TB3272

Getting Started with External High-Frequency Oscillator on AVR® DB MCUs

Introduction

Author: Egil Rotevatn, Microchip Technology Inc.

The External High-Frequency Oscillator (XOSCHF) enables the use of an external crystal or an external clock signal up to 32 MHz. This can be used as a clock source for the Main Clock (CLK_MAIN), the Real-Time Counter (RTC), and the 12-Bit Timer/Counter Type D (TCDn).

The Clock Failure Detection (CFD) feature can be used to detect if the output from a clock source stops, and can switch the Main Clock to a different clock source to continue operation or shutdown operation safely. This feature is most useful in applications where XOSCHF is in use, for instance to allow safe shutdown of a Functional Safety application.

This technical brief describes how the XOSCHF and CFD is used on the AVR® DB family of microcontrollers and covers the following use cases:

- **XOSCHF with External Crystal:**
  Initialize XOSCHF for external crystal and change the main clock source to XOSCHF.
- **XOSCHF with External Clock:**
  Initialize XOSCHF for external clock signal and change the main clock source to XOSCHF.
- **RTC with XOSCHF:**
  Use XOSCHF as clock source for the RTC.
- **TCD with XOSCHF:**
  Use XOSCHF as clock source for TCD0.
- **TCD with XOSCHF and PLL:**
  Initialize the PLL with XOSCHF as clock source and use the PLL as clock source for TCD0.
- **CFD on XOSCHF:**
  Initialize the CFD to monitor XOSCHF, enable the interrupt, and toggle an LED if the clock source fails.
- **CFD on Main Clock with NMI:**
  Initialize the CFD to monitor Main Clock with XOSCHF as clock source, enable the interrupt as a Non-Maskable Interrupt (NMI), and toggle an LED with frequency derived from Main Clock.

The code examples are available in GitHub:

© 2020 Microchip Technology Inc.

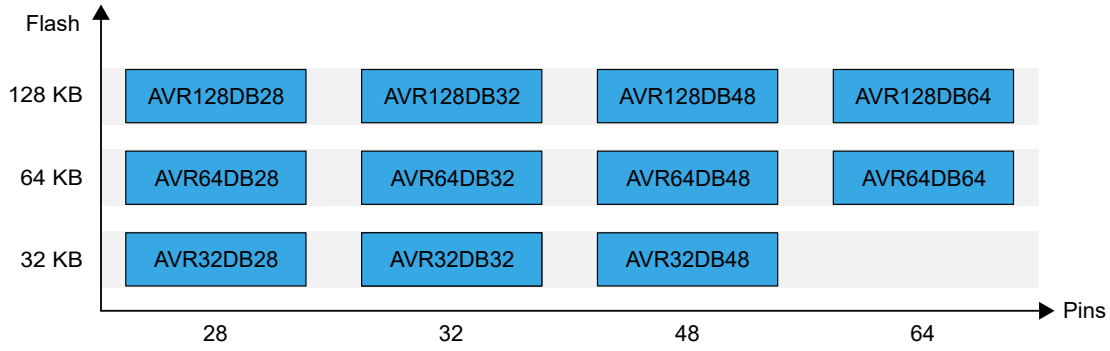
Table of Contents

1. Relevant Devices

This section lists the relevant devices for this document. The following figures show the different family devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features
- Horizontal migration to the left reduces the pin count and, therefore, the available features
- Devices with different Flash memory sizes typically also have different SRAM and EEPROM

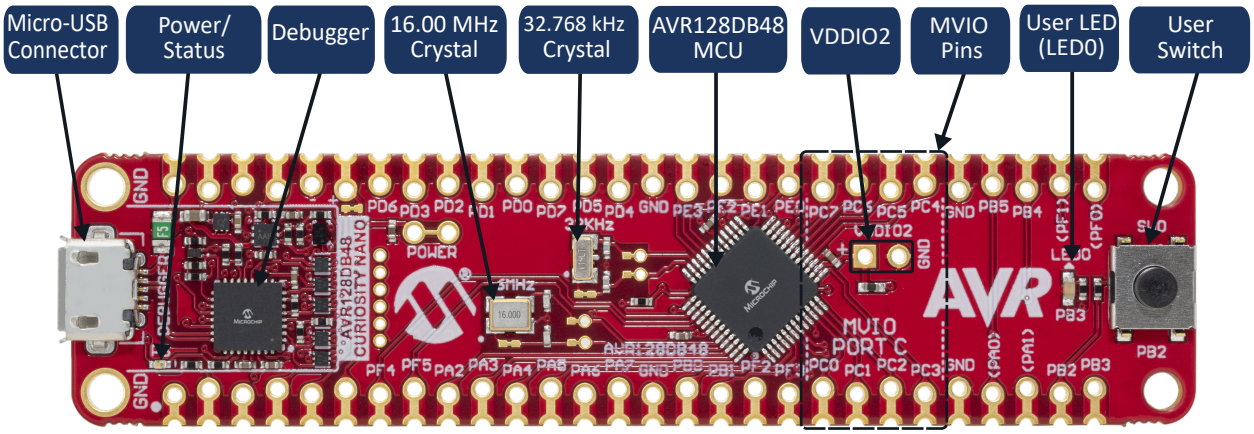
Figure 1-1. AVR® DB Family Overview



2. Hardware Configuration

The code examples were developed on the AVR128DB48 Curiosity Nano (EV35L43A) development board. It has a 16 MHz crystal on the board that can be used for most of the use-cases in this technical brief. The crystal must be disconnected if using the board with an external clock signal.

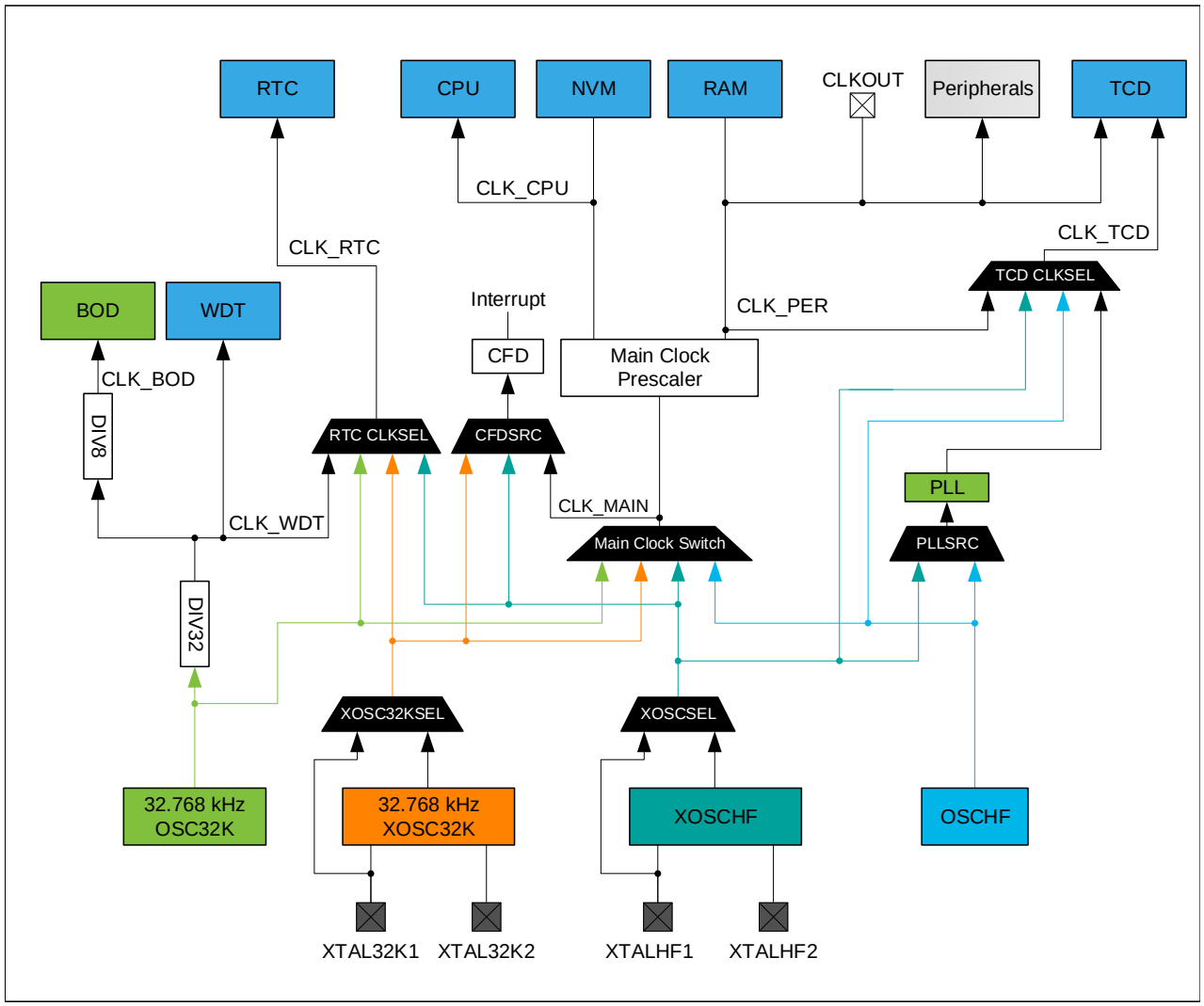
Figure 2-1. AVR128DB48 Curiosity Nano (EV35L43A) Development Board



3. Overview

The Clock Controller (CLKCTRL) block diagram shows how XOSCHF can be routed in AVR® DB family devices, as explained in the following sections.

Figure 3-1. CLKCTRL Block Diagram



4. XOSCHF with External Crystal

To use XOSCHF with external crystal, the SELHF bit in the External High-Frequency Oscillator Control A (CLKCTRL.XOSCHFCTRLA) register must be written to '0' and the ENABLE bit must be written to '1'.

In addition, the Frequency Range (FRQRANGE) bit field must be written to a setting equal to or higher than the frequency of the connected crystal, and the Crystal Start-up Time (CSUTHF) bit field can be written to select the number of cycles to wait before the clock is considered stable.

To make sure the oscillator runs as expected before it is requested by any modules, the Run Standby (RUNSTDBY) bit can be enabled.

This register has Configuration Change Protection (CCP), so the `ccp_write_io` function in `cpufunc.h` should be used to ensure correct timing for the unlock of the register.

Figure 4-1. CLKCTRL.XOSCHFCTRLA – Enable XOSCHF

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY		CSUTHF[1:0]		FRQRANGE[1:0]		SELHF	ENABLE
Access	R/W		R/W	R/W	R/W	R/W	R/W	R/W
Reset	0		0	0	0	0	0	0

Bit 7 – RUNSTDBY Run Standby
This bit controls whether the External High-Frequency Oscillator (XOSCHF) is always running or not, when the ENABLE bit is '1'.

Value	Description
0	The XOSCHF oscillator will only run when requested by a peripheral or by the main clock (1)
1	The XOSCHF oscillator will always run in Active, Idle and Standby sleep modes (2)

Notes:

1. The requesting peripheral, or the main clock, must take the oscillator start-up time into account.
2. The oscillator signal is only available if requested, and will be available after two XOSCHF cycles, if the initial crystal start-up time has already ended.

Bits 5:4 – CSUTHF[1:0] Crystal Start-up Time
This bit field controls the start-up time for the External High-Frequency Oscillator (XOSCHF), when the Source Select (SELHF) bit is '0'.

Value	Name	Description
0x0	256	256 XOSCHF cycles
0x1	1K	1K XOSCHF cycles
0x2	4K	4K XOSCHF cycles
0x3	-	Reserved

Note: This bit field is read-only when the ENABLE bit or the External Crystal/Clock Status (XOSCHFS) bit in the Main Clock Status (MCLKSTATUS) register is '1'.

Bits 3:2 – FRQRANGE[1:0] Frequency Range
This bit field controls the maximum frequency supported for the external crystal. The larger the range selected, the higher the current consumption by the oscillator.

Value	Name	Description
0x0	8M	Max. 8 MHz XTAL frequency
0x1	16M	Max. 16 MHz XTAL frequency
0x2	24M	Max. 24 MHz XTAL frequency
0x3	32M	Max. 32 MHz XTAL frequency

Note: If a crystal with a frequency larger than the maximum supported CLK_CPU frequency is used and used as the main clock, it is necessary to divide it down by writing the appropriate configuration to the PDIV bit field in the Main Clock Control B register.

Bit 1 – SELHF Source Select
This bit controls the source of the External High-Frequency Oscillator (XOSCHF).

Value	Name	Description
0	CRYSTAL	External Crystal on the XTALHF1 and XTALHF2 pins
1	EXTCLOCK	External Clock on the XTALHF1 pin

Note: This bit field is read-only when the ENABLE bit or the External Crystal/Clock Status (XOSCHFS) bit in the Main Clock Status (MCLKSTATUS) register is '1'.

Bit 0 – ENABLE Enable
This bit controls whether the External High-Frequency Oscillator (XOSCHF) is enabled or not.

Value	Description
0	The XOSCHF oscillator is disabled
1	The XOSCHF oscillator is enabled, and overrides normal port operation for the respective oscillator pins

```
/* Enable crystal oscillator with frequency range 16 MHz and 4K cycles start-up time */
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);
```

After enabling the XOSCHF, wait for the clock source to stabilize by polling the External Clock Status (EXTS) bit in the Main Clock Status (CLKCTRL.MCLKSTATUS) register until the bit is read as '1'.

Figure 4-2. CLKCTRL.MCLKSTATUS – Wait for XOSCHF to start

Bit	7	6	5	4	3	2	1	0
			PLLS	EXTS	XOSC32KS	OSC32KS	OSCHFS	SOSC
Access			R	R	R	R	R	R
Reset			0	0	0	0	0	0

Bit 4 – EXTS External Crystal/Clock Status

Value	Description
0	The external high-frequency crystal is not stable, when the Source Select (SELHF) bit in the External High-Frequency Oscillator Control A (CLKCTRL.XOSCHFCTRLA) register is '0'. The external high-frequency clock is not running, when the SELHF bit is '1'.
1	The external high-frequency crystal is stable, when the SELHF bit is '0'. The external high-frequency clock is running, when the SELHF bit is '1'.

```
/* Confirm crystal oscillator start-up */
while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}
```

The Main Clock prescaler shall be configured to give a maximum frequency of 24 MHz. If Main Clock will use XOSCHF with a crystal running faster than 24 MHz as the clock source, a prescaler setting can be written to the Prescaler Division (PDIV) bit field and '1' to the Prescaler Enable (PEN) bit in the Main Clock Control B (CLKCTRL.MCLKCTRLB) register. The AVR128DB48 Curiosity Nano has a 16 MHz crystal which allows disabling the prescaler by writing '0' to the PEN bit.

Main Clock Control B also has CCP.

Figure 4-3. CLKCTRL.MCLKCTRLB – Configure Main Clock prescaler

Bit	7	6	5	4	3	2	1	0
				PDIV[3:0]				PEN
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bits 4:1 – PDIV[3:0] Prescaler Division
This bit field controls the division ratio of the Main Clock (CLK_MAIN) prescaler, when the Prescaler (PEN) bit is '1'.

Value	Name	Description
0x0	2X	Divide by 2
0x1	4X	Divide by 4
0x2	8X	Divide by 8
0x3	16X	Divide by 16
0x4	32X	Divide by 32
0x5	64X	Divide by 64
0x8	6X	Divide by 6
0x9	10X	Divide by 10
0xA	12X	Divide by 12
0xB	24X	Divide by 24
0xC	48X	Divide by 48
Other	-	Reserved

Note: Configuration of the input frequency (CLK_MAIN) and prescaler settings must not exceed the allowed maximum frequency of the peripheral clock (CLK_PER) or CPU clock (CLK_CPU). Refer to the *Electrical Characteristics* section for further information.

Bit 0 – PEN Prescaler Enable
This bit controls whether the Main Clock (CLK_MAIN) prescaler is enabled or not.

Value	Description
0	The CLK_MAIN prescaler is disabled
1	The CLK_MAIN prescaler is enabled, and the division ratio is controlled by the Prescaler Division (PDIV) bit field

```
/* Clear Main Clock Prescaler */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, 0x00);
```

Change Main Clock source by writing the EXTCLK setting to the Clock Select (CLKSEL) bit field in the Main Clock Control A (CLKCTRL.MCLKCTRLA) register. The Main Clock Out (CLKOUT) bit can be written to '1' to output the Main Clock on the CLKOUT pin. This register also has CCP.

Figure 4-4. CLKCTRL.MCLKCTRLA – Change source to XOSCHF

Bit	7	6	5	4	3	2	1	0
	CLKOUT				CLKSEL[3:0]			
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

Bit 7 – CLKOUT Main Clock Out
This bit controls whether the main clock is available on the Main Clock Out (CLKOUT) pin or not, when the main clock is running.
This bit is cleared when a '0' is written to it or when a Clock Failure Detection (CFD) condition with the main clock as source occurs.
This bit is set when a '1' is written to it.

Value	Description
0	The main clock is not available on the CLKOUT pin
1	The main clock is available on the CLKOUT pin

Bits 3:0 – CLKSEL[3:0] Clock Select
This bit field controls the source for the Main Clock (CLK_MAIN).

Value	Name	Description
0x0	OSCHF	Internal high-frequency oscillator
0x1	OSC32K	32.768 kHz internal oscillator
0x2	XOSC32K	32.768 kHz external crystal oscillator
0x3	EXTCLK	External clock or external crystal, depending on the SELHF bit in XOSCHFCTRLA
Other	Reserved	Reserved

```
/* Set the main clock to use XOSCHF as source, and enable the CLKOUT pin */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA,
             CLKCTRL_CLKSEL_EXTCLK_gc | CLKCTRL_CLKOUT_bm);
```

After changing the Main Clock source, wait for the switch to complete by polling the Main Clock Oscillator Changing (SOSC) bit in the Main Clock Status (CLKCTRL.MCLKSTATUS) register until the bit is read as '0'.

Figure 4-5. CLKCTRL.MCLKSTATUS – Wait for Main Clock to change

Bit	7	6	5	4	3	2	1	0
			PLLS	EXTS	XOSC32KS	OSC32KS	OSCHFS	SOSC
Access			R	R	R	R	R	R
Reset			0	0	0	0	0	0

Bit 0 – SOSC Main Clock Oscillator Changing

Value	Description
0	The clock source for CLK_MAIN is not undergoing a switch
1	The clock source for CLK_MAIN is undergoing a switch and will change as soon as the new source is stable

```
/* Wait for system oscillator changing to complete */
while(CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
{
    ;
}
```

The Run Standby bit can now be cleared since the output is now is used by Main Clock and there is no need to force enable the oscillator.

```
    /* Clear RUNSTDBY for power save during sleep */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA,
                 CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

The code for this example is available in the **XOSCHF-with-external-crystal** folder in these github repositories:

5. XOSCHF with External Clock

To use XOSCHF with an external clock signal, the SELHF bit and the ENABLE bit in the External High-Frequency Oscillator Control A (CLKCTRL.XOSCHFCTRLA) register must be written to '1'.

This register has Configuration Change Protection (CCP), so the `ccp_write_io` function in `cpufunc.h` should be used to ensure correct timing for the unlock of the register.

Figure 5-1. CLKCTRL.XOSCHFCTRLA – Enable XOSCHF

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY		CSUTHF[1:0]		FRQRANGE[1:0]		SELHF	ENABLE
Access	R/W		R/W	R/W	R/W	R/W	R/W	R/W
Reset	0		0	0	0	0	0	0

Bit 1 – SELHF Source Select
This bit controls the source of the External High-Frequency Oscillator (XOSCHF).

Value	Name	Description
0	CRYSTAL	External Crystal on the XTALHF1 and XTALHF2 pins
1	EXTCLOCK	External Clock on the XTALHF1 pin

Note: This bit field is read-only when the ENABLE bit or the External Crystal/Clock Status (XOSCHFS) bit in the Main Clock Status (MCLKSTATUS) register is '1'.

Bit 0 – ENABLE Enable
This bit controls whether the External High-Frequency Oscillator (XOSCHF) is enabled or not.

Value	Description
0	The XOSCHF oscillator is disabled
1	The XOSCHF oscillator is enabled, and overrides normal port operation for the respective oscillator pins

```
/* Enable external clock input */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA,
             CLKCTRL_SELHF_EXTCLOCK_gc | CLKCTRL_ENABLE_bm);
```

If changing Main Clock to use XOSCHF as the clock source, first make sure to set the Main Clock prescaler to output maximum 24 MHz by writing a prescaler setting to the Prescaler Division (PDIV) bit field and '1' to the Prescaler Enable (PEN) bit in the Main Clock Control B (CLKCTRL.MCLKCTRLB) register. This register also has CCP.

Figure 5-3. CLKCTRL.MCLKCTRLB – Configure Main Clock prescaler

Bit	7	6	5	4	3	2	1	0
				PDIV[3:0]				PEN
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bits 4:1 – PDIV[3:0] Prescaler Division
This bit field controls the division ratio of the Main Clock (CLK_MAIN) prescaler, when the Prescaler (PEN) bit is '1'.

Value	Name	Description
0x0	2X	Divide by 2
0x1	4X	Divide by 4
0x2	8X	Divide by 8
0x3	16X	Divide by 16
0x4	32X	Divide by 32
0x5	64X	Divide by 64
0x8	6X	Divide by 6
0x9	10X	Divide by 10
0xA	12X	Divide by 12
0xB	24X	Divide by 24
0xC	48X	Divide by 48
Other	-	Reserved

Note: Configuration of the input frequency (CLK_MAIN) and prescaler settings must not exceed the allowed maximum frequency of the peripheral clock (CLK_PER) or CPU clock (CLK_CPU). Refer to the *Electrical Characteristics* section for further information.

Bit 0 – PEN Prescaler Enable
This bit controls whether the Main Clock (CLK_MAIN) prescaler is enabled or not.

Value	Description
0	The CLK_MAIN prescaler is disabled
1	The CLK_MAIN prescaler is enabled, and the division ratio is controlled by the Prescaler Division (PDIV) bit field

```
/* Set Main Clock Prescaler */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB,
             CLKCTRL_PDIV_2X_gc | CLKCTRL_PEN_bm);
```

Change Main Clock source by writing the EXTCLK setting to the Clock Select (CLKSEL) bit field in the Main Clock Control A (CLKCTRL.MCLKCTRLA) register. The Main Clock Out (CLKOUT) bit can be written to '1' to output the Main Clock on the CLKOUT pin. This register also has CCP.

Figure 5-4. CLKCTRL.MCLKCTRLA – Change source to XOSCHF

Bit	7	6	5	4	3	2	1	0
	CLKOUT				CLKSEL[3:0]			
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

Bit 7 – CLKOUT Main Clock Out
This bit controls whether the main clock is available on the Main Clock Out (CLKOUT) pin or not, when the main clock is running.
This bit is cleared when a '0' is written to it or when a Clock Failure Detection (CFD) condition with the main clock as source occurs.
This bit is set when a '1' is written to it.

Value	Description
0	The main clock is not available on the CLKOUT pin
1	The main clock is available on the CLKOUT pin

Bits 3:0 – CLKSEL[3:0] Clock Select
This bit field controls the source for the Main Clock (CLK_MAIN).

Value	Name	Description
0x0	OSCHF	Internal high-frequency oscillator
0x1	OSC32K	32.768 kHz internal oscillator
0x2	XOSC32K	32.768 kHz external crystal oscillator
0x3	EXTCLK	External clock or external crystal, depending on the SELHF bit in XOSCHFCTRLA
Other	Reserved	Reserved

```
/* Set the main clock to use XOSCHF as source, and enable the CLKOUT pin */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA,
             CLKCTRL_CLKSEL_EXTCLK_gc | CLKCTRL_CLKOUT_bm);
```

After changing the Main Clock source, wait for the switch to complete by polling the Main Clock Oscillator Changing (SOSC) bit in the Main Clock Status (CLKCTRL.MCLKSTATUS) register until the bit is read as '0'.

Figure 5-5. CLKCTRL.MCLKSTATUS – Wait for Main Clock to change

Bit	7	6	5	4	3	2	1	0
			PLLS	EXTS	XOSC32KS	OSC32KS	OSCHFS	SOSC
Access			R	R	R	R	R	R
Reset			0	0	0	0	0	0

Bit 0 – SOSC Main Clock Oscillator Changing

Value	Description
0	The clock source for CLK_MAIN is not undergoing a switch
1	The clock source for CLK_MAIN is undergoing a switch and will change as soon as the new source is stable

```
/* Wait for system oscillator changing to complete */
while(CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
{
    ;
}
```

The code for this example is available in the **XOSCHF-with-external-clock** folder in these github repositories:

6. RTC with XOSCHF

To use XOSCHF as clock source for the Real-Time Counter (RTC), first enable the XOSCHF similarly to the first two use cases. The crystal or input clock frequency can be up to ¼ of the main clock frequency.

⚠WARNING The 16 MHz crystal on the ATtiny1627 Curiosity Nano will generate a too high frequency for this example. Replace the crystal or use an external clock with maximum ¼ of the main clock frequency to run within spec.

```
/* Enable crystal oscillator with frequency range 8 MHz and 1K cycles start-up time */
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);

/* Confirm crystal oscillator start-up */
while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}

/* Clear RUNSTDBY for power save during sleep */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA,
             CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

To select the XOSCHF as clock source for RTC, the EXTCLK setting must be written to the Clock Select (RTC.CLKSEL) register.

Figure 6-1. RTC.CLKSEL – Select XOSCHF as source

Bit	7	6	5	4	3	2	1	0
							CLKSEL[1:0]	
Access							R/W	R/W
Reset							0	0

Bits 1:0 – CLKSEL[1:0] Clock Select
Writing these bits select the source for the RTC clock (CLK_RTC).
When configuring the RTC to use either XOSC32K or the external clock on XTAL32K1, XOSC32K needs to be enabled, and the Source Select (SEL) bit and Run Standby (RUNSTDBY) bit in the XOSC32K Control A of the Clock Controller (CLKCTRL.XOSC32KCTRLA) register must be configured accordingly.

Value	Name	Description
0x0	OSC32K	32.768 kHz from OSC32K
0x1	OSC1K	1.024 kHz from OSC32K
0x2	XTAL32K	32.768 kHz from XOSC32K or external clock from XTAL32K1
0x3	EXTCLK	External clock from the EXTCLK pin

```
/* Configure RTC to use XOSCHF as source */
RTC.CLKSEL = RTC_CLKSEL_EXTCLK_gc;
```

Now the RTC can be configured and enabled with XOSCHF as clock source.

The code for this example is available in the **RTC-with-XOSCHF** folder in these github repositories:

View Code Example on GitHub
Click to browse repository

7. TCD with XOSCHF

To use XOSCHF as clock source for TCD0, first enable the XOSCHF similarly to the first two use cases. The crystal or input clock frequency can be up to 32 MHz.

```
/* Enable crystal oscillator with frequency range 16 MHz and 4K cycles start-up time */
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);

/* Confirm crystal oscillator start-up */
while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}

/* Clear RUNSTDBY for power save during sleep */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA,
             CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

To select the XOSCHF as clock source for TCD0, the EXTCLK setting must be written to the Clock Select (CLKSEL) bit field in the Control A (TCD0.CTRLA) register.

Figure 7-1. TCD0.CTRLA – Enable TCD0 with PLL as source

Bit	7	6	5	4	3	2	1	0
		CLKSEL[1:0]		CNTPRES[1:0]		SYNCPRES[1:0]		ENABLE
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bits 6:5 – CLKSEL[1:0] Clock Select
The Clock Select bits select the clock source of the TCD clock.

Value	Name	Description
0x0	OSCHF	Internal High-Frequency Oscillator
0x1	PLL	PLL
0x2	EXTCLK	External clock
0x3	CLK_PER	Peripheral clock

Bits 4:3 – CNTPRES[1:0] Counter Prescaler
The Counter Prescaler bits select the division factor of the TCD counter clock.

Value	Name	Description
0x0	DIV1	Division factor 1
0x1	DIV4	Division factor 4
0x2	DIV32	Division factor 32
0x3	-	Reserved

Bits 2:1 – SYNCPRES[1:0] Synchronization Prescaler
The Synchronization Prescaler bits select the division factor of the TCD clock.

Value	Name	Description
0x0	DIV1	Division factor 1
0x1	DIV2	Division factor 2
0x2	DIV4	Division factor 4
0x3	DIV8	Division factor 8

```
/* Configure the TCD with XOSCHF (16 MHz) as source */
TCD0.CTRLA = TCD_CLKSEL_EXTCLK_gc | TCD_CNTPRES_DIV1_gc | TCD_SYNCPRES_DIV1_gc;

/* Replace with your application configuration */

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

The code for this example is available in the **TCD-with-XOSCHF** folder in these github repositories:

8. TCD with XOSCHF and PLL

To use XOSCHF as clock source for the Phase-Lock Loop (PLL), first enable the XOSCHF similarly to first two use cases. The crystal or input clock frequency must be at least 16 MHz and can be multiplied two or three times, giving an output frequency up to 48 MHz.

```
/* Enable crystal oscillator with frequency range 16 MHz and 4K cycles start-up time */
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);

/* Confirm crystal oscillator start-up */
while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
{
    ;
}

/* Clear RUNSTDBY for power save during sleep */
ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA,
             CLKCTRL.XOSCHFCTRLA & ~CLKCTRL_RUNSTDBY_bm);
```

Select XOSCHF as source by writing a '1' to the SOURCE bit in the PLL Control A (CLKCTRL.PLLCTRLA) register, and select the multiplication factor by writing to the MULFAC bit field in the same register.

This register has Configuration Change Protection (CCP), so the `ccp_write_io` function in `cpufunc.h` should be used to ensure correct timing for the unlock of the register.

Figure 8-1. CLKCTRL.PLLCTRLA – Enable PLL with XOSCHF as source

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY	SOURCE					MULFAC[1:0]	
Access	R/W	R/W					R/W	R/W
Reset	0	0					0	0

Bit 6 – SOURCE Select Source for PLL
This bit controls the Phase-Locked Loop (PLL) clock source.

Value	Name	Description
0	OSCHF	High-frequency internal oscillator as PLL source
1	XOSCHF	High-frequency external clock or external high-frequency oscillator as PLL source

Bits 1:0 – MULFAC[1:0] Multiplication Factor
This bit field controls the multiplication factor for the Phased-Locked Loop (PLL).

Value	Name	Description
0x0	DISABLE	PLL is disabled
0x1	2x	2 x multiplication factor
0x2	3x	3 x multiplication factor
0x3	-	Reserved

```
/* Set the PLL to use XOSCHF as source, and select 3x multiplication factor */
ccp_write_io((uint8_t *) &CLKCTRL.PLLCTRLA,
             CLKCTRL_SOURCE_bm | CLKCTRL_MULFAC_3x_gc);
```

To select the PLL as clock source for TCD0, the PLL setting must be written to the Clock Select (CLKSEL) bit field in the Control A (TCD0.CTRLA) register.

Figure 8-3. TCD0.CTRLA – Enable TCD0 with PLL as source

Bit	7	6	5	4	3	2	1	0
		CLKSEL[1:0]		CNTPRES[1:0]		SYNCPRES[1:0]		ENABLE
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

Bits 6:5 – CLKSEL[1:0] Clock Select
The Clock Select bits select the clock source of the TCD clock.

Value	Name	Description
0x0	OSCHF	Internal High-Frequency Oscillator
0x1	PLL	PLL
0x2	EXTCLK	External clock
0x3	CLK_PER	Peripheral clock

Bits 4:3 – CNTPRES[1:0] Counter Prescaler
The Counter Prescaler bits select the division factor of the TCD counter clock.

Value	Name	Description
0x0	DIV1	Division factor 1
0x1	DIV4	Division factor 4
0x2	DIV32	Division factor 32
0x3	-	Reserved

Bits 2:1 – SYNCPRES[1:0] Synchronization Prescaler
The Synchronization Prescaler bits select the division factor of the TCD clock.

Value	Name	Description
0x0	DIV1	Division factor 1
0x1	DIV2	Division factor 2
0x2	DIV4	Division factor 4
0x3	DIV8	Division factor 8

```
/* Configure the TCD with PLL (48 MHz) as source */
TCD0.CTRLA = TCD_CLKSEL_PLL_gc | TCD_CNTPRES_DIV1_gc | TCD_SYNCPRES_DIV1_gc;

/* Replace with your application configuration */

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

The code for this example is available in the **TCD-with-XOSCHF-and-PLL** folder in these github repositories:

View Code Example on GitHub
Click to browse repository

9. CFD on XOSCHF

To use Clock Failure Detection (CFD) on the External High-Frequency Oscillator (XOSCHF), write the XOSCHF setting to the Clock Failure Detection Source (CFDSRC) bit field and '1' to the Clock Failure Detection Enable (CFDEN) bit in the Main Clock Control C (CLKCTRL.MCLKCTRLC) register.

This register has Configuration Change Protection (CCP), so the `ccp_write_io` function in `cpufunc.h` should be used to ensure correct timing for the unlock of the register.

Figure 9-1. CLKCTRL.MCLKCTRLC – Enable CFD on XOSCHF

Bit	7	6	5	4	3	2	1	0
					CFDSRC[1:0]		CFDTST	CFDEN
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:2 – CFDSRC[1:0] Clock Failure Detection Source
This bit field controls which clock source to monitor, when the Clock Failure Detection Enable (CFDEN) bit is '1'.

Value	Name	Description
0x0	CLKMAIN	Main Clock
0x1	XOSCHF	External High Frequency Oscillator
0x2	XOSC32K	External 32.768 kHz Oscillator
Other	Reserved	Reserved

Note: This bit field is read-only when the CFDEN bit is '1', and both the Clock Failure Detection (CFD) interrupt enable bit and Interrupt Type (INTTYPE) bit in the Main Clock Interrupt Control (CLKCTRL.MCLKINTCTRL) are '1'. This bit will remain read-only until a System Reset occurs.

Bit 0 – CFDEN Clock Failure Detection Enable
This bit controls whether Clock Failure Detection (CFD) is enabled or not.

Value	Description
0	CFD is disabled
1	CFD is enabled

Note: This bit is read-only when this bit is '1', and both the Clock Failure Detection (CFD) interrupt enable bit and Interrupt Type (INTTYPE) bit in the Main Clock Interrupt Control (CLKCTRL.MCLKINTCTRL) are '1'. This bit will remain read-only until a System Reset occurs.

```
/* Enable Clock Failure Detection on XOSCHF */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC,
             CLKCTRL_CFDSRC_XOSCHF_gc | CLKCTRL_CFDEN_bm);
```

To enable the regular CFD interrupt, write '0' to the Interrupt Type (INTTYPE) bit and '1' to the Clock Failure Detection (CFD) bit in the Main Clock Interrupt Control (CLKCTRL.MCLKINTCTRL) register. This register also has CCP.

Figure 9-2. CLKCTRL.MCLKINTCTRL – Enable regular CFD interrupt

Bit	7	6	5	4	3	2	1	0
	INTTYPE							CFD
Access	R/W							R/W
Reset	0							0

Bit 7 – INTTYPE Interrupt Type
This bit controls the type of the Clock Failure Detection (CFD) interrupt.

Value	Name	Description
0	INT	Regular Interrupt
1	NMI	Non-Maskable Interrupt

Note: This bit is read-only when the Clock Failure Detection Enable (CFDEN) bit in the Main Clock Control C (CLKCTRL.MCLKCTRLC) register is '1', and both the Clock Failure Detection (CFD) interrupt enable bit and this bit are '1.' This bit will remain read-only until a System Reset occurs.

Bit 0 – CFD Clock Failure Detection Interrupt Enable
This bit controls whether the Clock Failure Detection (CFD) interrupt is enabled or not.

Value	Description
0	The CFD interrupt is disabled
1	The CFD interrupt is enabled

Note: This bit is read-only when the Clock Failure Detection Enable (CFDEN) bit in the Main Clock Control C (CLKCTRL.MCLKCTRLC) register is '1', and both the Interrupt Type (INTTYPE) bit and this bit are '1.' This bit will remain read-only until a System Reset occurs.

```
    /* Enable regular interrupt for CFD */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_CFD_bm);
```

The CFD vector will be called when the interrupt is triggered. Before returning from the Interrupt Service Routine (ISR), the Clock Failure Detection (CFD) interrupt flag in the Main Clock Interrupt Flags (CLKCTRL.MCLKINTFLAGS) register must be cleared. This is done by writing a '1' to the flag.

Figure 9-3. CLKCTRL.MCLKINTFLAGS – Clear CFD interrupt flag

Bit	7	6	5	4	3	2	1	0
								CFD
Access								R/W
Reset								0

Bit 0 – CFD Clock Failure Detection Interrupt Flag
This flag is cleared by writing a '1' to it.
This flag is set when a clock failure is detected.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Clock Failure Detection (CFD) interrupt flag.

```
ISR(CLKCTRL_CFD_vect)
{
    /* This interrupt will trigger every time the CFD detects XOSCHF has stopped
     * The Main Clock source is OSCHF so the CPU is not affected
     */
    LED0_toggle();

    /* Clear the CFD interrupt flag */
    CLKCTRL.MCLKINTFLAGS = CLKCTRL_CFD_bm;
}
```

The code for this example is available in the **CFD-on-XOSCHF** folder in these github repositories:

10. CFD on Main Clock with NMI

To use Clock Failure Detection (CFD) on the Main Clock, write the CLKMAIN setting to the Clock Failure Detection Source (CFDSRC) bit field and '1' to the Clock Failure Detection Enable (CFDEN) bit in the Main Clock Control C (CLKCTRL.MCLKCTRLC) register.

This register has Configuration Change Protection (CCP), so the `ccp_write_io` function in `cpufunc.h` should be used to ensure correct timing for the unlock of the register.

Figure 10-1. CLKCTRL.MCLKCTRLC – Enable CFD on Main Clock

Bit	7	6	5	4	3	2	1	0
					CFDSRC[1:0]		CFDTST	CFDEN
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:2 – CFDSRC[1:0] Clock Failure Detection Source
This bit field controls which clock source to monitor, when the Clock Failure Detection Enable (CFDEN) bit is '1'.

Value	Name	Description
0x0	CLKMAIN	Main Clock
0x1	XOSCHF	External High Frequency Oscillator
0x2	XOSC32K	External 32.768 kHz Oscillator
Other	Reserved	Reserved

Note: This bit field is read-only when the CFDEN bit is '1', and both the Clock Failure Detection (CFD) interrupt enable bit and Interrupt Type (INTTYPE) bit in the Main Clock Interrupt Control (CLKCTRL.MCLKINTCTRL) are '1'. This bit will remain read-only until a System Reset occurs.

Bit 0 – CFDEN Clock Failure Detection Enable
This bit controls whether Clock Failure Detection (CFD) is enabled or not.

Value	Description
0	CFD is disabled
1	CFD is enabled

Note: This bit is read-only when this bit is '1', and both the Clock Failure Detection (CFD) interrupt enable bit and Interrupt Type (INTTYPE) bit in the Main Clock Interrupt Control (CLKCTRL.MCLKINTCTRL) are '1'. This bit will remain read-only until a System Reset occurs.

```
/* Enable Clock Failure Detection on main clock */
ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC,
             CLKCTRL_CFDSRC_CLKMAIN_gc | CLKCTRL_CFDEN_bm);
```

To enable the CFD interrupt as a Non-Maskable Interrupt (NMI), write '1' to both the Interrupt Type (INTTYPE) bit and the Clock Failure Detection (CFD) bit in the Main Clock Interrupt Control (CLKCTRL.MCLKINTCTRL) register. This register also has CCP.

Figure 10-2. CLKCTRL.MCLKINTCTRL – Enable CFD interrupt as NMI

Bit	7	6	5	4	3	2	1	0
	INTTYPE							CFD
Access	R/W							R/W
Reset	0							0

Bit 7 – INTTYPE Interrupt Type
This bit controls the type of the Clock Failure Detection (CFD) interrupt.

Value	Name	Description
0	INT	Regular Interrupt
1	NMI	Non-Maskable Interrupt

Note: This bit is read-only when the Clock Failure Detection Enable (CFDEN) bit in the Main Clock Control C (CLKCTRL.MCLKCTRLC) register is '1', and both the Clock Failure Detection (CFD) interrupt enable bit and this bit are '1.' This bit will remain read-only until a System Reset occurs.

Bit 0 – CFD Clock Failure Detection Interrupt Enable
This bit controls whether the Clock Failure Detection (CFD) interrupt is enabled or not.

Value	Description
0	The CFD interrupt is disabled
1	The CFD interrupt is enabled

Note: This bit is read-only when the Clock Failure Detection Enable (CFDEN) bit in the Main Clock Control C (CLKCTRL.MCLKCTRLC) register is '1', and both the Interrupt Type (INTTYPE) bit and this bit are '1.' This bit will remain read-only until a System Reset occurs.

```
    /* Enable Non-Maskable Interrupt for CFD */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL,
                 CLKCTRL_INTTYPE_bm | CLKCTRL_CFD_bm);
```

When the CFD interrupt has been configured as an NMI, the NMI interrupt vector will be called instead of the CFD vector when the interrupt is triggered.

When entering the Interrupt Service Routine (ISR) for the NMI, the interrupt source must be found if more than one NMI source is enabled. The CFD interrupt flag in the Main Clock Interrupt Flags (CLKCTRL.MCLKINTFLAGS) register is '1' if the CFD was the trigger.

Figure 10-3. CLKCTRL.MCLKINTFLAGS – Check CFD interrupt flag

Bit	7	6	5	4	3	2	1	0
								CFD
Access								R/W
Reset								0

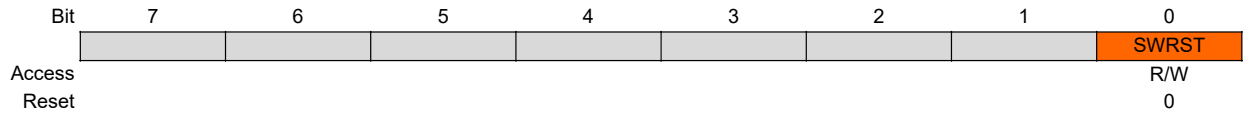
Bit 0 – CFD Clock Failure Detection Interrupt Flag
This flag is cleared by writing a '1' to it.
This flag is set when a clock failure is detected.
Writing a '0' to this bit has no effect.
Writing a '1' to this bit will clear the Clock Failure Detection (CFD) interrupt flag.

```
ISR(NMI_vect)
{
    /* Check which NMI has triggered*/
    if(CLKCTRL.MCLKINTFLAGS & CLKCTRL_CFD_bm)
    {
        /* This interrupt will trigger if the source for the main clock fails
        * and the CFD is able to switch to a different working clock.
```

```
        * In this case that means XOSCHF has failed and is replaced by OSCHF.
        * The main clock is therefore reduced to 4 MHz / 2 = 2 MHz.
        */

        /* Toggle the LED forever */
        while(1)
        {
            LED0_toggle();
            _delay_ms(200); // 200 ms calculated from 16 MHz == 1600 ms
        }

    }
    else
    {
        /* A different NMI has been triggered */
    }

    /* Non-Maskable Interrupt bits can only be cleared by a reset */
}
```

A software reset can be triggered by writing a '1' to the Software Reset (SWRST) bit in the Software Reset Register (RSTCTRL.SWRR). This register also has CCP.

Figure 10-4. RSTCTRL.SWRR – Trigger Software Reset



Bit 0 – SWRST Software Reset
When this bit is written to '1', a Software Reset will occur.
This bit will always read as '0'.

```
    /* Software Reset */
    ccp_write_io((uint8_t *) &RSTCTRL.SWRR, RSTCTRL_SWRST_bm);
```

The code for this example is available in the **CFD-on-main-clock-with-NMI** folder in these github repositories:

View Code Example on GitHub
Click to browse repository

11. Revision History

Revision	Date	Description
A	9/2020	Initial document release

12. Appendix

Here are the code examples for each case used in the technical brief.

Example 12-1. XOSCHF with External Crystal

```
#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();

    /* Turn on LED0 to show successful clock init */
    LED0_init();

    /* Replace with your application code */
    while(1)
    {
    }
}


void CLOCK_XOSCHF_crystal_init(void)
{
    /* Enable crystal oscillator with frequency range 16 MHz and 4K cycles
start-up time */
    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);

    /* Confirm crystal oscillator start-up */
    while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* Clear Main Clock Prescaler */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, 0x00);

    /* Set the main clock to use XOSCHF as source, and enable the CLKOUT pin */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc |
CLKCTRL_CLKOUT_bm);

    /* Wait for system oscillator changing to complete */
    while(CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }

    /* Clear RUNSTDBY for power save during sleep */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA &
~CLKCTRL_RUNSTDBY_bm);

    /* Change complete and the main clock is 16 MHz */
}
```

Example 12-2. XOSCHF with External Clock

```
#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();

    /* Turn on LED0 to show successful clock init */
    LED0_init();

    /* Replace with your application code */
    while(1)
    {
    }
}


void CLOCK_XOSCHF_clock_init(void)
{
    /* Enable external (32 MHz) clock input */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL_SELHF_EXTCLOCK_gc |
CLKCTRL_ENABLE_bm);

    /* Set Main Clock Prescaler */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, CLKCTRL_PDIV_2X_gc |
CLKCTRL_PEN_bm);

    /* Set the main clock to use XOSCHF as source, and enable the CLKOUT pin */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc |
CLKCTRL_CLKOUT_bm);

    /* Wait for system oscillator changing to complete */
    while(CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }

    /* Change complete and the main clock is 32 MHz / 2 = 16 MHz */
}
```

Example 12-3. RTC with XOSCHF

```
#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();

    /* Enable global interrupts */
    sei();

    /* Replace with your application code */
    while(1)
    {
    }
}


void CLOCK_XOSCHF_crystal_init(void)
{
    /* Enable crystal oscillator with frequency range 8 MHz and 1K cycles
start-up time */
    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);

    /* Confirm crystal oscillator start-up */
    while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* Clear RUNSTDBY for power save during sleep */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA &
~CLKCTRL_RUNSTDBY_bm);
}


void TIMER_RTC_init(void)
{
    while(RTC.STATUS > 0)
    {
        /* Wait for RTC registers to synchronize */
    }

    /* Configure RTC to use XOSCHF as source */
    RTC.CLKSEL = RTC_CLKSEL_EXTCLK_gc;

    /* Replace with your application configuration */
    RTC.PER = 0xffff;
    RTC.INTCTRL = RTC_OVF_bm;
    RTC.CTRLA = RTC_PRESCALER_DIV32_gc | RTC_RTCEN_bm;
}


ISR(RTC_CNT_vect)
{
    /* This interrupt will trigger every time the RTC overflows */
    LED0_toggle();

    /* Clear the RTC overflow interrupt flag */
    RTC.INTFLAGS = RTC_OVF_bm;
}
```

Example 12-4. TCD with XOSCHF

```
#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();

    /* Replace with your application code */
    while(1)
    {
    }
}


void CLOCK_XOSCHF_crystal_init(void)
{
    /* Enable crystal oscillator with frequency range 16 MHz and 4K cycles
start-up time */
    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);

    /* Confirm crystal oscillator start-up */
    while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* Clear RUNSTDBY for power save during sleep */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA &
~CLKCTRL_RUNSTDBY_bm);
}


void TIMER_TCD0_init(void)
{
    /* Configure the TCD with XOSCHF (16 MHz) as source */
    TCD0.CTRLA = TCD_CLKSEL_EXTCLK_gc | TCD_CNTPRES_DIV1_gc |
TCD_SYNCPRES_DIV1_gc;

    /* Replace with your application configuration */

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

Example 12-5. TCD with XOSCHF and PLL

```
#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();

    /* Replace with your application code */
    while(1)
    {
    }
}


void CLOCK_XOSCHF_crystal_PLL_init(void)
{
    /* Enable crystal oscillator with frequency range 16 MHz and 4K cycles
start-up time */
    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);
```

```
    /* Confirm crystal oscillator start-up */
    while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* Set the PLL to use XOSCHF as source, and select 3x multiplication
factor */
    ccp_write_io((uint8_t *) &CLKCTRL.PLLCTRLA, CLKCTRL_SOURCE_bm |
CLKCTRL_MULFAC_3x_gc);

    /* Clear RUNSTDBY for power save during sleep */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA &
~CLKCTRL_RUNSTDBY_bm);
}


void TIMER_TCD0_init(void)
{
    /* Configure the TCD with PLL (48 MHz) as source */
    TCD0.CTRLA = TCD_CLKSEL_PLL_gc | TCD_CNTPRES_DIV1_gc |
TCD_SYNCPRES_DIV1_gc;

    /* Replace with your application configuration */

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

Example 12-6. CFD on XOSCHF

```
#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();

    /* Enable global interrupts */
    sei();

    /* Replace with your application code */
    while(1)
    {
    }
}


void CLOCK_XOSCHF_crystal_init(void)
{
    /* Enable crystal oscillator
     * with frequency range 16 MHz and 4K cycles start-up time
     */
```

```
    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);

    /* Confirm crystal oscillator start-up */
    while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }
}


void CLOCK_CFD_XOSCHF_init(void)
{
    /* Enable Clock Failure Detection on XOSCHF */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_XOSCHF_gc |
CLKCTRL_CFDEN_bm);

    /* Enable regular interrupt for CFD */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_CFD_bm);
}


ISR(CLKCTRL_CFD_vect)
{
    /* This interrupt will trigger every time the CFD detects XOSCHF has
stopped
     * The Main Clock source is OSCHF so the CPU is not affected
     */
    LED0_toggle();

    /* Clear the CFD interrupt flag */
    CLKCTRL.MCLKINTFLAGS = CLKCTRL_CFD_bm;
}
```

Example 12-7. CFD on Main Clock with NMI

```
#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();

    /* Enable global interrupts */
    sei();

    /* Replace with your application code */
    while(1)
    {
        LED0_toggle();
        _delay_ms(200);
```

```
    }
}


void CLOCK_XOSCHF_crystal_init(void)
{
    /* Enable crystal oscillator
     * with frequency range 16 MHz and 4K cycles start-up time
     */
    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);

    /* Confirm crystal oscillator start-up */
    while(!(CLKCTRL.MCLKSTATUS & CLKCTRL_EXTS_bm))
    {
        ;
    }

    /* Clear Main Clock Prescaler */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLB, 0x00);

    /* Set the main clock to use XOSCHF as source, and enable the CLKOUT pin */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLA, CLKCTRL_CLKSEL_EXTCLK_gc
                 | CLKCTRL_CLKOUT_bm);

    /* Wait for system oscillator changing to complete */
    while(CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
    {
        ;
    }

    /* Clear RUNSTDBY for power save when not in use */
    ccp_write_io((uint8_t *) &CLKCTRL.XOSCHFCTRLA, CLKCTRL.XOSCHFCTRLA &
~CLKCTRL_RUNSTDBY_bm);

    /* Change complete and the main clock is 16 MHz */
}


void CLOCK_CFD_CLKMAIN_init(void)
{
    /* Enable Clock Failure Detection on main clock */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKCTRLC, CLKCTRL_CFDSRC_CLKMAIN_gc
                 | CLKCTRL_CFDEN_bm);

    /* Enable interrupt for CFD */
    ccp_write_io((uint8_t *) &CLKCTRL.MCLKINTCTRL, CLKCTRL_INTTYPE_bm
                 | CLKCTRL_CFD_bm);
}


ISR(NMI_vect)
{
    /* Check which NMI has triggered*/
    if(CLKCTRL.MCLKINTFLAGS & CLKCTRL_CFD_bm)
    {
        /* This interrupt will trigger if the source for the main clock fails
        * and the CFD is able to switch to a different working clock.
        * In this case that means XOSCHF has failed and is replaced by OSCHF.
        * The main clock is therefore reduced to 4 MHz / 2 = 2 MHz.
        */

        /* Toggle the LED forever */
        while(1)
        {
            LED0_toggle();
            _delay_ms(200); // 200 ms calculated from 16 MHz == 1600 ms
        }

    }
    else
    {
        /* A different NMI has been triggered */
```

```
    }

    /* Non-Maskable Interrupt bits can only be cleared by a reset */
}
```

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

© 2020 Microchip Technology Inc.

Legal Notice

Information contained in this publication is provided for the sole purpose of designing with and using Microchip products. Information regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL LOSS, DAMAGE, COST OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

© 2020, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

ISBN: 978-1-5224-6763-2

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820