Clock configuration setup in XMC7000 MCU family

About this document

Scope and purpose

This application note describes how to set up the various clock sources in XMC7000 family MCUs and provides examples including configuring PLL/FLL and calibrating the ILO.

Intended audience

This document is intended for anyone using XMC7000 family MCUs.

Associated part family

XMC7000 family of XMC™ industrial microcontrollers.

Table of contents

Table of contents

1 Introduction

XMC7000 family MCUs are targeted at industrial systems. These products enable a secure computing platform and incorporate Infineon low-power flash memory along with multiple high-performance analog and digital functions.

XMC7000 family clock system supports high-speed and low-speed clocks using both internal and external clock sources. One of the typical use cases for clock input is for internal real-time clock (RTC). XMC7000 family supports phase-locked loop (PLL) and frequency-locked loop (FLL) to generate clocks that operate the internal circuit at a high speed. XMC7000 family also supports a function to monitor the clock operation and to measure the clock difference of each clock with reference to a known clock.

To understand more on the functionality described and terminology used in this application note, see the "Clocking System" chapter in the XMC7000 MCU family architecture reference manual.

2 Clock system for XMC7000 family MCUs

2.1 Overview

The clock system in this series of MCUs is divided into two blocks. One selects the clock resources (such as external oscillator and internal oscillator) and multiplies the clock (using FLL and PLL). The other distributes and divides clocks to the CPU cores and other peripheral functions. However, there are some exceptions such as RTC that can connect directly to a clock resource.

Figure 1 shows the overview of the clock system.

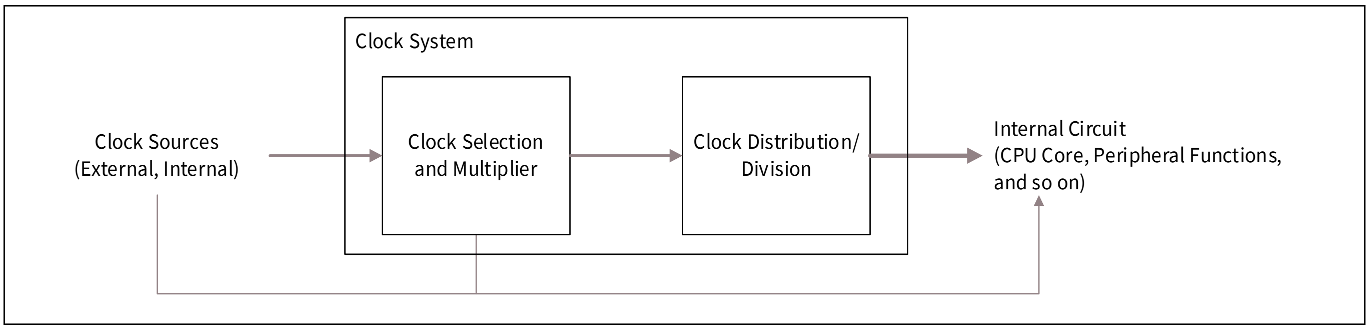

Clock System
Clock Sources (External, Internal)
Clock Selection and Multiplier
Clock Distribution/ Division
Internal Circuit (CPU Core, Peripheral Functions, and so on)

Figure 1 Clock system structure

2.2 Clock resources

The MCU supports two types of resource inputs – internal and external. Each of these internally supports three types of clocks, respectively.

- Internal clock sources (all clocks are enabled by default):
  - Internal main oscillator (IMO): Built-in clock with a frequency of 8 MHz (Typical)
  - Internal low-speed oscillator 0 (ILO0): Built-in clock with a frequency of 32.768 kHz (Typical)
  - Internal low-speed oscillator 1 (ILO1): Has the same function as ILO0, but ILO1 can monitor the clock of ILO0
- External clock sources (all clocks are disabled by default):
  - External crystal oscillator (ECO): Uses an external oscillator whose input frequency range is between 8 and 33.34 MHz
  - Watch crystal oscillator (WCO): Uses an external oscillator whose frequency is stable 32.768 kHz, mainly used by the RTC module
  - External clock (EXT_CLK): The EXT_CLK is a 0.25 MHz to 80 MHz range clock that can be sourced from a signal on a designed I/O pin. This clock can be used as the source clock for either PLL or FLL or can be used directly by the high-frequency clocks.

For more details on functions such as IMO, PLL, and so on, and numerical values such as frequency, see the XMC7000 MCU family architecture reference manual and the datasheet.

2.3 Clock system functions

Figure 2 shows the details of the clock selection and multiplier block. This block generates root frequency clocks CLK_HF0 to CLK_HF7 from the clock resources. This block has the capability to select one of the supported clock resources – FLL and PLL to generate the required high-speed clock. This MCU supports two types of PLLs: PLL without spread spectrum clock generation (SSCG) and fractional operation (PLL200#x), and PLL with SSCG and fractional operation (PLL400#x).

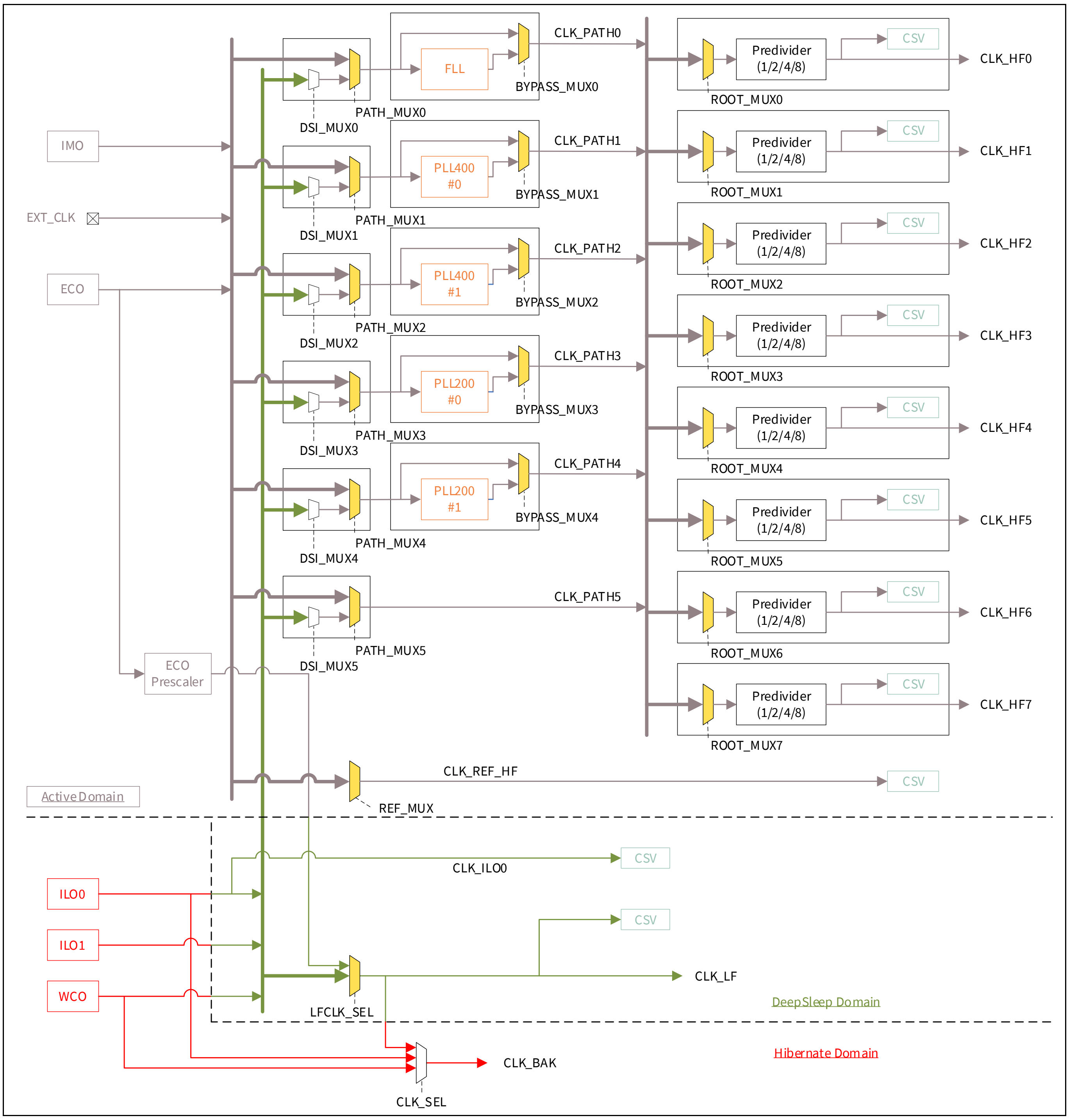


Figure 2 Clock system block diagram

Domain/block	Description
Active domain	Region of operation only in Active power mode
DeepSleep domain	Region of operation in Active and DeepSleep power modes
Hibernate domain	Region of operation in the All Power mode
ECO prescaler	Divides the ECO and creates a clock that can be used with the CLK_LF clock. The division function has a 10-bit integer divider and an 8-bit fractional divider
DSI_MUX	Selects a clock from ILO0, ILO1, and WCO
PATH_MUX	Selects a clock from IMO, ECO, EXT_CLK, and DSI_MUX output
CLK_PATH	CLK_PATHs 0 through 5 are used as the input sources for high-frequency clocks
CLK_HF	CLK_HFs 0 through 7 are recognized as high-frequency clocks
FLL	Generates the high-frequency clock
PLL	Generates the high-frequency clock. There are two kinds of PLL: – PLL200: Without SSCG and fractional operation – PLL400: With SSCG and fractional operation
BYPASS_MUX	Selects the clock to be output to CLK_PATH. In case of FLL, the clock that can be selected is either the FLL output or clock input to FLL.
ROOT_MUX	Selects the clock source of CLK_HFx. The clocks that can be selected are CLK_PATHs 0 through 5.
Predivider	Predivider (divided by 1, 2, 4, or 8) is available to divide the selected CLK_PATH.
REF_MUX	Selects the CLK_REF_HF clock source
CLK_REF_HF	Used to monitor CSV of CLK_HF
LFCLK_SEL	Selects the CLK_LF clock source
CLK_LF	MCWDT source clock
CLK_SEL	Selects the clock to be input to the RTC
CLK_BAK	Mainly input to the RTC
CSV	Clock supervision to monitor the clock operation

Note: *The clock configuration is done one time via the ModusToolbox™ Device Configurator; the settings code is auto generated.*

Figure 3 shows the distribution of CLK_HF0.

CLK_HF0 is the root clock for the CPU subsystem (CPUSS) and peripheral clock dividers. For more details on Figure 3, see the XMC7000 MCU family architecture reference manual and the datasheet.

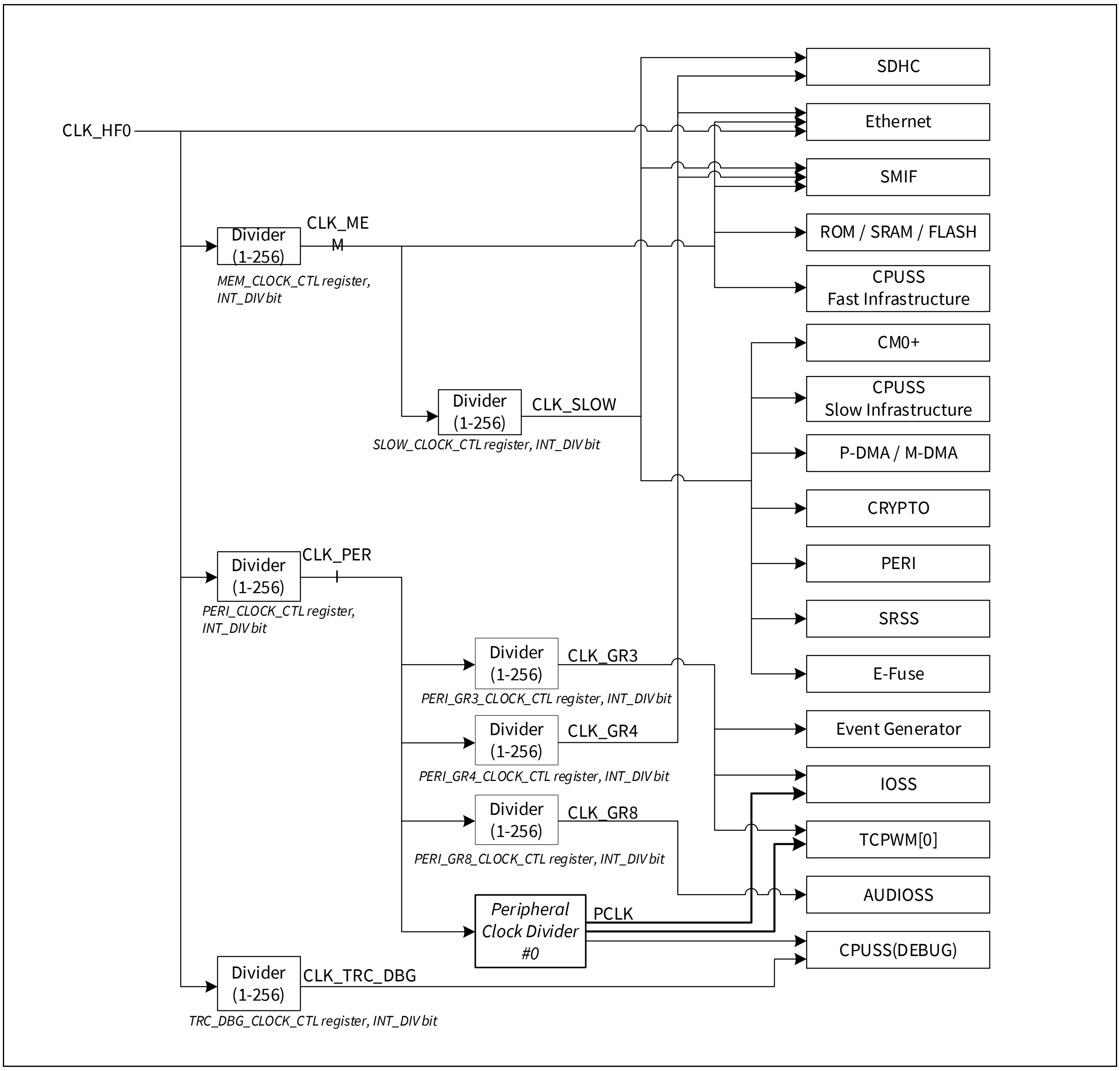

CLK_HF0
Divider (1-256)
CLK_MEM
MEM_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_SLOW
SLOW_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_PER
PERI_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_GR3
PERI_GR3_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_GR4
PERI_GR4_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_GR8
PERI_GR8_CLOCK_CTL register, INT_DIV bit
Peripheral Clock Divider #0
PCLK
Divider (1-256)
CLK_TRC_DBG
TRC_DBG_CLOCK_CTL register, INT_DIV bit
SDHC
Ethernet
SMIF
ROM / SRAM / FLASH
CPUSS Fast Infrastructure
CM0+
CPUSS Slow Infrastructure
P-DMA / M-DMA
CRYPTO
PERI
SRSS
E-Fuse
Event Generator
IOSS
TCPWM[0]
AUDIOSS
CPUSS(DEBUG)

Figure 3 CLK_HF0 block diagram

Clock input	Description
CLK_MEM	Clock input to the CPUSS of Fast Infrastructure, Ethernet, and Serial Memory Interface (SMIF)
CLK_PERI	Clock source for CLK_GR and peripheral clock divider
CLK_SLOW	Clock input to the CPUSS of Cortex®-M0+ and Slow Infrastructure, SDHC, and SMIF
CLK_GR	Clock input to peripheral functions. CLK_GR is grouped by the clock gater. CLK_GR has six groups.
PCLK	Peripheral clock used in peripheral functions. PCLK can be configured for each channel of IPs independently; it can select one divider to generate PCLK.
CLK_TRC_DBG	Clock input to CPUSS(DEBUG)
Divider	Divider has a function to divide each clock. It can be configured from 1 division to 256 divisions.

Figure 4 shows details of the peripheral clock divider #0.

This MCU needs a clock to each peripheral unit (the serial communication block (SCB), the Timer, Counter, and PWM (TCPWM), etc.) and its respective channel. The clocks are controlled by their respective dividers.

This peripheral clock divider #0 has many peripheral clock dividers to generate the peripheral clock (PCLK). See the datasheet for the number of dividers. The output of any of these dividers can be routed to any peripheral. Note that the dividers already in use cannot be used for other peripherals or channels.

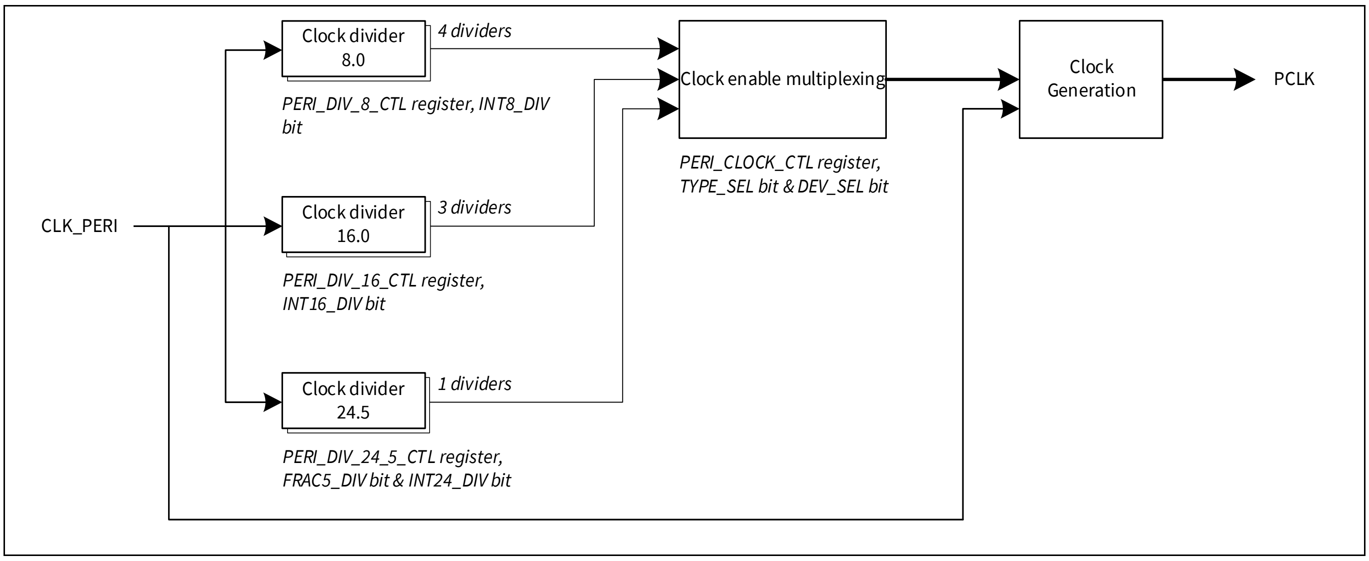

Clock divider 8.0
4 dividers
PERI_DIV_8_CTL register, INT8_DIV bit
Clock enable multiplexing
Clock Generation
PCLK
PERI_CLOCK_CTL register, TYPE_SEL bit & DEV_SEL bit
CLK_PERI
Clock divider 16.0
3 dividers
PERI_DIV_16_CTL register, INT16_DIV bit
Clock divider 24.5
1 dividers
PERI_DIV_24_5_CTL register, FRAC5_DIV bit & INT24_DIV bit

Figure 4 Peripheral clock divider #0 block diagram

Block	Description
Clock divider8.0	Divides a clock by 8
Clock divider16.0	Divides a clock by 16
Clock divider24.5	Divides a clock by 24.5
Clock Enable Multiplexing	Enables the signal output from clock divider
Clock Generator	Divides CLK_PERI based on clock divider

Figure 5 shows the distribution of CLK_HF1.

CLK_HF1 is a clock source for CLK_FAST_0 and CLK_FAST_1. The clock distribution of CLK_HF1 is shown in Figure 5. CLK_FAST_0 and CLK_FAST_1 are the input sources for the CM7_0 and CM7_1 respectively.

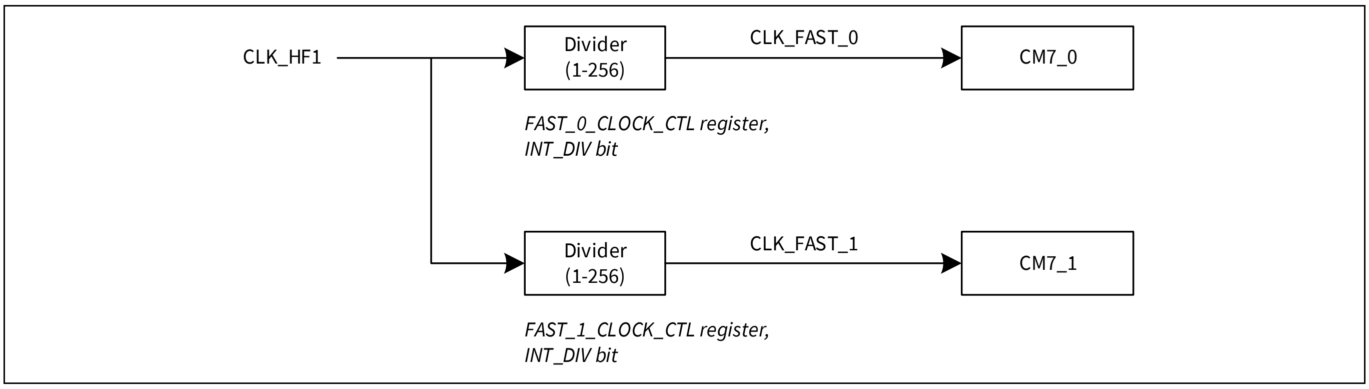

CLK_HF1
Divider (1-256)
CLK_FAST_0
CM7_0
FAST_0_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_FAST_1
CM7_1
FAST_1_CLOCK_CTL register, INT_DIV bit

Figure 5 **CLK_HF1 block diagram**

Figure 6 shows the distribution of CLK_HF2.

CLK_HF2 is a clock source for CLK_GR and PCLK. The clock distribution of CLK_HF2 is shown in Figure 6.

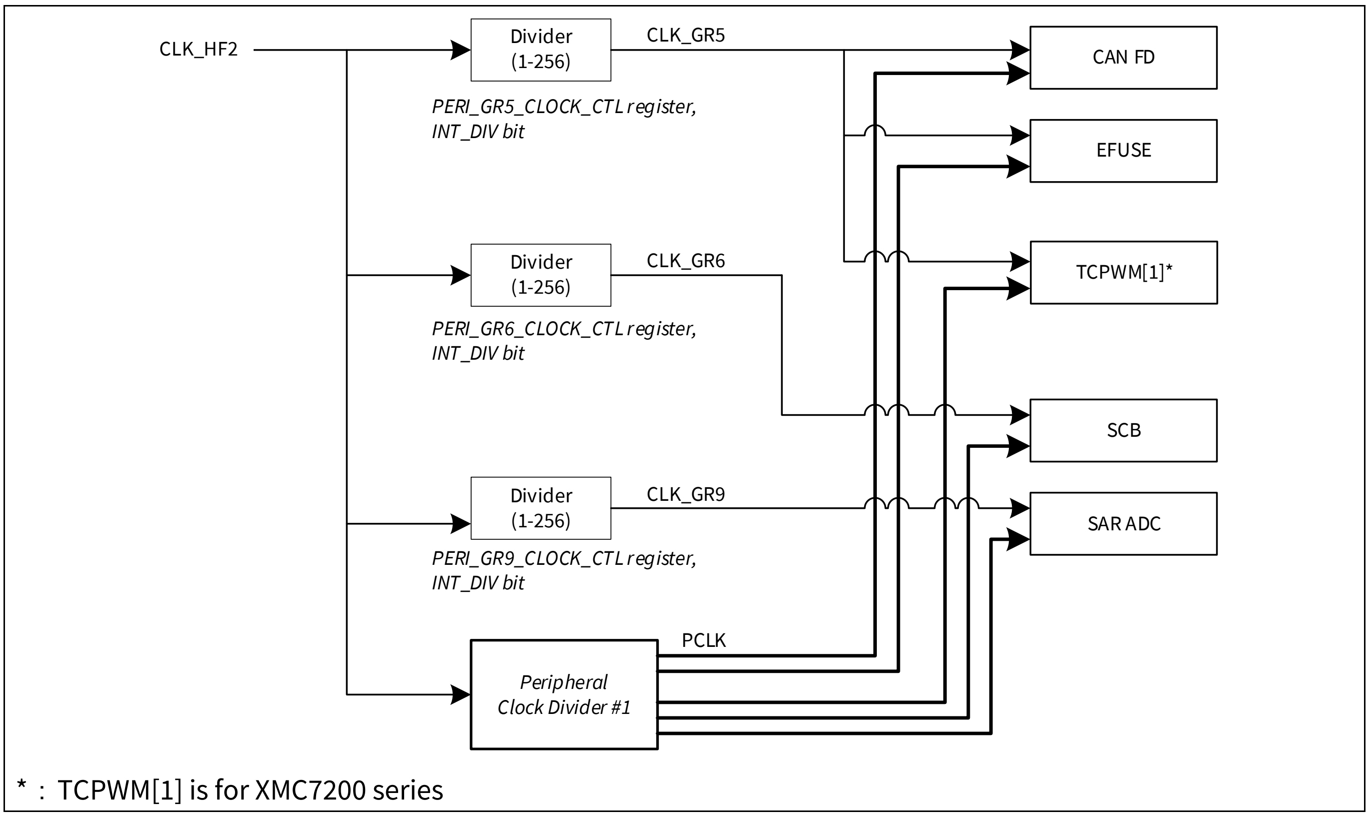

CLK_HF2
Divider (1-256)
CLK_GR5
PERI_GR5_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_GR6
PERI_GR6_CLOCK_CTL register, INT_DIV bit
Divider (1-256)
CLK_GR9
PERI_GR9_CLOCK_CTL register, INT_DIV bit
PCLK
Peripheral Clock Divider #1
CAN FD
EFUSE
TCPWM[1]*
SCB
SAR ADC
* : TCPWM[1] is for XMC7200 series

Figure 6 **CLK_HF2 block diagram**

Figure 7 shows details of peripheral clock divider #1.

Peripheral clock divider #1 has many peripheral clock dividers to generate PCLK. See the datasheet for the number of each divider. The output of any of these dividers can be routed to any peripheral. Note that dividers already in use cannot be used for other peripherals or channels.

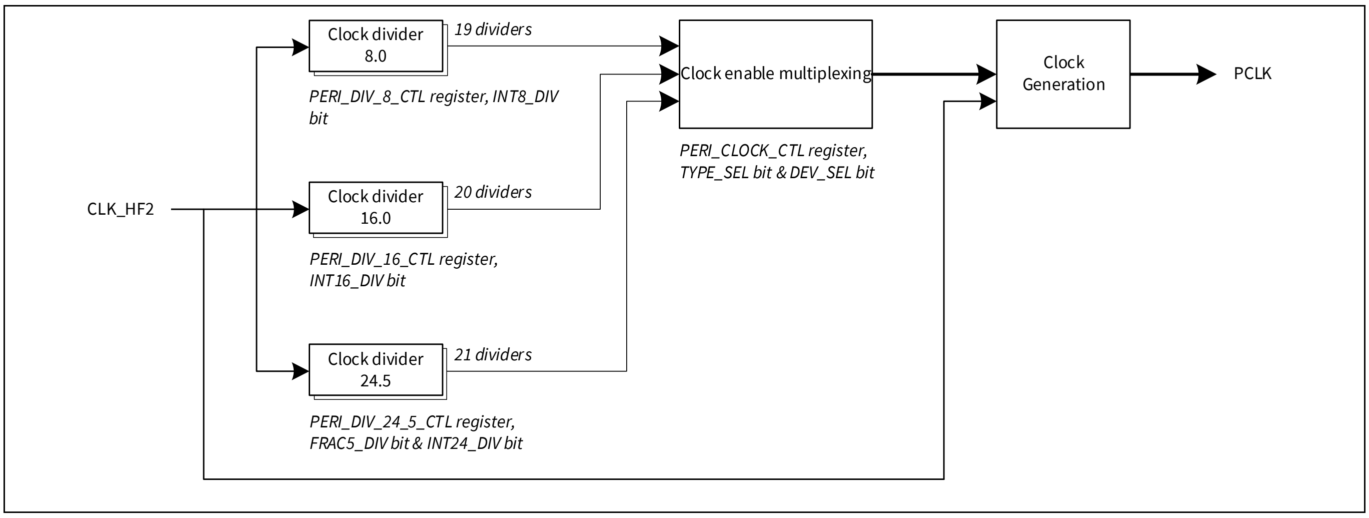

CLK_HF2
Clock divider 8.0
19 dividers
PERI_DIV_8_CTL register, INT8_DIV bit
Clock divider 16.0
20 dividers
PERI_DIV_16_CTL register, INT16_DIV bit
Clock divider 24.5
21 dividers
PERI_DIV_24_5_CTL register, FRAC5_DIV bit & INT24_DIV bit
Clock enable multiplexing
PERI_CLOCK_CTL register, TYPE_SEL bit & DEV_SEL bit
Clock Generation
PCLK

Figure 7 Peripheral clock divider #1 block diagram

Block	Description
Clock divider8.0	Divides a clock by 8
Clock divider16.0	Divides a clock by 16
Clock divider24.5	Divides a clock by 24.5
Clock Enable Multiplexing	Enables the signal output from the clock divider
Clock Generator	Divides CLK_PERI based on the clock divider

Figure 8 shows the distribution of CLK_HF3, CLK_HF4, CLK_HF5, and CLK_HF6. For more details on these functions in Figure 8, see the XMC7000 MCU family architecture reference manual.

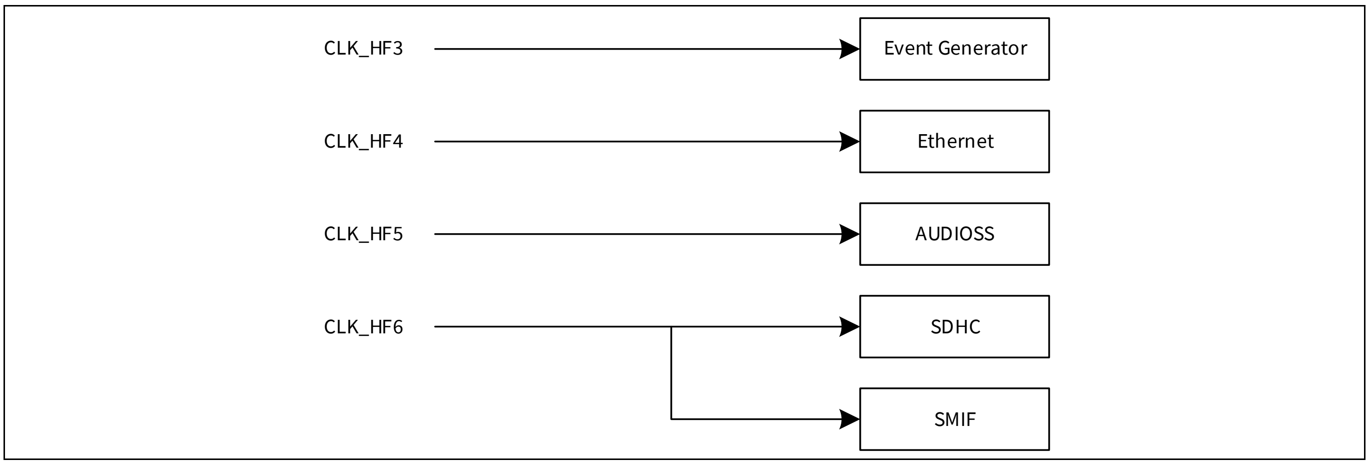


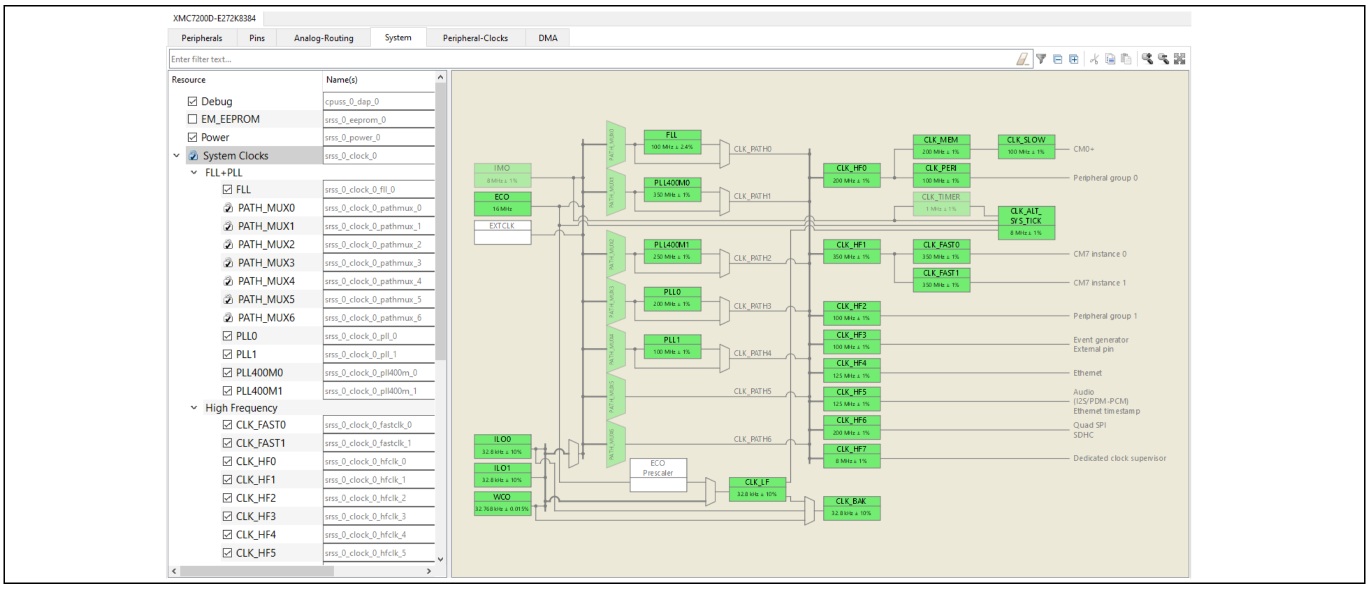
Figure 8 CLK_HF3, CLK_HF4, CLK_HF5, and CLK_HF6 block diagram

CLK_HF7 is dedicated to CSV. See the XMC7000 MCU family architecture reference manual for CSV description.

2.4 Basic clock system settings

This section displays the configuration of the clock system in the Device Configurator of ModusToolbox™ software.

You can configure the clock system according to your system in the Device Configurator as follows:



Resource	Name(s)
☑ Debug	cpuss_0_dap_0
☐ EM_EEPROM	srss_0_eeprom_0
☑ Power	srss_0_power_0
System Clocks	srss_0_clock_0
FLL+PLL	
☑ FLL	srss_0_clock_0_fll_0
PATH_MUX0	srss_0_clock_0_pathmux_0
PATH_MUX1	srss_0_clock_0_pathmux_1
PATH_MUX2	srss_0_clock_0_pathmux_2
PATH_MUX3	srss_0_clock_0_pathmux_3
PATH_MUX4	srss_0_clock_0_pathmux_4
PATH_MUX5	srss_0_clock_0_pathmux_5
PATH_MUX6	srss_0_clock_0_pathmux_6
☑ PLL0	srss_0_clock_0_pll_0
☑ PLL1	srss_0_clock_0_pll_1
☑ PLL400M0	srss_0_clock_0_pll400m_0
☑ PLL400M1	srss_0_clock_0_pll400m_1
High Frequency	
☑ CLK_FAST0	srss_0_clock_0_fastclk_0
☑ CLK_FAST1	srss_0_clock_0_fastclk_1
☑ CLK_HF0	srss_0_clock_0_hfclk_0
☑ CLK_HF1	srss_0_clock_0_hfclk_1
☑ CLK_HF2	srss_0_clock_0_hfclk_2
☑ CLK_HF3	srss_0_clock_0_hfclk_3
☑ CLK_HF4	srss_0_clock_0_hfclk_4
☑ CLK_HF5	srss_0_clock_0_hfclk_5

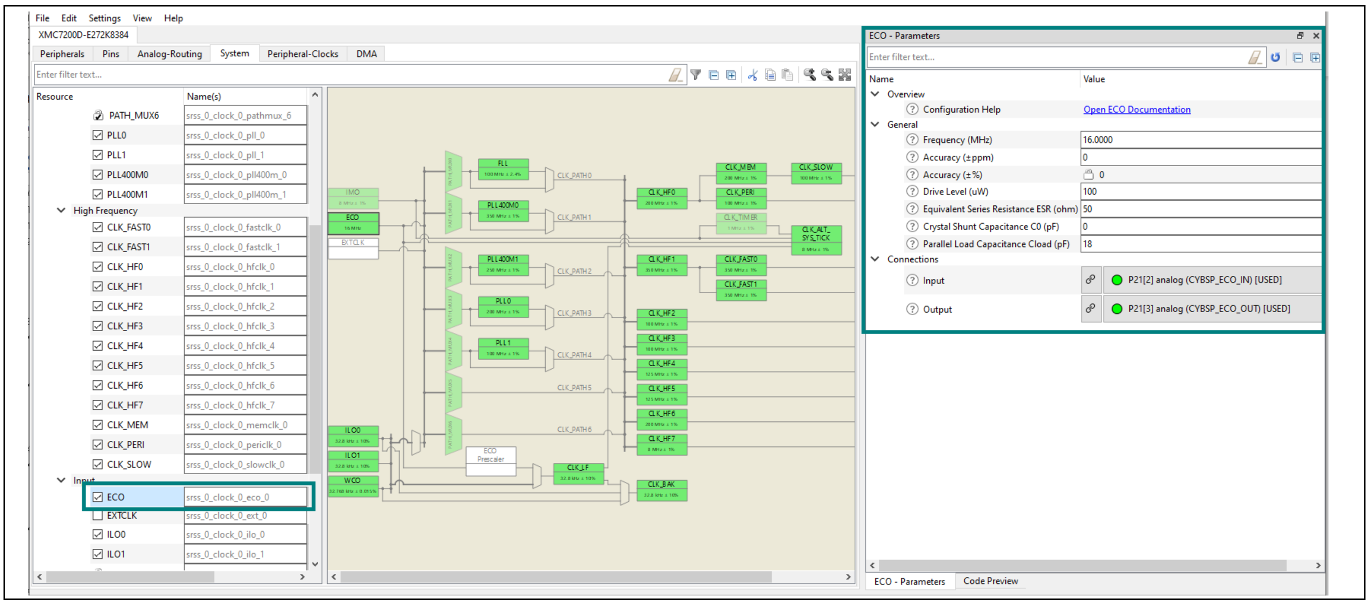
Figure 9 Basic clock system setting

3 Configuring the clock resources

3.1 ECO configuration

The ECO is disabled by default and needs to be enabled for usage. Also, trimming is necessary to use the ECO. This device can set the trimming parameters that control the oscillator according to the crystal unit and ceramic resonator. The method to determine the parameters differs between the crystal unit and ceramic resonator.

You can configure these parameters in the **System** tab of the ModusToolbox™ Device Configurator. ModusToolbox™ software will automatically generate the corresponding code.



Resource	Name(s)
PATH_MUX6	srss_0_clock_0_pathmux_6
☑ PLL0	srss_0_clock_0_pll_0
☑ PLL1	srss_0_clock_0_pll_1
☑ PLL400M0	srss_0_clock_0_pll400m_0
☑ PLL400M1	srss_0_clock_0_pll400m_1
High Frequency	
☑ CLK_FAST0	srss_0_clock_0_fastclk_0
☑ CLK_FAST1	srss_0_clock_0_fastclk_1
☑ CLK_HF0	srss_0_clock_0_hfclk_0
☑ CLK_HF1	srss_0_clock_0_hfclk_1
☑ CLK_HF2	srss_0_clock_0_hfclk_2
☑ CLK_HF3	srss_0_clock_0_hfclk_3
☑ CLK_HF4	srss_0_clock_0_hfclk_4
☑ CLK_HF5	srss_0_clock_0_hfclk_5
☑ CLK_HF6	srss_0_clock_0_hfclk_6
☑ CLK_HF7	srss_0_clock_0_hfclk_7
☑ CLK_MEM	srss_0_clock_0_memclk_0
☑ CLK_PERI	srss_0_clock_0_periclk_0
☑ CLK_SLOW	srss_0_clock_0_slowclk_0
Input	
☑ ECO	srss_0_clock_0_eco_0
☐ EXTCLK	srss_0_clock_0_ext_0
☑ ILO0	srss_0_clock_0_ilo_0
☑ ILO1	srss_0_clock_0_ilo_1

ECO - Parameters

Name	Value
Overview	
Configuration Help	Open ECO Documentation
General	
Frequency (MHz)	16.0000
Accuracy (±ppm)	0
Accuracy (±%)	0
Drive Level (uW)	100
Equivalent Series Resistance ESR (ohm)	50
Crystal Shunt Capacitance C0 (pF)	0
Parallel Load Capacitance Cload (pF)	18
Connections	
Input	P21[2] analog (CYBSP_ECO_IN) [USED]
Output	P21[3] analog (CYBSP_ECO_OUT) [USED]

Figure 10 ECO configuration

3.1.1 Use case

Oscillator to use	**Crystal unit**
Fundamental frequency	16 MHz
Maximum drive level	300.0 µW
Equivalent series resistance	150.0 ohm
Shunt capacitance	0.530 pF
Parallel load capacitance	8.000 pF
Crystal unit vendor's recommended value of negative resistance	1500 ohm
Automatic gain control	OFF

Note: These values are decided in consultation with the crystal unit vendor.

You can configure these parameters in the **System** tab of the Device Configurator; based on these, ModusToolbox™ software automatically generates the corresponding code.

Configuring the clock resources

Double-click the **Device Configurator on the left Quick Panel.**

(you can also double-click *design.modus* to open the Device Configurator in your project,*\bsps\TARGET_APP_KIT_XMC72_EVK\config\design.modus*)

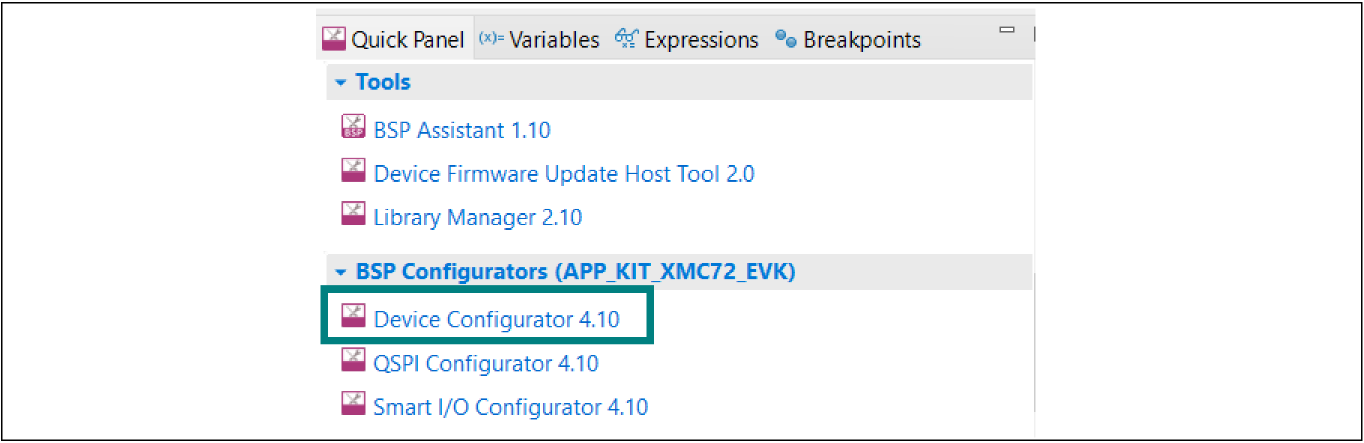
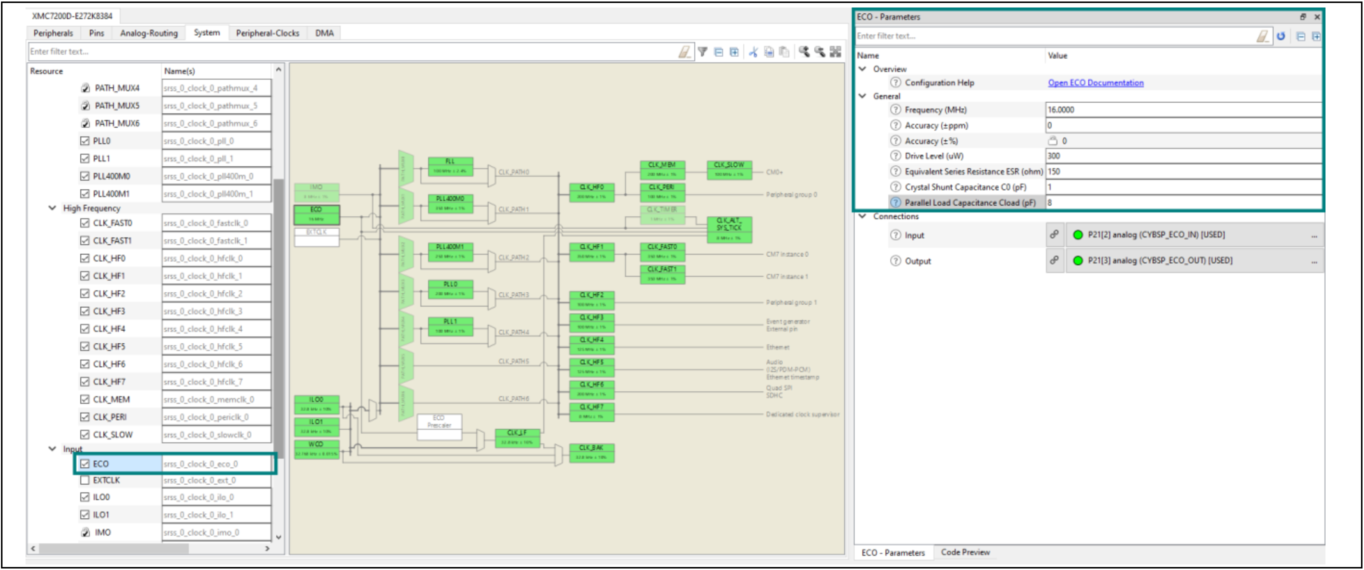


Figure 11 **Quick Panel**



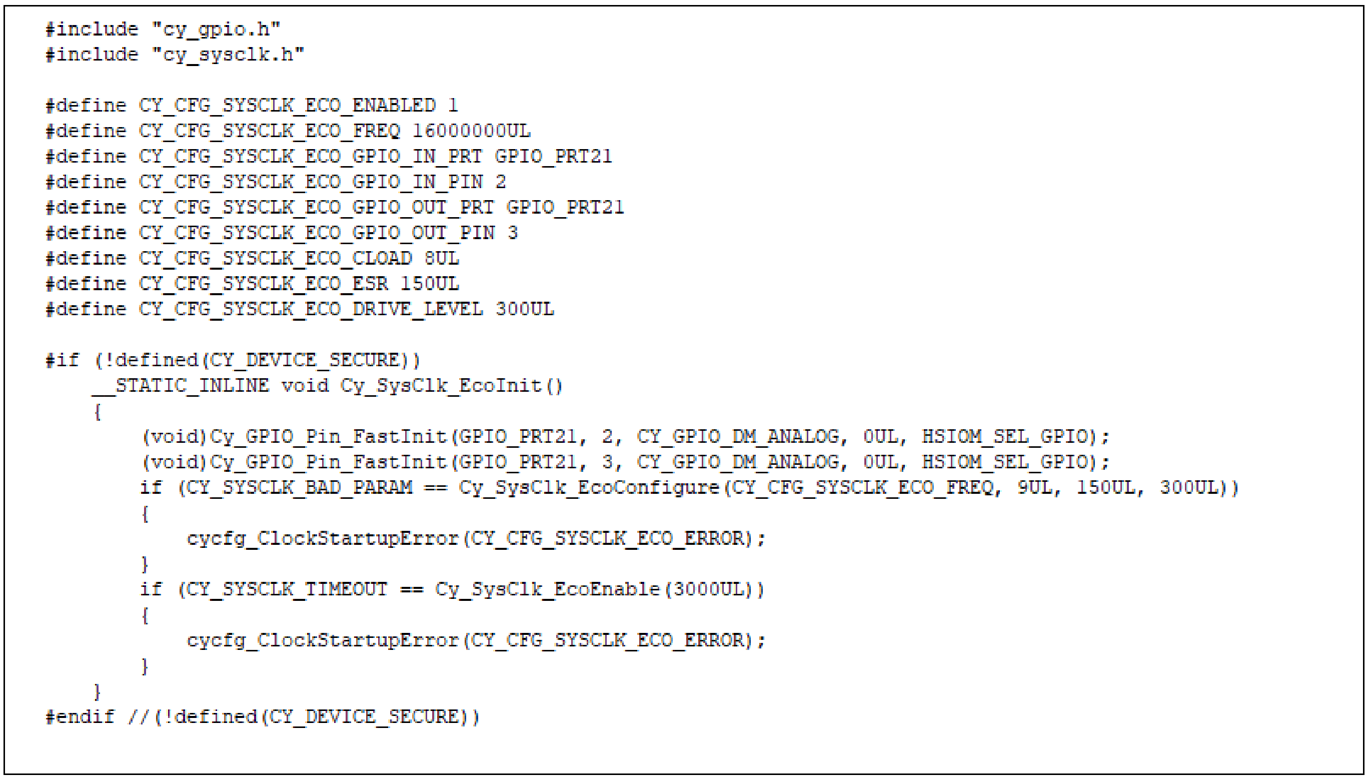
Resource	Name(s)
PATH_MUX4	srss_0_clock_0_pathmux_4
PATH_MUX5	srss_0_clock_0_pathmux_5
PATH_MUX6	srss_0_clock_0_pathmux_6
☑ PLL0	srss_0_clock_0_pll_0
☑ PLL1	srss_0_clock_0_pll_1
☑ PLL400M0	srss_0_clock_0_pll400m_0
☑ PLL400M1	srss_0_clock_0_pll400m_1
High Frequency	
☑ CLK_FAST0	srss_0_clock_0_fastclk_0
☑ CLK_FAST1	srss_0_clock_0_fastclk_1
☑ CLK_HF0	srss_0_clock_0_hfclk_0
☑ CLK_HF1	srss_0_clock_0_hfclk_1
☑ CLK_HF2	srss_0_clock_0_hfclk_2
☑ CLK_HF3	srss_0_clock_0_hfclk_3
☑ CLK_HF4	srss_0_clock_0_hfclk_4
☑ CLK_HF5	srss_0_clock_0_hfclk_5
☑ CLK_HF6	srss_0_clock_0_hfclk_6
☑ CLK_HF7	srss_0_clock_0_hfclk_7
☑ CLK_MEM	srss_0_clock_0_memclk_0
☑ CLK_PERI	srss_0_clock_0_periclk_0
☑ CLK_SLOW	srss_0_clock_0_slowclk_0
Input	
☑ ECO	srss_0_clock_0_eco_0
☐ EXTCLK	srss_0_clock_0_ext_0
☑ ILO0	srss_0_clock_0_ilo_0
☑ ILO1	srss_0_clock_0_ilo_1
IMO	srss_0_clock_0_imo_0

Name	Value
Overview	
Configuration Help	Open ECO Documentation
General	
Frequency (MHz)	16.0000
Accuracy (±ppm)	0
Accuracy (±%)	0
Drive Level (uW)	300
Equivalent Series Resistance ESR (ohm)	150
Crystal Shunt Capacitance C0 (pF)	1
Parallel Load Capacitance Cload (pF)	8
Connections	
Input	P21[2] analog (CYBSP_ECO_IN) [USED]
Output	P21[3] analog (CYBSP_ECO_OUT) [USED]

Figure 12 **ECO use case**

3.1.2 Code preview

This code is auto generated after configuration in the Device Configurator.



```
#include "cy_gpio.h"
#include "cy_sysclk.h"

#define CY_CFG_SYSCLK_ECO_ENABLED 1
#define CY_CFG_SYSCLK_ECO_FREQ 16000000UL
#define CY_CFG_SYSCLK_ECO_GPIO_IN_PRT GPIO_PRT21
#define CY_CFG_SYSCLK_ECO_GPIO_IN_PIN 2
#define CY_CFG_SYSCLK_ECO_GPIO_OUT_PRT GPIO_PRT21
#define CY_CFG_SYSCLK_ECO_GPIO_OUT_PIN 3
#define CY_CFG_SYSCLK_ECO_CLOAD 8UL
#define CY_CFG_SYSCLK_ECO_ESR 150UL
#define CY_CFG_SYSCLK_ECO_DRIVE_LEVEL 300UL

#if (!defined(CY_DEVICE_SECURE))
    __STATIC_INLINE void Cy_SysClk_EcoInit()
    {
        (void)Cy_GPIO_Pin_FastInit(GPIO_PRT21, 2, CY_GPIO_DM_ANALOG, 0UL, HSIOM_SEL_GPIO);
        (void)Cy_GPIO_Pin_FastInit(GPIO_PRT21, 3, CY_GPIO_DM_ANALOG, 0UL, HSIOM_SEL_GPIO);
        if (CY_SYSCLK_BAD_PARAM == Cy_SysClk_EcoConfigure(CY_CFG_SYSCLK_ECO_FREQ, 9UL, 150UL, 300UL))
        {
            cycfg_ClockStartupError(CY_CFG_SYSCLK_ECO_ERROR);
        }
        if (CY_SYSCLK_TIMEOUT == Cy_SysClk_EcoEnable(3000UL))
        {
            cycfg_ClockStartupError(CY_CFG_SYSCLK_ECO_ERROR);
        }
    }
#endif //(!defined(CY_DEVICE_SECURE))
```

Figure 13 **Code preview**

3.2 WCO configuration

3.2.1 Overview

The WCO is disabled by default. Accordingly, WCO cannot be used unless it is enabled. Figure 14 shows how to configure the registers for enabling the WCO.

To disable the WCO, write '0' to the WCO_EN bit of the BACKUP_CTL register.

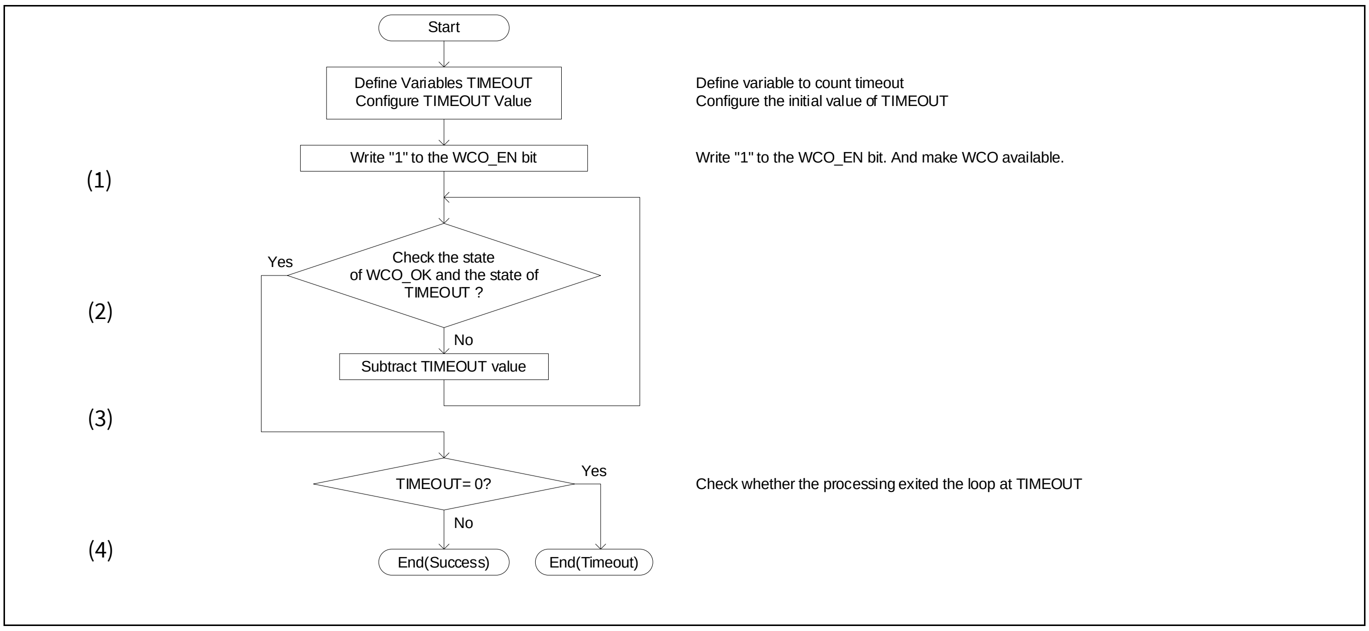
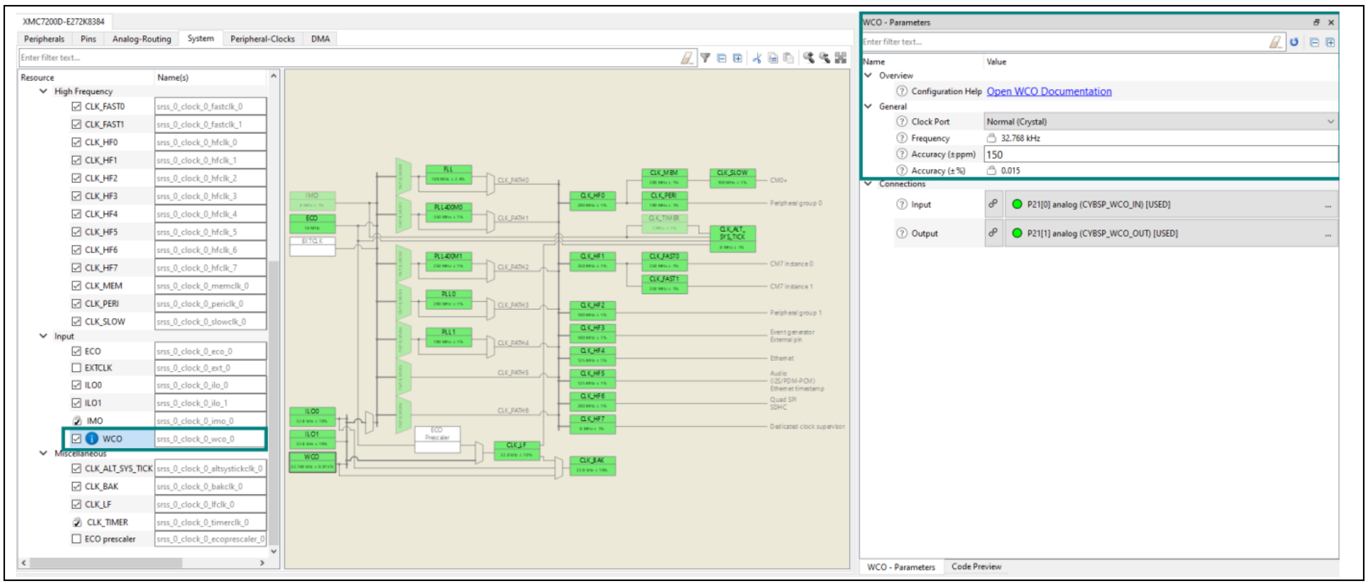

Start
Define Variables TIMEOUT
Configure TIMEOUT Value
Define variable to count timeout
Configure the initial value of TIMEOUT
(1)
Write "1" to the WCO_EN bit
Write "1" to the WCO_EN bit. And make WCO available.
(2)
Yes
Check the state of WCO_OK and the state of TIMEOUT ?
No
Subtract TIMEOUT value
(3)
TIMEOUT= 0?
Yes
No
Check whether the processing exited the loop at TIMEOUT
(4)
End(Success)
End(Timeout)

Figure 14 **Enabling the WCO**

You can enable the WCO in the **System** tab of the Device Configurator. ModusToolbox™ software automatically generates the corresponding code.

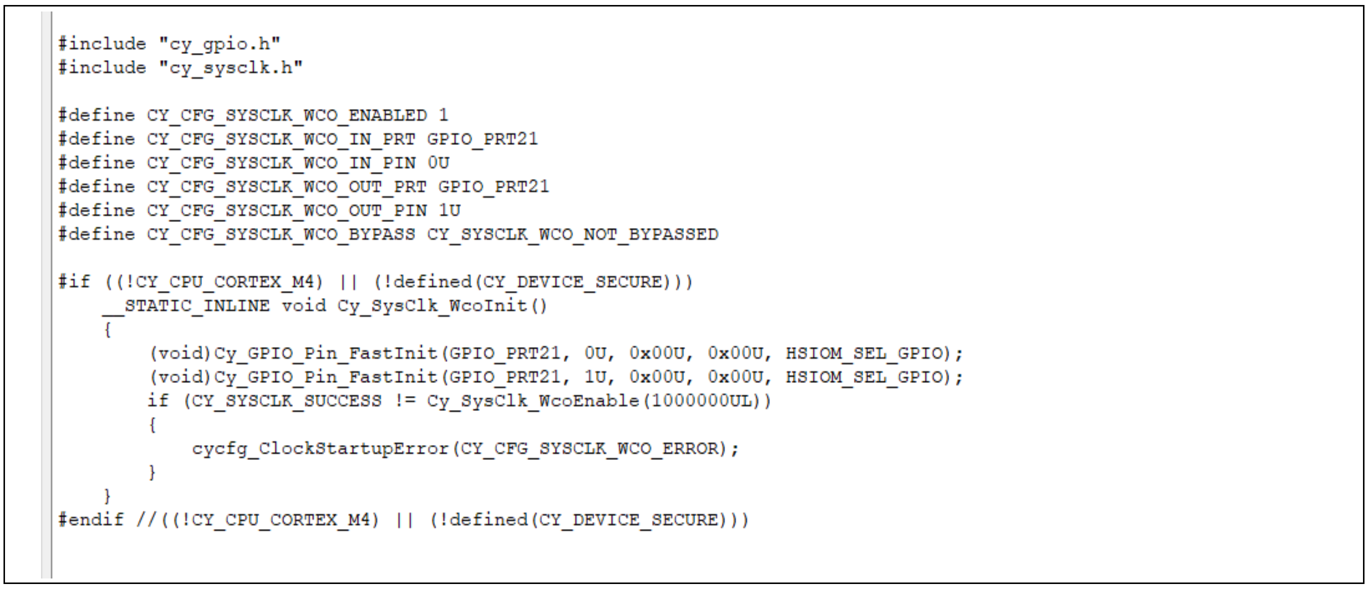


Resource	Name(s)
High Frequency	
☑ CLK_FAST0	srss_0_clock_0_fastclk_0
☑ CLK_FAST1	srss_0_clock_0_fastclk_1
☑ CLK_HF0	srss_0_clock_0_hfclk_0
☑ CLK_HF1	srss_0_clock_0_hfclk_1
☑ CLK_HF2	srss_0_clock_0_hfclk_2
☑ CLK_HF3	srss_0_clock_0_hfclk_3
☑ CLK_HF4	srss_0_clock_0_hfclk_4
☑ CLK_HF5	srss_0_clock_0_hfclk_5
☑ CLK_HF6	srss_0_clock_0_hfclk_6
☑ CLK_HF7	srss_0_clock_0_hfclk_7
☑ CLK_MEM	srss_0_clock_0_memclk_0
☑ CLK_PERI	srss_0_clock_0_periclk_0
☑ CLK_SLOW	srss_0_clock_0_slowclk_0
Input	
☑ ECO	srss_0_clock_0_eco_0
☐ EXTCLK	srss_0_clock_0_ext_0
☑ ILO0	srss_0_clock_0_ilo_0
☑ ILO1	srss_0_clock_0_ilo_1
✓ IMO	srss_0_clock_0_imo_0
☑ WCO	srss_0_clock_0_wco_0
Miscellaneous	
☑ CLK_ALT_SYS_TICK	srss_0_clock_0_altsystickclk_0
☑ CLK_BAK	srss_0_clock_0_bakclk_0
☑ CLK_LF	srss_0_clock_0_lfclk_0
✓ CLK_TIMER	srss_0_clock_0_timerclk_0
☐ ECO prescaler	srss_0_clock_0_ecoprescaler_0

Name	Value
Overview	
Configuration Help	Open WCO Documentation
General	
Clock Port	Normal (Crystal)
Frequency	32.768 kHz
Accuracy (±ppm)	150
Accuracy (±%)	0.015
Connections	
Input	● P21[0] analog (CYBSP_WCO_IN) [USED]
Output	● P21[1] analog (CYBSP_WCO_OUT) [USED]

Figure 15 **WCO Configuration**

This code is auto generated after configuration in the Device Configurator.



```
#include "cy_gpio.h"
#include "cy_sysclk.h"

#define CY_CFG_SYSCLK_WCO_ENABLED 1
#define CY_CFG_SYSCLK_WCO_IN_PRT GPIO_PRT21
#define CY_CFG_SYSCLK_WCO_IN_PIN 0U
#define CY_CFG_SYSCLK_WCO_OUT_PRT GPIO_PRT21
#define CY_CFG_SYSCLK_WCO_OUT_PIN 1U
#define CY_CFG_SYSCLK_WCO_BYPASS CY_SYSCLK_WCO_NOT_BYPASSED

#if ((!CY_CPU_CORTEX_M4) || (!defined(CY_DEVICE_SECURE)))
    __STATIC_INLINE void Cy_SysClk_WcoInit()
    {
        (void)Cy_GPIO_Pin_FastInit(GPIO_PRT21, 0U, 0x00U, 0x00U, HSIOM_SEL_GPIO);
        (void)Cy_GPIO_Pin_FastInit(GPIO_PRT21, 1U, 0x00U, 0x00U, HSIOM_SEL_GPIO);
        if (CY_SYSCLK_SUCCESS != Cy_SysClk_WcoEnable(1000000UL))
        {
            cycfg_ClockStartupError(CY_CFG_SYSCLK_WCO_ERROR);
        }
    }
#endif //((!CY_CPU_CORTEX_M4) || (!defined(CY_DEVICE_SECURE)))
```

Figure 16 Code preview

3.3 Setting the IMO

The IMO is enabled by default so that all functions operate properly. The IMO will automatically be disabled during DeepSleep, Hibernate, and XRES modes. Therefore, it is not required to set IMO explicitly.

3.4 Setting ILO0/ILO1

ILO0 and ILO1 are enabled by default.

Note: *ILO0 is used as the operating clock of the watchdog timer (WDT). Therefore, if ILO0 is disabled, you should disable the WDT. To disable ILO0, write '0b01' to the WDT_LOCK bit of the WDT_CTL register, and then write '0b00' to the ENABLE bit of the CLK_ILO0_CONFIG register.*

4 FLL and PLL configuration

This section shows the configuration of FLL and PLL in the clock system.

4.1 Setting the FLL

4.1.1 Overview

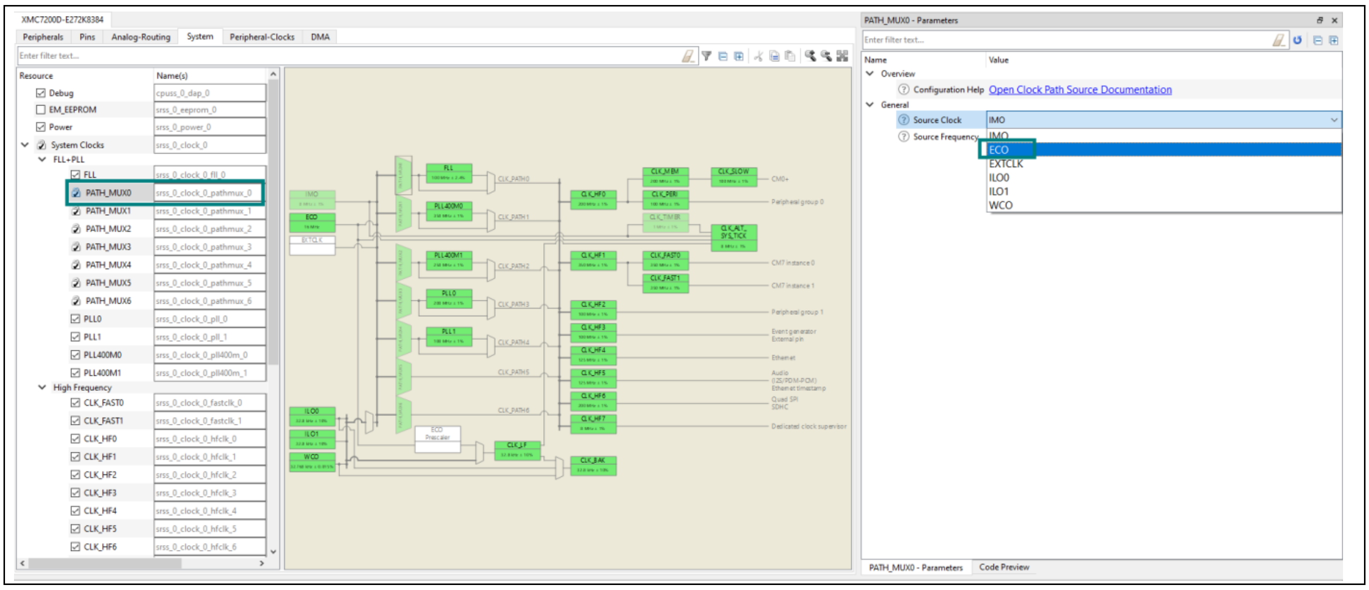
The FLL must be configured before it is used. The FLL has a current-controlled oscillator (CCO); the output frequency of this CCO is controlled by adjusting the trim of the CCO.

4.1.2 Use case

Input clock frequency	16 MHz
Output clock frequency	100 MHz

You can configure these parameters in the **System** tab of the Device Configurator; based on these, ModusToolbox™ software automatically generates the corresponding code.

1. In the Device Configurator, select PATH_MUX0.
2. On the PATH_MUX Parameters pane, select ECO (16 MHz) as the source clock.



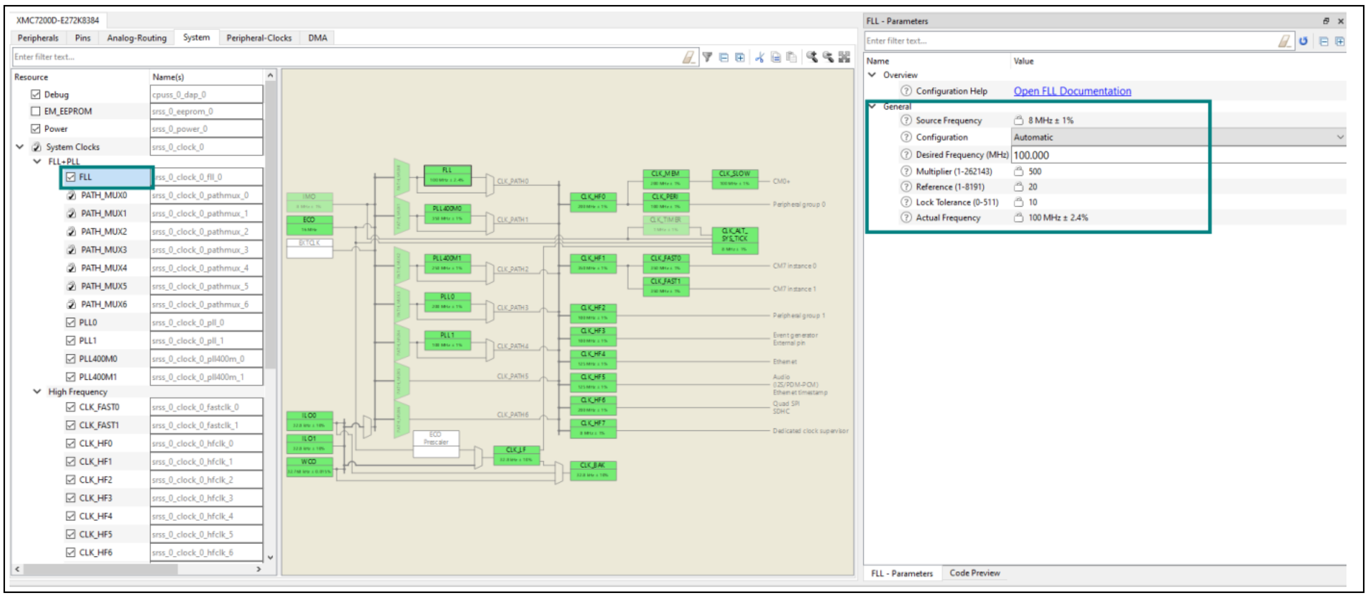
PATH_MUX0 - Parameters

Name	Value
Overview	
Configuration Help	Open Clock Path Source Documentation
General	
Source Clock	IMO
Source Frequency	

IMO
ECO
EXTCLK
ILO0
ILO1
WCO

Figure 17 Setting ECO (16 MHz) as the source clock

Select FLL, and in the FLL – Parameters pane, under General, enter the **Desired Frequency (MHz)** as 100.000 to set the frequency as 100 MHz.

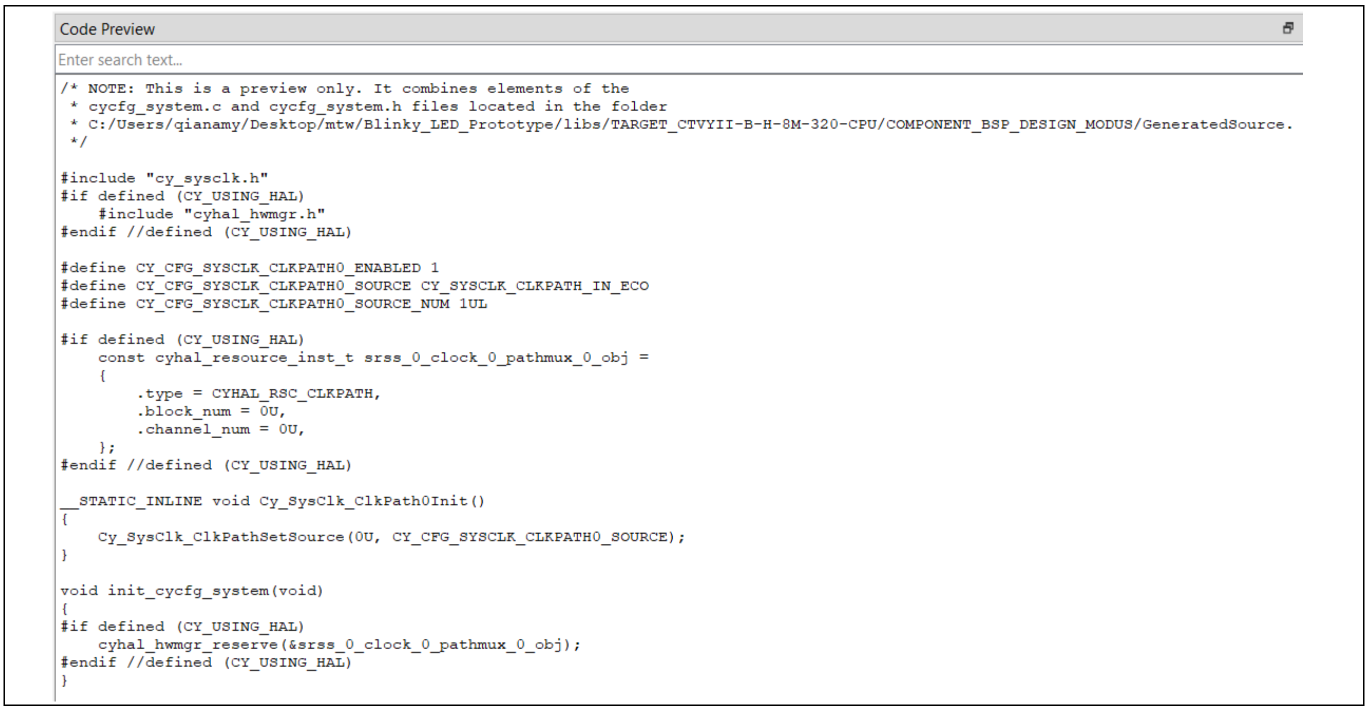


FLL - Parameters

Name	Value
Overview	
Configuration Help	Open FLL Documentation
General	
Source Frequency	8 MHz ± 1%
Configuration	Automatic
Desired Frequency (MHz)	100.000
Multiplier (1-262143)	500
Reference (1-8191)	20
Lock Tolerance (0-511)	10
Actual Frequency	100 MHz ± 2.4%

Figure 18 Setting output clock frequency as 100.000 MHz

4.1.3 Code preview



Code Preview

```
/* NOTE: This is a preview only. It combines elements of the
 * cycfg_system.c and cycfg_system.h files located in the folder
 * C:/Users/qianamy/Desktop/mtw/Blinky_LED_Prototype/libs/TARGET_CTVYII-B-H-8M-320-CPU/COMPONENT_BSP_DESIGN_MODUS/GeneratedSource.
 */

#include "cy_sysclk.h"
#if defined (CY_USING_HAL)
    #include "cyhal_hwmgr.h"
#endif //defined (CY_USING_HAL)

#define CY_CFG_SYSCLK_CLKPATH0_ENABLED 1
#define CY_CFG_SYSCLK_CLKPATH0_SOURCE CY_SYSCLK_CLKPATH_IN_ECO
#define CY_CFG_SYSCLK_CLKPATH0_SOURCE_NUM 1UL

#if defined (CY_USING_HAL)
    const cyhal_resource_inst_t srss_0_clock_0_pathmux_0_obj =
    {
        .type = CYHAL_RSC_CLKPATH,
        .block_num = 0U,
        .channel_num = 0U,
    };
#endif //defined (CY_USING_HAL)

__STATIC_INLINE void Cy_SysClk_ClkPath0Init()
{
    Cy_SysClk_ClkPathSetSource(0U, CY_CFG_SYSCLK_CLKPATH0_SOURCE);
}

void init_cycfg_system(void)
{
#if defined (CY_USING_HAL)
    cyhal_hwmgr_reserve(&srss_0_clock_0_pathmux_0_obj);
#endif //defined (CY_USING_HAL)
}
```

Figure 19 Code preview

4.2 Setting the PLL

4.2.1 Overview

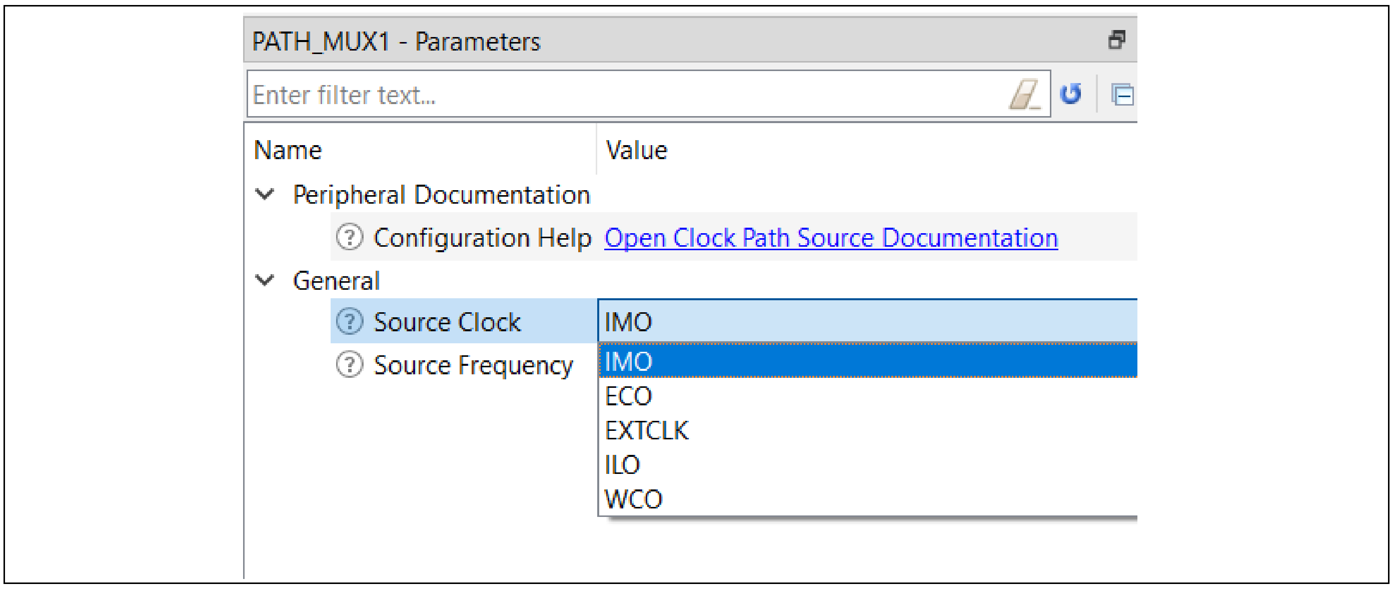
You must configure the PLL before using it.

You can configure these parameters in the System tab of the Device Configurator; based on these, ModusToolbox™ software automatically generates the corresponding code.

4.2.2 Use case

Parameter	Value
Input clock frequency	8.000 MHz
Output clock frequency	350.000 MHz (PLL400 #0) 250 MHz (PLL400 #1) 200.000 MHz (PLL200 #0) 100.000 MHz (PLL200 #1)

First, select the source clock for PLL400M#0/PLL400M#1/PLL200M#0/PLL200M#1 in PATH_MUX1/PATH_MUX2/PATH_MUX3/PATH_MUX4.

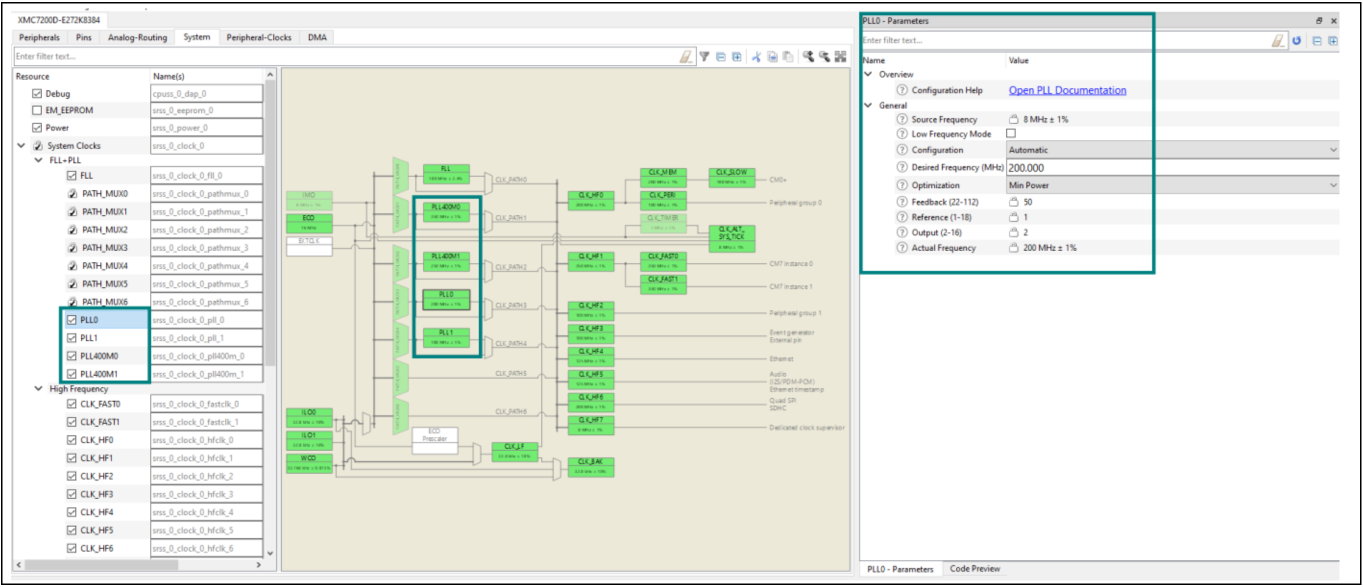

PATH_MUX1 - Parameters

Enter filter text...

Name	Value
Peripheral Documentation	
Configuration Help	Open Clock Path Source Documentation
General	
Source Clock	IMO
Source Frequency	

Figure 20 Setting 'IMO' in 'Parameters'

Configure the corresponding desired frequency as follows:



XMC7200D-E272K8384

Peripherals | Pins | Analog-Routing | System | Peripheral-Clocks | DMA

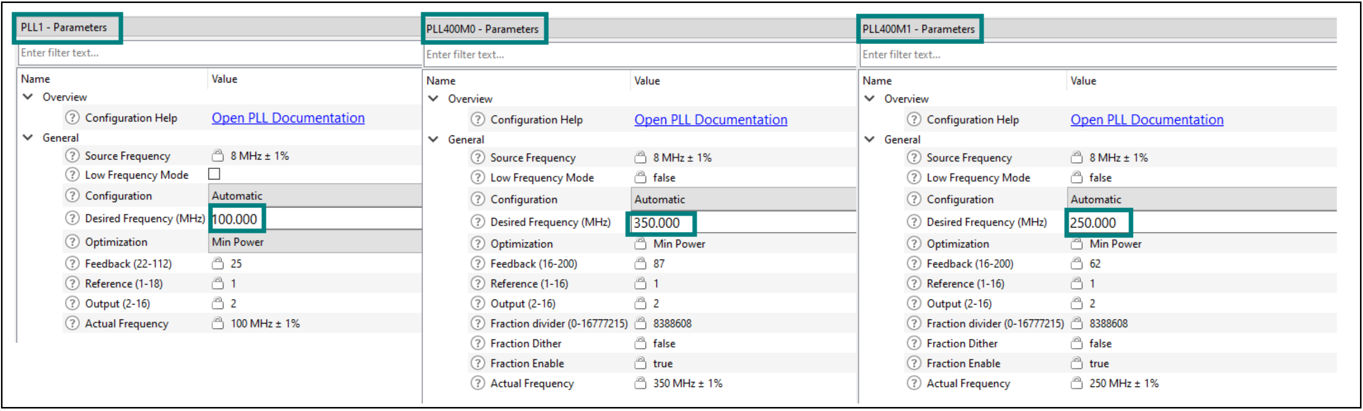
Resource	Name(s)
☑ Debug	cpuss_0_dap_0
☐ EM_EEPROM	srss_0_eeprom_0
☑ Power	srss_0_power_0
System Clocks	srss_0_clock_0
FLL+PLL	
☑ FLL	srss_0_clock_0_fll_0
PATH_MUX0	srss_0_clock_0_pathmux_0
PATH_MUX1	srss_0_clock_0_pathmux_1
PATH_MUX2	srss_0_clock_0_pathmux_2
PATH_MUX3	srss_0_clock_0_pathmux_3
PATH_MUX4	srss_0_clock_0_pathmux_4
PATH_MUX5	srss_0_clock_0_pathmux_5
PATH_MUX6	srss_0_clock_0_pathmux_6
☑ PLL0	srss_0_clock_0_pll_0
☑ PLL1	srss_0_clock_0_pll_1
☑ PLL400M0	srss_0_clock_0_pll400m_0
☑ PLL400M1	srss_0_clock_0_pll400m_1
High Frequency	
☑ CLK_FAST0	srss_0_clock_0_fastclk_0
☑ CLK_FAST1	srss_0_clock_0_fastclk_1
☑ CLK_HF0	srss_0_clock_0_hfclk_0
☑ CLK_HF1	srss_0_clock_0_hfclk_1
☑ CLK_HF2	srss_0_clock_0_hfclk_2
☑ CLK_HF3	srss_0_clock_0_hfclk_3
☑ CLK_HF4	srss_0_clock_0_hfclk_4
☑ CLK_HF5	srss_0_clock_0_hfclk_5
☑ CLK_HF6	srss_0_clock_0_hfclk_6

PLL0 - Parameters

Name	Value
Overview	
Configuration Help	Open PLL Documentation
General	
Source Frequency	8 MHz ± 1%
Low Frequency Mode	☐
Configuration	Automatic
Desired Frequency (MHz)	200.000
Optimization	Min Power
Feedback (22-112)	50
Reference (1-18)	1
Output (2-16)	2
Actual Frequency	200 MHz ± 1%

PLL0 - Parameters | Code Preview

Figure 21 Configuring output frequency in ‘Parameters’



PLL1 - Parameters

Name	Value
Overview	
Configuration Help	Open PLL Documentation
General	
Source Frequency	8 MHz ± 1%
Low Frequency Mode	☐
Configuration	Automatic
Desired Frequency (MHz)	100.000
Optimization	Min Power
Feedback (22-112)	25
Reference (1-18)	1
Output (2-16)	2
Actual Frequency	100 MHz ± 1%

PLL400M0 - Parameters

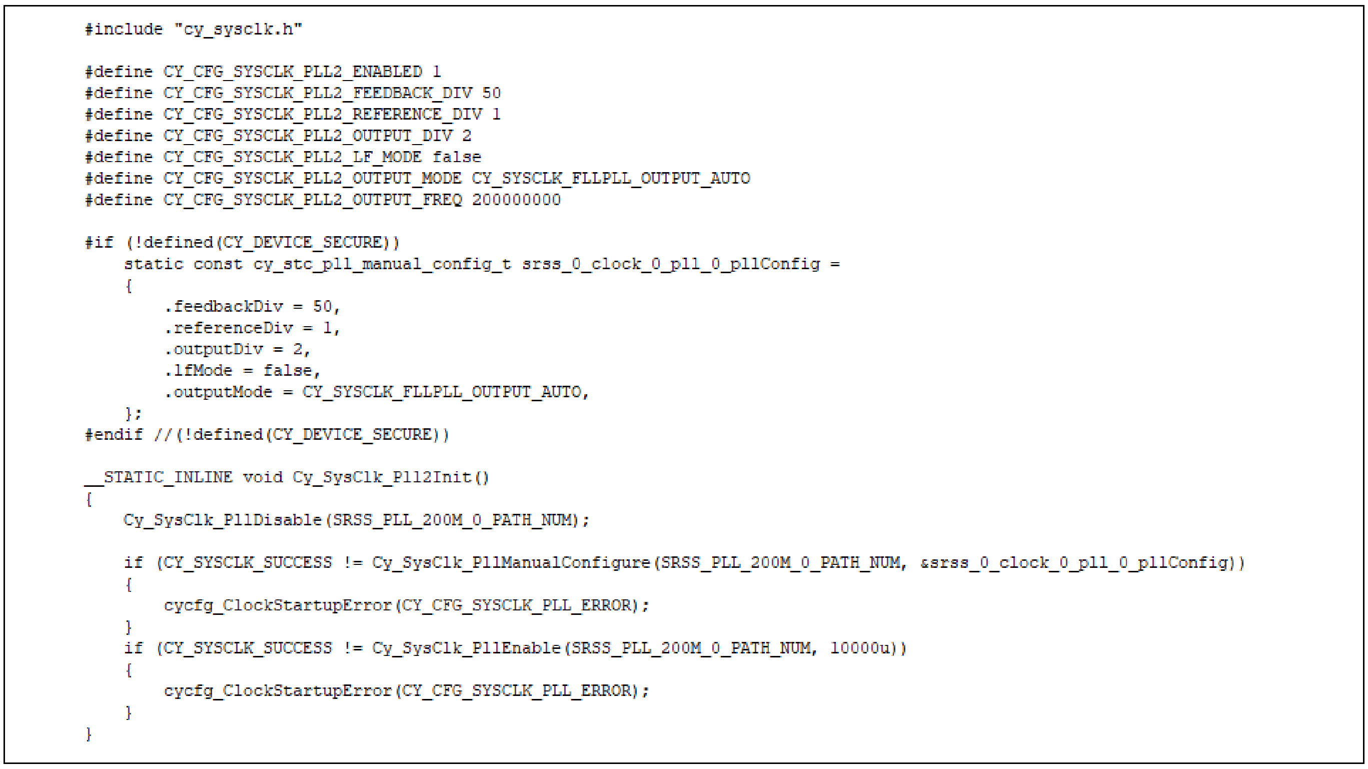
Name	Value
Overview	
Configuration Help	Open PLL Documentation
General	
Source Frequency	8 MHz ± 1%
Low Frequency Mode	false
Configuration	Automatic
Desired Frequency (MHz)	350.000
Optimization	Min Power
Feedback (16-200)	87
Reference (1-16)	1
Output (2-16)	2
Fraction divider (0-16777215)	8388608
Fraction Dither	false
Fraction Enable	true
Actual Frequency	350 MHz ± 1%

PLL400M1 - Parameters

Name	Value
Overview	
Configuration Help	Open PLL Documentation
General	
Source Frequency	8 MHz ± 1%
Low Frequency Mode	false
Configuration	Automatic
Desired Frequency (MHz)	250.000
Optimization	Min Power
Feedback (16-200)	62
Reference (1-16)	1
Output (2-16)	2
Fraction divider (0-16777215)	8388608
Fraction Dither	false
Fraction Enable	true
Actual Frequency	250 MHz ± 1%

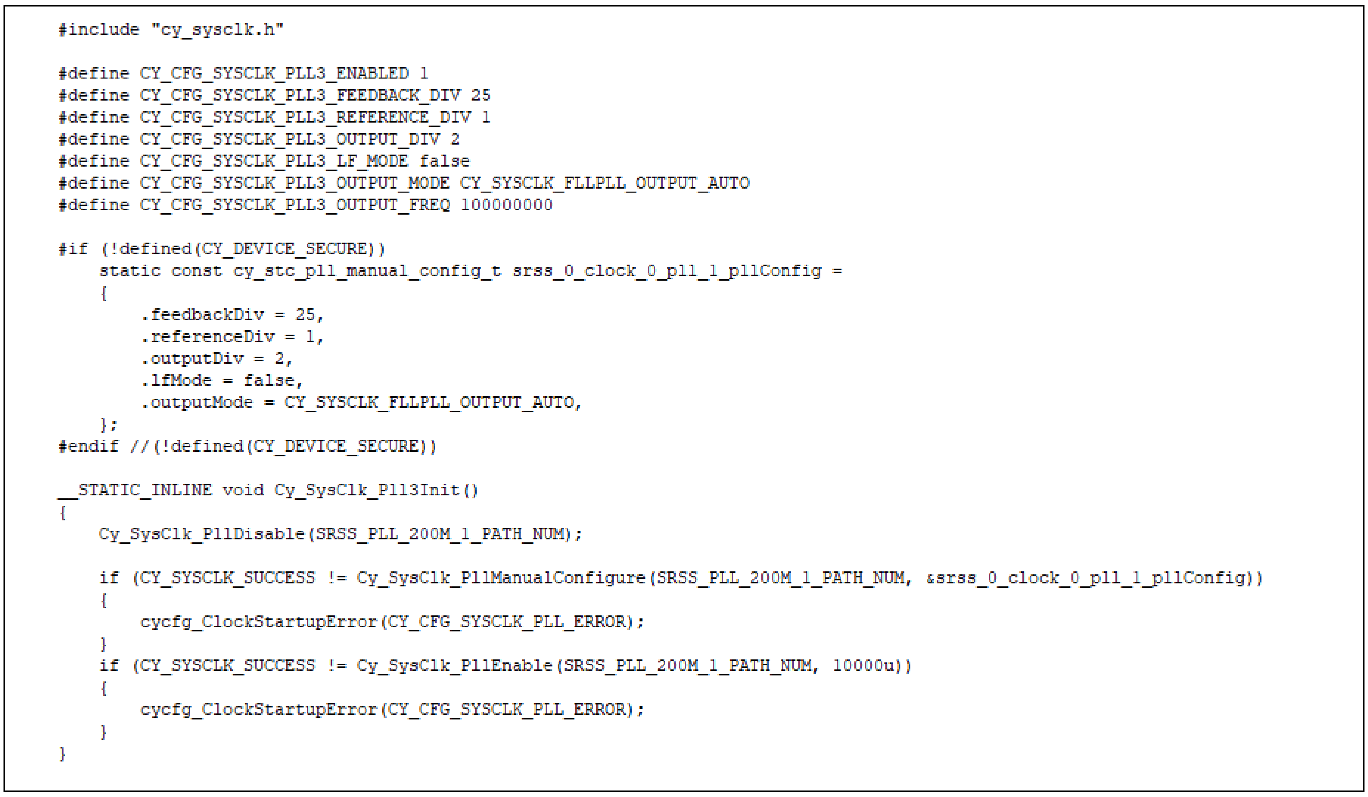
Figure 22 Configuring output frequency in ‘Parameters’

4.2.3 Code preview



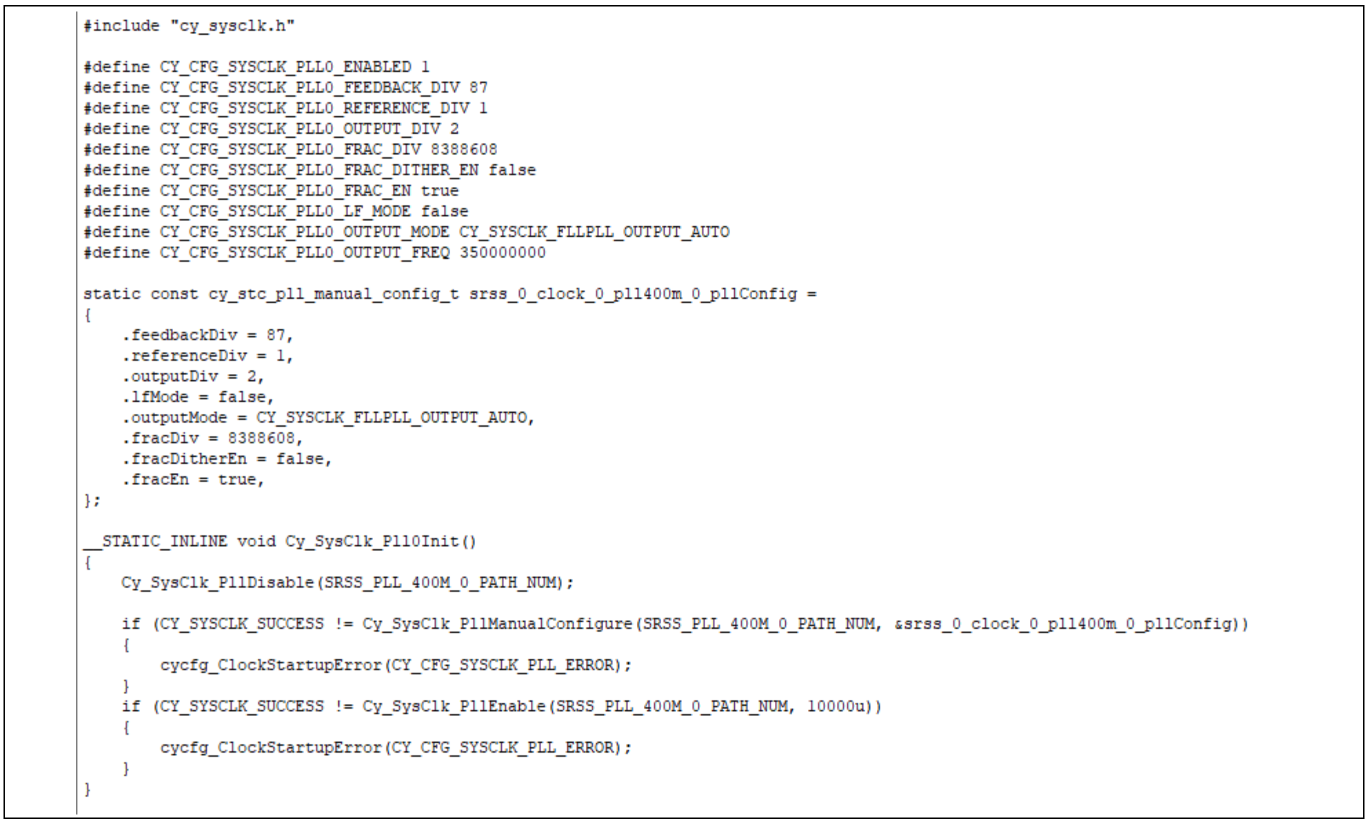
```
#include "cy_sysclk.h"

#define CY_CFG_SYSCLK_PLL2_ENABLED 1
#define CY_CFG_SYSCLK_PLL2_FEEDBACK_DIV 50
#define CY_CFG_SYSCLK_PLL2_REFERENCE_DIV 1
#define CY_CFG_SYSCLK_PLL2_OUTPUT_DIV 2
#define CY_CFG_SYSCLK_PLL2_LF_MODE false
#define CY_CFG_SYSCLK_PLL2_OUTPUT_MODE CY_SYSCLK_FLLPLL_OUTPUT_AUTO
#define CY_CFG_SYSCLK_PLL2_OUTPUT_FREQ 200000000

#if (!defined(CY_DEVICE_SECURE))
    static const cy_stc_pll_manual_config_t srss_0_clock_0_pll_0_pllConfig =
    {
        .feedbackDiv = 50,
        .referenceDiv = 1,
        .outputDiv = 2,
        .lfMode = false,
        .outputMode = CY_SYSCLK_FLLPLL_OUTPUT_AUTO,
    };
#endif //(!defined(CY_DEVICE_SECURE))

__STATIC_INLINE void Cy_SysClk_Pll2Init()
{
    Cy_SysClk_PllDisable(SRSS_PLL_200M_0_PATH_NUM);

    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllManualConfigure(SRSS_PLL_200M_0_PATH_NUM, &srss_0_clock_0_pll_0_pllConfig))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllEnable(SRSS_PLL_200M_0_PATH_NUM, 10000u))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
}
```

Figure 23 Code preview for PLL200M#0 configuration



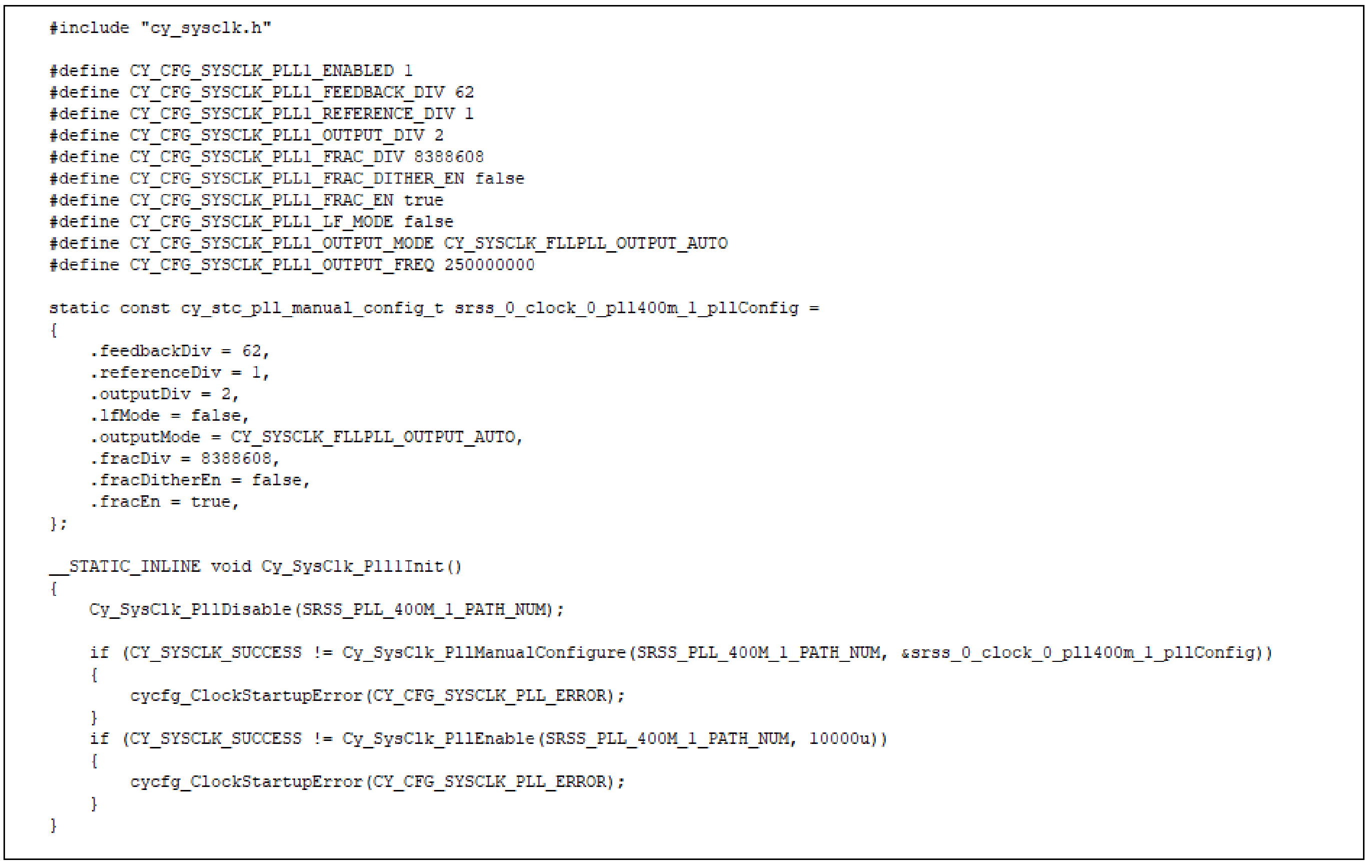
```
#include "cy_sysclk.h"

#define CY_CFG_SYSCLK_PLL3_ENABLED 1
#define CY_CFG_SYSCLK_PLL3_FEEDBACK_DIV 25
#define CY_CFG_SYSCLK_PLL3_REFERENCE_DIV 1
#define CY_CFG_SYSCLK_PLL3_OUTPUT_DIV 2
#define CY_CFG_SYSCLK_PLL3_LF_MODE false
#define CY_CFG_SYSCLK_PLL3_OUTPUT_MODE CY_SYSCLK_FLLPLL_OUTPUT_AUTO
#define CY_CFG_SYSCLK_PLL3_OUTPUT_FREQ 100000000

#if (!defined(CY_DEVICE_SECURE))
    static const cy_stc_pll_manual_config_t srss_0_clock_0_pll_1_pllConfig =
    {
        .feedbackDiv = 25,
        .referenceDiv = 1,
        .outputDiv = 2,
        .lfMode = false,
        .outputMode = CY_SYSCLK_FLLPLL_OUTPUT_AUTO,
    };
#endif //(!defined(CY_DEVICE_SECURE))

__STATIC_INLINE void Cy_SysClk_Pll3Init()
{
    Cy_SysClk_PllDisable(SRSS_PLL_200M_1_PATH_NUM);

    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllManualConfigure(SRSS_PLL_200M_1_PATH_NUM, &srss_0_clock_0_pll_1_pllConfig))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllEnable(SRSS_PLL_200M_1_PATH_NUM, 10000u))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
}
```

Figure 24 Code preview for PLL200M#1 configuration



```
#include "cy_sysclk.h"

#define CY_CFG_SYSCLK_PLL0_ENABLED 1
#define CY_CFG_SYSCLK_PLL0_FEEDBACK_DIV 87
#define CY_CFG_SYSCLK_PLL0_REFERENCE_DIV 1
#define CY_CFG_SYSCLK_PLL0_OUTPUT_DIV 2
#define CY_CFG_SYSCLK_PLL0_FRAC_DIV 8388608
#define CY_CFG_SYSCLK_PLL0_FRAC_DITHER_EN false
#define CY_CFG_SYSCLK_PLL0_FRAC_EN true
#define CY_CFG_SYSCLK_PLL0_LF_MODE false
#define CY_CFG_SYSCLK_PLL0_OUTPUT_MODE CY_SYSCLK_FLLPLL_OUTPUT_AUTO
#define CY_CFG_SYSCLK_PLL0_OUTPUT_FREQ 350000000

static const cy_stc_pll_manual_config_t srss_0_clock_0_pll400m_0_pllConfig =
{
    .feedbackDiv = 87,
    .referenceDiv = 1,
    .outputDiv = 2,
    .lfMode = false,
    .outputMode = CY_SYSCLK_FLLPLL_OUTPUT_AUTO,
    .fracDiv = 8388608,
    .fracDitherEn = false,
    .fracEn = true,
};

__STATIC_INLINE void Cy_SysClk_Pll0Init()
{
    Cy_SysClk_PllDisable(SRSS_PLL_400M_0_PATH_NUM);

    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllManualConfigure(SRSS_PLL_400M_0_PATH_NUM, &srss_0_clock_0_pll400m_0_pllConfig))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllEnable(SRSS_PLL_400M_0_PATH_NUM, 10000u))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
}
```

Figure 25 Code preview for PLL400M#0 configuration



```
#include "cy_sysclk.h"

#define CY_CFG_SYSCLK_PLL1_ENABLED 1
#define CY_CFG_SYSCLK_PLL1_FEEDBACK_DIV 62
#define CY_CFG_SYSCLK_PLL1_REFERENCE_DIV 1
#define CY_CFG_SYSCLK_PLL1_OUTPUT_DIV 2
#define CY_CFG_SYSCLK_PLL1_FRAC_DIV 8388608
#define CY_CFG_SYSCLK_PLL1_FRAC_DITHER_EN false
#define CY_CFG_SYSCLK_PLL1_FRAC_EN true
#define CY_CFG_SYSCLK_PLL1_LF_MODE false
#define CY_CFG_SYSCLK_PLL1_OUTPUT_MODE CY_SYSCLK_FLLPLL_OUTPUT_AUTO
#define CY_CFG_SYSCLK_PLL1_OUTPUT_FREQ 250000000

static const cy_stc_pll_manual_config_t srss_0_clock_0_pll400m_1_pllConfig =
{
    .feedbackDiv = 62,
    .referenceDiv = 1,
    .outputDiv = 2,
    .lfMode = false,
    .outputMode = CY_SYSCLK_FLLPLL_OUTPUT_AUTO,
    .fracDiv = 8388608,
    .fracDitherEn = false,
    .fracEn = true,
};

__STATIC_INLINE void Cy_SysClk_Pll1Init()
{
    Cy_SysClk_PllDisable(SRSS_PLL_400M_1_PATH_NUM);

    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllManualConfigure(SRSS_PLL_400M_1_PATH_NUM, &srss_0_clock_0_pll400m_1_pllConfig))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
    if (CY_SYSCLK_SUCCESS != Cy_SysClk_PllEnable(SRSS_PLL_400M_1_PATH_NUM, 10000u))
    {
        cycfg_ClockStartupError(CY_CFG_SYSCLK_PLL_ERROR);
    }
}
```

Figure 26 Code preview for PLL400M#1 configuration

5 Internal clock configuration

5.1 Configuring CLK_PATHx

CLK_PATHx is used as the input source for root clocks CLK_HFx. CLK_PATHx can select all clock resources including the FLL and PLL using DSI_MUX and PATH_MUX. CLK_PATH5 cannot select the FLL and PLL, but other clock resources can be selected.

Figure 27 shows the generation diagram for CLK_PATHx.

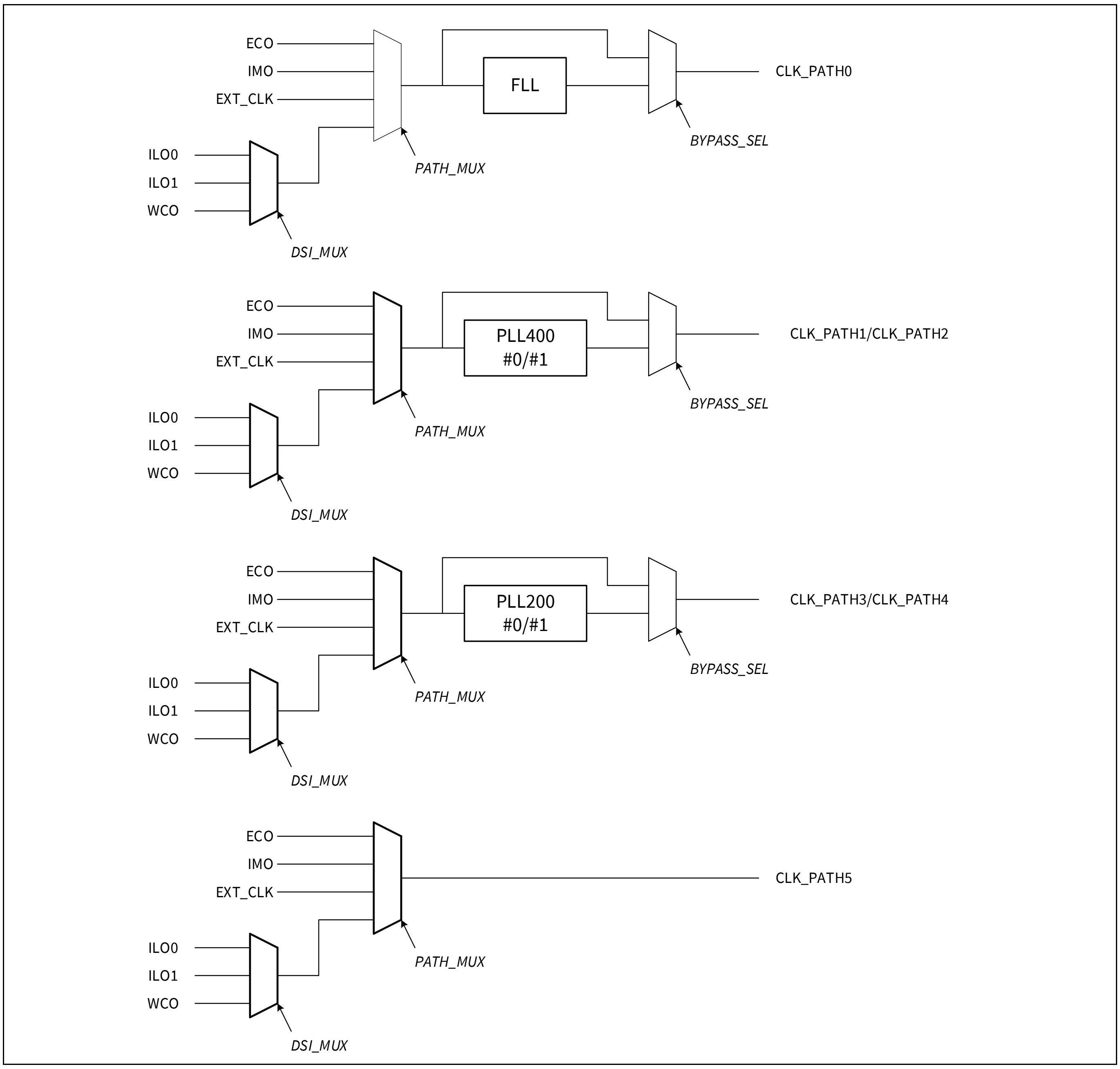
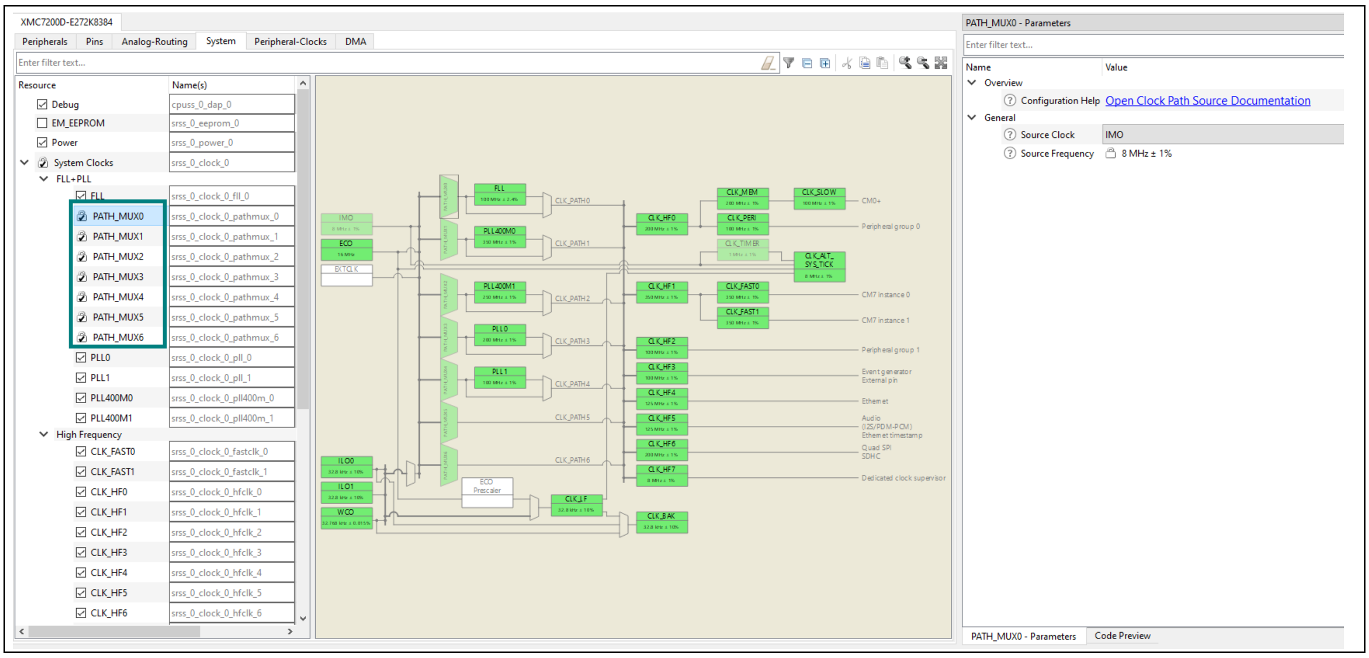


Figure 27 Generating CLK_PATHx

You can configure CLK_PATHx in the **System** tab of the Device Configurator. ModusToolbox™ software automatically generates the corresponding code.



Resource	Name(s)
Debug	cpuss_0_dap_0
EM_EEPROM	srss_0_eeprom_0
Power	srss_0_power_0
System Clocks	srss_0_clock_0
FLL+PLL	
FLL	srss_0_clock_0_fll_0
PATH_MUX0	srss_0_clock_0_pathmux_0
PATH_MUX1	srss_0_clock_0_pathmux_1
PATH_MUX2	srss_0_clock_0_pathmux_2
PATH_MUX3	srss_0_clock_0_pathmux_3
PATH_MUX4	srss_0_clock_0_pathmux_4
PATH_MUX5	srss_0_clock_0_pathmux_5
PATH_MUX6	srss_0_clock_0_pathmux_6
PLL0	srss_0_clock_0_pll_0
PLL1	srss_0_clock_0_pll_1
PLL400M0	srss_0_clock_0_pll400m_0
PLL400M1	srss_0_clock_0_pll400m_1
High Frequency	
CLK_FAST0	srss_0_clock_0_fastclk_0
CLK_FAST1	srss_0_clock_0_fastclk_1
CLK_HF0	srss_0_clock_0_hfclk_0
CLK_HF1	srss_0_clock_0_hfclk_1
CLK_HF2	srss_0_clock_0_hfclk_2
CLK_HF3	srss_0_clock_0_hfclk_3
CLK_HF4	srss_0_clock_0_hfclk_4
CLK_HF5	srss_0_clock_0_hfclk_5
CLK_HF6	srss_0_clock_0_hfclk_6

PATH_MUX0 - Parameters

Name	Value
Overview	
Configuration Help	Open Clock Path Source Documentation
General	
Source Clock	IMO
Source Frequency	8 MHz ± 1%

Figure 28 CLK_PATHx configuration

5.1.1 Code preview

```
#include "cy_sysclk.h"
#if defined (CY_USING_HAL)
    #include "cyhal_hwmgr.h"
#endif //defined (CY_USING_HAL)

#define CY_CFG_SYSCLK_CLKPATH0_ENABLED 1
#define CY_CFG_SYSCLK_CLKPATH0_SOURCE CY_SYSCLK_CLKPATH_IN_ECO
#define CY_CFG_SYSCLK_CLKPATH0_SOURCE_NUM 1UL

#if defined (CY_USING_HAL)
    const cyhal_resource_inst_t srss_0_clock_0_pathmux_0_obj =
    {
        .type = CYHAL_RSC_CLKPATH,
        .block_num = 0U,
        .channel_num = 0U,
    };
#endif //defined (CY_USING_HAL)

__STATIC_INLINE void Cy_SysClk_ClkPath0Init()
{
    Cy_SysClk_ClkPathSetSource(0U, CY_CFG_SYSCLK_CLKPATH0_SOURCE);
}

void init_cycfg_system(void)
{
#if defined (CY_USING_HAL)
    cyhal_hwmgr_reserve(&srss_0_clock_0_pathmux_0_obj);
#endif //defined (CY_USING_HAL)
}
```

Figure 29 Code preview for CLK_PATHx configuration

5.2 Configuring CLK_HFx

CLK_HFx (x=0,1,2,3,4,5,6,7) can be selected from any CLK_PATHy (y=0,1,2,3,4,5). A predivider is available to divide the selected CLK_PATHx. CLK_HF0 is always enabled because it is the source clock for the CPU cores. CLK_HFx can be disabled.

To enable CLK_HFx, write '1' to the ENABLE bit of each CLK_ROOT_SELECT register. To disable CLK_HFx, write '0' to the ENABLE bit of each CLK_ROOT_SELECT register.

The ROOT_DIV bit of the CLK_ROOT register sets the predivider values from the options: No division, divide by 2, by 4, and by 8.

Figure 30 shows the details of ROOT_MUX and the predivider.

o

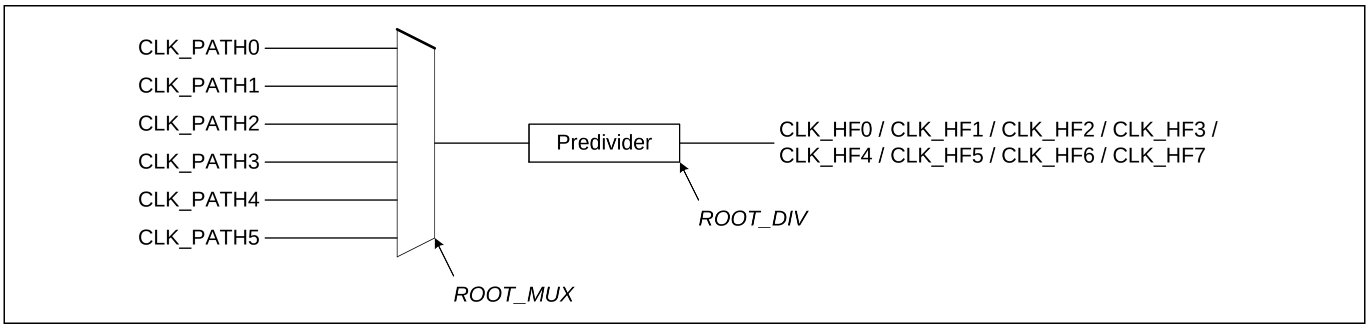
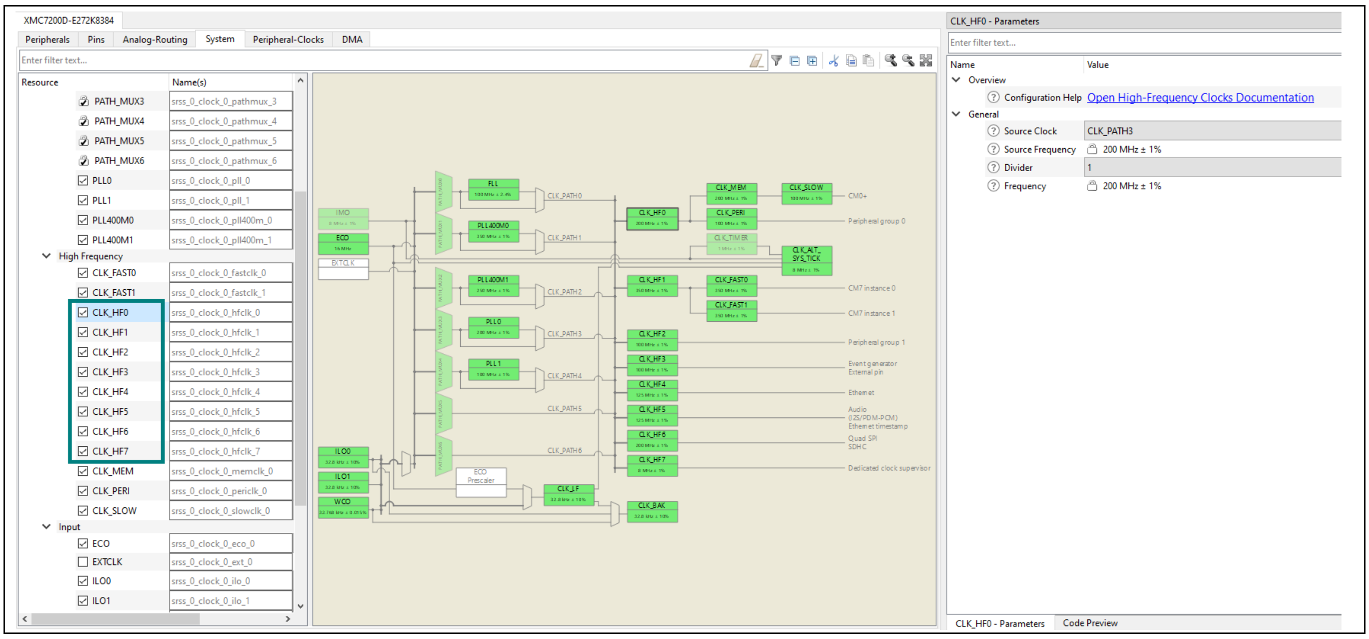


Figure 30 ROOT_MUX and predivider

You can configure CLK_HFx in the **System** tab of the Device Configurator. ModusToolbox™ software automatically generates the corresponding code.



Resource	Name(s)
PATH_MUX3	srss_0_clock_0_pathmux_3
PATH_MUX4	srss_0_clock_0_pathmux_4
PATH_MUX5	srss_0_clock_0_pathmux_5
PATH_MUX6	srss_0_clock_0_pathmux_6
☑ PLL0	srss_0_clock_0_pll_0
☑ PLL1	srss_0_clock_0_pll_1
☑ PLL400M0	srss_0_clock_0_pll400m_0
☑ PLL400M1	srss_0_clock_0_pll400m_1
High Frequency	
☑ CLK_FAST0	srss_0_clock_0_fastclk_0
☑ CLK_FAST1	srss_0_clock_0_fastclk_1
☑ CLK_HF0	srss_0_clock_0_hfclk_0
☑ CLK_HF1	srss_0_clock_0_hfclk_1
☑ CLK_HF2	srss_0_clock_0_hfclk_2
☑ CLK_HF3	srss_0_clock_0_hfclk_3
☑ CLK_HF4	srss_0_clock_0_hfclk_4
☑ CLK_HF5	srss_0_clock_0_hfclk_5
☑ CLK_HF6	srss_0_clock_0_hfclk_6
☑ CLK_HF7	srss_0_clock_0_hfclk_7
☑ CLK_MEM	srss_0_clock_0_memclk_0
☑ CLK_PERI	srss_0_clock_0_periclk_0
☑ CLK_SLOW	srss_0_clock_0_slowclk_0
Input	
☑ ECO	srss_0_clock_0_eco_0
☐ EXTCLK	srss_0_clock_0_ext_0
☑ ILO0	srss_0_clock_0_ilo_0
☑ ILO1	srss_0_clock_0_ilo_1

CLK_HF0 - Parameters

Name	Value
Overview	
Configuration Help	Open High-Frequency Clocks Documentation
General	
Source Clock	CLK_PATH3
Source Frequency	200 MHz ± 1%
Divider	1
Frequency	200 MHz ± 1%

Figure 31 CLK_HFx configuration

5.2.1 Code preview

```
#include "cy_sysclk.h"

#define CY_CFG_SYSCLK_CLKHF0_ENABLED 1
#define CY_CFG_SYSCLK_CLKHF0_DIVIDER CY_SYSCLK_CLKHF_NO_DIVIDE
#define CY_CFG_SYSCLK_CLKHF0 0UL
#define CY_CFG_SYSCLK_CLKHF0_FREQ_MHZ 200UL
#define CY_CFG_SYSCLK_CLKHF0_CLKPATH CY_SYSCLK_CLKHF_IN_CLKPATH3
#define CY_CFG_SYSCLK_CLKHF0_CLKPATH_NUM 3UL

__STATIC_INLINE void Cy_SysClk_ClkHf0Init()
{
    Cy_SysClk_ClkHfSetSource(0U, CY_CFG_SYSCLK_CLKHF0_CLKPATH);
    Cy_SysClk_ClkHfSetDivider(0U, CY_SYSCLK_CLKHF_NO_DIVIDE);
    Cy_SysClk_ClkHfDirectSel(CY_CFG_SYSCLK_CLKHF0, false);
}
```

Figure 32 Code preview for CLK_HFx configuration

5.3 Configuring CLK_LF

CLK_LF can be selected from one of the possible sources: WCO, ILO0, ILO1, and ECO_Prescaler. CLK_LF cannot be set when the WDT_LOCK bit in the WDT_CTL register is disabled because CLK_LF can select ILO0 that is input clock for the WDT.

Figure 33 shows the details of LFCLK_SEL that CLK_LF is configured.

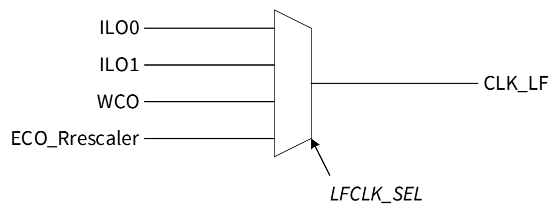


Figure 33 LFCLK_SEL

Table 1 shows the required registers for CLK_LF. See the XMC7000 MCU family architecture reference manual for more details.

Table 1 Configuring of CLK_LF

Register name	Bit name	Value	Selected item
CLK_SELECT	LFCLK_SEL[2:0]	0	ILO0
		1	WCO
		5	ILO1
		6	ECO_Prescaler
		Other	Reserved. Do not use.

5.4 Configuring CLK_FAST_0/CLK_FAST_1

CLK_FAST_0 and CLK_FAST_1 are generated by dividing CLK_HF1 by (x+1). When configuring CLK_FAST_0 and CLK_FAST_1, configure a value (x = 0..255) divided by the FRAC_DIV bit and INT_DIV bit of the CPUSS_FAST_0_CLOCK_CTL register and CPUSS_FAST_1_CLOCK_CTL register.

5.5 Configuring CLK_MEM

CLK_MEM is generated by dividing CLK_HF0; its frequency is configured by the value obtained by dividing CLK_HF0 by (x+1). When configuring CLK_MEM, configure a value (x = 0..255) divided by the INT_DIV bit of the CPUSS_MEM_CLOCK_CTL register.

5.6 Configuring CLK_PERI

CLK_PERI is the clock input to the peripheral clock divider and CLK_GR. CLK_PERI is generated by dividing CLK_HF0; its frequency is configured by the value obtained by dividing CLK_HF0 by (x+1). When configuring CLK_PERI, configure a value (x = 0..255) divided by the INT_DIV bit of the CPUSS_PERI_CLOCK_CTL register.

5.7 Configuring CLK_SLOW

CLK_SLOW is generated by dividing CLK_MEM; its frequency is configured by the value obtained by dividing CLK_MEM by (x+1). After configuring CLK_MEM, configure a value divided (x = 0..255) by the INT_DIV bit of the CPUSS_SLOW_CLOCK_CTL register.

5.8 Configuring CLK_GR

The clock source of CLK_GP is CLK_PERI in Group 3, 4, 8, and CLK_HF2 in Group 5, 6, and 9. Groups 3, 4, and 8 are clocks divided by CLK_PERI. To generate CLK_GR3, CLK_GR4, and CLK_GR8, write the division value (from 1 to 255) to divide the INT8_DIV bit of the CPUSS_PERI_GRx_CLOCK_CTL register.

5.9 Configuring PCLK

PCLK is a clock that activates each peripheral function. Peripheral Clock Dividers have a function to divide CLK_PERI and generate a clock to be supplied to each peripheral function. For assignment of the peripheral clocks, see the “Peripheral Clocks” section in the datasheet.

Figure 34 shows the flow to set peripheral clock dividers. See the XMC7000 MCU family architecture reference manual for more details.

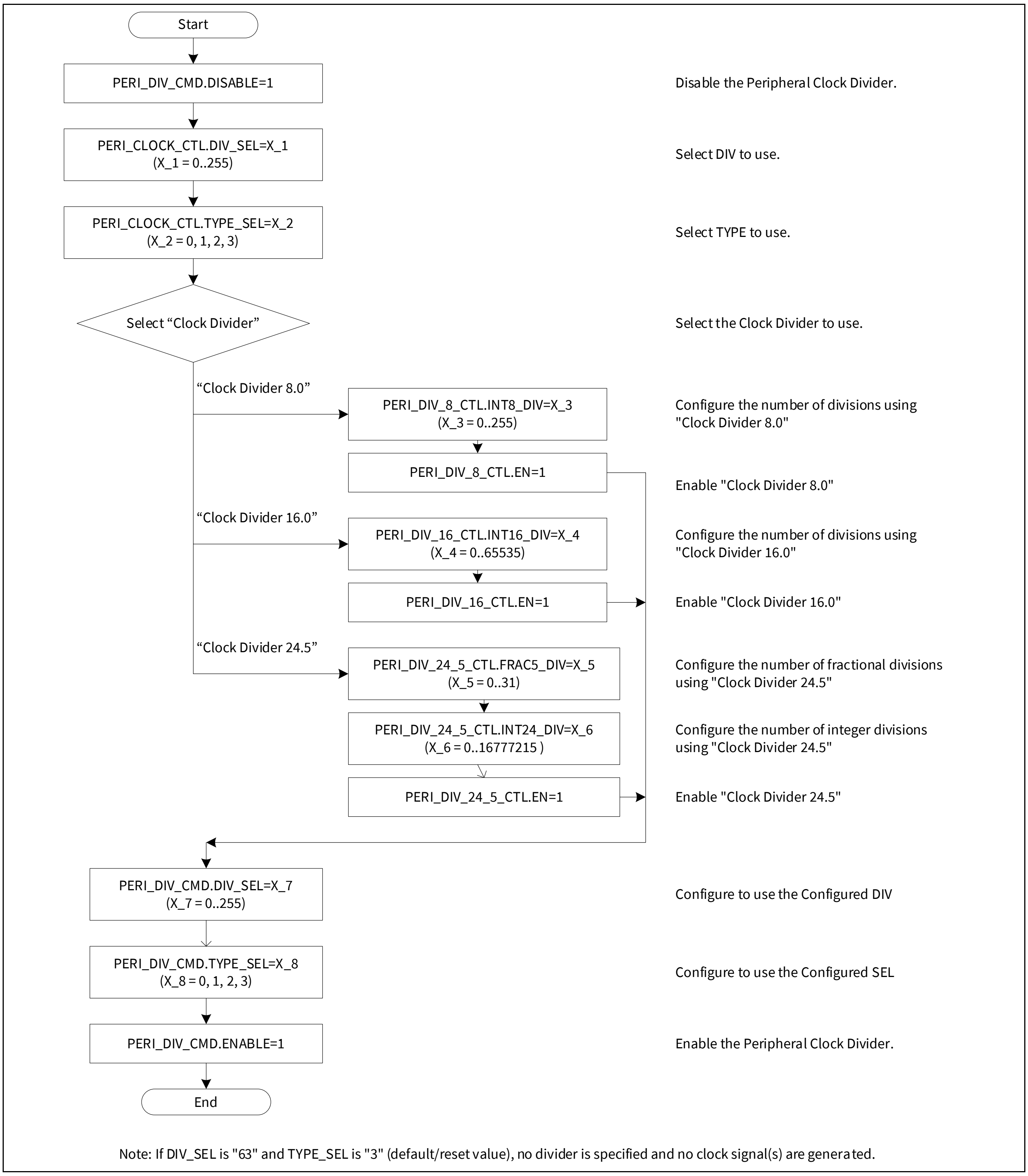

Start
PERI_DIV_CMD.DISABLE=1
Disable the Peripheral Clock Divider.
PERI_CLOCK_CTL.DIV_SEL=X_1
(X_1 = 0..255)
Select DIV to use.
PERI_CLOCK_CTL.TYPE_SEL=X_2
(X_2 = 0, 1, 2, 3)
Select TYPE to use.
Select "Clock Divider"
Select the Clock Divider to use.
"Clock Divider 8.0"
PERI_DIV_8_CTL.INT8_DIV=X_3
(X_3 = 0..255)
Configure the number of divisions using "Clock Divider 8.0"
PERI_DIV_8_CTL.EN=1
Enable "Clock Divider 8.0"
"Clock Divider 16.0"
PERI_DIV_16_CTL.INT16_DIV=X_4
(X_4 = 0..65535)
Configure the number of divisions using "Clock Divider 16.0"
PERI_DIV_16_CTL.EN=1
Enable "Clock Divider 16.0"
"Clock Divider 24.5"
PERI_DIV_24_5_CTL.FRAC5_DIV=X_5
(X_5 = 0..31)
Configure the number of fractional divisions using "Clock Divider 24.5"
PERI_DIV_24_5_CTL.INT24_DIV=X_6
(X_6 = 0..16777215)
Configure the number of integer divisions using "Clock Divider 24.5"
PERI_DIV_24_5_CTL.EN=1
Enable "Clock Divider 24.5"
PERI_DIV_CMD.DIV_SEL=X_7
(X_7 = 0..255)
Configure to use the Configured DIV
PERI_DIV_CMD.TYPE_SEL=X_8
(X_8 = 0, 1, 2, 3)
Configure to use the Configured SEL
PERI_DIV_CMD.ENABLE=1
Enable the Peripheral Clock Divider.
End
Note: If DIV_SEL is "63" and TYPE_SEL is "3" (default/reset value), no divider is specified and no clock signal(s) are generated.

Figure 34 Procedure to generate PCLK

5.9.1 PCLK configuration example

5.9.1.1 Use case

Parameter	Value
Input clock frequency	80 MHz
Output clock frequency	2 MHz
Divider type	Clock divider 16.0
Used divider	Clock divider 16.0#0
Peripheral clock output number	31 (TCPWM0, Group#0, Counter#0)

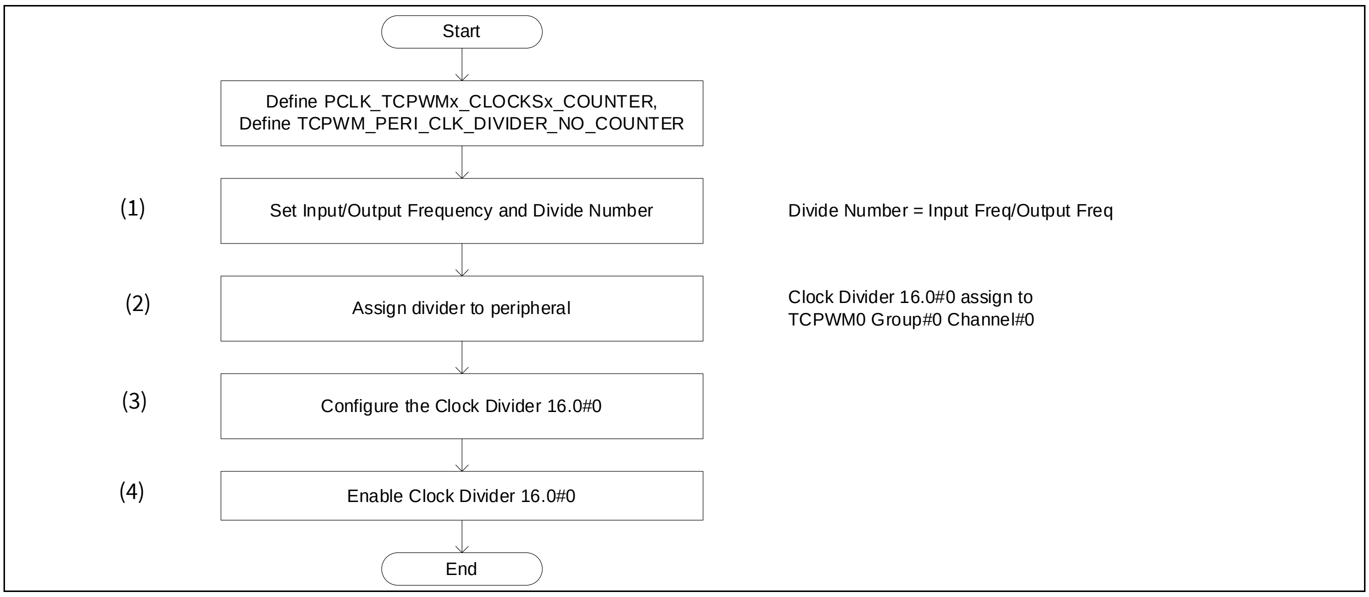
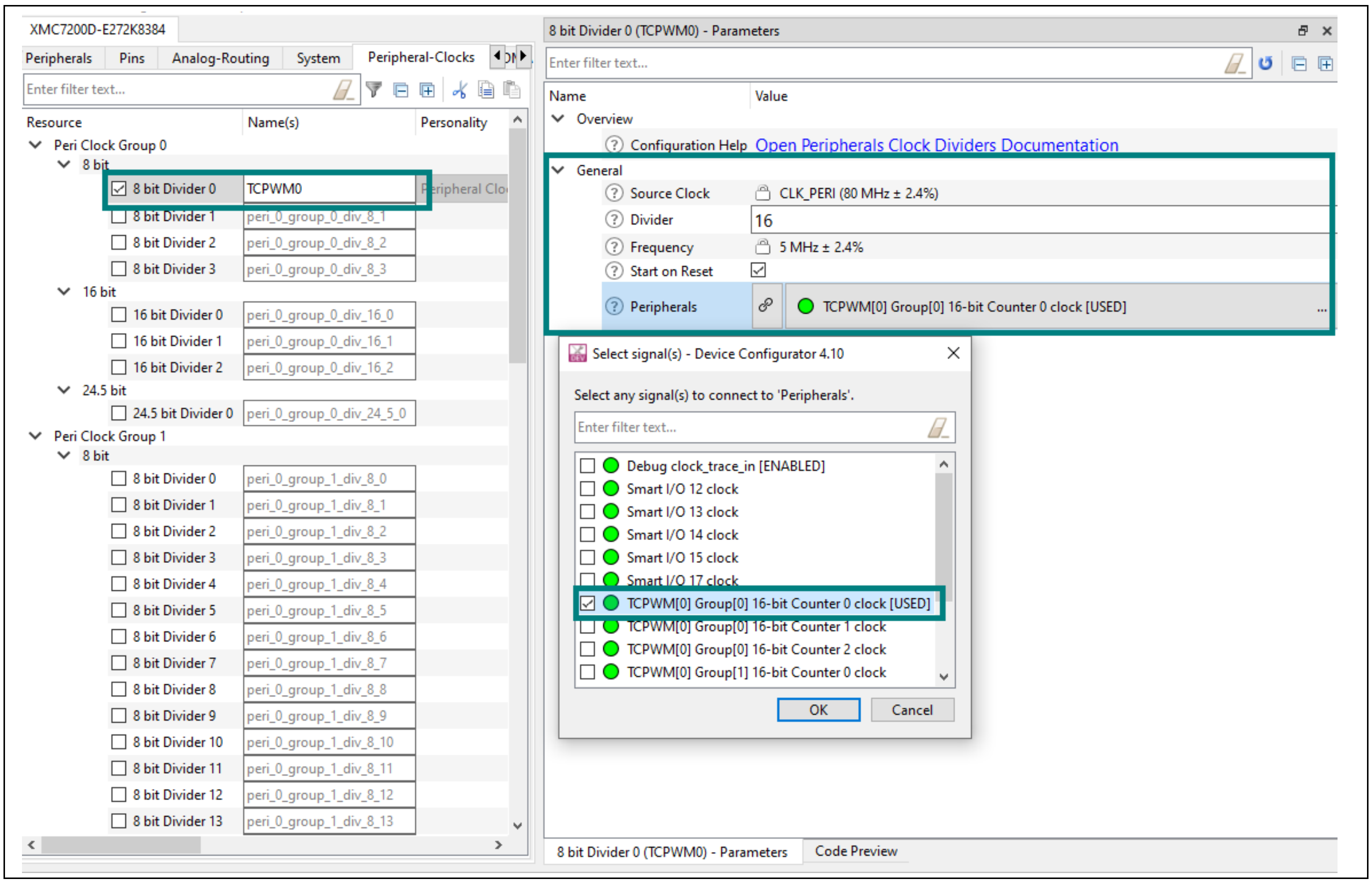

Start
Define PCLK_TCPWMx_CLOCKSx_COUNTER,
Define TCPWM_PERI_CLK_DIVIDER_NO_COUNTER
(1)
Set Input/Output Frequency and Divide Number
Divide Number = Input Freq/Output Freq
(2)
Assign divider to peripheral
Clock Divider 16.0#0 assign to
TCPWM0 Group#0 Channel#0
(3)
Configure the Clock Divider 16.0#0
(4)
Enable Clock Divider 16.0#0
End

Figure 35 Example procedure for setting PCLK

5.9.1.2 Configuration

You can configure these parameters in the **Peripheral-Clocks** tab of the Device Configurator; based on these, ModusToolbox™ software automatically generates the corresponding code.

1. In the Device Configurator, select the **System** tab.
2. Select **FLL**, and in the **FLL – Parameters** pane, under **General**, enter the **Desired Frequency (MHz)** as `80.000` to set the frequency as 80 MHz.
3. Select **CLK_HF0** and select **CLK_PATH0** in Source Clock
4. Select **CLK_PERI**, under **General**, enter '1' in Divider
5. In **Peripheral-Clocks** tab, set the parameters as follows:



XMC7200D-E272K8384

Peripherals | Pins | Analog-Routing | System | Peripheral-Clocks

Resource	Name(s)	Personality
Peri Clock Group 0		
8 bit		
☑ 8 bit Divider 0	TCPWM0	Peripheral Clo
☐ 8 bit Divider 1	peri_0_group_0_div_8_1	
☐ 8 bit Divider 2	peri_0_group_0_div_8_2	
☐ 8 bit Divider 3	peri_0_group_0_div_8_3	
16 bit		
☐ 16 bit Divider 0	peri_0_group_0_div_16_0	
☐ 16 bit Divider 1	peri_0_group_0_div_16_1	
☐ 16 bit Divider 2	peri_0_group_0_div_16_2	
24.5 bit		
☐ 24.5 bit Divider 0	peri_0_group_0_div_24_5_0	
Peri Clock Group 1		
8 bit		
☐ 8 bit Divider 0	peri_0_group_1_div_8_0	
☐ 8 bit Divider 1	peri_0_group_1_div_8_1	
☐ 8 bit Divider 2	peri_0_group_1_div_8_2	
☐ 8 bit Divider 3	peri_0_group_1_div_8_3	
☐ 8 bit Divider 4	peri_0_group_1_div_8_4	
☐ 8 bit Divider 5	peri_0_group_1_div_8_5	
☐ 8 bit Divider 6	peri_0_group_1_div_8_6	
☐ 8 bit Divider 7	peri_0_group_1_div_8_7	
☐ 8 bit Divider 8	peri_0_group_1_div_8_8	
☐ 8 bit Divider 9	peri_0_group_1_div_8_9	
☐ 8 bit Divider 10	peri_0_group_1_div_8_10	
☐ 8 bit Divider 11	peri_0_group_1_div_8_11	
☐ 8 bit Divider 12	peri_0_group_1_div_8_12	
☐ 8 bit Divider 13	peri_0_group_1_div_8_13	

8 bit Divider 0 (TCPWM0) - Parameters

Name	Value
Overview	
Configuration Help	Open Peripherals Clock Dividers Documentation
General	
Source Clock	CLK_PERI (80 MHz ± 2.4%)
Divider	16
Frequency	5 MHz ± 2.4%
Start on Reset	☑
Peripherals	● TCPWM[0] Group[0] 16-bit Counter 0 clock [USED]

Select signal(s) - Device Configurator 4.10

Select any signal(s) to connect to 'Peripherals'.

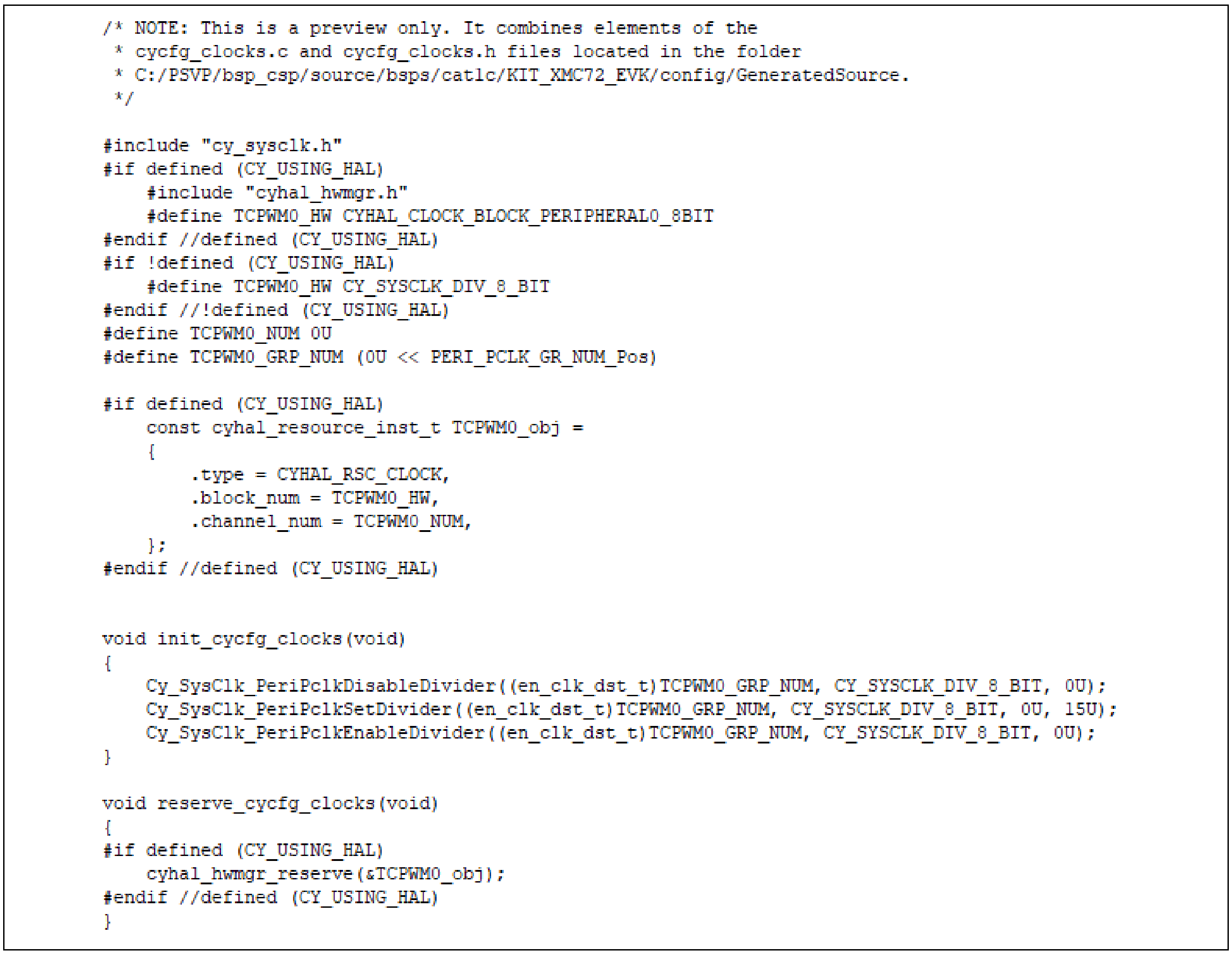
- ☐ ● Debug clock_trace_in [ENABLED]
- ☐ ● Smart I/O 12 clock
- ☐ ● Smart I/O 13 clock
- ☐ ● Smart I/O 14 clock
- ☐ ● Smart I/O 15 clock
- ☐ ● Smart I/O 17 clock
- ☑ ● TCPWM[0] Group[0] 16-bit Counter 0 clock [USED]
- ☐ ● TCPWM[0] Group[0] 16-bit Counter 1 clock
- ☐ ● TCPWM[0] Group[0] 16-bit Counter 2 clock
- ☐ ● TCPWM[0] Group[1] 16-bit Counter 0 clock

OK | Cancel

8 bit Divider 0 (TCPWM0) - Parameters | Code Preview

Figure 36 Setting parameters for PCLK configuration

5.9.2 Code preview



```
/* NOTE: This is a preview only. It combines elements of the
 * cycfg_clocks.c and cycfg_clocks.h files located in the folder
 * C:/PSVP/bsp_csp/source/bsps/cat1c/KIT_XMC72_EVK/config/GeneratedSource.
 */

#include "cy_sysclk.h"
#if defined (CY_USING_HAL)
    #include "cyhal_hwmgr.h"
    #define TCPWM0_HW CYHAL_CLOCK_BLOCK_PERIPHERAL0_8BIT
#endif //defined (CY_USING_HAL)
#if !defined (CY_USING_HAL)
    #define TCPWM0_HW CY_SYSCLK_DIV_8_BIT
#endif //!defined (CY_USING_HAL)
#define TCPWM0_NUM 0U
#define TCPWM0_GRP_NUM (0U << PERI_PCLK_GR_NUM_Pos)

#if defined (CY_USING_HAL)
    const cyhal_resource_inst_t TCPWM0_obj =
    {
        .type = CYHAL_RSC_CLOCK,
        .block_num = TCPWM0_HW,
        .channel_num = TCPWM0_NUM,
    };
#endif //defined (CY_USING_HAL)


void init_cycfg_clocks(void)
{
    Cy_SysClk_PeriPclkDisableDivider((en_clk_dst_t)TCPWM0_GRP_NUM, CY_SYSCLK_DIV_8_BIT, 0U);
    Cy_SysClk_PeriPclkSetDivider((en_clk_dst_t)TCPWM0_GRP_NUM, CY_SYSCLK_DIV_8_BIT, 0U, 15U);
    Cy_SysClk_PeriPclkEnableDivider((en_clk_dst_t)TCPWM0_GRP_NUM, CY_SYSCLK_DIV_8_BIT, 0U);
}

void reserve_cycfg_clocks(void)
{
#if defined (CY_USING_HAL)
    cyhal_hwmgr_reserve(&TCPWM0_obj);
#endif //defined (CY_USING_HAL)
}
```

Figure 37 Code preview for PCLK configuration

5.10 Configuring the ECO Prescaler

5.10.1 Overview

The ECO_Prescaler divides the ECO and creates a clock that can be used with CLK_LF. The division function has a 10-bit integer divider and an 8-bit fractional divider.

Figure 38 shows the steps to enable ECO_Prescaler. For details on the ECO_Prescaler, see XMC7000 MCU family architecture reference manual.

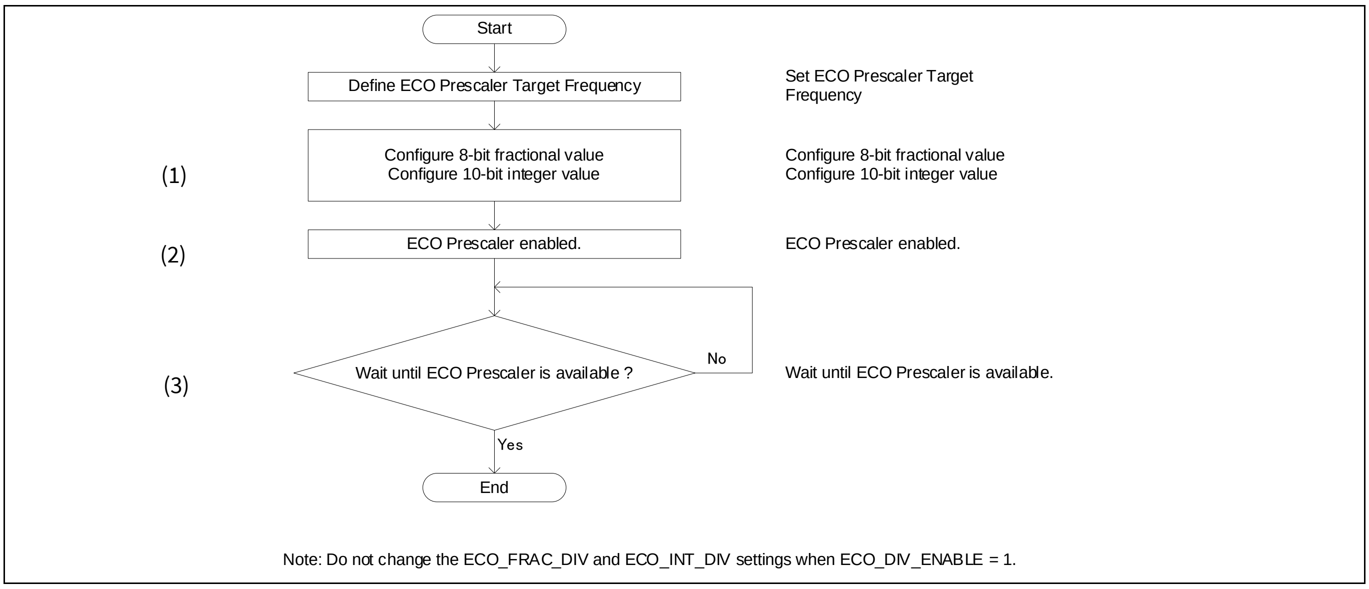

Start
Define ECO Prescaler Target Frequency
Set ECO Prescaler Target Frequency
(1)
Configure 8-bit fractional value
Configure 10-bit integer value
Configure 8-bit fractional value
Configure 10-bit integer value
(2)
ECO Prescaler enabled.
ECO Prescaler enabled.
(3)
Wait until ECO Prescaler is available ?
No
Wait until ECO Prescaler is available.
Yes
End
Note: Do not change the ECO_FRAC_DIV and ECO_INT_DIV settings when ECO_DIV_ENABLE = 1.

Figure 38 Enabling the ECO_Prescaler

Figure 39 shows the steps to disable the ECO_Prescalar. For details on the ECO_Prescaler, see XMC7000 MCU family architecture reference manual.

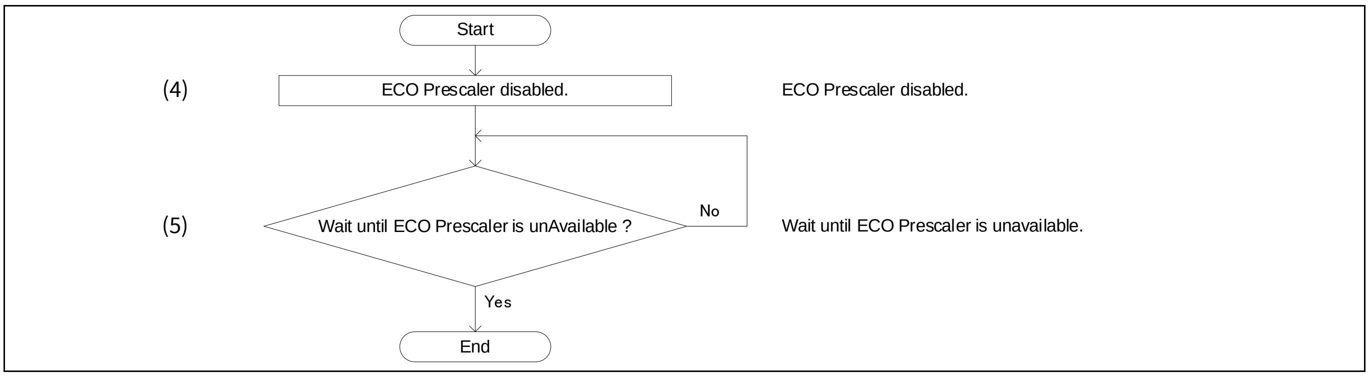

Start
(4)
ECO Prescaler disabled.
ECO Prescaler disabled.
(5)
Wait until ECO Prescaler is unAvailable ?
No
Wait until ECO Prescaler is unavailable.
Yes
End

Figure 39 Disabling the ECO_Prescaler

6 Supplementary information

6.1 Input clocks in peripheral functions

You can view the clock inputs for each peripheral function in the **System** tab of the Device Configurator.

For detailed values of PCLK, see the "Peripheral Clocks" section of the datasheet.

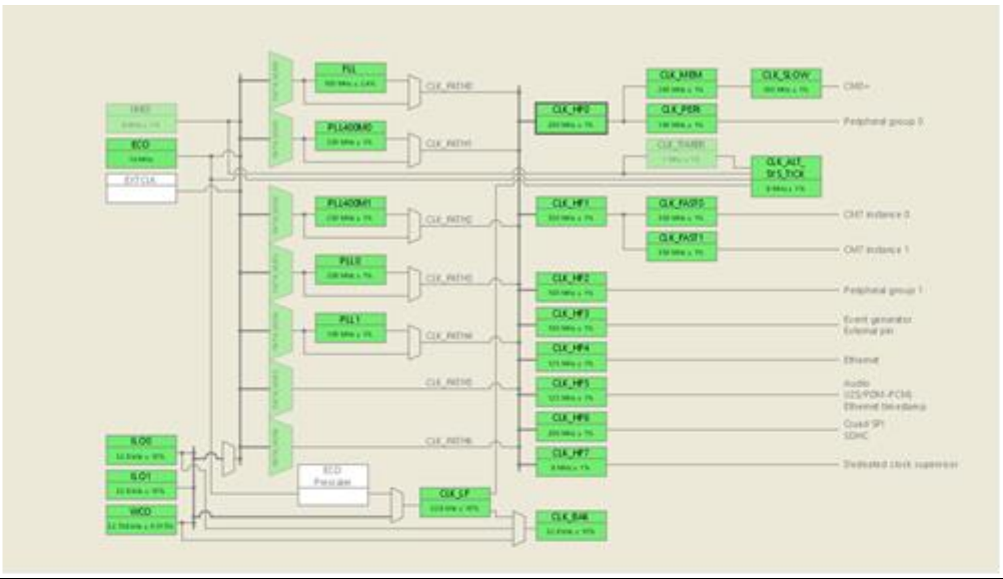


Figure 40 Clock inputs

6.2 Clock calibration counter function

6.2.1 Overview

The clock calibration counter has two counters that can be used to compare the frequency of two clock sources. All clock sources are available as a source for these two clocks.

1. Calibration Counter1 counts clock pulses from Calibration Clock1 (the high-accuracy clock used as the reference clock). Counter1 counts in decreasing order.
2. Calibration Counter2 counts clock pulses from Calibration Clock2 (measurement clock). This counter counts in increasing order.
3. When Calibration Counter1 reaches 0, Calibration Counter2 stops counting, and its value can be read.
4. The frequency of Calibration Counter2 can be obtained by using the value and the following equation:

$$CalibrationClock2 = \frac{Counter2value}{Counter1value} \times CalibrationClock1$$

Figure 41 shows an example of the clock calibration counter function when ILO0 and ECO are used. ILO0 and ECO must be enabled. See ILO0 and ECO for 3.4 Setting ILO0/ILO1 and 3.1 ECO.

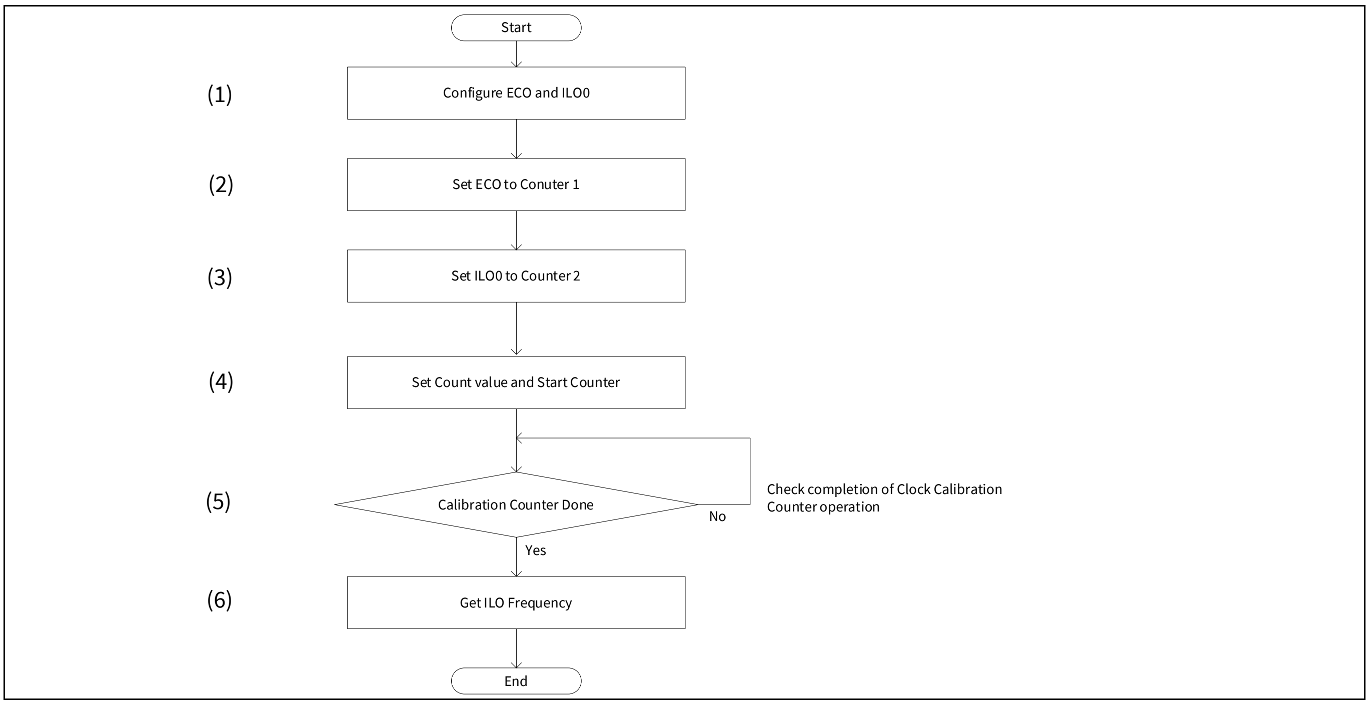


Figure 41 **Example of clock calibration counter with ILO0 and ECO**

6.2.2 Use case

Parameter	Value
Measurement clock	ILO0 clock frequency 32.768 kHz
Reference clock	ECO clock frequency 16 MHz
Reference clock count value	40000ul

6.2.3 Configuration

Table 2 lists the parameters and Table 3 lists the functions of the configuration part in ModusToolbox™ CAT1 Peripheral Driver Library for Clock Calibration Counter with ILO0 and ECO settings.

Table 2 **Configuration parameters for Clock calibration counter with ILO0 and ECO**

Parameters	Description	Value
`ILO_0`	Define the ILO_0 configuration parameter	0ul
`ILO_1`	Define the ILO_1 configuration parameter	1ul
`ILONo`	Define the measurement clock	ILO_0
`clockMeasuredInfo[].name`	Measurement clock	CY_SYSCLK_MEAS_CLK_ILO0 = 1ul
`clockMeasuredInfo[].measuredFreq`	Store the measurement clock frequency	–
`counter1`	Reference clock count value	40000ul
`CLK_FREQ_ECO`	ECO clock frequency	16000000ul (16 MHz)

Table 3 Functions for clock calibration counter with ILO0 and ECO

Functions	Description	Value
`GetILOClockFreq()`	Get the ILO 0 Frequency	–

`Cy_SysClk_StartClkMeasurementCounters(clk1, count1, clk2)`	Set and start calibration
	Clk1: Reference clock
	Count1: Measurement period
	Clk2: Measurement clock
		[Set the counter]
		clk1 = CY_SYSCLK_MEAS_CLK_ECO = 0x101ul
		count1 = counter1
		clk2 = clockMeasuredInfo[].name

`Cy_SysClk_ClkMeasurementCountersDone()`	Check if counter measurement is done	–

`Cy_SysClk_ClkMeasurementCountersGetFreq(MesauredFreq, refClkFreq)`	Get the measurement clock frequency
	MesauredFreq: Stored measurement clock frequency
	refClkFreq: Reference clock frequency
		
		MesauredFreq = clockMeasuredInfo[].measuredFreq
		refClkFreq = CLK_FREQ_ECO

6.2.3.1 Sample code for initial configuration of clock calibration counter with ILO0 and ECO

The following is the sample code:

```
    /* Scenario: ILO frequency may be drifting. Measure the ILO frequency
using
                 the clock measurement counters, with the IMO as the
reference. */
    #define  ECO_FREQ        16000000UL   /* 16 MHz ECO */
    #define  ILO_NOMINAL     32768UL     /* 32.768 kHz ILO */
    /* Start the ILO clock measurement using the ECO */
    (void)Cy_SysClk_StartClkMeasurementCounters(CY_SYSCLK_MEAS_CLK_ILO,
/* Counter 1 clock  = ILO */
                                                0x7FUL,
/* Counter 1 period = 128 */
                                                CY_SYSCLK_MEAS_CLK_ECO);
/* Counter 2 clock  = IMO */
    /* Wait for counter 1 to reach 0 */
    while(!Cy_SysClk_ClkMeasurementCountersDone()){}
    /* Measure clock 1 with the ECO clock cycles (counter 2) */
    uint32_t measuredFreq = Cy_SysClk_ClkMeasurementCountersGetFreq(false,
ECO_FREQ);
    if(measuredFreq != ILO_NOMINAL)
    {
        /* Take appropriate action such as trimming the ILO or changing the
LFCLK source */
    }
```

6.3 CSV diagram and relationship of monitored clock and reference clock

Figure 42 shows the clock diagram with monitored clock and reference clock for CSV. Table 4 shows the relationship between monitored clock and reference clock.

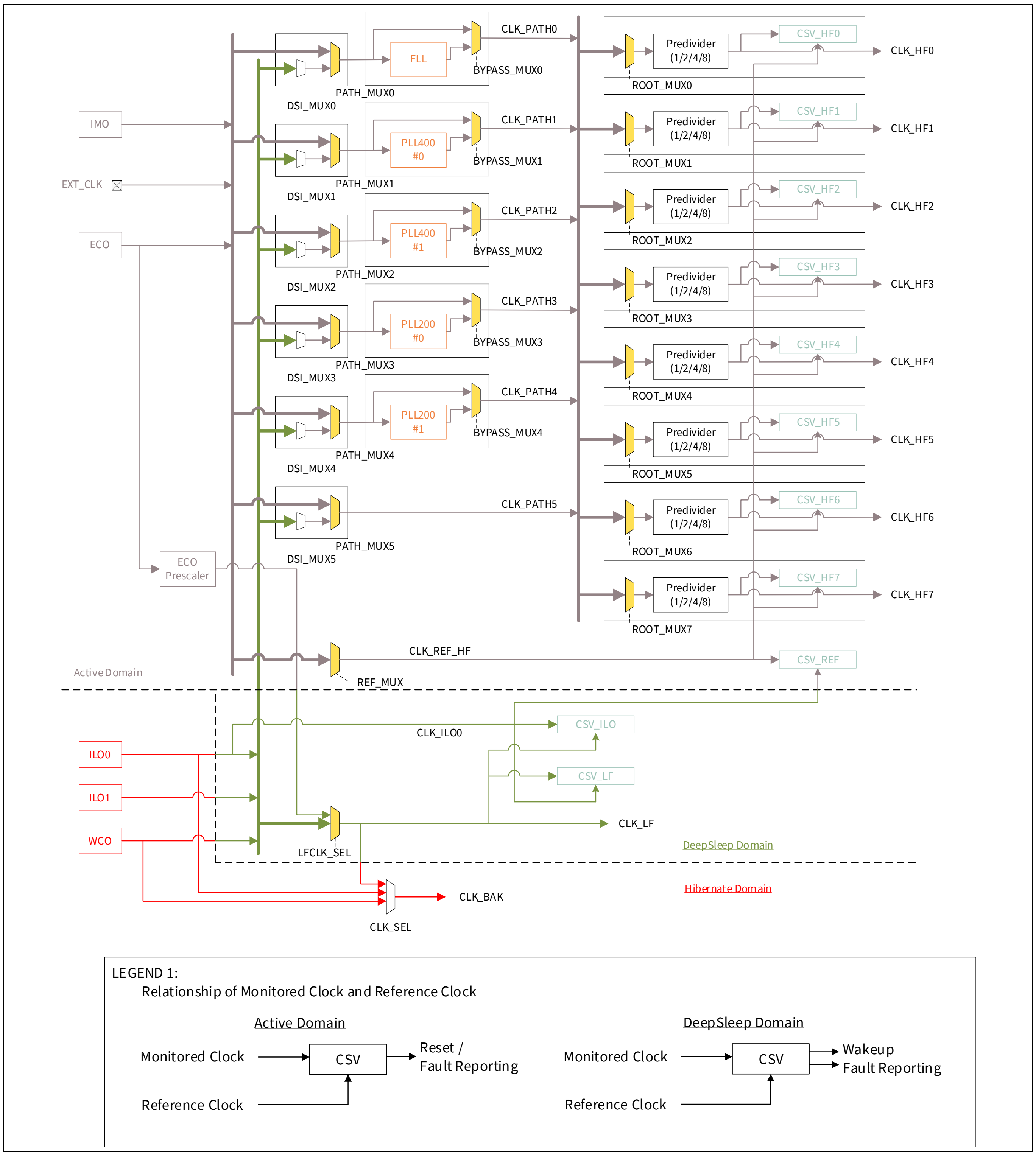


Figure 42 CSV diagram

Table 4 Monitored clock and reference clock

CSV components	Monitor clock	Reference clock	Notes
CSV_HF0	CSV_HF0	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_HF1	CLK_HF1	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_HF2	CLK_HF2	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_HF3	CLK_HF3	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_HF4	CLK_HF4	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_HF5	CLK_HF5	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_HF6	CLK_HF6	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_HF7	CLK_HF7	CLK_REF_HF	CLK_REF_HF is selected from CLK_IMO, EXT_CLK, or CLK_ECO.
CSV_REF	CLK_REF_HF	ILO0 (CLK_ILO0)	–
CSV_ILO	ILO0 (CLK_ILO0)	CLK_LF	CLK_LF is selected from WCO, ILO1, or ECO Prescaler.
CSV_LF	CLK_LF	ILO0 (CLK_ILO0)	–

Glossary

Table 5 Glossary

Terms	Description
AUDIOSS	Audio Subsystem. See the "Audio Subsystem" chapter of the XMC7000 MCU family architecture reference manual for details.
CAN FD	CAN FD is the CAN with Flexible Data rate, and CAN is the Controller Area Network. See the "CAN FD Controller" chapter of the XMC7000 MCU family architecture reference manual for details.
CLK_FAST_0	Fast lock. CLK_FAST is used for the CM7 and CPUSS Fast Infrastructure.
CLK_FAST_1	Fast clock. CLK_FAST is used for the CM7 and CPUSS Fast Infrastructure.
CLK_HF	High-frequency clock. Both CLK_FAST and CLK_SLOW are derived from CLK_HF. CLK_HF, CLK_FAST and CLK_SLOW are synchronous to each other.
CLK_GR	Group clock. CLK_GR is the clock input to peripheral functions.
CLK_MEM	Memory clock. CLK_MEM clocks the CPUSS fast infrastructure.
CLK_PERI	Peripheral clock. CLK_PERI is the clock source for CLK_SLOW, CLK_GR and peripheral clock divider.
CLK_SLOW	Slow clock. CLK_FAST is used for the CM0+ and CPUSS slow Infrastructure.
Clock calibration counter	Clock calibration counter has a function to calibrate the clock using two clocks.
CSV	Clock supervision
ECO	External crystal oscillator
EXT_CLK	External clock
FLL	Frequency-locked loop
ILO	Internal low-speed oscillator
IMO	Internal main oscillator
Peripheral Clock Divider	Peripheral clock divider derives a clock for the use of each peripheral function.
PLL200	Phase-locked loop. This PLL is not implemented for SSCG and fractional operation.
PLL400	Phase-locked Loop. This PLL is implemented for SSCG and fractional operation.
SAR ADC	Successive approximation register analog-to-digital converter. See the "SAR ADC" chapter of the XMC7000 MCU family architecture reference manual for details.
SCB	Serial communication block. See the "Serial Communication Block (SCB)" chapter of the XMC7000 MCU family manual for details.
SDHC	Secure Digital High-Capacity Host Controller. See the "SDHC Host Controller" chapter of the XMC7000 MCU family architecture reference manual for details.
SMIF	Serial Memory Interface. See the "Serial Memory Interface" chapter of the XMC7000 MCU family architecture reference manual for details.
TCPWM	Timer, counter, and pulse width modulator. See the "Timer, Counter, and PWM" chapter of the XMC7000 MCU family architecture reference manual for details.
WCO	Watch crystal oscillator

References

[1] Datasheet

- XMC7100 XMC7000 microcontroller 32-bit Arm® Cortex®-M7
- XMC7200 XMC7000 microcontroller 32-bit Arm® Cortex®-M7

[2] Architecture reference manual

- XMC7000 MCU family architecture reference manual

[3] Registers reference manual

Note: Contact Technical support to obtain these documents.

- 002-33817: XMC7100 registers reference manual
- 002-33812: XMC7200 registers reference manual

Revision history

Document revision	Date	Description of changes
**	2021-12-13	Initial release
*A	2022-05-06	Updated 5.9 Configuring PCLK Updated 5.10 Configuring the ECO Prescaler Updated Supplementary information
*B	2023-01-09	Updated 3.1.1 Use case
*C	2023-11-08	Updated 2.2 Clock resources Updated 2.4 Basic clock system settings Updated 3.1.1 Use case and 3.2.1 Overview Updated 4.1.2 Use case and 4.2.2 Use case Updated 5.9.1 PCLK configuration example Added Associated part family Updated the References section

Edition 2023-11-08

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-34253 Rev. *C