は未定義です。***

2.2 初期化コード

ソフトウェア関数 TSE_Calibration_Init() を呼び出して、キャリブレーションコードを初期化します。この関数は、オンチップ温度センサーを有効にし、パラメータを割り当てます。

```
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 キャリブレーション ルーチン

キャリブレーションルーチンは、ソフトウェア関数 TSE_DCO_Calibration()で提供されます。

TSE_DCO_Calibration()は実行時に呼び出され、内部発振器を調整します。これを実行する 1 つの方法は、システムタイマー割込みを使用して定期的に呼び出すことです。

デバイスの温度は、ルーチンの実行時に測定されます。オンチップ温度センサーから温度測定値を取得するには、温度センサーライブラリファイルが必要です。現在の温度が測定されると、それを使用して新しいオフセット値が計算されます。このドキュメントで説明するオフセット計算には、計算値を最も近い整数に丸めることが含まれます。

この新しいオフセット値が以前のオフセット値と異なる場合は、ANAOFFSET レジスタに書き込む必要があります。

- 計算されたオフセット値が 0 より小さい場合は、0 を使用します。
- 計算されたオフセット値が 8 より大きい場合は、8 を使用します。

内部発振器の周波数は、新しいオフセット値が以前のオフセット値と異なる場合、自動的に調整されます。

```
#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 キャリブレーション コードの温度テスト

温度テストは、キャリブレーションコードの有無にかかわらずマイクロコントローラで実施されました。テストは、-40°C～105°Cの温度範囲をカバーしました。デバイスは 100 kHz の PWM 信号を出力し、この PWM 信号の周波数変動が観察されました。

2.4.1 テストのセットアップ

XMC1302-T038X0200 マイクロコントローラは恒温槽でテストされました。まず、マイクロコントローラをキャリブレーションコードなしでテストし、次にキャリブレーションコードを追加して再度テストしました。

テストは室温 (25°C) で開始し、その後ゆっくりと 105°C まで上げ、その温度で 15 分間維持しました。その後、温度を-40°C までゆっくりと下げ、その温度を 15 分間維持しました。

テスト全体を通して PWM 信号の周波数変動が観察されました。

2.4.2 テスト結果

グラフは、温度範囲にわたる 100 kHz PWM 信号の周波数の変化を示しています。

キャリブレーションコード無し

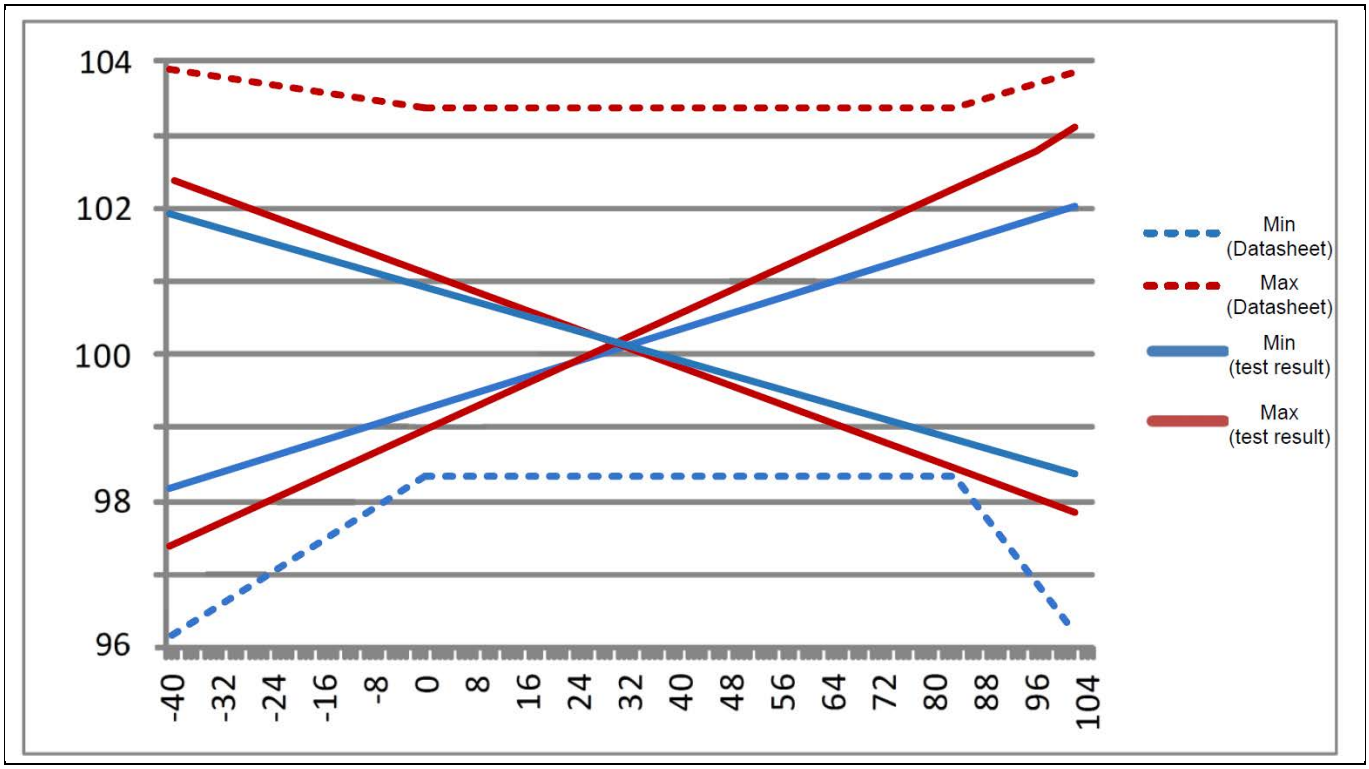


Figure 12 100kHz PWM 出力 (DCO1 キャリブレーション無し)

2 温度に対する DCO1 のキャリブレーション

キャリブレーション有り

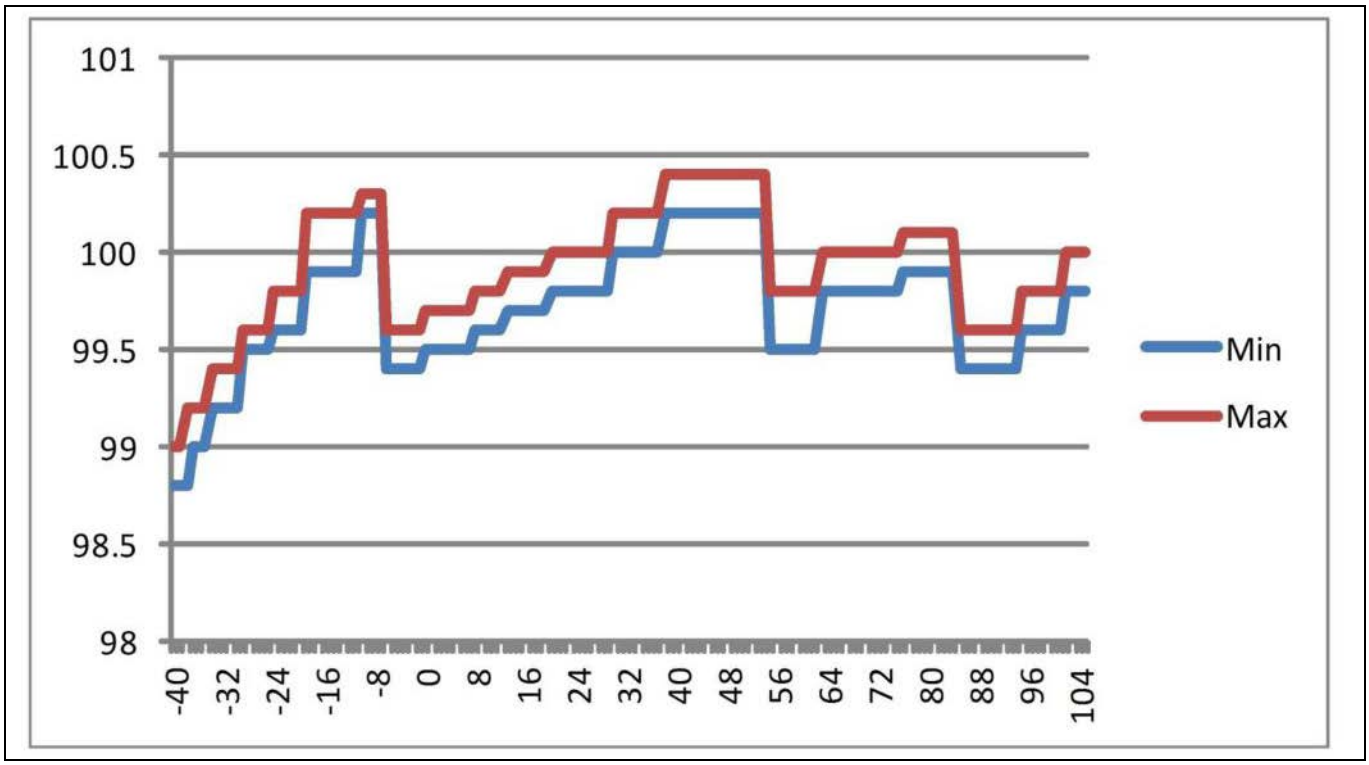


Figure 13　100kHz PWM 出力 (DCO1 キャリブレーション有り)

この観察結果から、100 kHz PWM の周波数は、キャリブレーションコードがない場合、105°C で最大 101.4 kHz、-40°C で最小 98.1 kHz まで変化する可能性があることがわかります。

キャリブレーションコードを追加した後、観測された最高周波数は 38°C ～ 53°C で 100.4 kHz となり、最低周波数は-40°C で 98.8 kHz となりました。

結論としては、キャリブレーションコードを追加した後、周波数変動が大幅に減少するということです。

通常、ノミナルの周波数に対するクロック精度は、0°C を超える温度では 2 倍に向上すると予想されます。

参考資料

[1] Infineon: XMC1000 Reference Manual, http://www.infineon.com/XMC1000 タブ: Documents

[2] Infineon: XMC1000 Datasheet, http://www.infineon.com/XMC1000 タブ: Documents

[3] Infineon: DAVE™, http://www.infineon.com/DAVE

改訂履歴

版数	発行日	変更内容
1.6	2024-07-26	本版は英語版 AP32321 Oscillator handling Revision 1.6 について、インフィニオン DEVELOPER COMMUNITY の参画者によって日本語に翻訳されたドキュメントです。

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-07-26
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2024 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32321_JA

重要事項
本手引書に記載された本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項
技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。
インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。