TB3272

AVR® DB MCUの外部高周波数発振器での開始に際して

序説

著者: Egil Rotevatn, Microchip Technology Inc.

外部高周波数発振器(XOSCHF)は32MHzまでの外部ｸﾘｽﾀﾙまたは外部ｸﾛｯｸ信号の使用を許します。これは主ｸﾛｯｸ(CLK_MAIN)、実時間計数器(RTC)、12ﾋﾞｯﾄ ﾀｲﾏ/ｶｳﾝﾀD型(TCDn)用のｸﾛｯｸ元として使うことができます。

ｸﾛｯｸ障害検出(CFD:Clock Failure Detection)機能はｸﾛｯｸ元からの出力が止まる場合を検出して動作を続けるために主ｸﾛｯｸを違うｸﾛｯｸ元に切り替えるか、または安全に動作を停止することができます。この機能はXOSCHFが使われる応用、例えば、機能安全応用の安全な停止を許すのに最も有用です。

この技術概説はAVR® DB系ﾏｲｸﾛ ｺﾝﾄﾛｰﾗでXOSCHFとCFDがどう使われるかを記述し、以下の使用事例を網羅します。

- **外部ｸﾘｽﾀﾙとのXOSCHF:**

  XOSCHFを外部ｸﾘｽﾀﾙ用に初期化して主ｸﾛｯｸ元をXOSCHFに変更

- **外部ｸﾛｯｸとのXOSCHF:**

  XOSCHFを外部ｸﾛｯｸ信号用に初期化して主ｸﾛｯｸ元をXOSCHFに変更

- **XOSCHFとのRTC:**

  RTC用ｸﾛｯｸ元としてXOSCHFを使用

- **XOSCHFとのTCD:**

  TCD用ｸﾛｯｸ元としてXOSCHFを使用

- **XOSCHFとPLLとのTCD:**

  ｸﾛｯｸ元としてXOSCHFでPLLを初期化してTCD0用ｸﾛｯｸ元としてPLLを使用

- **XOSCHFでのCFD:**

  XOSCHFを監視するようにCFDを初期化し、ｸﾛｯｸ元障害の場合にLEDを交互切り替え

- **NMIとで主ｸﾛｯｸでののでのCFD:**

  ｸﾛｯｸ元としてXOSCHFでの主ｸﾛｯｸをを監視するようにCFDを初期化し、割り込みを遮蔽不可割り込み(NMI:Non-Maskable Interrupt)として許可し、主ｸﾛｯｸから派生した周波数でLEDを交互切り替え

ｺｰﾄﾞ例はGitHubで入手可能です

GitHubでｺｰﾄﾞ例を見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

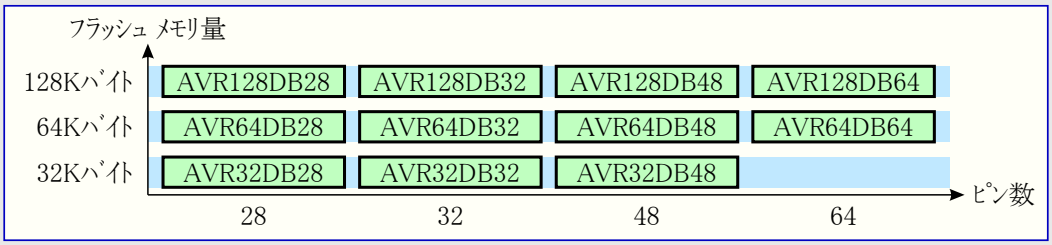
目次

1. 関連ﾃﾞﾊﾞｲｽ

本章はこの文書に関連するﾃﾞﾊﾞｲｽを一覧にします。下図はﾋﾟﾝ数の変種とﾒﾓﾘ量を展開して各種系列ﾃﾞﾊﾞｲｽを示します。

- これらのﾃﾞﾊﾞｲｽがﾋﾟﾝ互換で同じまたはより多くの機能を提供するため、垂直上方向移植はｺｰﾄﾞ変更なしで可能です。
- 左への水平方向移植はﾋﾟﾝ数、従って利用可能な機能を減らします。
- 異なるﾌﾗｯｼｭ ﾒﾓﾘ量を持つﾃﾞﾊﾞｲｽは一般的に異なるSRAMとEEPROMも持ちます。

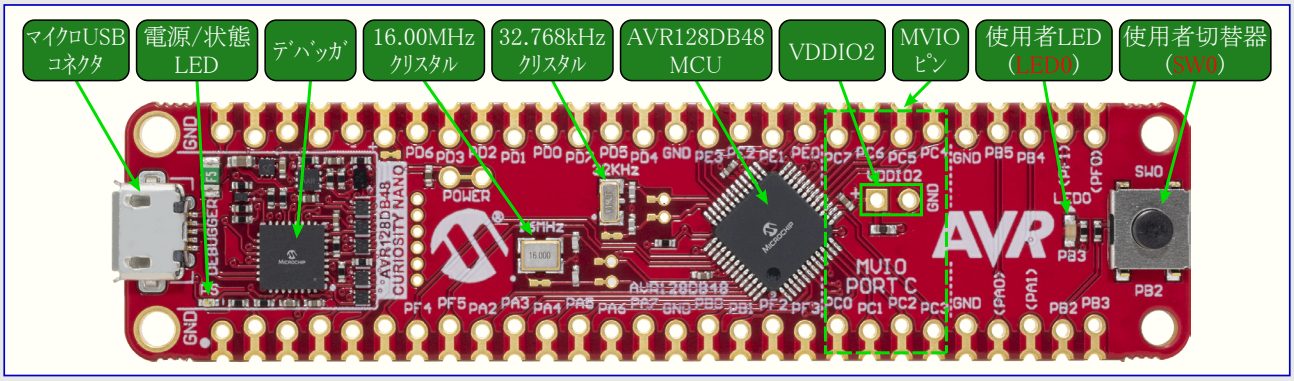
図1-1. AVR® DB系概要



2. ﾊｰﾄﾞｳｪｱ構成設定

ｺｰﾄﾞ例はAVR128DB48 Curiosity Nano (EV25L43A)開発基板で開発されました。これはこの技術概説での使用事例の殆どに使うことができる基板上の16MHzｸﾘｽﾀﾙを持ちます。このｸﾘｽﾀﾙは基板を外部ｸﾛｯｸ信号で使う場合に切断されなければなりません。

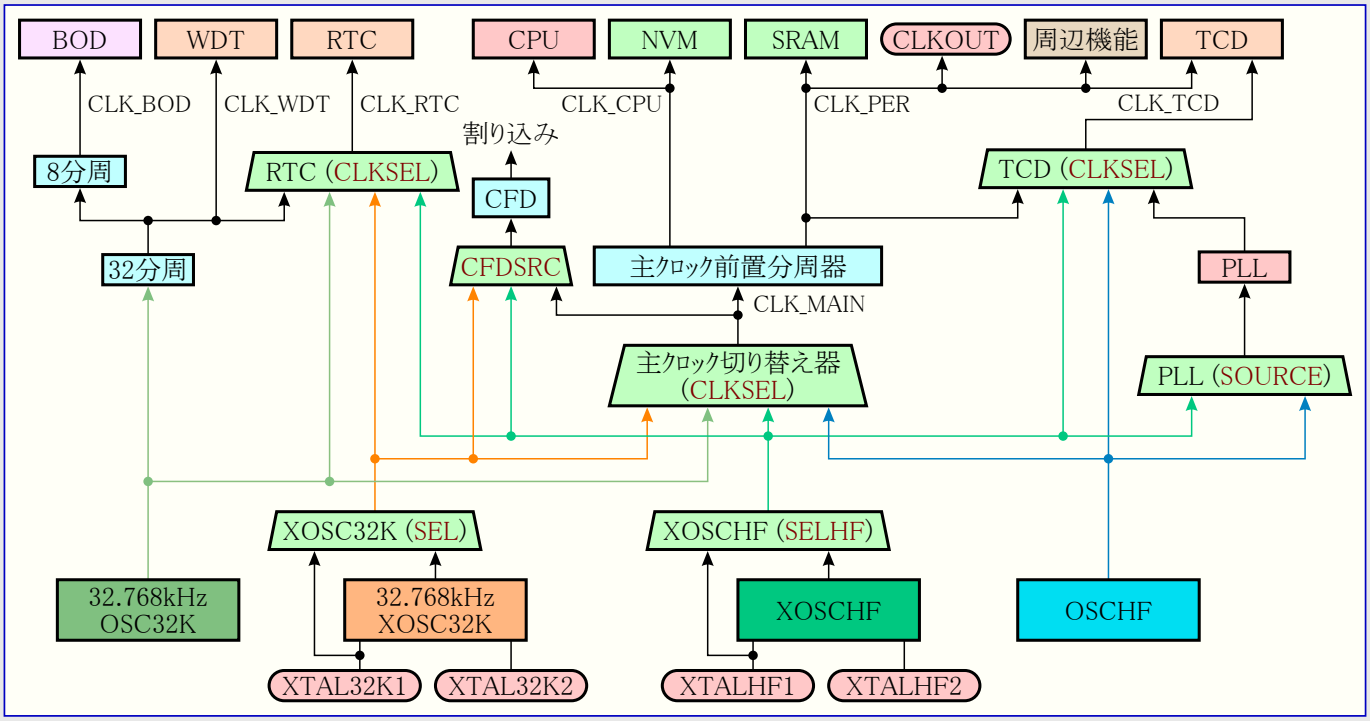
図2-1. AVR128DB48 Curiosity Nano (EV35L43A)開発基板



3. 概要

ｸﾛｯｸ制御器(CLKCTRL)構成図は以降の部分で説明されるようにAVR® DB系ﾃﾞﾊﾞｲｽでXOSCHFがどう配線され得るかを示します。

図3-1. CLKCTRL構成図



4. 外部クリスタルとのXOSCHF

外部クリスタルでXOSCHFを使うには外部高周波数クリスタル用発振器制御A(CLKCTRL.XOSCHFCTRLA)レジスタで供給元選択(SELHF)ビットが'0'を、許可(ENABLE)ビットが'1'を書かれなければなりません。

加えて、周波数範囲(FRQRANGE)ビット領域が接続されるクリスタルの周波数と等しいかより高い設定を書かれなければならず、クリスタル始動時間(CSUTHF)ビット領域はクロックが安定と見做される前に待つためのクロック数を選ぶように書くことができます。

発振器がどれかの単位部によって要求される前に意図されるように動いているのを確実にするため、スタンバイ時走行(RUNSTDBY)ビットを許可することができます。

このレジスタは構成設定変更保護(CCP:Configuration Change Protection)を持ち、故にレジスタの解錠用の正しいタイミングを保証するためにcpufunc.hのccp_write_io関数が使われるべきです。

図4-1. CLKCTRL.XOSCHFCTRLA – XOSCHF許可

ビット	7	6	5	4	3	2	1	0
	RUNSTDBY		CSUTHF1,0		FRQRANGE1,0		SELHF	ENABLE
アクセス種別	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
リセット値	0	0	0	0	0	0	0	0

● **ビット7 – RUNSTDBY : スタンバイ時走行** (Run Standby)

このビットは許可(ENABLE)ビットが'1'の時に外部高周波数発振器(XOSCHF)が常に動くか否かを制御します。

値	0	1
説明	XOSCHF発振器は周辺機能または主クロックによって要求される時にだけ動きます。(注1)	XOSCHF発振器は常に活動動作及びアイドルとスタンバイの休止動作で動きます。(注2)

注1: 要求する周辺機能や主クロックは発振器始動時間を考慮しなければなりません。

注2: 発振器信号は要求された場合にだけ利用可能で、初期クリスタル始動時間が既に終わった場合、2 XOSCHF周期後に利用可能です。

● **ビット5,4 – CSUTHF1,0 : クリスタル始動時間** (Crystal Start-Up Time)

このビット領域は供給元選択(SELHF)ビットが'0'の時に外部高周波数発振器(XOSCHF)の始動時間を制御します。

値	0 0	0 1	1 0	1 1
名称	256	1K	4K	–
説明	256周期	1K周期	4K周期	(予約)

注: このビット領域は許可(ENABLE)ビットまたは主クロック状態(CLKCTRL.MCLKSTATUS)レジスタのXOSCHF状態(XOSCHFS)ビットが'1'の時に読み込み専用です。

● **ビット3,2 – FRQRANGE1,0 : 周波数範囲** (Frequency Range)

このビット領域は外部クリスタルに対して支援される最大周波数を制御します。選んだ範囲が大きいほど発振器による消費電流が大きくなります。

値	0 0	0 1	1 0	1 1
名称	8M	16M	24M	32M
説明	最大8MHzクリスタル周波数	最大16MHzクリスタル周波数	最大24MHzクリスタル周波数	最大32MHzクリスタル周波数

注: 支援される最大CLK_CPU周波数よりも高い周波数のクリスタルが主クロックとして使われる場合、主クロック制御B(CLKCTRL.MCLKCTRLB)レジスタの前置分周器分周値(PDIV)ビット領域に適切な構成設定を書くことによって分周することが必要です。

● **ビット1 – SELHF : 供給元選択** (Source Select)

このビットは外部高周波数発振器(XOSCHF)の供給元を制御します。

値	0	1
名称	CRYSTAL	EXTCLOCK
説明	XTALHF1とXTAKHF2のピンでの外部クリスタル	XTALHF1ピンでの外部クロック

注: このビット領域は許可(ENABLE)ビットまたは主クロック状態(CLKCTRL.MCLKSTATUS)レジスタのXOSCHF状態(XOSCHFS)ビットが'1'の時に読み込み専用です。

次頁へ続く

図4-1 (続き). CLKCTRL.XOSCHFCTRLA – XOSCHF許可

● **ビット0 – ENABLE : 許可** (Enable)

このビットは外部高周波数発振器(XOSCHF)が許可されるか否かを制御します。

値	0	1
説明	XOSCHF発振器禁止	XOSCHF発振器許可、各々の発振器ピンに対する標準ピン操作を無効にします。

```
/* 周波数範囲16Mzと4K周期始動時間でクリスタル用発振器許可 */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
              | CLKCTRL_CSUTHF_4K_gc
              | CLKCTRL_FRQRANGE_16M_gc
              | CLKCTRL_SELHF_CRYSTAL_gc
              | CLKCTRL_ENABLE_bm);
```

XOSCHF許可後、主クロック状態(CLKCTRL.MCLKSTATUS)レジスタの外部クリスタル/クロック状態(EXTS/XOSCHFS)ビットをこのビットが'1'として読まれるまでポーリングすることによってクロック元の安定を待ってください。

図4-2. CLKCTRL.MCLKSTATUS – XOSCHF開始待ち

ビット	7	6	5	4	3	2	1	0
			PLLS	EXTS/XOSCHFS	XOSC32KS	OSC32KS	OSCHFS	SOSC
アクセス種別	R	R	R	R	R	R	R	R
リセット値	0	0	0	0	0	0	0	0

● **ビット4 – EXTS/XOSCHFS : 外部クリスタル/クロック状態** (External Crystal/Clock Status)

値	0	1
説明	外部高周波数発振器制御A(CLKCTRL.XOSCHFCTRLA)レジスタの供給元選択(SELHF)ビットが'0'の時に外部高周波数クリスタルは安定ではありません。 SELHFビットが'1'の時に外部高周波数クロックは動いていません。	SELHFビットが'0'の時に外部高周波数クリスタルは安定です。 SELHFビットが'1'の時に外部高周波数クロックは動いています。

```
/* クリスタル用発振器始動確認 */
while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}
```

主クロック前置分周器は24MHzの最大周波数を与えるように構成設定されるでしょう。主クロックがクロック元として24MHzよりも速く動くクリスタルでXOSCHFを使う場合、前置分周器設定は主クロック制御B(CLKCTRL.MCLKCTRLB)レジスタで前置分周器分周値(PDIV)ビット領域に、それと前置分周器許可(PEN)に'1'を書くことができます。AVR128DB48 Curiosity NanoはPENビットへの'0'書き込みによって前置分周器の禁止を可能にする16MHzクリスタルを持ちます。

主クロック制御BもCCPを持ちます。

図4-3. CLKCTRL.MCLKCTRLB – 主クロック前置分周器構成設定

ビット	7	6	5	4	3	2	1	0
				PDIV3~0				PEN
アクセス種別	R	R	R	R/W	R/W	R/W	R/W	R/W
リセット値	0	0	0	0	0	0	0	0

● **ビット4~1 – PDIV3~0 : 前置分周器分周値** (Prescaler Division)

このビット領域は前置分周器許可(PEN)ビットが'1'の時に主クロック(CLK_MAIN)前置分周器の分周比を制御します。

値	0000	0001	0010	0011	0100	0101	1000	1001	1010	1011	1100	その他
名称	2X	4X	8X	16X	32X	64X	6X	10X	12X	24X	48X	–
説明 (CLK_MAIN分周数)	2	4	8	16	32	64	6	10	12	24	48	(予約)

次頁へ続く

図4-3 (続き). CLKCTRL.MCLKCTRLB – 主ｸﾛｯｸ前置分周器構成設定

注: 入力周波数(CLK_MAIN)の構成設定と前置分周器設定は許された最大周波数の周辺機能ｸﾛｯｸ(CLK_PER)やCPUｸﾛｯｸ(CLK_CPU)を超えてはなりません。更なる情報については「電気的特性」章を参照してください。

● **ﾋﾞｯﾄ0 – PEN : 前置分周器許可** (Prescaler Enable)

このﾋﾞｯﾄは主ｸﾛｯｸ(CLK_MAIN)前置分周器が許可されるか否かを制御します。

値	0	1
説明	CLK_MAIN前置分周器禁止	CLK_MAIN前置分周器許可、分周比は前置分周器分周値(PDIV)ﾋﾞｯﾄ領域によって制御されます。

```
/* 主ｸﾛｯｸ前置分周器解消 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, 0x00);
```

主ｸﾛｯｸ制御A(CLKCTRL.MCLKCTRLA)ﾚｼﾞｽﾀのｸﾛｯｸ選択(CLKSEL)ﾋﾞｯﾄ領域にEXTCLK設定を書くことによって主ｸﾛｯｸ元を変更してください。主ｸﾛｯｸ出力(CLKOUT)ﾋﾞｯﾄはCLKOUTﾋﾟﾝで主ｸﾛｯｸを出力するのに'1'を書くことができます。このﾚｼﾞｽﾀもCCPを持ちます。

図4-4. CLKCTRL.MCLKCTRLA – 供給元をXOSCHFに変更

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
	CLKOUT				CLKSEL3~0			
ｱｸｾｽ種別	R/W	R	R	R	R/W	R/W	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ7 – CLKOUT : 主ｸﾛｯｸ出力** (Main Clock Out)

このﾋﾞｯﾄは主ｸﾛｯｸが動いている時に主ｸﾛｯｸが主ｸﾛｯｸ出力(CLKOUT)ﾋﾟﾝで利用可能か否かを制御します。

このﾋﾞｯﾄはこれに'0'が書かれる時か、または供給元として主ｸﾛｯｸでｸﾛｯｸ障害検出(CFD)状態が起きた時に解除(0)されます。

このﾋﾞｯﾄはこれに'1'が書かれる時に設定(1)されます。

値	0	1
説明	主ｸﾛｯｸはCLKOUTﾋﾟﾝで利用不能です。	主ｸﾛｯｸはCLKOUTﾋﾟﾝで利用可能です。

● **ﾋﾞｯﾄ3~0 – CLKSEL3~0 : ｸﾛｯｸ選択** (Clock Select)

このﾋﾞｯﾄ領域は主ｸﾛｯｸ(CLK_MAIN)用の供給元を選びます。

値	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	その他
名称	OSCHF	OSC32K	XOSC32K	EXTCLK	–
説明	内部高周波数発振器	32.768kHz内部発振器	32.768kHz外部ｸﾘｽﾀﾙ用発振器	XOSCHFCTRLAのSELHFﾋﾞｯﾄに応じて外部ｸﾛｯｸまたは外部高周波数ｸﾘｽﾀﾙ用発振器	(予約)

```
/* 供給元としてXOSCHFを使うように主ｸﾛｯｸを設定し、CLKOUTﾋﾟﾝを許可 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc | CLKCTRL_CLKOUT_bm);
```

主ｸﾛｯｸ元変更後、主ｸﾛｯｸ状態(CLKCTRL.MCLKSTATUS)ﾚｼﾞｽﾀの主ｸﾛｯｸ発振器変更(SOSC)ﾋﾞｯﾄをこのﾋﾞｯﾄが'0'として読まれるまでﾎﾟｰﾘﾝｸﾞすることによって切替完了を待ってください。

図4-5. CLKCTRL.MCLKSTATUS – 主ｸﾛｯｸ変更待ち

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
			PLLS	EXTS/XOSCHFS	XOSC32KS	OSC32KS	OSCHFS	SOSC
ｱｸｾｽ種別	R	R	R	R	R	R	R	R
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ0 – SOSC : 主ｸﾛｯｸ発振器変更** (Main Clock Oscillator Changing)

値	0	1
説明	CLK_MAIN用ｸﾛｯｸ元は切り替えを体験していません。	CLK_MAIN用ｸﾛｯｸ元は切り替えを体験し、新供給元が安定すると直ぐに変更します。

```
/* システム発振器変更完了待ち */
while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
{
    ;
}
```

これで出力が主クロックによって使われ、発振器を強制的に許可する必要がないため、今やスタンバイ時走行(RUNSTDBY)ビットを解除(0)することができます。

```
/* 休止中の節電のためRUNSTDBYを解除(0) */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

この例用のコードはそれらのGitHub貯蔵庫のXOSCHF-with-external-crystalフォルダで入手可能です。

GitHubでコード例を見てください。
貯蔵庫を閲覧するにはクリックしてください。

5. 外部クロックとのXOSCHF

外部クロック信号でXOSCHFを使うには外部高周波数クリスタル用発振器制御A(CLKCTRL.XOSCHFCTRLA)レジスタで供給元選択(SELHF)ビットと許可(ENABLE)ビットが'1'を書かれなければなりません。

このレジスタは構成設定変更保護(CCP)を持ち、故にレジスタの解錠用の正しいタイミングを保証するためにcpufunc.hのccp_write_io関数が使われるべきです。

図5-1. CLKCTRL.XOSCHFCTRLA – XOSCHF許可

ビット	7	6	5	4	3	2	1	0
	RUNSTDBY		CSUTHF1,0		FRQRANGE1,0		SELHF	ENABLE
アクセス種別	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
リセット値	0	0	0	0	0	0	0	0

注: 支援される最大CLK_CPU周波数よりも高い周波数のクリスタルが主クロックとして使われる場合、主クロック制御B(CLKCTRL.MCLKCTRLB)レジスタの前置分周器分周値(PDIV)ビット領域に適切な構成設定を書くことによって分周することが必要です。

● **ビット1 – SELHF : 供給元選択** (Source Select)

このビットは外部高周波数発振器(XOSCHF)の供給元を制御します。

値	0	1
名称	CRYSTAL	EXTCLOCK
説明	XTALHF1とXTAKHF2のピンでの外部クリスタル	XTALHF1ピンでの外部クロック

注: このビット領域は許可(ENABLE)ビットまたは主クロック状態(CLKCTRL.MCLKSTATUS)レジスタのXOSCHF状態(XOSCHFS)ビットが'1'の時に読み込み専用です。

● **ビット0 – ENABLE : 許可** (Enable)

このビットは外部高周波数発振器(XOSCHF)が許可されるか否かを制御します。

値	0	1
説明	XOSCHF発振器禁止	XOSCHF発振器許可、各々の発振器ピンに対する標準ピン操作を無効にします。

```
/* 外部クロック入力許可 */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_SELHF_EXTCLOCK_gc | CLKCTRL_ENABLE_bm);
```

クロック元としてXOSCHFを使うように主クロックを変更する場合、最初に主クロック制御B(CLKCTRL.MCLKCTRLB)レジスタで前置分周器分周値(PDIV)ビット領域に前置分周値設定と前置分周器許可(PEN)に'1'を書くことによって最大24MHzを出力するように主クロック前置分周器を設定することを確実にしてください。このレジスタもCCPを持ちます。

図5-3. CLKCTRL.MCLKCTRLB – 主ｸﾛｯｸ前置分周器構成設定

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
				PDIV3~0				PEN
ｱｸｾｽ種別	R	R	R	R/W	R/W	R/W	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ4~1 – PDIV3~0 : 前置分周器分周値** (Prescaler Division)

このﾋﾞｯﾄ領域は前置分周器許可(PEN)ﾋﾞｯﾄが'1'の時に主ｸﾛｯｸ(CLK_MAIN)前置分周器の分周比を制御します。

値	0000	0001	0010	0011	0100	0101	1000	1001	1010	1011	1100	その他
名称	2X	4X	8X	16X	32X	64X	6X	10X	12X	24X	48X	–
説明 (CLK_MAIN分周数)	2	4	8	16	32	64	6	10	12	24	48	(予約)

注: 入力周波数(CLK_MAIN)の構成設定と前置分周器設定は許された最大周波数の周辺機能ｸﾛｯｸ(CLK_PER)やCPUｸﾛｯｸ(CLK_CPU)を超えてはなりません。更なる情報については「電気的特性」章を参照してください。

● **ﾋﾞｯﾄ0 – PEN : 前置分周器許可** (Prescaler Enable)

このﾋﾞｯﾄは主ｸﾛｯｸ(CLK_MAIN)前置分周器が許可されるか否かを制御します。

値	0	1
説明	CLK_MAIN前置分周器禁止	CLK_MAIN前置分周器許可、分周比は前置分周器分周値(PDIV)ﾋﾞｯﾄ領域によって制御されます。

```
/* 主ｸﾛｯｸ前置分周器設定 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_2X_gc | CLKCTRL_PEN_bm);
```

主ｸﾛｯｸ制御A(CLKCTRL.MCLKCTRLA)ﾚｼﾞｽﾀのｸﾛｯｸ選択(CLKSEL)ﾋﾞｯﾄ領域にEXTCLK設定を書くことによって主ｸﾛｯｸ元を変更してください。主ｸﾛｯｸ出力(CLKOUT)ﾋﾞｯﾄはCLKOUTﾋﾟﾝで主ｸﾛｯｸを出力するのに'1'を書くことができます。このﾚｼﾞｽﾀもCCPを持ちます。

図5-4. CLKCTRL.MCLKCTRLA – 供給元をXOSCHFに変更

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
	CLKOUT				CLKSEL3~0			
ｱｸｾｽ種別	R/W	R	R	R	R/W	R/W	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ7 – CLKOUT : 主ｸﾛｯｸ出力** (Main Clock Out)

このﾋﾞｯﾄは主ｸﾛｯｸが動いている時に主ｸﾛｯｸが主ｸﾛｯｸ出力(CLKOUT)ﾋﾟﾝで利用可能か否かを制御します。

このﾋﾞｯﾄはこれに'0'が書かれる時か、または供給元として主ｸﾛｯｸでｸﾛｯｸ障害検出(CFD)状態が起きた時に解除(0)されます。

このﾋﾞｯﾄはこれに'1'が書かれる時に設定(1)されます。

値	0	1
説明	主ｸﾛｯｸはCLKOUTﾋﾟﾝで利用不能です。	主ｸﾛｯｸはCLKOUTﾋﾟﾝで利用可能です。

● **ﾋﾞｯﾄ3~0 – CLKSEL3~0 : ｸﾛｯｸ選択** (Clock Select)

このﾋﾞｯﾄ領域は主ｸﾛｯｸ(CLK_MAIN)用の供給元を選びます。

値	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	その他
名称	OSCHF	OSC32K	XOSC32K	EXTCLK	–
説明	内部高周波数発振器	32.768kHz内部発振器	32.768kHz外部ｸﾘｽﾀﾙ用発振器	XOSCHFCTRLAのSELHFﾋﾞｯﾄに応じて外部ｸﾛｯｸまたは外部高周波数ｸﾘｽﾀﾙ用発振器	(予約)

```
/* 供給元としてXOSCHFを使うように主ｸﾛｯｸを設定し、CLKOUTﾋﾟﾝを許可 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc | CLKCTRL_CLKOUT_bm);
```

主ｸﾛｯｸ元変更後、主ｸﾛｯｸ状態(CLKCTRL.MCLKSTATUS)ﾚｼﾞｽﾀの主ｸﾛｯｸ発振器変更(SOSC)ﾋﾞｯﾄをこのﾋﾞｯﾄが'0'として読まれるまでﾎﾟｰﾘﾝｸﾞすることによって切替完了を待ってください。

図5-5. CLKCTRL.MCLKSTATUS – 主ｸﾛｯｸ変更待ち

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
			PLLS	EXTS/XOSCHFS	XOSC32KS	OSC32KS	OSCHFS	SOSC
ｱｸｾｽ種別	R	R	R	R	R	R	R	R
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ0 – SOSC : 主ｸﾛｯｸ発振器変更** (Main Clock Oscillator Changing)

値	0	1
説明	CLK_MAIN用ｸﾛｯｸ元は切り替えを体験していません。	CLK_MAIN用ｸﾛｯｸ元は切り替えを体験し、 新供給元が安定すると直ぐに変更します。

```
/* ｼｽﾃﾑ発振器変更完了待ち */
while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
{
    ;
}
```

この例用のｺｰﾄﾞはそれらのGitHub貯蔵庫のXOSCHF-with-external-clockﾌｫﾙﾀﾞで入手可能です。

GitHubでｺｰﾄﾞ例を見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

6. XOSCHFとのRTC

実時間計数器(RTC:Real-Time Couter)用ｸﾛｯｸ元としてXOSCHFを使うには最初に先の2つの使用事例と同様にXOSCHFを許可してください。ｸﾘｽﾀﾙまたは入力ｸﾛｯｸの周波数は主ｸﾛｯｸ周波数の1/4までが可能です。

⚠警告 AVR128DB48 Curiosity Nano上の16MHzｸﾘｽﾀﾙはこの例に対して高すぎる周波数を生成します。仕様内で動かすために最大で主ｸﾛｯｸ周波数の1/4のｸﾘｽﾀﾙで置き換えるか、または外部信号を使ってください。

```
/* 周波数範囲8Mzと1K周期始動時間でｸﾘｽﾀﾙ用発振器許可 */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
             | CLKCTRL_CSUTHF_1K_gc
             | CLKCTRL_FRQRANGE_8M_gc
             | CLKCTRL_SELHF_CRYSTAL_gc
             | CLKCTRL_ENABLE_bm);

/* ｸﾘｽﾀﾙ用発振器始動確認 */
while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}

/* 休止中の節電のためRUNSTDBYを解除(0) */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

RTC用のｸﾛｯｸ元としてXOSCHFを選ぶにはEXTCLK設定がｸﾛｯｸ選択(RTC.CLKSEL)ﾚｼﾞｽﾀに書かれなければなりません。

図6-1. RTC.CLKSEL – 供給元としてXOSCHFを選択

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
							CLKSEL1,0	
ｱｸｾｽ種別	R	R	R	R	R	R	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ1,0 – CLKSEL1,0 : ｸﾛｯｸ選択** (Clock Select)

これらのﾋﾞｯﾄ書き込みはRTCｸﾛｯｸ(CLK_RTC)用の供給元を選びます。

次頁へ続く

図6-1 (続き). RTC.CLKSEL – 供給元としてXOSCHFを選択

値	0 0	0 1	1 0	1 1
名称	OSC32K	OSC1K	XOSC32K	EXTCLK
説明	OSC32Kからの 32.768kHz	OSC32Kからの 1.024kHz	XOSC32Kからの 32.768kHz	EXTCLK/XTALHF1ﾋﾟﾝ からの外部ｸﾛｯｸ

```
/* 供給元としてXOSCHFを使うようにRTCを構成設定 */
RTC.CLKSEL = RTC_CLKSEL_EXTCLK_gc;
```

これでRTCはｸﾛｯｸ元としてXOSCHFで構成設定して許可することができます。

この例用のｺｰﾄﾞはそれらのGitHub貯蔵庫のRTC-with-XOSCHFﾌｫﾙﾀﾞで入手可能です。

GitHubでｺｰﾄﾞ例を見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

7. XOSCHFとのTCD

TCD0用ｸﾛｯｸ元としてXOSCHFを使うには最初に先の2つの使用事例と同様にXOSCHFを許可してください。ｸﾘｽﾀﾙまたは入力ｸﾛｯｸの周波数は32MHzまでが可能です。

```
/* 周波数範囲16Mzと4K周期始動時間でｸﾘｽﾀﾙ用発振器許可 */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
             | CLKCTRL_CSUTHF_4K_gc
             | CLKCTRL_FRQRANGE_16M_gc
             | CLKCTRL_SELHF_CRYSTAL_gc
             | CLKCTRL_ENABLE_bm);

/* ｸﾘｽﾀﾙ用発振器始動確認 */
while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}

/* 休止中の節電のためRUNSTDBYを解除(0) */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

TCD0用のｸﾛｯｸ元としてXOSCHFを選ぶにはEXTCLK設定が制御A(TCD0.CTRLA)ﾚｼﾞｽﾀのｸﾛｯｸ選択(CLKSEL)ﾋﾞｯﾄ領域に書かれなければなりません。

図7-1. TCD0.CTRLA – TCD0ｸﾛｯｸ供給元としてXOSCHFを選択

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
		CLKSEL1,0		CNTPRES1,0		SYNCPRES1,0		ENABLE
ｱｸｾｽ種別	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ6,5 – CLKSEL1,0 : ｸﾛｯｸ選択** (Clock Select)

ｸﾛｯｸ選択ﾋﾞｯﾄはTCDｸﾛｯｸのｸﾛｯｸ元を選びます。

値	0 0	0 1	1 0	1 1
名称	OSCHF	PLL	EXTCLK	CLKPER
説明	内部高周波数発振器	PLL	外部ｸﾛｯｸまたは外部ｸﾘｽﾀﾙ用発振器	前置分周後主ｸﾛｯｸ(CLK_PER)

● **ﾋﾞｯﾄ4,3 – CNTPRES1,0 : 計数器前置分周器** (Counter Prescaler)

計数器前置分周器ﾋﾞｯﾄはTCD計数器ｸﾛｯｸの分周係数を選びます。

次頁へ続く

図7-1 (続き). TCD0.CTRLA – TCD0ｸﾛｯｸ供給元としてXOSCHFを選択

値	0 0	0 1	1 0	1 1
名称	DIV1	DIV4	DIV32	–
説明	1分周(分周なし)	4分周	32分周	(予約)

● ﾋﾞｯﾄ2,1 – SYNCPRES1,0 : 同期前置分周器 (Syncronization Prescaler)

同期前置分周器ﾋﾞｯﾄはTCDｸﾛｯｸの分周係数を選びます。

値	0 0	0 1	1 0	1 1
名称	DIV1	DIV2	DIV4	DIV8
説明	1分周(分周なし)	2分周	4分周	8分周

```
/* 供給元としてXOSCHF(16MHz)でTCDを構成設定 */
TCD0.CTRLA = TCD_CLKSEL_EXTCLK_gc | TCD_CNTPRES_DIV1_gc | TCD_SYNCPRES_DIV1_gc;

/* あなたの応用構成設定で置き換えてください。 */

/* TCD0許可 */
TCD0.CTRLA |= TCD_ENABLE_bm;
```

この例用のｺｰﾄﾞはそれらのGitHub貯蔵庫のTCD-with-XOSCHFﾌｫﾙﾀﾞで入手可能です。

GitHubでｺｰﾄﾞ例を見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

8. XOSCHFとPLLとのTCD

位相固定化閉路(PLL:Phase-Lock Loop)用ｸﾛｯｸ元としてXOSCHFを使うには最初に先の2つの使用事例と同様にXOSCHFを許可してください。ｸﾘｽﾀﾙまたは入力ｸﾛｯｸの周波数は最低16MHzでなければならず、48MHzまでの出力周波数を与える2倍または3倍に逓倍することができます。

```
/* 周波数範囲16Mzと4K周期始動時間でｸﾘｽﾀﾙ用発振器許可 */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
             | CLKCTRL_CSUTHF_4K_gc
             | CLKCTRL_FRQRANGE_16M_gc
             | CLKCTRL_SELHF_CRYSTAL_gc
             | CLKCTRL_ENABLE_bm);

/* ｸﾘｽﾀﾙ用発振器始動確認 */
while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}

/* 休止中の節電のためRUNSTDBYを解除(0) */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

PLL制御A(CLKCTRL.PLLCTRLA)ﾚｼﾞｽﾀのPLL用供給元選択(SOURCE)ﾋﾞｯﾄに'1'を書くことによって供給元としてXOSCHFを選び、同じﾚｼﾞｽﾀの逓倍係数(MULFAC)ﾋﾞｯﾄ領域に書くことによって逓倍係数を選んでください。

このﾚｼﾞｽﾀは構成設定変更保護(CCP)を持ち、故にﾚｼﾞｽﾀの解錠用の正しいﾀｲﾐﾝｸﾞを保証するためにcpufunc.hのccp_write_io関数が使われるべきです。

図8-1. CLKCTRL.PLLCTRLA – 供給元としてXOSCHFでPLLを許可

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
	RUNSTDBY	SOURCE					MULFAC1,0	
ｱｸｾｽ種別	R/W	R/W	R	R	R	R	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

次頁へ続く

図8-1 (続き). CLKCTRL.PLLCTRLA – 供給元としてXOSCHFでPLLを許可

● ビット6 – SOURCE : PLL用供給元選択 (Select Soutce for PLL)

このビットは位相固定化閉路(PLL)クロック元を制御します。

値	0	1
名称	OSCHF	XOSCHF
説明	PLL供給元として高周波数内部発振器	PLL供給元として高周波数外部クロックまたは外部高周波数発振器

● ビット1,0 – MULFAC1,0 : 逓倍係数 (Frequency Select)

このビット領域は位相固定化閉路(PLL)に対する倍率を制御します。

値	0 0	0 1	1 0	1 1
名称	DISABLE	2x	3x	–
説明	PLL禁止	2逓倍	3逓倍	(予約)

```
/* 供給元としてXOSCHFを使うようにPLLを設定し、3倍の逓倍係数を選択 */
ccp_write_io((uint8_t *) &CLKCTRL.PLLCTRLA, CLKCTRL_SOURCE_bm | CLKCTRL_MULFAC_3x_gc);
```

TCD0用クロック元としてPLLを選ぶには制御A(TCD0.CTRLA)レジスタのクロック選択(CLKSEL)ビット領域にPLL設定が書かれなければなりません。

図8-2. TCD0.CTRLA – 供給元としてPLLでTCD0を許可

ビット	7	6	5	4	3	2	1	0
		CLKSEL1,0		CNTPRES1,0		SYNCPRES1,0		ENABLE
アクセス種別	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
リセット値	0	0	0	0	0	0	0	0

● ビット6,5 – CLKSEL1,0 : クロック選択 (Clock Select)

クロック選択ビットはTCDクロックのクロック元を選びます。

値	0 0	0 1	1 0	1 1
名称	OSCHF	PLL	EXTCLK	CLKPER
説明	内部高周波数発振器	PLL	外部クロックまたは外部クリスタル用発振器	前置分周後主クロック(CLK_PER)

● ビット4,3 – CNTPRES1,0 : 計数器前置分周器 (Counter Prescaler)

計数器前置分周器ビットはTCD計数器クロックの分周係数を選びます。

値	0 0	0 1	1 0	1 1
名称	DIV1	DIV4	DIV32	–
説明	1分周(分周なし)	4分周	32分周	(予約)

● ビット2,1 – SYNCPRES1,0 : 同期前置分周器 (Syncronization Prescaler)

同期前置分周器ビットはTCDクロックの分周係数を選びます。

値	0 0	0 1	1 0	1 1
名称	DIV1	DIV2	DIV4	DIV8
説明	1分周(分周なし)	2分周	4分周	8分周

```
/* 供給元としてPLL(48MHz)でTCDを構成設定 */
TCD0.CTRLA = TCD_CLKSEL_PLL_gc | TCD_CNTPRES_DIV1_gc | TCD_SYNCPRES_DIV1_gc;

/* あなたの応用構成設定で置き換えてください。 */

/* TCD0許可 */
TCD0.CTRLA |= TCD_ENABLE_bm;
```

この例用のｺｰﾄﾞはそれらのGitHub貯蔵庫のTCD-with-XOSCHF-and-PLLﾌｫﾙﾀﾞで入手可能です。

GitHubでｺｰﾄﾞ例を見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

9. XOSCHFでのCFD

外部高周波数発振器(OXOSCHF)でｸﾛｯｸ障害検出(CFD:Clock Failure Detection)を使うには主ｸﾛｯｸ制御C(CLKCTRL.MCLKCTRLC)ﾚｼﾞｽﾀでｸﾛｯｸ障害検出元(CFDSRC)ﾋﾞｯﾄ領域にXOSCHF設定とｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄに'1'を書いてください。

このﾚｼﾞｽﾀは構成設定変更保護(CCP)を持ち、故にﾚｼﾞｽﾀの解錠用の正しいﾀｲﾐﾝｸﾞを保証するためにcpufunc.hのccp_write_io関数が使われるべきです。

図9-1. CLKCTRL.MCLKCTRLC – XOSCHFでCFDを許可

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
					CFDSRC1,0		CFDTST	CFDEN
アクセス種別	R	R	R	R	R/W	R/W	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ3,2 – CFDSRC1,0 : ｸﾛｯｸ障害検出元** (Clock Failure Detection Source)

このﾋﾞｯﾄ領域はｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄが'1'の時にどのｸﾛｯｸ元が監視されるかを制御します。

値	0 0	0 1	1 0	1 1
名称	CLKMAIN	XOSCHF	XOSC32K	–
説明	主ｸﾛｯｸ	外部高周波数発振器	外部32.768kHz発振器	(予約)

注: このﾋﾞｯﾄ領域はCFDENﾋﾞｯﾄが'1'で、主ｸﾛｯｸ割り込み制御(CLKCTRL.MCLKINTCTRL)のｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄと割り込み型(INTTYPE)ﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

● **ﾋﾞｯﾄ0 – CFDEN : ｸﾛｯｸ障害検出許可** (Clock Failure Detection Enable)

このﾋﾞｯﾄはｸﾛｯｸ障害検出(CFD)が許可されるか否かを制御します。

値	0	1
説明	CFDは禁止	CFDは許可

注: このﾋﾞｯﾄ領域はCFDENﾋﾞｯﾄが'1'で、主ｸﾛｯｸ割り込み制御(CLKCTRL.MCLKINTCTRL)のｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄと割り込み型(INTTYPE)ﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

```
/* XOSCHFでｸﾛｯｸ障害検出を許可 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_XOSCHF_gc | CLKCTRL_CFDEN_bm);
```

通常のCFD割り込みを許可するには主ｸﾛｯｸ割り込み制御(CLKCTRL.MCLKINTCTRL)ﾚｼﾞｽﾀで割り込み型(INTTYPE)ﾋﾞｯﾄに'0'とｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄに'1'を書いてください。このﾚｼﾞｽﾀもCCPを持ちます。

図9-2. CLKCTRL.MCLKINTCTRL – 通常CFD割り込みを許可

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
	INTTYPE							CFD
アクセス種別	R/W	R	R	R	R	R	R	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ7 – INTTYPE : 割り込み型** (Interrupt Type)

このﾋﾞｯﾄはｸﾛｯｸ障害検出(CFD)割り込みの型を制御します。

値	0	1
名称	INT	NMI
説明	通常割り込み	遮蔽不可割り込み

次頁へ続く

図9-2 (続き). CLKCTRL.MCLKINTCTRL – 通常CFD割り込みを許可

注: このﾋﾞｯﾄ領域は主ｸﾛｯｸ制御C(CLKCTRL.MCLKCTRLC)ﾚｼﾞｽﾀのｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄが'1'で、ｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄとこのﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

● **ﾋﾞｯﾄ0 – CFD : ｸﾛｯｸ障害検出割り込み許可** (Clock Failure Detection Interrupt Enable)

このﾋﾞｯﾄはｸﾛｯｸ障害検出(CFD)割り込みが許可されるか否かを制御します。

値	0	1
説明	CFD割り込みは禁止	CFD割り込みは許可

注: このﾋﾞｯﾄ領域は主ｸﾛｯｸ制御C(CLKCTRL.MCLKCTRLC)ﾚｼﾞｽﾀのｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄが'1'で、割り込み型(INTTYPE)ﾋﾞｯﾄとこのﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

```
/* CFDに対して通常割り込みを許可 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_CFD_bm);
```

割り込みが起動された時にCFDﾍﾞｸﾀが呼ばれます。割り込み処理ﾙｰﾁﾝ(ISR)から戻る前に、主ｸﾛｯｸ割り込み要求ﾌﾗｸﾞ(CLKCTRL.MCLKINTFLAGS)ﾚｼﾞｽﾀのｸﾛｯｸ障害検出(CFD)割り込み要求ﾌﾗｸﾞが解除(0)されなければなりません。これはこのﾌﾗｸﾞに'1'を書くことによって行われます。

図9-3. CLKCTRL.MCLKINTFLAGS – CFD割り込み要求ﾌﾗｸﾞを解除(0)

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
								CFD
ｱｸｾｽ種別	R	R	R	R	R	R	R	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ7 – CFD : ｸﾛｯｸ障害検出割り込み要求ﾌﾗｸﾞ** (Clock Failure Detection Interrupt Flag)

このﾋﾞｯﾄはこれに'1'を書くことによって解除(0)されます。このﾌﾗｸﾞはｸﾛｯｸ障害が検出された時に設定(1)されます。このﾋﾞｯﾄへの'0'書き込みは無効です。このﾋﾞｯﾄへの'1'書き込みはｸﾛｯｸ障害検出割り込み要求(CFD)ﾌﾗｸﾞを解除(0)します。

```
ISR(CLKCTRL_CFD_vect)
{
    /* この割り込みはCFDがXOSCHF停止を検出する毎に起動します。
     * 主ｸﾛｯｸ元はOSCHFで、故にCPUは影響を及ぼされません。 */
    LED0_toggle();

    /* CFD割り込み要求ﾌﾗｸﾞを解除(0) */
    CLKCTRL.MCLKINTFLAGS = CLKCTRL_CFD_bm;
}
```

この例用のｺｰﾄﾞはそれらのGitHub貯蔵庫のCFD-on-XOSCHFﾌｫﾙﾀﾞで入手可能です。

GitHubでｺｰﾄﾞ例を見てください。
貯蔵庫を閲覧するにはｸﾘｯｸしてください。

10. NMIとの主ｸﾛｯｸでのCFD

主ｸﾛｯｸでｸﾛｯｸ障害検出(CFD)を使うには主ｸﾛｯｸ制御C(CLKCTRL.MCLKCTR LC)ﾚｼﾞｽﾀでｸﾛｯｸ障害検出元(CFDSRC)ﾋﾞｯﾄ領域にCLKMAIN設定とｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄに'1'を書いてください。

このﾚｼﾞｽﾀは構成設定変更保護(CCP)を持ち、故にﾚｼﾞｽﾀの解錠用の正しいﾀｲﾐﾝｸﾞを保証するためにcpufunc.hのccp_write_io関数が使われるべきです。

図10-1. CLKCTRL.MCLKCTRLC – 主ｸﾛｯｸでCFDを許可

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
					CFDSRC1,0		CFDTST	CFDEN
ｱｸｾｽ種別	R	R	R	R	R/W	R/W	R/W	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ3,2 – CFDSRC1,0 : ｸﾛｯｸ障害検出元** (Clock Failure Detection Source)

このﾋﾞｯﾄ領域はｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄが'1'の時にどのｸﾛｯｸ元が監視されるかを制御します。

値	0 0	0 1	1 0	1 1
名称	CLKMAIN	XOSCHF	XOSC32K	–
説明	主ｸﾛｯｸ	外部高周波数発振器	外部32.768kHz発振器	(予約)

注: このﾋﾞｯﾄ領域はCFDENﾋﾞｯﾄが'1'で、主ｸﾛｯｸ割り込み制御(CLKCTRL.MCLKINTCTRL)のｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄと割り込み型(INTTYPE)ﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

● **ﾋﾞｯﾄ0 – CFDEN : ｸﾛｯｸ障害検出許可** (Clock Failure Detection Enable)

このﾋﾞｯﾄはｸﾛｯｸ障害検出(CFD)が許可されるか否かを制御します。

値	0	1
説明	CFDは禁止	CFDは許可

注: このﾋﾞｯﾄ領域はCFDENﾋﾞｯﾄが'1'で、主ｸﾛｯｸ割り込み制御(CLKCTRL.MCLKINTCTRL)のｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄと割り込み型(INTTYPE)ﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

```
/* 主ｸﾛｯｸでｸﾛｯｸ障害検出を許可 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_CLKMAIN_gc | CLKCTRL_CFDEN_bm);
```

遮蔽不可割り込み(NMI:Non-Maskable Interrupt)としてCFD割り込みを許可するには主ｸﾛｯｸ割り込み制御(CLKCTRL.MCLKINTCTRL)ﾚｼﾞｽﾀで割り込み型(INTTYPE)ﾋﾞｯﾄとｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄの両方に'1'を書いてください。このﾚｼﾞｽﾀもCCPを持ちます。

図10-2. CLKCTRL.MCLKINTCTRL – NMIとしてCFD割り込みを許可

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
	INTTYPE							CFD
ｱｸｾｽ種別	R/W	R	R	R	R	R	R	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ7 – INTTYPE : 割り込み型** (Interrupt Type)

このﾋﾞｯﾄはｸﾛｯｸ障害検出(CFD)割り込みの型を制御します。

値	0	1
名称	INT	NMI
説明	通常割り込み	遮蔽不可割り込み

注: このﾋﾞｯﾄ領域は主ｸﾛｯｸ制御C(CLKCTRL.MCLKCTRLC)ﾚｼﾞｽﾀのｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄが'1'で、ｸﾛｯｸ障害検出割り込み許可(CFD)ﾋﾞｯﾄとこのﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

● **ﾋﾞｯﾄ0 – CFD : ｸﾛｯｸ障害検出割り込み許可** (Clock Failure Detection Interrupt Enable)

このﾋﾞｯﾄはｸﾛｯｸ障害検出(CFD)割り込みが許可されるか否かを制御します。

値	0	1
説明	CFD割り込みは禁止	CFD割り込みは許可

注: このﾋﾞｯﾄ領域は主ｸﾛｯｸ制御C(CLKCTRL.MCLKCTRLC)ﾚｼﾞｽﾀのｸﾛｯｸ障害検出許可(CFDEN)ﾋﾞｯﾄが'1'で、割り込み型(INTTYPE)ﾋﾞｯﾄとこのﾋﾞｯﾄの両方が'1'の時に読み込み専用です。このﾋﾞｯﾄはｼｽﾃﾑ ﾘｾｯﾄが起こるまで読み込み専用に留まります。

```
/* CFDに対して遮蔽不可割り込みを許可 */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_INTTYPE_bm | CLKCTRL_CFD_bm);
```

CFD割り込みがNMIとして構成設定されてしまうと、割り込みが起動された時にCFDﾍﾞｸﾀの代わりにNMIﾍﾞｸﾀが呼ばれます。

NMI用割り込み処理ルーチン(ISR)に入ると、複数のNMI元が許可されている場合、割り込み元は見つけ出されなければなりません。主ｸﾛｯｸ割り込み要求ﾌﾗｸﾞ(CLKCTRL. MCLKINTFLAGS)ﾚｼﾞｽﾀのｸﾛｯｸ障害検出(CFD)割り込み要求ﾌﾗｸﾞはCFDが起動した場合に'1'です。

図10-3. CLKCTRL.MCLKINTFLAGS – CFD割り込み要求ﾌﾗｸﾞを調査

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
								CFD
ｱｸｾｽ種別	R	R	R	R	R	R	R	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ7 – CFD : ｸﾛｯｸ障害検出割り込み要求ﾌﾗｸﾞ** (Clock Failure Detection Interrupt Flag)

このﾋﾞｯﾄはこれに'1'を書くことによって解除(0)されます。このﾌﾗｸﾞはｸﾛｯｸ障害が検出された時に設定(1)されます。このﾋﾞｯﾄへの'0'書き込みは無効です。このﾋﾞｯﾄへの'1'書き込みはｸﾛｯｸ障害検出割り込み要求(CFD)ﾌﾗｸﾞを解除(0)します。

```
ISR(NMI_vect)
{
    /* どのNMIが起動されたか調査 */
    if (CLKCTRL.MCLKINTFLAGS & CLKCTRL_CFD_bm)
    {
        /* この割り込みは主ｸﾛｯｸ用供給元が途切れた場合に起動し、CFDは動いている違うｸﾛｯｸに切り替えることができます。
         * この場合はXOSCHFが途切れてOSCHFによって置き換えられます。
         * 従って主ｸﾛｯｸは4MHz/2=2MHzに減らされます。 */

        /* 永遠にLEDを交互切り替え */
        while (1)
        {
            LED0_toggle();
            _delay_ms(200);        // 16MHz=1600msから計算された200ms
        }
    }
    else
    {
        /* 違うNMIが起動されました。 */
    }
    /* 遮蔽不可割り込みﾋﾞｯﾄはﾘｾｯﾄによってのみ解除(0)することができます。 */
}
```

ｿﾌﾄｳｪｱ ﾘｾｯﾄはｿﾌﾄｳｪｱ ﾘｾｯﾄ ﾚｼﾞｽﾀ(RSTCTRL.SWRR)のｿﾌﾄｳｪｱ ﾘｾｯﾄ(SWRST)ﾋﾞｯﾄに'1'を書くことによって起動することができます。このﾚｼﾞｽﾀもCCPを持ちます。

図10-4. RSTCTRL.SWRR – ｿﾌﾄｳｪｱ ﾘｾｯﾄを起動

ﾋﾞｯﾄ	7	6	5	4	3	2	1	0
								SWRST
ｱｸｾｽ種別	R	R	R	R	R	R	R	R/W
ﾘｾｯﾄ値	0	0	0	0	0	0	0	0

● **ﾋﾞｯﾄ0 – SWRST : ｿﾌﾄｳｪｱ ﾘｾｯﾄ** (Software Reset)

このﾋﾞｯﾄが'1'を書かかれると、ｿﾌﾄｳｪｱ ﾘｾｯﾄが起こります。

このﾋﾞｯﾄは常に'0'として読みます。

```
/* ソフトウェア リセット */
ccp_write_io((uint8_t *) &RSTCTRL.SWRR, RSTCTRL_SWRST_bm);
```

この例用のｺｰﾄﾞはそれらのGitHub貯蔵庫のCFD-on-main-clock-with-NMIﾌｫﾙﾀﾞで入手可能です。

11. 改訂履歴

改訂	日付	注釈
A	2020年9月	初版文書公開

12. 追補

ここはこの技術概説で使われた各事例用のｺｰﾄﾞ例です。

例12-1. 外部ｸﾘｽﾀﾙとのXOSCHF

```
#define F_CPU 16000000ul

#include <avr/io.h>
#include <avr/cpufunc.h>

void CLOCK_XOSCHF_crystal_init(void);

static inline void LED0_init(void)
{
    PORTB.DIRSET = PIN3_bm;
}

int main(void)
{
    CLOCK_XOSCHF_crystal_init();

    /* ｸﾛｯｸ初期化成功を示すためにLED0をON */
    LED0_init();

    /* あなたの応用ｺｰﾄﾞで置き換えてください。 */
    while (1)
    {
    }
}

void CLOCK_XOSCHF_crystal_init(void)
{
    /* 周波数範囲16MHzと4K周期始動時間でｸﾘｽﾀﾙ用発振器を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                | CLKCTRL_CSUTHF_4K_gc
                | CLKCTRL_FRQRANGE_16M_gc
                | CLKCTRL_SELHF_CRYSTAL_gc
                | CLKCTRL_ENABLE_bm);

    /* ｸﾘｽﾀﾙ用発振器始動確認 */
    while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* 主ｸﾛｯｸ前置分周器解消 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, 0x00);

    /* 供給元としてXOSCHFを使うように主ｸﾛｯｸを設定、CLKOUTﾋﾟﾝ許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc | CLKCTRL_CLKOUT_bm);

    /* ｼｽﾃﾑ発振器変更完了待ち */
    while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }

    /* 休止中の節電のためにRUNSTDBYを解除(0) */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);

    /* 変更完了、主ｸﾛｯｸは16MHzです。 */
}
```

例12-2. 外部クロックとのXOSCHF

```
#define F_CPU 16000000ul

#include <avr/io.h>
#include <avr/cpufunc.h>

void CLOCK_XOSCHF_clock_init(void);

static inline void LED0_init(void)
{
    PORTB.DIRSET = PIN3_bm;
}

int main(void)
{
    CLOCK_XOSCHF_clock_init();

    /* クロック初期化成功を示すためにLED0をON */
    LED0_init();

    /* あなたの応用コードで置き換えてください。 */
    while (1)
    {
    }
}

void CLOCK_XOSCHF_clock_init(void)
{
    /* 外部(32MHz)クロック入力を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_SELHF_EXTCLOCK_gc
                                                 | CLKCTRL_ENABLE_bm);

    /* 主クロック前置分周器を設定 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_2X_gc | CLKCTRL_PEN_bm);

    /* 供給元としてXOSCHFを使うように主クロックを設定、CLKOUTピン許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc | CLKCTRL_CLKOUT_bm);

    /* システム発振器変更完了待ち */
    while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }

    /* 変更完了、主クロックは32MKHz/2=16MHzです。 */
}
```

例12-3. XOSCHFとのRTC

```
#include <avr/io.h>
#include <avr/cpufunc.h>
#include <avr/interrupt.h>

void CLOCK_XOSCHF_crystal_init(void);
void TIMER_RTC_init(void);

static inline void LED0_init(void)
{
    PORTB.DIRSET = PIN3_bm;
    PORTB.OUTSET = PIN3_bm;
}
```

```
static inline void LED0_toggle(void)
{
    PORTB.OUTTGL = PIN3_bm;
}

int main(void)
{
    CLOCK_XOSCHF_crystal_init();
    TIMER_RTC_init();
    LED0_init();

    /* 全体割り込み許可 */
    sei();

    /* あなたの応用ｺｰﾄﾞで置き換えてください。 */
    while (1)
    {
    }
}

void CLOCK_XOSCHF_crystal_init(void)
{
    /* 周波数範囲8MHzと1K周期始動時間でｸﾘｽﾀﾙ用発振器を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                | CLKCTRL_CSUTHF_1K_gc
                | CLKCTRL_FRQRANGE_8M_gc
                | CLKCTRL_SELHF_CRYSTAL_gc
                | CLKCTRL_ENABLE_bm);

    /* ｸﾘｽﾀﾙ用発振器始動確認 */
    while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* 休止中の節電のためRUNSTDBYを解除(0) */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
}

void TIMER_RTC_init(void)
{
    while (RTC.STATUS > 0)
    {
        /* RTCﾚｼﾞｽﾀ同期待ち */
    }

    /* 供給元としてXOSCHFを使うようにRTCを構成設定 */
    RTC.CLKSEL = RTC_CLKSEL_EXTCLK_gc;

    /* あなたの応用構成設定で置き換えてください。 */
    RTC.PER = 0xffff;
    RTC.INTCTRL = RTC_OVF_bm;
    RTC.CTRLA = RTC_PRESCALER_DIV32_gc | RTC_RTCEN_bm;
}

ISR(RTC_CNT_vect)
{
    /* この割り込みはRTC溢れ毎に起動します。 */
    LED0_toggle();
```

```
    /* RTC溢れ割り込み要求ﾌﾗｸﾞ解除(0) */
    RTC.INTFLAGS = RTC_OVF_bm;
}
```

例12-4. XOSCHFとのTCD

```
#include <avr/io.h>
#include <avr/cpufunc.h>

void CLOCK_XOSCHF_crystal_init(void);
void TIMER_TCD0_init(void);

int main(void)
{
    CLOCK_XOSCHF_crystal_init();
    TIMER_TCD0_init();

    /* あなたの応用ｺｰﾄﾞで置き換えてください。 */
    while (1)
    {
    }
}

void CLOCK_XOSCHF_crystal_init(void)
{
    /* 周波数範囲16MHzと4K周期始動時間でｸﾘｽﾀﾙ用発振器を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                | CLKCTRL_CSUTHF_4K_gc
                | CLKCTRL_FRQRANGE_16M_gc
                | CLKCTRL_SELHF_CRYSTAL_gc
                | CLKCTRL_ENABLE_bm);

    /* ｸﾘｽﾀﾙ用発振器始動確認 */
    while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* 休止中の節電のためRUNSTDBYを解除(0) */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
}

void TIMER_TCD0_init(void){
    /* 供給元としてXOSCHF(16MHz)でTCDを構成設定 */
    TCD0.CTRLA = TCD_CLKSEL_EXTCLK_gc | TCD_CNTPRES_DIV1_gc | TCD_SYNCPRES_DIV1_gc;

    /* あなたの応用構成設定で置き換えてください。 */

    /* TCD0許可 */
    TCD0.CTRLA |= TCD_ENABLE_bm;
}
```

例12-5. XOSCHFとPLLとのTCD

```
#include <avr/io.h>
#include <avr/cpufunc.h>

void CLOCK_XOSCHF_crystal_PLL_init(void);
void TIMER_TCD0_init(void);

int main(void)
{
```

```
    CLOCK_XOSCHF_crystal_PLL_init();
    TIMER_TCD0_init();

    /* あなたの応用ｺｰﾄﾞで置き換えてください。 */
    while (1)
    {
    }
}

void CLOCK_XOSCHF_crystal_PLL_init(void)
{
    /* 周波数範囲16MHzと4K周期始動時間でｸﾘｽﾀﾙ用発振器を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                | CLKCTRL_CSUTHF_4K_gc
                | CLKCTRL_FRQRANGE_16M_gc
                | CLKCTRL_SELHF_CRYSTAL_gc
                | CLKCTRL_ENABLE_bm);

    /* ｸﾘｽﾀﾙ用発振器始動確認 */
    while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* 供給元としてXOSCHFを使うようにPLLを設定、3倍の逓倍係数を選択 */
    ccp_write_io((uint8_t *) &CLKCTRL.PLLCTRLA, CLKCTRL_SOURCE_bm | CLKCTRL_MULFAC_3x_gc);

    /* 休止中の節電のためRUNSTDBYを解除(0) */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
}

void TIMER_TCD0_init(void)
{
    /* 供給元としてPLL(48MHz)でTCDを構成設定 */
    TCD0.CTRLA = TCD_CLKSEL_PLL_gc | TCD_CNTPRES_DIV1_gc | TCD_SYNCPRES_DIV1_gc;

    /* あなたの応用構成設定で置き換えてください。 */

    /* TCD0許可 */
    TCD0.CTRLA |= TCD_ENABLE_bm;
}
```

例12-6. XOSCHFでのCFD

```
#define F_CPU 16000000ul

#include <avr/io.h>
#include <avr/cpufunc.h>
#include <avr/interrupt.h>

void CLOCK_XOSCHF_crystal_init(void);
void CLOCK_CFD_XOSCHF_init(void);

static inline void LED0_init(void)
{
    PORTB.DIRSET = PIN3_bm;
    PORTB.OUTSET = PIN3_bm;
}

static inline void LED0_toggle(void)
{
```

```
    PORTB.OUTTGL = PIN3_bm;
}

int main(void)
{
    CLOCK_XOSCHF_crystal_init();
    CLOCK_CFD_XOSCHF_init();
    LED0_init();

    /* 全体割り込み許可 */
    sei();

    /* あなたの応用ｺｰﾄﾞで置き換えてください。 */
    while (1)
    {
    }
}

void CLOCK_XOSCHF_crystal_init(void)
{
    /* 周波数範囲16MHzと4K周期始動時間でｸﾘｽﾀﾙ用発振器を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                | CLKCTRL_CSUTHF_4K_gc
                | CLKCTRL_FRQRANGE_16M_gc
                | CLKCTRL_SELHF_CRYSTAL_gc
                | CLKCTRL_ENABLE_bm);

    /* ｸﾘｽﾀﾙ用発振器始動確認 */
    while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }
}

void CLOCK_CFD_XOSCHF_init(void)
{
    /* XOSCHFでｸﾛｯｸ障害検出を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_XOSCHF_gc | CLKCTRL_CFDEN_bm);

    /* CFDに対して通常割り込みを許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_CFD_bm);
}

ISR(CLKCTRL_CFD_vect)
{
    /* この割り込みはXOSCHF停止をCFDが検出する毎に起動します。
     * 主ｸﾛｯｸ元はOSCHFで、故にCPUは影響を及ぼされません。 */
    LED0_toggle();

    /* CFD割り込み要求ﾌﾗｸﾞを解除(0) */
    CLKCTRL.MCLKINTFLAGS = CLKCTRL_CFD_bm;
}
```

例12-7. NMIとの主ｸﾛｯｸでのCFD

```
#define F_CPU 16000000ul

#include <avr/io.h>
#include <avr/cpufunc.h>
#include <avr/interrupt.h>
#include <util/delay.h>
```

```
void CLOCK_XOSCHF_crystal_init(void);
void CLOCK_CFD_CLKMAIN_init(void);

static inline void LED0_init(void)
{
    PORTB.DIRSET = PIN3_bm;
}

static inline void LED0_toggle(void)
{
    PORTB.OUTTGL = PIN3_bm;
}

int main(void)
{
    CLOCK_XOSCHF_crystal_init();
    CLOCK_CFD_CLKMAIN_init();
    LED0_init();

    /* 全体割り込み許可 */
    sei();

    /* あなたの応用ｺｰﾄﾞで置き換えてください。 */
    while (1)
    {
        LED0_toggle();
        _delay_ms(200);
    }
}

void CLOCK_XOSCHF_crystal_init(void)
{
    /* 周波数範囲16MHzと4K周期始動時間でｸﾘｽﾀﾙ用発振器を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_RUNSTDBY_bm
                | CLKCTRL_CSUTHF_4K_gc
                | CLKCTRL_FRQRANGE_16M_gc
                | CLKCTRL_SELHF_CRYSTAL_gc
                | CLKCTRL_ENABLE_bm);

    /* ｸﾘｽﾀﾙ用発振器始動確認 */
    while (!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* 主ｸﾛｯｸ前置分周器を解消 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, 0x00);

    /* 供給元としてXOSCHFを使うように主ｸﾛｯｸを設定、CLKOUTﾋﾟﾝを許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc | CLKCTRL_CLKOUT_bm);

    /* ｼｽﾃﾑ発振器変更完了待ち */
    while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }

    /* 使用していない時に節電するためRUNSTDBYを解除(0) */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

```
    /* 変更完了で主ｸﾛｯｸは16MHz */
}

void CLOCK_CFD_CLKMAIN_init(void)
{
    /* 主ｸﾛｯｸでｸﾛｯｸ障害検出を許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_CLKMAIN_gc | CLKCTRL_CFDEN_bm);

    /* CFDに対して割り込みを許可 */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_INTTYPE_bm | CLKCTRL_CFD_bm);
}

ISR(NMI_vect)
{
    /* どのNMIが起動されたか調査 */
    if (CLKCTRL.MCLKINTFLAGS & CLKCTRL_CFD_bm)
    {
        /* この割り込みは主ｸﾛｯｸ用供給元停止の場合に起動し、CFDは別の動いているｸﾛｯｸに切り替えることができます。
         * この場合はXOSCHFが停止したことを意味し、OSCHFによって置き換えられます。
         * 従って主ｸﾛｯｸは4MHz/2=2MHzに減らされます。 */

        /* 永遠にLEDを交互切り替え */
        while (1)
        {
            LED0_toggle();
            _delay_ms(200);                    // 16MHz=1600msから計算された200ms
        }
    }
    else
    {
        /* 違うNMIが起動されました。 */
    }

    /* 遮蔽不可割り込みﾋﾞｯﾄはﾘｾｯﾄによってのみ解除(0)することができます。 */
}
```

Microchipウェブ サイト

Microchipはwww.microchip.com/で当社のウェブ サイト経由でのオンライン支援を提供します。このウェブ サイトはお客様がファイルや情報を容易に利用可能にするのに使われます。利用可能な情報のいくつかは以下を含みます。

- **製品支援** – データシートと障害情報、応用記述と試供プログラム、設計資源、使用者の手引きとハードウェア支援資料、最新ソフトウェア配布と保管されたソフトウェア
- **全般的な技術支援** – 良くある質問(FAQ)、技術支援要求、オンライン検討グループ、Microchip設計協力課程会員一覧
- **Microchipの事業** – 製品選択器と注文の手引き、最新Microchip報道発表、セミナーとイベントの一覧、Microchip営業所の一覧、代理店と代表する工場

製品変更通知サービス

Microchipの製品変更通知サービスはMicrochip製品を最新に保つのに役立ちます。加入者は指定した製品系統や興味のある開発ツールに関連する変更、更新、改訂、障害情報がある場合に必ず電子メール通知を受け取ります。

登録するにはwww.microchip.com/pcnへ行って登録指示に従ってください。

お客様支援

Microchip製品の使用者は以下のいくつかのチャネルを通して支援を受け取ることができます。

- 代理店または販売会社
- 最寄りの営業所
- 組み込み解決技術者(ESE:Embedded Solutions Engineer)
- 技術支援

お客様は支援に関してこれらの代理店、販売会社、またはESEに連絡を取るべきです。最寄りの営業所もお客様の手助けに利用できます。営業所と位置の一覧はこの資料の後ろに含まれます。

技術支援はwww.microchip.com/supportでのウェブ サイトを通して利用できます。

Microchipデバイス コード保護機能

Microchipデバイスでの以下のコード保護機能の詳細に注意してください。

- Microchip製品はそれら特定のMicrochipデータシートに含まれる仕様に合致します。
- Microchipは意図した方法と通常条件下で使われる時に、その製品系統が安全であると考えます。
- Microchipデバイスのコード保護機能を破ろうとする試みに使われる不正でおそらく違法な方法があります。当社はこれらの方法がMicrochipのデータシートに含まれた動作仕様外の方法でMicrochip製品を使うことが必要とされると確信しています。これらのコード保護機能を破ろうとする試みは、おそらく、Microchipの知的財産権に違反することなく達成することはできません。
- Microchipはそれのコードの完全性について心配されている何れのお客様とも共に働きたいと思います。
- Microchipや他のどの半導体製造業者もそれのコードの安全を保証することはできません。コード保護は製品が”破ることができない”ことを当社が保証すると言うことを意味しません。コード保護は常に進化しています。Microchipは当社製品のコード保護機能を継続的に改善することを約束します。Microchipのコード保護機能を破る試みはデジタル ミレニアム著作権法に違反するかもしれません。そのような行為があなたのソフトウェアや他の著作物に不正なアクセスを許す場合、その法律下の救済のために訴権を持つかもしれません。

法的通知

この刊行物に含まれる情報はMicrochip製品を使って設計する唯一の目的のために提供されます。デバイス応用などに関する情報は皆さまの便宜のためにだけ提供され、更新によって取り換えられるかもしれません。皆さまの応用が皆さまの仕様に合致するのを保証するのは皆さまの責任です。

この情報はMicrochipによって「現状そのまま」で提供されます。Microchipは非侵害、商品性、特定目的に対する適合性の何れの黙示的保証やその条件、品質、性能に関する保証を含め、明示的にも黙示的にもその情報に関連して書面または表記された書面または黙示の如何なる表明や保証もしません。

如何なる場合においても、Microchipは情報またはその使用に関連するあらゆる種類の間接的、特別的、懲罰的、偶発的または結果的な損失、損害、費用または経費に対して責任を負わないものとします。法律で認められている最大限の範囲で、情報またはその使用に関連する全ての請求に対するMicrochipの全責任は、もしあれば、情報のためにMicrochipへ直接支払った料金を超えないものとします。生命維持や安全応用でのMicrochipデバイスの使用は完全に購入者の危険性で、購入者はそのような使用に起因する全ての損害、請求、訴訟、費用からMicrochipを擁護し、補償し、免責にすることに同意します。他に言及されない限り、Microchipのどの知的財産権下でも暗黙的または違う方法で許認可は譲渡されません。

商標

品質管理ｼｽﾃﾑ

Microchipの品質管理ｼｽﾃﾑに関する情報についてはwww.microchip.com/qualityを訪ねてください。

日本語© *HERO* 2020.

本技術概説はMicrochipのTB3272技術概説(DS90003272A-2020年9月)の翻訳日本語版です。日本語では不自然となる重複する形容表現は省略されている場合があります。日本語では難解となる表現は大幅に意訳されている部分もあります。必要に応じて一部加筆されています。頁割の変更により、原本より頁数が少なくなっています。

必要と思われる部分には( )内に英語表記や略称などを残す形で表記しています。

青字の部分はﾘﾝｸとなっています。一般的に赤字の0,1は論理0,1を表します。その他の赤字は重要な部分を表します。

世界的な販売とサービス

米国	亜細亜/太平洋	亜細亜/太平洋	欧州
本社 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 技術支援: www.microchip.com/support ｳｪﾌﾞｱﾄﾞﾚｽ: www.microchip.com **ｱﾄﾗﾝﾀ** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **ｵｰｽﾁﾝ** TX Tel: 512-257-3370 **ﾎﾞｽﾄﾝ** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **ｼｶｺﾞ** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **ﾀﾞﾗｽ** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **ﾃﾞﾄﾛｲﾄ** Novi, MI Tel: 248-848-4000 **ﾋｭｰｽﾄﾝ** TX Tel: 281-894-5983 **ｲﾝﾃﾞｱﾅﾎﾟﾘｽ** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **ﾛｻﾝｾﾞﾙｽ** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **ﾛｰﾘｰ** NC Tel: 919-844-7510 **ﾆｭｰﾖｰｸ** NY Tel: 631-435-6000 **ｻﾝﾎｾ** CA Tel: 408-735-9110 Tel: 408-436-4270 **ｶﾅﾀﾞ - ﾄﾛﾝﾄ** Tel: 905-695-1980 Fax: 905-695-2078	**ｵｰｽﾄﾗﾘｱ - ｼﾄﾞﾆｰ** Tel: 61-2-9868-6733 **中国 - 北京** Tel: 86-10-8569-7000 **中国 - 成都** Tel: 86-28-8665-5511 **中国 - 重慶** Tel: 86-23-8980-9588 **中国 - 東莞** Tel: 86-769-8702-9880 **中国 - 広州** Tel: 86-20-8755-8029 **中国 - 杭州** Tel: 86-571-8792-8115 **中国 - 香港特別行政区** Tel: 852-2943-5100 **中国 - 南京** Tel: 86-25-8473-2460 **中国 - 青島** Tel: 86-532-8502-7355 **中国 - 上海** Tel: 86-21-3326-8000 **中国 - 瀋陽** Tel: 86-24-2334-2829 **中国 - 深圳** Tel: 86-755-8864-2200 **中国 - 蘇州** Tel: 86-186-6233-1526 **中国 - 武漢** Tel: 86-27-5980-5300 **中国 - 西安** Tel: 86-29-8833-7252 **中国 - 厦門** Tel: 86-592-2388138 **中国 - 珠海** Tel: 86-756-3210040	**ｲﾝﾄﾞ - ﾊﾝｶﾞﾛｰﾙ** Tel: 91-80-3090-4444 **ｲﾝﾄﾞ - ﾆｭｰﾃﾞﾘｰ** Tel: 91-11-4160-8631 **ｲﾝﾄﾞ - ﾌﾟﾈｰ** Tel: 91-20-4121-0141 **日本 - 大阪** Tel: 81-6-6152-7160 **日本 - 東京** Tel: 81-3-6880-3770 **韓国 - 大邱** Tel: 82-53-744-4301 **韓国 - ソウル** Tel: 82-2-554-7200 **ﾏﾚｰｼｱ - ｸｱﾗﾙﾝﾌﾟｰﾙ** Tel: 60-3-7651-7906 **ﾏﾚｰｼｱ - ﾍﾟﾅﾝ** Tel: 60-4-227-8870 **ﾌｨﾘﾋﾟﾝ - ﾏﾆﾗ** Tel: 63-2-634-9065 **ｼﾝｶﾞﾎﾟｰﾙ** Tel: 65-6334-8870 **台湾 - 新竹** Tel: 886-3-577-8366 **台湾 - 高雄** Tel: 886-7-213-7830 **台湾 - 台北** Tel: 886-2-2508-8600 **ﾀｲ - ﾊﾞﾝｺｸ** Tel: 66-2-694-1351 **ﾍﾞﾄﾅﾑ - ﾎｰﾁﾐﾝ** Tel: 84-28-5448-2100	**ｵｰｽﾄﾘｱ - ｳﾞｪﾙｽ** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **ﾃﾞﾝﾏｰｸ - ｺﾍﾟﾝﾊｰｹﾞﾝ** Tel: 45-4485-5910 Fax: 45-4485-2829 **ﾌｨﾝﾗﾝﾄﾞ - ｴｽﾎﾟｰ** Tel: 358-9-4520-820 **ﾌﾗﾝｽ - ﾊﾟﾘ** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **ﾄﾞｲﾂ - ｶﾞﾙﾋﾝｸﾞ** Tel: 49-8931-9700 **ﾄﾞｲﾂ - ﾊｰﾝ** Tel: 49-2129-3766400 **ﾄﾞｲﾂ - ﾊｲﾙﾌﾞﾛﾝ** Tel: 49-7131-72400 **ﾄﾞｲﾂ - ｶｰﾙｽﾙｰｴ** Tel: 49-721-625370 **ﾄﾞｲﾂ - ﾐｭﾝﾍﾝ** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **ﾄﾞｲﾂ - ﾛｰｾﾞﾝﾊｲﾑ** Tel: 49-8031-354-560 **ｲｽﾗｴﾙ - ﾗｰﾅﾅ** Tel: 972-9-744-7705 **ｲﾀﾘｱ - ﾐﾗﾉ** Tel: 39-0331-742611 Fax: 39-0331-466781 **ｲﾀﾘｱ - ﾊﾟﾄﾞﾊﾞ** Tel: 39-049-7625286 **ｵﾗﾝﾀﾞ - ﾃﾞﾙｰﾈﾝ** Tel: 31-416-690399 Fax: 31-416-690340 **ﾉﾙｳｪｰ - ﾄﾛﾝﾊｲﾑ** Tel: 47-72884388 **ﾎﾟｰﾗﾝﾄﾞ - ﾜﾙｼｬﾜ** Tel: 48-22-3325737 **ﾙｰﾏﾆｱ - ﾌﾞｶﾚｽﾄ** Tel: 40-21-407-87-50 **ｽﾍﾟｲﾝ - ﾏﾄﾞﾘｰﾄﾞ** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **ｽｳｪｰﾃﾞﾝ - ｲｪｰﾃﾎﾞﾘ** Tel: 46-31-704-60-40 **ｽｳｪｰﾃﾞﾝ - ｽﾄｯｸﾎﾙﾑ** Tel: 46-8-5090-4654 **ｲｷﾞﾘｽ - ｳｫｰｷﾝｶﾞﾑ** Tel: 44-118-921-5800 Fax: 44-118-921-5820