

Clock configuration setup in TRAVEO™ T2G Body High family

About this document

Scope and purpose

This application note describes how to set up various clock sources in TRAVEO™ T2G Body High family MCUs and provides examples including configuring phase-locked loop (PLL)/frequency locked loop (FLL) and calibrating the internal low-speed oscillator.

Intended audience

This document is intended for users who use the clock configuration setup in TRAVEO™ T2G Body High family.

Associated part family

TRAVEO™ T2G Body High family

Table of contents

Table of contents

	About this document	1
	Table of contents	2
1	Introduction	4
2	Clock system for TRAVEO™ T2G family MCUs	5
2.1	Overview of clock system	5
2.2	Clock resources	5
2.3	Clock system function	5
2.4	Basic clock system settings	11
3	Configuration of the clock resources	12
3.1	Setting the ECO	12
3.1.1	Use case	12
3.1.2	Sample code for initial configuration of ECO settings	14
3.2	Setting WCO	24
3.2.1	Operation overview	24
3.2.2	Configuration	25
3.2.3	Sample code	26
3.3	Setting IMO	28
3.4	Setting ILO0/ILO1	28
4	Configuration of FLL and PLL	30
4.1	Setting FLL	30
4.1.1	Operation overview	30
4.1.2	Use case	31
4.1.3	Configuration	31
4.1.4	Sample code	31
4.2	Setting PLL	39
4.2.1	Operation overview	39
4.2.2	Use case	40
4.2.3	Configuration	40
4.2.4	Sample code	45
5	Configuration of the internal clock	63
5.1	Configuring the CLK_PATHx	63
5.2	Configuring the CLK_HFx	64
5.3	Configuring the CLK_LF	65
5.4	Configuring the CLK_FAST_0/CLK_FAST_1	66
5.5	Configuring the CLK_MEM	66
5.6	Configuring the CLK_PERI	66
5.7	Configuring the CLK_SLOW	66
5.8	Configuring the CLK_GR	66

Table of contents

5.9	Configuring the PCLK	66
5.9.1	Example of PCLK setting	68
5.9.1.1	Use case	68
5.9.1.2	Configuration	68
5.9.2	Sample code (Example of the TCPWM timer)	69
5.10	Setting ECO_Prescaler	72
5.10.1	Operation overview	72
5.10.2	Use case	72
5.10.3	Configuration	73
5.10.4	Sample code	73
6	Supplementary information	78
6.1	Input clocks in peripheral functions	78
6.2	Use case of clock calibration counter function	80
6.2.1	How to use the clock calibration counter	80
6.2.1.1	Operation overview	80
6.2.1.2	Use case	81
6.2.1.3	Configuration	81
6.2.1.4	Sample code for initial configuration of clock calibration counter with ILO0 and ECO settings	82
6.2.2	ILO0 calibration using clock calibration counter function	86
6.2.2.1	Operation overview	86
6.2.2.2	Configuration	87
6.2.2.3	Sample code for initial configuration of ILO0 calibration using clock calibration counter settings	88
6.3	CSV diagram and relationship of monitored clock and reference clock	90
	Glossary	93
	References	95
	Revision history	96
	Disclaimer	97

1 Introduction

1 Introduction

The TRAVEO™ T2G family MCUs, intended for automotive systems such as body control units, are 32-bit automotive microcontrollers based on the Arm® Cortex®-M7 processor with floating-point unit (single and dual precision) and manufactured in advanced 40 nm process technology. These products enable a secure computing platform and incorporate Infineon low-power flash memory along with multiple high-performance analog and digital functions.

The TRAVEO™ T2G clock system supports high- 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). TRAVEO™ T2G supports phase-locked loop (PLL) and frequency locked loop (FLL) to generate clocks that operate the internal circuit at a high speed.

The TRAVEO™ T2G also supports a function to monitor clock operation and to measure the clock difference of each clock with reference to a known clock.

To learn more about the functionality described and terminologies used in this application note, see the Clocking system chapter in the [architecture reference manual](#).

In this document, TRAVEO™ T2G family MCUs refers to Body High family.

2 Clock system for TRAVEO™ T2G family MCUs

2 Clock system for TRAVEO™ T2G family MCUs

2.1 Overview of clock system

The clock system in this series of MCUs are divided into two blocks. One block selects the clock resources (such as external oscillation and internal oscillation) and multiplies the clock (using FLL and PLL). The other block 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 structure.

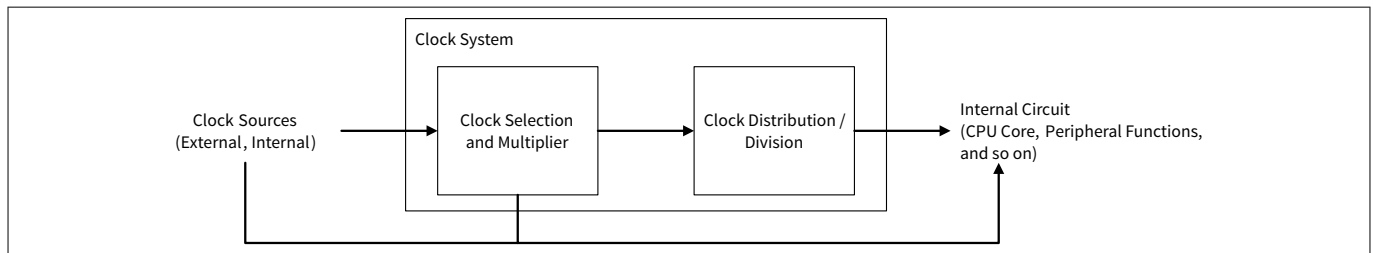


Figure 1 Overview of the clock system structure

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 of these clocks are enabled by default):
 - Internal main oscillator (IMO) - This is a built-in clock with a frequency of 8 MHz (TYP)
 - Internal low-speed oscillator 0 (ILO0) - This is a built-in clock with a frequency of 32.768 kHz (TYP)
 - Internal low-speed oscillator 1 (ILO1) - This clock has the same function as ILO0, but ILO1 can monitor the clock of ILO0
- External clock sources (All of these clocks are disabled by default):
 - External crystal oscillator (ECO) - This clock uses an external oscillator whose input frequency range is in between 8 MHz and 33.34 MHz
 - Watch crystal oscillator (WCO) - This also 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 about functions such as IMO, PLL, and numerical values such as frequency, see the TRAVEO™ T2G [architecture reference manual](#) and the [datasheet](#).

2.3 Clock system function

This section explains the functions of the clock system.

Figure 2 shows the details of the clock selection and the 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 generator (SSCG), fractional operation (PLL200#x), PLL with SSCG, and fractional operation (PLL400#x).

2 Clock system for TRAVEO™ T2G family MCUs

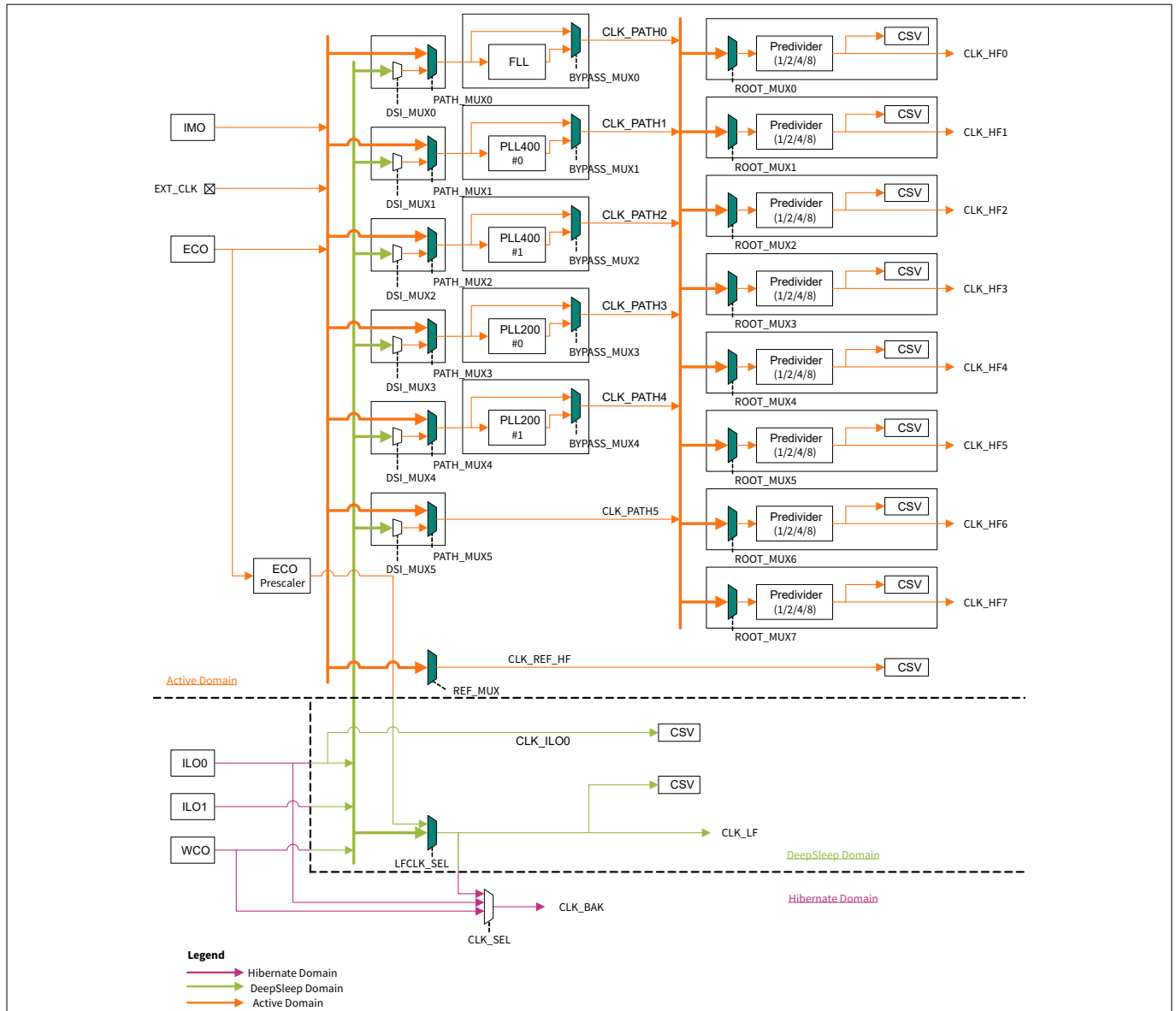


Figure 2 Block diagram

Active domain	Region of operation in only Active power mode
DeepSleep domain	Region of operation in Active and DeepSleep power modes
Hibernate domain	Region of operation in 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 high-frequency clock

2 Clock system for TRAVEO™ T2G family MCUs

PLL	Generates high-frequency clock. There are two kinds of PLL—PLL200 and PLL400. PLL200 is without SSCG and fractional operation and PLL400 is 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 FLL output or clock input to FLL
ROOT_MUX	Selects the clock source of CLK_HF _x . The clocks that can be selected are CLK_PATHs 0 through 5
Predivider	The 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 RTC
CLK_BAK	Mainly input to RTC
CSV	Clock supervision. to monitor the clock operation

[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 [architecture reference manual](#) and [datasheet](#).

2 Clock system for TRAVEO™ T2G family MCUs

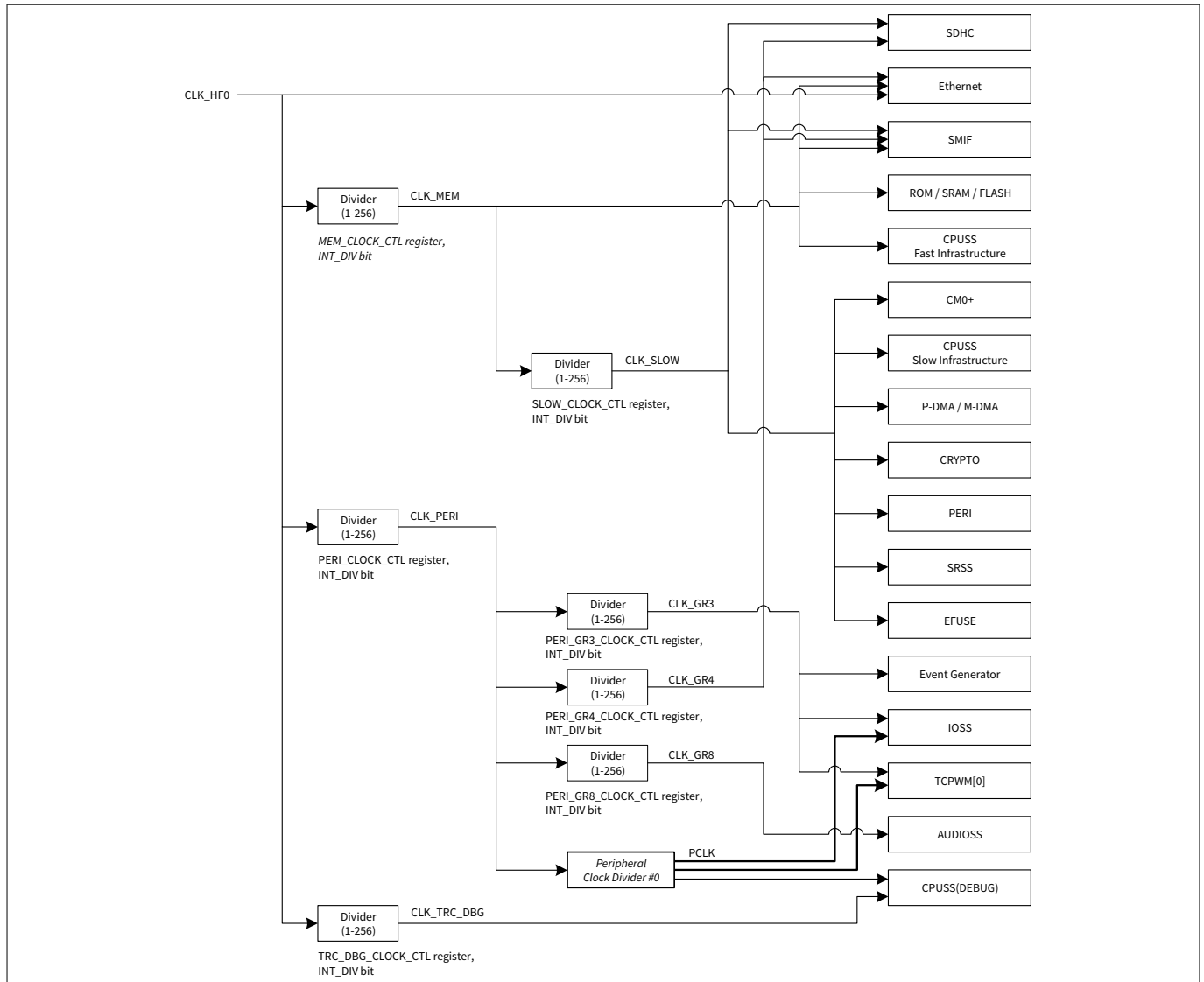


Figure 3 Block diagram for CLK_HF0

CLK_MEM	Clock input to 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 clock gater. CLK_GR has six groups
PCLK	Peripheral clock used in peripheral functions. PCLK can be configured with each channel of IPs independently and select one divider to generate PCLK
CLK_TRC_DBG	Clock input to CPUS(DEBUG)
Divider	Divides 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 (for example, the serial communication block (SCB), the timer, counter, and PWM (TCPWM), etc.) and its respective channel. The clocks are controlled by their respective dividers.

2 Clock system for TRAVEO™ T2G family MCUs

This peripheral clock divider #0 has many peripheral clock dividers to generate the peripheral clock (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 the dividers already in use cannot be used for other peripherals or channels.

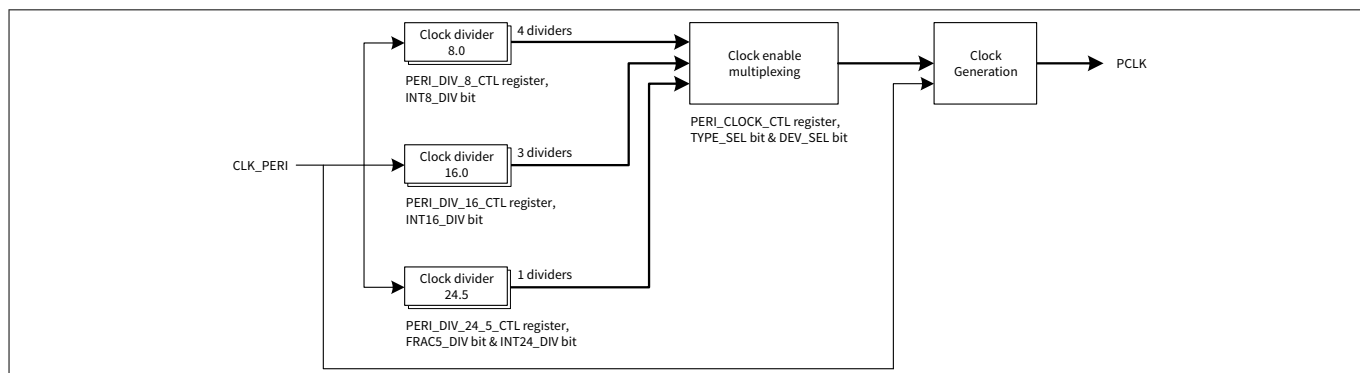


Figure 4 Block diagram for the peripheral clock divider #0

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, CLK_FAST_1, CLK_FAST_2*, and CLK_FAST_3*. * Only CYT6B series
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_x (x = 1 for CYT4B series, x=3 for CYT6B series respectively).

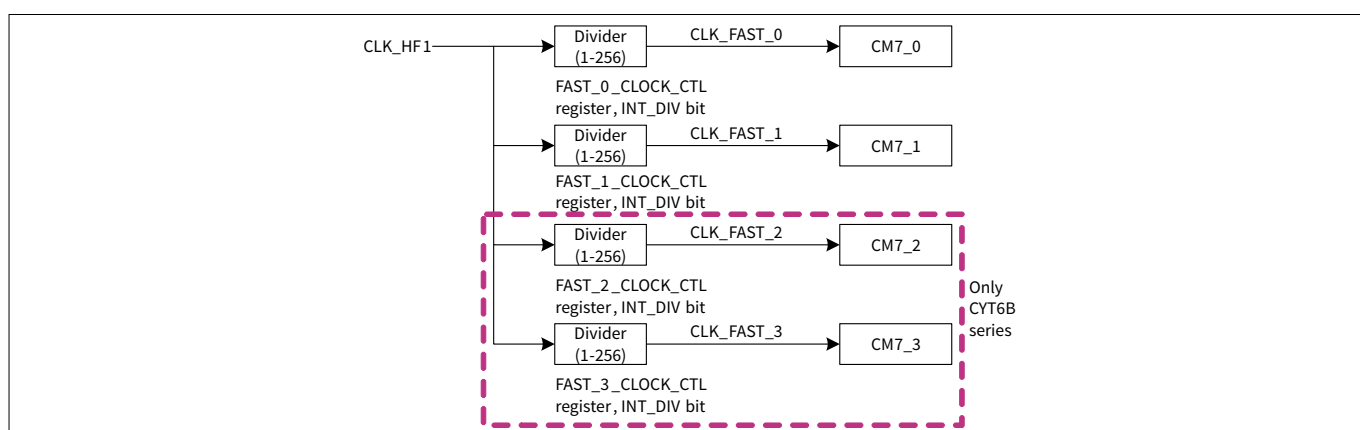


Figure 5 Block diagram for CLK_HF1

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.

2 Clock system for TRAVEO™ T2G family MCUs

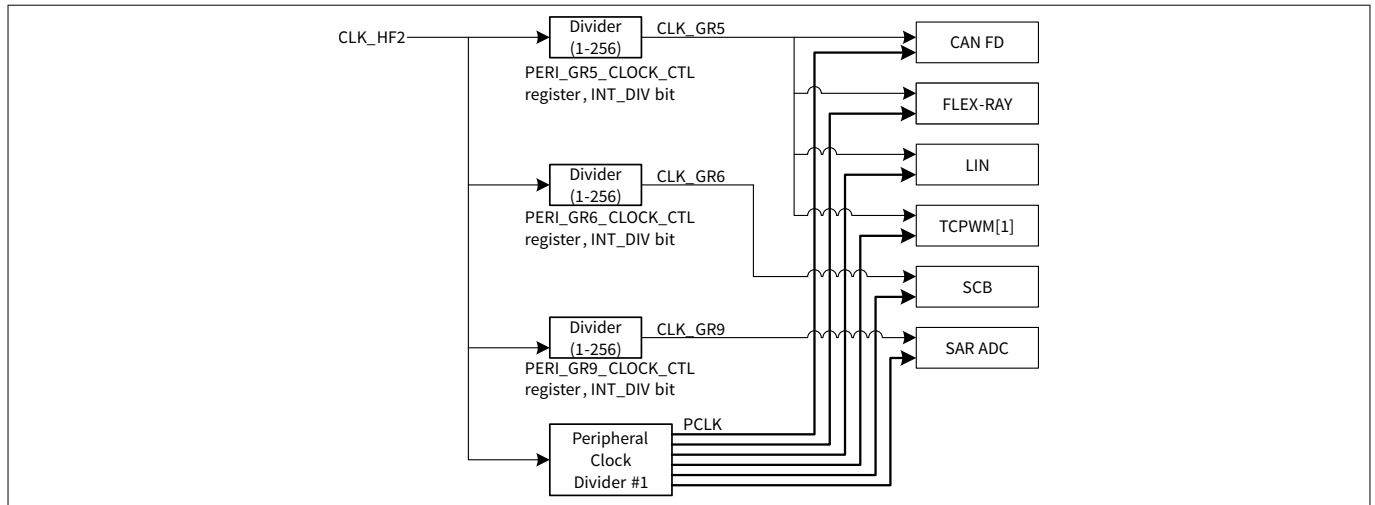


Figure 6 Block diagram for CLK_HF2

Figure 7 shows details of the peripheral clock divider #1.

The 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.

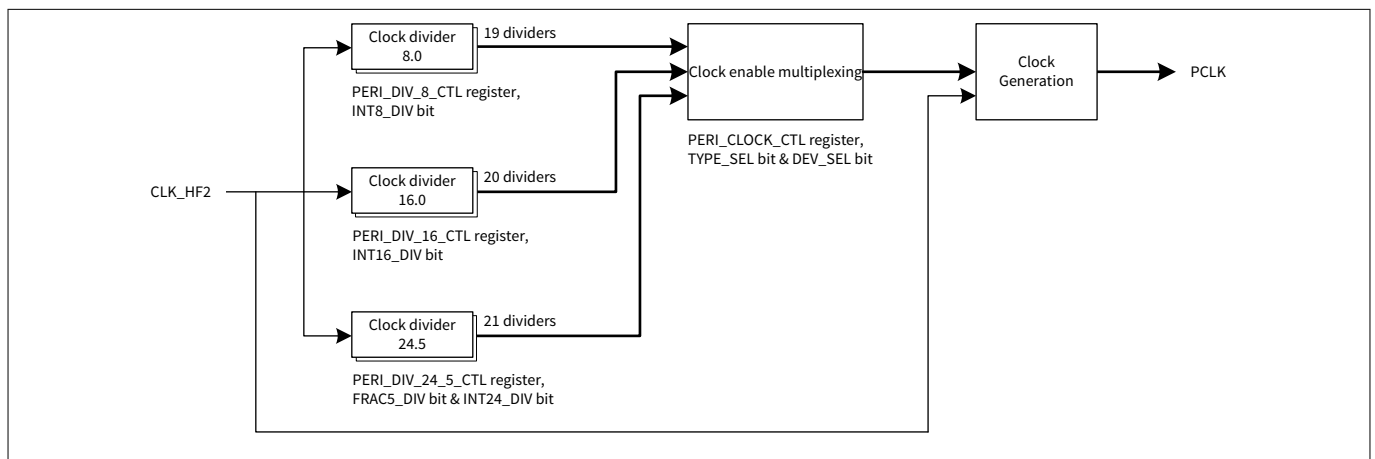


Figure 7 Block diagram for the peripheral clock divider #1

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 8 shows the distribution of CLK_HF3, CLK_HF4, CLK_HF5, and CLK_HF6. For more details on these functions in the [Figure 8](#), see the [architecture reference manual](#).

2 Clock system for TRAVEO™ T2G family MCUs

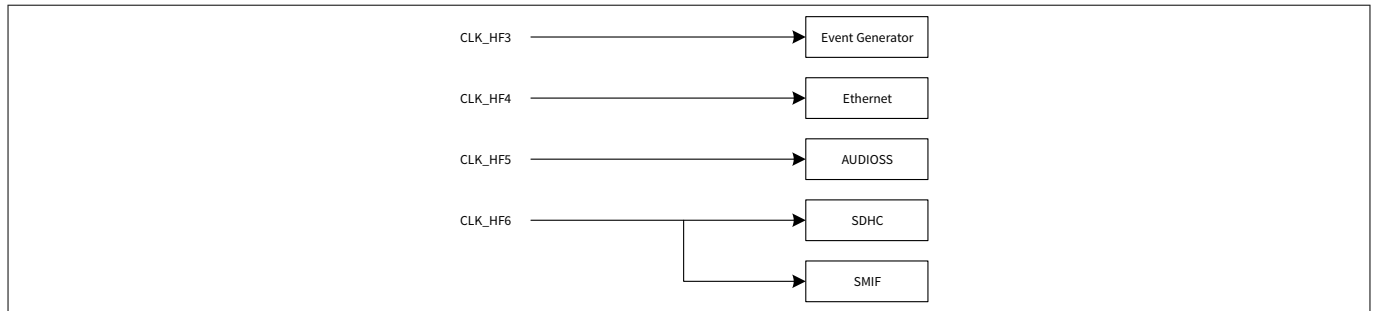


Figure 8 Block diagram for CLK_HF3, CLK_HF4, CLK_HF5, and CLK_HF6

CLK_HF7 is dedicated to CSV. See the [architecture reference manual](#) for CSV description.

2.4 Basic clock system settings

This section describes how to configure the clock system based on a use case using the sample driver library (SDL) provided by Infineon. The code snippets in this application note are part of SDL. For more information, see [References](#).

SDL has a configuration part and a driver part. The configuration part configures the parameter values for the desired operation.

The driver part configures each register based on the parameter values in the configuration part.

You can configure the configuration part according to your system.

3 Configuration of the clock resources

3 Configuration of the clock resources

This section explains the function of the clock.

3.1 Setting the ECO

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 crystal unit and ceramic resonator. The method to determine the parameters differs between crystal unit and ceramic resonator. For more information, see ["Setting ECO parameters" section in the TRAVEO™ T2G user guide](#).

Figure 9 shows ECO setting steps.

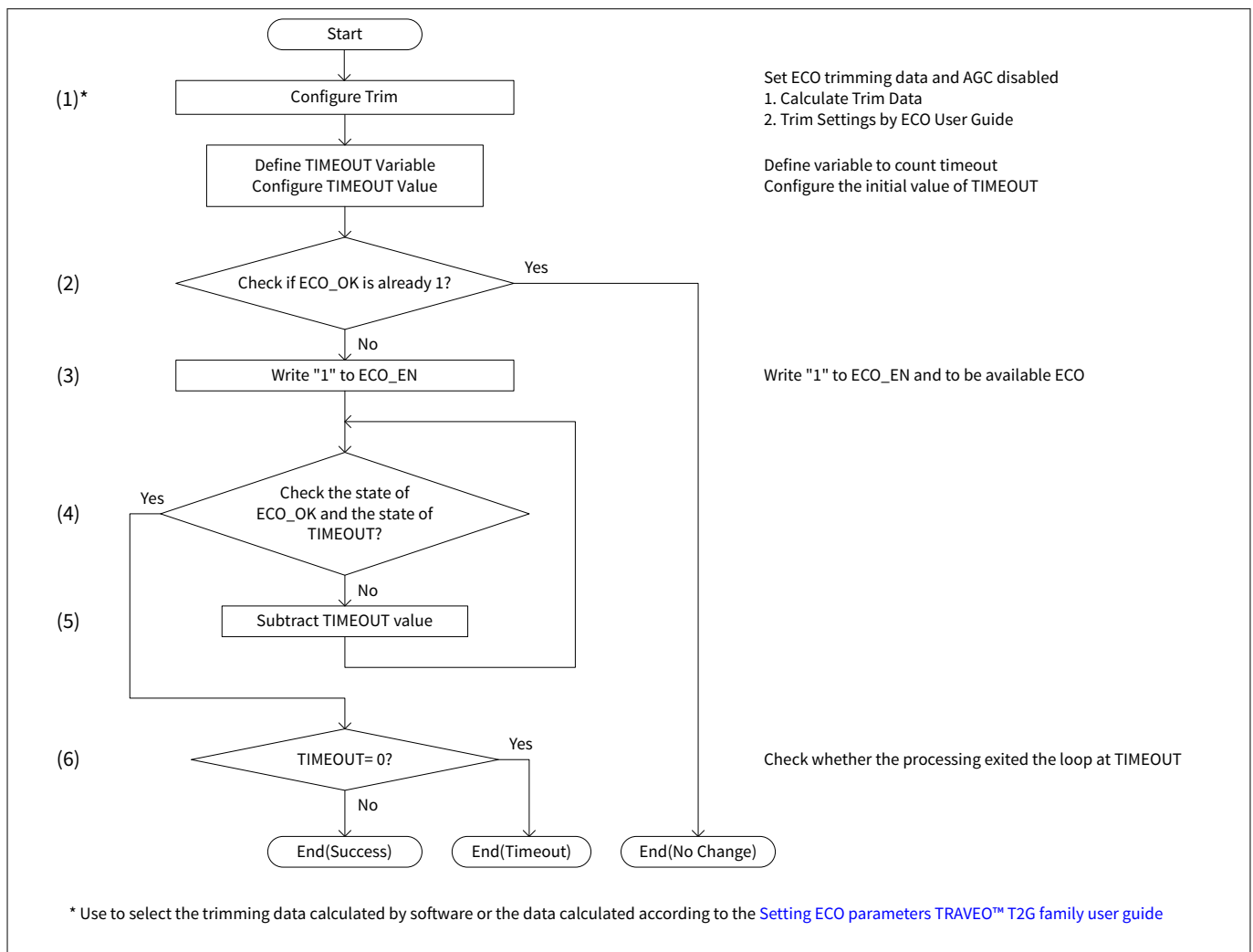


Figure 9 Enabling ECO

3.1.1 Use case

- Oscillator to use: Crystal unit
- Fundamental frequency: 16 MHz
- Maximum drive level: 300.0 μ W
- Equivalent series resistance: 150.0 ohm
- Shunt capacitance: 0.530 pF
- Parallel load capacitance: 8.000 pF

3 Configuration of the clock resources

- 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.

Table 1 lists the parameters and Table 2 lists the functions of the configuration part of in SDL for ECO trim settings.

Table 1 List of ECO trim settings parameters

Parameters	Description	Value
CLK_ECO_CONFIG2.WDTRIM	Watchdog trim Calculated from Setting ECO parameters in TRAVEO™ T2G user guide	7ul
CLK_ECO_CONFIG2.ATRIM	Amplitude trim Calculated from Setting ECO parameters in TRAVEO™ T2G user guide	0ul
CLK_ECO_CONFIG2.FTRIM	Filter trim of 3 rd harmonic oscillation Calculated from Setting ECO parameters in TRAVEO™ T2G user guide	3ul
CLK_ECO_CONFIG2.RTRIM	Feedback resistor trim Calculated from Setting ECO parameters in TRAVEO™ T2G user guide	3ul
CLK_ECO_CONFIG2.GTRIM	Startup time of the gain trim Calculated from Setting ECO parameters in TRAVEO™ T2G user guide	3ul
CLK_ECO_CONFIG.AGC_EN	Automatic gain control (AGC) disabled Calculated from Setting ECO parameters in TRAVEO™ T2G user guide	0ul [OFF]
WAIT_FOR_STABILIZATION	Waiting for stabilization	10000ul
PLL_400M_0_PATH_NO	PLL number for PLL_400M_0	1ul
PLL_400M_1_PATH_NO	PLL number for PLL_400M_1	2ul
PLL_200M_0_PATH_NO	PLL number for PLL_200M_0	3ul
PLL_200M_1_PATH_NO	PLL number for PLL_200M_1	4ul
CLK_FREQ_ECO	Source clock frequency	16000000ul
SUM_LOAD_SHUNT_CAP_IN_PF	Sum of load shunt capacity (pF)	17ul
ESR_IN_OHM	Equivalent series resistance (ESR) (ohm)	250ul
MAX_DRIVE_LEVEL_IN_UW	Maximum drive level (uW)	100ul
MIN_NEG_RESISTANCE	Minimum negative resistance	5 * ESR_IN_OHM

Table 2 List of ECO trim settings functions

Functions	Description	Value
Cy_WDT_Disable()	Disable watchdog timer	–

(table continues...)

3 Configuration of the clock resources

Table 2 (continued) List of ECO trim settings functions

Functions	Description	Value
Cy_SysClk_FllDisable Sequence(Wait Cycle)	Disable FLL	Wait cycle = WAIT_FOR_STABILIZATION
Cy_SysClk_Pll400M Disable(PLL Number)	Disable PLL400M_0	PLL number = PLL_400M_0_PATH_NO
	Disable PLL400M_1	PLL number = PLL_400M_1_PATH_NO
Cy_SysClk_PllDisable (PLL Number)	Disable PLL200M_0	PLL number = PLL_200M_0_PATH_NO
	Disable PLL200M_1	PLL number = PLL_200M_1_PATH_NO
AllClockConfiguration ()	Clock configuration	–
Cy_SysClk_EcoEnable (Timeout value)	Set ECO enable and timeout value	Timeout value = WAIT_FOR_STABILIZATION
Cy_SysLib_DelayUs(Wait Time)	Delays by the specified number of microseconds	Wait time = 1u (1us)

3.1.2 Sample code for initial configuration of ECO settings

There is a sample code as shown [Code Listing 1](#).

The following description will help you to understand the register notation of the driver part of SDL:

- SRSS->unCLK_ECO_CONFIG.stcField.u1ECO_EN is the SRSS_CLK_ECO_CONFIG.ECO_EN mentioned in the [registers reference manual](#). Other registers are also described in the same manner

- Performance improvement measures

For register setting performance improvement, the SDL writes a complete 32-bit data to the register. Each bit field is generated in advance in a bit-writable buffer and written to the register as the final 32-bit data

```
tempTrimEcoCtlReg.u32Register      = SRSS->unCLK_ECO_CONFIG2.u32Register;
tempTrimEcoCtlReg.stcField.u3WDTRIM = wdtrim;
tempTrimEcoCtlReg.stcField.u4ATRIM  = atrim;
tempTrimEcoCtlReg.stcField.u2FTRIM  = ftrim;
tempTrimEcoCtlReg.stcField.u2RTRIM  = rtrim;
tempTrimEcoCtlReg.stcField.u3GTRIM  = gtrim;
SRSS->unCLK_ECO_CONFIG2.u32Register = tempTrimEcoCtlReg.u32Register;
```

See *cyip_srss_v2.h* under *hdr/rev_x/ip* for more information on the union and structure representation of registers.

3 Configuration of the clock resources

Code Listing 1 General configuration of ECO settings

```

:
/** Wait time definition */
#define WAIT_FOR_STABILIZATION (10000ul) /* Define TIMEOUT Variable. */
:
#define CLK_FREQ_ECO          (16000000ul) /* Define oscillator parameters to use for software
calculation. */
:
#define PLL_400M_0_PATH_NO    (1ul) /* Define PLL number. */
#define PLL_400M_1_PATH_NO    (2ul) /* Define PLL number. */
#define PLL_200M_0_PATH_NO    (3ul) /* Define PLL number. */
#define PLL_200M_1_PATH_NO    (4ul) /* Define PLL number. */
:
#define SUM_LOAD_SHUNT_CAP_IN_PF      (17ul)
:
#define ESR_IN_OHM                (250ul)
:
#define MIN_NEG_RESISTANCE         (5 * ESR_IN_OHM)
#define MAX_DRIVE_LEVEL_IN_UW      (100ul)
:
static void AllClockConfiguration(void);
:

int main(void)
{
    /* disable watchdog timer */
    Cy_WDT_Disable(); /* Watchdog Timer disable. */
    :
    /* Disable Fll */
    CY_ASSERT(Cy_SysClk_FllDisableSequence(WAIT_FOR_STABILIZATION) == CY_SYSClk_SUCCESS); /*
Disable FLL */

    /* Disable Pll */
    CY_ASSERT(Cy_SysClk_Pll400MDisable(PLL_400M_0_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable
PLL */
    CY_ASSERT(Cy_SysClk_Pll400MDisable(PLL_400M_1_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable
PLL */
    CY_ASSERT(Cy_SysClk_PllDisable(PLL_200M_0_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable PLL */
    CY_ASSERT(Cy_SysClk_PllDisable(PLL_200M_1_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable PLL */

    /* Enable interrupt */
    __enable_irq();

    /* Set Clock Configuring registers */
    AllClockConfiguration(); /* Trim and ECO setting. See Code Listing 2. */
    :
    /* Please ensure output clock frequency using oscilloscope */
    for(;;);
}

```

3 Configuration of the clock resources

Code Listing 2 AllClockConfiguration() function

```
static void AllClockConfiguration(void)
{
    :
    /***** ECO setting *****/
    cy_en_sysclk_status_t ecoStatus;
    ecoStatus = Cy_SysClk_EcoConfigureWithMinRneg( /* (1)-1. Trim settings for software
    calculation. See Code Listing 4. */
        CLK_FREQ_ECO, /* (1)-1. Trim settings for software calculation. See
    Code Listing 4. */
        SUM_LOAD_SHUNT_CAP_IN_PF, /* (1)-1. Trim settings for software
    calculation. See Code Listing 4. */
        ESR_IN_OHM, /* (1)-1. Trim settings for software calculation. See
    Code Listing 4. */
        MAX_DRIVE_LEVEL_IN_UW, /* (1)-1. Trim settings for software
    calculation. See Code Listing 4. */
        MIN_NEG_RESISTANCE /* (1)-1. Trim settings for software calculation.
    See Code Listing 4. */
        ); /* (1)-1. Trim settings for software calculation. See Code
    Listing 4. */
    CY_ASSERT(ecoStatus == CY_SYSClk_SUCCESS);
    {
        SRSS->unCLK_ECO_CONFIG2.stcField.u3WDTRIM = 7ul; /* (1)-2. Trim settings according to the
    ECO User Guide */
        SRSS->unCLK_ECO_CONFIG2.stcField.u4ATRIM = 0ul; /* (1)-2. Trim settings according to the
    ECO User Guide */
        SRSS->unCLK_ECO_CONFIG2.stcField.u2FTRIM = 3ul; /* (1)-2. Trim settings according to the
    ECO User Guide */
        SRSS->unCLK_ECO_CONFIG2.stcField.u2RTRIM = 3ul; /* (1)-2. Trim settings according to the
    ECO User Guide */
        SRSS->unCLK_ECO_CONFIG2.stcField.u3GTRIM = 0ul; /* (1)-2. Trim settings according to the
    ECO User Guide */
        SRSS->unCLK_ECO_CONFIG.stcField.u1AGC_EN = 0ul; /* (1)-2. Trim settings according to the
    ECO User Guide */

        ecoStatus = Cy_SysClk_EcoEnable(WAIT_FOR_STABILIZATION); /* ECO Enable. See Code Listing
    3. */
        CY_ASSERT(ecoStatus == CY_SYSClk_SUCCESS);
    }
    :
    return;
}
```

Either (1)-1 or (1)-2 can be used.

Comment out or delete unused code snippets in (1)-1 or (1)-2.

3 Configuration of the clock resources

Code Listing 3 Cy_SysClk_EcoEnable() function

```
cy_en_sysclk_status_t Cy_SysClk_EcoEnable(uint32_t timeoutus)
{
    cy_en_sysclk_status_t rtnval;

    /* invalid state error if ECO is already enabled */
    if (SRSS->unCLK_ECO_CONFIG.stcField.u1ECO_EN != 0ul) /* 1 = enabled */ /* (2) Check if
ECO_OK is already enabled. */
    {
        return CY_SYSCLOCK_INVALID_STATE;
    }

    /* first set ECO enable */
    SRSS->unCLK_ECO_CONFIG.stcField.u1ECO_EN = 1ul; /* 1 = enable */ /* (3) Write "1" to the
ECO_EN bit and make ECO available. */

    /* now do the timeout wait for ECO_STATUS, bit ECO_OK */
    for (;
        (SRSS->unCLK_ECO_STATUS.stcField.u1ECO_OK == 0ul) &&(timeoutus != 0ul); /* (4) Check
the state of ECO_OK and the state of TIMEOUT. */
        timeoutus--) /* (5) Subtract TIMEOUT value. */
    {
        Cy_SysLib_DelayUs(1u); /* Wait for 1 us. */
    }

    rtnval = ((timeoutus == 0ul) ? CY_SYSCLOCK_TIMEOUT : CY_SYSCLOCK_SUCCESS); /* (6) Check whether
the processing exited the loop at TIMEOUT. */
    return rtnval;
}
```

3 Configuration of the clock resources

Code Listing 4 Cy_SysClk_EcoConfigureWithMinRneg() function

```
cy_en_sysclk_status_t Cy_SysClk_EcoConfigureWithMinRneg(uint32_t freq, uint32_t cSum, uint32_t
esr, uint32_t driveLevel, uint32_t minRneg) /* Trim Calculation by software */
{
    /* Check if ECO is disabled */
    if(SRSS->unCLK_ECO_CONFIG.stcField.u1ECO_EN == 1ul)
    {
        return(CY_SYSCLOCK_INVALID_STATE);
    }

    /* calculate intermediate values */
    float32_t freqMHz      = (float32_t)freq / 1000000.0f;
    float32_t maxAmplitude = (1000.0f * ((float32_t)sqrt((float64_t)((float32_t)driveLevel /
(2.0f * (float32_t)esr)))))) /
                                (M_PI * freqMHz * (float32_t)cSum);

    float32_t gm_min       = (157.91367042f /*4 * M_PI * M_PI * 4*/ * minRneg * freqMHz *
freqMHz * (float32_t)cSum * (float32_t)cSum) /
                                1000000000.0f;

    /* Get trim values according to calculated values */
    uint32_t atrim, agcen, wdtrim, gtrim, rtrim, ftrim;
    atrim = Cy_SysClk_SelectEcoAtrim(maxAmplitude); /* Get Atrim Value. See Code Listing 5. */
    if(atrim == CY_SYSCLOCK_INVALID_TRIM_VALUE)
    {
        return(CY_SYSCLOCK_BAD_PARAM);
    }

    agcen = Cy_SysClk_SelectEcoAGCEN(maxAmplitude); /* Get AGC enable setting. See Code Listing
6. */
    if(agcen == CY_SYSCLOCK_INVALID_TRIM_VALUE)
    {
        return(CY_SYSCLOCK_BAD_PARAM);
    }

    wdtrim = Cy_SysClk_SelectEcoWDtrim(maxAmplitude); /* Get Wdtrim Value. See Code Listing 7.
*/
    if(wdtrim == CY_SYSCLOCK_INVALID_TRIM_VALUE)
    {
        return(CY_SYSCLOCK_BAD_PARAM);
    }

    gtrim = Cy_SysClk_SelectEcoGtrim(gm_min); /* Get Gtrim Value. See Code Listing 8. */
    if(gtrim == CY_SYSCLOCK_INVALID_TRIM_VALUE)
    {
        return(CY_SYSCLOCK_BAD_PARAM);
    }

    rtrim = Cy_SysClk_SelectEcoRtrim(freqMHz); /* Get Rtrim Value. See Code Listing 9. */
    if(rtrim == CY_SYSCLOCK_INVALID_TRIM_VALUE)
    {
        return(CY_SYSCLOCK_BAD_PARAM);
    }
}
```

3 Configuration of the clock resources

```
}

ftrim = Cy_SysClk_SelectEcoFtrim(atrim); /* Get Ftrim Value. See Code Listing 10. */

/* update all fields of trim control register with one write, without changing the ITRIM
field: */
un_CLK_ECO_CONFIG2_t tempTrimEcoCtlReg;
tempTrimEcoCtlReg.u32Register = SRSS->unCLK_ECO_CONFIG2.u32Register;
tempTrimEcoCtlReg.stcField.u3WDTRIM = wdtrim;
tempTrimEcoCtlReg.stcField.u4ATRIM = atrim;
tempTrimEcoCtlReg.stcField.u2FTRIM = ftrim;
tempTrimEcoCtlReg.stcField.u2RTRIM = rtrim;
tempTrimEcoCtlReg.stcField.u3GTRIM = gtrim;
SRSS->unCLK_ECO_CONFIG2.u32Register = tempTrimEcoCtlReg.u32Register;

SRSS->unCLK_ECO_CONFIG.stcField.u1AGC_EN = agcen;

return(CY_SYSCLK_SUCCESS);
}
```

3 Configuration of the clock resources

Code Listing 5 Cy_SysClk_SelectEcoAtrim () function

```
__STATIC_INLINE uint32_t Cy_SysClk_SelectEcoAtrim(float32_t maxAmplitude) /* Get Atrim Value. */
{
    if((0.50f <= maxAmplitude) && (maxAmplitude < 0.55f))
    {
        return(0x04u1);
    }
    else if(maxAmplitude < 0.60f)
    {
        return(0x05u1);
    }
    else if(maxAmplitude < 0.65f)
    {
        return(0x06u1);
    }
    else if(maxAmplitude < 0.70f)
    {
        return(0x07u1);
    }
    else if(maxAmplitude < 0.75f)
    {
        return(0x08u1);
    }
    else if(maxAmplitude < 0.80f)
    {
        return(0x09u1);
    }
    else if(maxAmplitude < 0.85f)
    {
        return(0x0Au1);
    }
    else if(maxAmplitude < 0.90f)
    {
        return(0x0Bu1);
    }
    else if(maxAmplitude < 0.95f)
    {
        return(0x0Cu1);
    }
    else if(maxAmplitude < 1.00f)
    {
        return(0x0Du1);
    }
    else if(maxAmplitude < 1.05f)
    {
        return(0x0Eu1);
    }
    else if(maxAmplitude < 1.10f)
    {
        return(0x0Fu1);
    }
    else if(1.1f <= maxAmplitude)
    {
        return(0x10u1);
    }
}
```

3 Configuration of the clock resources

```

{
    return(0x00u1);
}
else
{
    // invalid input
    return(CY_SYSCLK_INVALID_TRIM_VALUE);
}
}

```

Code Listing 6 Cy_SysClk_SelectEcoAGCEN() function

```

__STATIC_INLINE uint32_t Cy_SysClk_SelectEcoAGCEN(float32_t maxAmplitude) /* Get AGC enable
setting. */
{
    if((0.50f <= maxAmplitude) && (maxAmplitude < 1.10f))
    {
        return(0x01u1);
    }
    else if(1.10f <= maxAmplitude)
    {
        return(0x00u1);
    }
    else
    {
        return(CY_SYSCLK_INVALID_TRIM_VALUE);
    }
}

```

3 Configuration of the clock resources

Code Listing 7 Cy_SysClk_SelectEcoWDtrim() function

```
__STATIC_INLINE uint32_t Cy_SysClk_SelectEcoWDtrim(float32_t amplitude) /* Get Wdtrim value. */
{
    if( (0.50f <= amplitude) && (amplitude < 0.60f))
    {
        return(0x02ul);
    }
    else if(amplitude < 0.7f)
    {
        return(0x03ul);
    }
    else if(amplitude < 0.8f)
    {
        return(0x04ul);
    }
    else if(amplitude < 0.9f)
    {
        return(0x05ul);
    }
    else if(amplitude < 1.0f)
    {
        return(0x06ul);
    }
    else if(amplitude < 1.1f)
    {
        return(0x07ul);
    }
    else if(1.1f <= amplitude)
    {
        return(0x07ul);
    }
    else
    {
        // invalid input
        return(CY_SYSCLK_INVALID_TRIM_VALUE);
    }
}
```

3 Configuration of the clock resources

Code Listing 8 Cy_SysClk_SelectEcoGtrim() function

```
__STATIC_INLINE uint32_t Cy_SysClk_SelectEcoGtrim(float32_t gm_min) /* Get Gtrim value. */
{
    if( (0.0f <= gm_min) && (gm_min < 2.2f))
    {
        return(0x00ul+1ul);
    }
    else if(gm_min < 4.4f)
    {
        return(0x01ul+1ul);
    }
    else if(gm_min < 6.6f)
    {
        return(0x02ul+1ul);
    }
    else if(gm_min < 8.8f)
    {
        return(0x03ul+1ul);
    }
    else if(gm_min < 11.0f)
    {
        return(0x04ul+1ul);
    }
    else if(gm_min < 13.2f)
    {
        return(0x05ul+1ul);
    }
    else if(gm_min < 15.4f)
    {
        return(0x06ul+1ul);
    }
    else if(gm_min < 17.6f)
    {
        // invalid input
        return(CY_SYSCLK_INVALID_TRIM_VALUE);
    }
    else
    {
        // invalid input
        return(CY_SYSCLK_INVALID_TRIM_VALUE);
    }
}
```

3 Configuration of the clock resources

Code Listing 9 Cy_SysClk_SelectEcoRtrim() function

```
__STATIC_INLINE uint32_t Cy_SysClk_SelectEcoRtrim(float32_t freqMHz) /* Get Rtrim value. */
{
    if(freqMHz > 28.6f)
    {
        return(0x00u1);
    }
    else if(freqMHz > 23.33f)
    {
        return(0x01u1);
    }
    else if(freqMHz > 16.5f)
    {
        return(0x02u1);
    }
    else if(freqMHz > 0.0f)
    {
        return(0x03u1);
    }
    else
    {
        // invalid input
        return(CY_SYSCLK_INVALID_TRIM_VALUE);
    }
}
```

Code Listing 10 Cy_SysClk_SelectEcoFtrim() function

```
__STATIC_INLINE uint32_t Cy_SysClk_SelectEcoFtrim(uint32_t atrim) /* Get Ftrim value. */
{
    return(0x03u1);
}
```

3.2 Setting WCO

3.2.1 Operation overview

WCO is disabled by default. Accordingly, WCO cannot be used unless it is enabled. [Figure 10](#) shows how to configure registers for enabling WCO.

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

3 Configuration of the clock resources

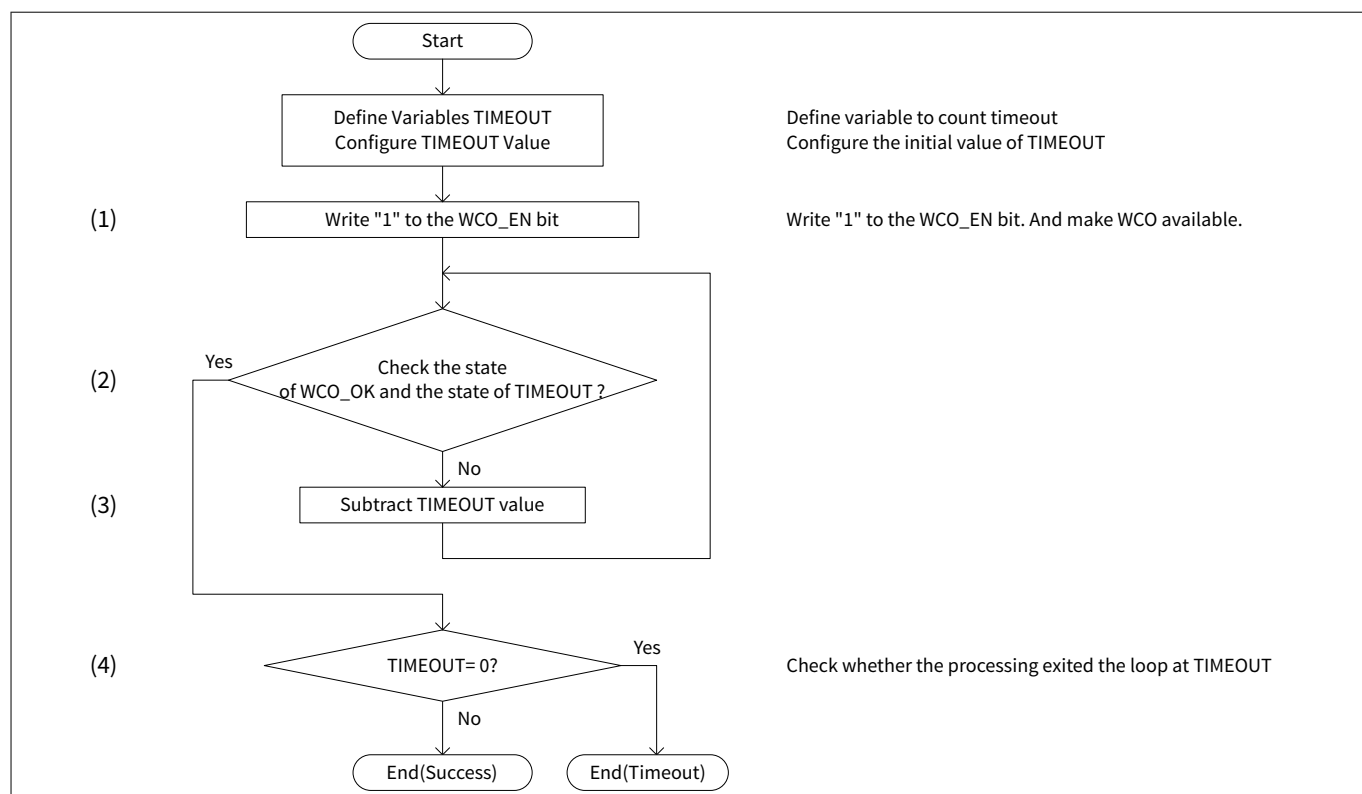


Figure 10 Enabling WCO

3.2.2 Configuration

Table 3 lists the parameters and Table 4 lists the functions of the configuration part of in SDL for WCO settings.

Table 3 List of WCO settings parameters

Parameters	Description	Value
WAIT_FOR_STABILIZATION	Waiting for stabilization	10000ul
PLL_400M_0_PATH_NO	PLL number for PLL_400M_0	1ul
PLL_400M_1_PATH_NO	PLL number for PLL_400M_1	2ul
PLL_200M_0_PATH_NO	PLL number for PLL_200M_0	3ul
PLL_200M_1_PATH_NO	PLL number for PLL_200M_1	4ul

Table 4 List of WCO settings functions

Functions	Description	Value
Cy_WDT_Disable()	Disable watchdog timer	–
Cy_SysClk_FllDisableSequence(Wait Cycle)	Disable FLL	Wait cycle = WAIT_FOR_STABILIZATION
Cy_SysClk_Pll400MDisable(PLL Number)	Disable PLL400M_0	PLL number = PLL_400M_0_PATH_NO
	Disable PLL400M_1	PLL number = PLL_400M_1_PATH_NO

(table continues...)

3 Configuration of the clock resources

Table 4 (continued) List of WCO settings functions

Functions	Description	Value
Cy_SysClk_PllDisable(PLL Number)	Disable PLL200M_0	PLL number = PLL_200M_0_PATH_NO
	Disable PLL200M_1	PLL number = PLL_200M_1_PATH_NO
AllClockConfigurationw()	Clock configuration	–
Cy_SysClk_WcoEnable(Timeout value)	Set WCO enable and timeout value	Timeout value = WAIT_FOR_STABILIZATION
Cy_SysLib_DelayUs(Wait Time)	Delays by the specified number of microseconds	Wait time = 1u (1us)

3.2.3 Sample code

There is a sample code as shown [Code Listing 11](#) to [Code Listing 13](#).

3 Configuration of the clock resources

Code Listing 11 General configuration of WCO settings

```

:
/** Wait time definition */
#define WAIT_FOR_STABILIZATION (10000ul) /* Define TIMEOUT variable. */
:
#define PLL_400M_0_PATH_NO    (1ul) /* Define PLL number. */
#define PLL_400M_1_PATH_NO    (2ul) /* Define PLL number. */
#define PLL_200M_0_PATH_NO    (3ul) /* Define PLL number. */
#define PLL_200M_1_PATH_NO    (4ul) /* Define PLL number. */
:
static void AllClockConfiguration(void);
:
int main(void)
{
    /* disable watchdog timer */
    Cy_WDT_Disable(); /* Watchdog Timer disable. */
:
    /* Disable Fll */
    CY_ASSERT(Cy_SysClk_FllDisableSequence(WAIT_FOR_STABILIZATION) == CY_SYSClk_SUCCESS); /*
Disable FLL */

    /* Disable Pll */
    CY_ASSERT(Cy_SysClk_Pll400MDisable(PLL_400M_0_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable
PLL. */
    CY_ASSERT(Cy_SysClk_Pll400MDisable(PLL_400M_1_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable
PLL. */
    CY_ASSERT(Cy_SysClk_PllDisable(PLL_200M_0_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable PLL. */
    CY_ASSERT(Cy_SysClk_PllDisable(PLL_200M_1_PATH_NO) == CY_SYSClk_SUCCESS); /* Disable PLL. */

    /* Enable interrupt */
    __enable_irq();

    /* Set Clock Configuring registers */
    AllClockConfiguration(); /* WCO setting. See Code Listing 12. */
:
    /* Please check clock output using oscilloscope after CPU reached here. */
    for(;;);
}

```

3 Configuration of the clock resources

Code Listing 12 AllClockConfiguration () function

```
static void AllClockConfiguration(void)
{
:
    /***** WCO setting *****/
    {
        cy_en_sysclk_status_t wcoStatus;
        wcoStatus = Cy_SysClk_WcoEnable(WAIT_FOR_STABILIZATION*10ul); /* WCO Enable See Code
Listing 13. */
        CY_ASSERT(wcoStatus == CY_SYSClk_SUCCESS);
    }

    return;
}
```

Code Listing 13 Cy_Sysclk_WcoEnable() function

```
:
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_WcoEnable(uint32_t timeoutus)
{
    cy_en_sysclk_status_t rtnval = CY_SYSClk_TIMEOUT;

    BACKUP->unCTL.stcField.u1WCO_EN = 1ul; /* (1) Write "1" to the WCO_EN bit and make WCO
available. */

    /* now do the timeout wait for STATUS, bit WCO_OK */
    for (; (Cy_SysClk_WcoOkay() == false) && (timeoutus != 0ul); timeoutus--) /* (2) Check the
state of WCO_OK and the state of TIMEOUT. */ /* (3) Subtract TIMEOUT value. */
    {
        Cy_SysLib_DelayUs(1u); /* Wait for 1 us. */
    }
    if (timeoutus != 0ul) /* (4) Check whether the processing exited the loop at TIMEOUT.
valueCheck whether the processing */
    {
        rtnval = CY_SYSClk_SUCCESS;
    }

    return (rtnval);
}
```

3.3 Setting IMO

IMO is enabled by default so that all functions operate properly. 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 is enabled by default.

3 Configuration of the clock resources

Note that ILO0 is used as the operating clock of the watchdog timer (WDT). Therefore, if ILO0 is disabled, it is necessary to disable 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.

ILO1 is disabled by default. If ILO1 is enabled, write '1' to the ENABLE bit of the CLK_ILO1_CONFIG register.

4 Configuration of FLL and PLL

4 Configuration of FLL and PLL

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

4.1 Setting FLL

4.1.1 Operation overview

To use FLL, it is necessary to set FLL. FLL has a current-controlled oscillator (CCO); the output frequency of this CCO is controlled by adjusting the trim of the CCO. [Figure 11](#) shows the steps to set FLL.

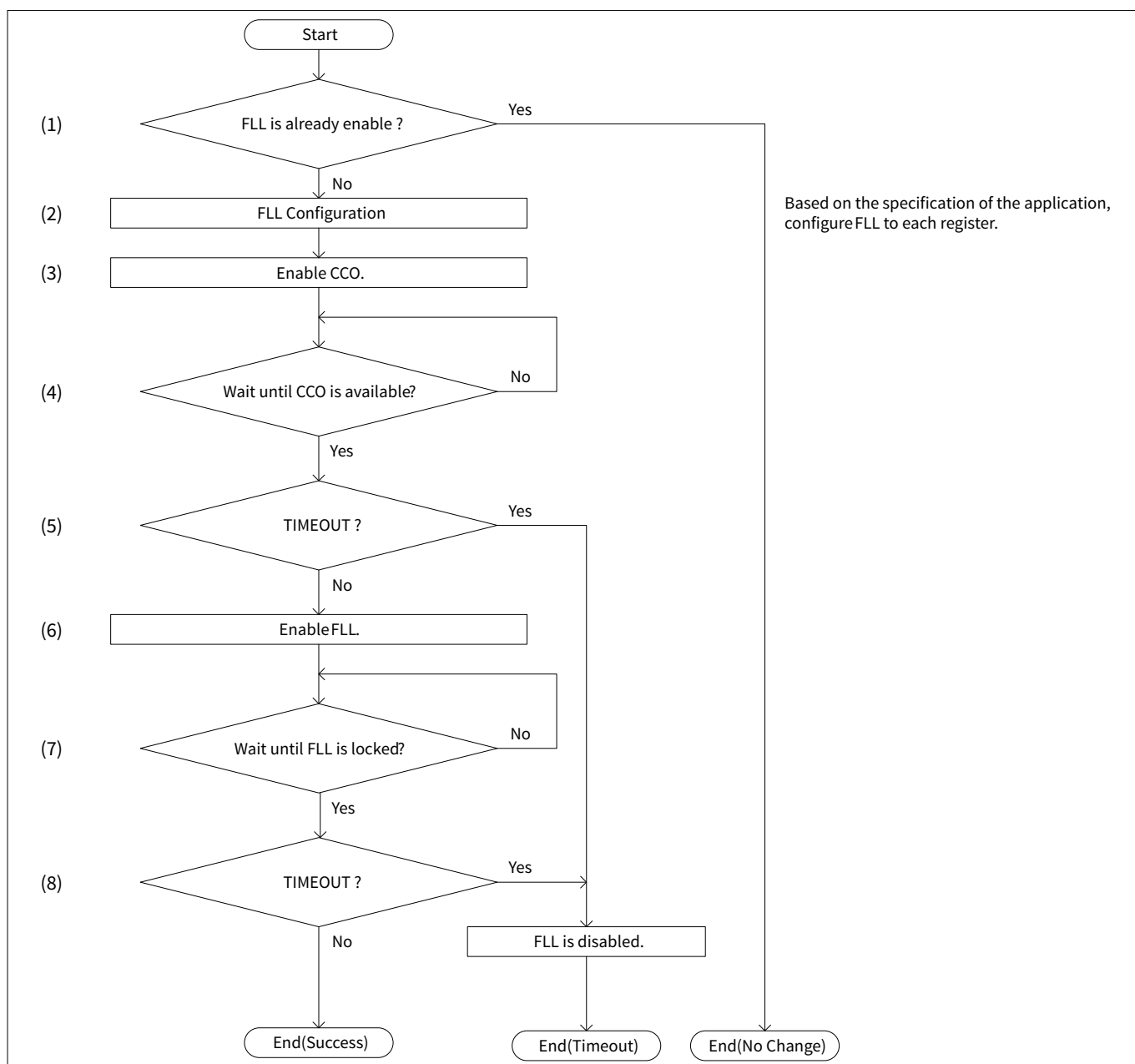


Figure 11 Procedure for setting FLL

For details of FLL and FLL setting registers, see the [architecture reference manual](#) and [registers reference manual](#).

4 Configuration of FLL and PLL

4.1.2 Use case

- Input clock frequency: 16 MHz
- Output clock frequency: 100 MHz

4.1.3 Configuration

[Table 5](#) lists the parameters and [Table 6](#) lists the functions of the configuration part of in SDL for FLL settings.

Table 5 List of FLL settings parameters

Parameters	Description	Value
WAIT_FOR_STABILIZATION	Waiting for stabilization	10000ul
FLL_PATH_NO	FLL number	0u
FLL_TARGET_FREQ	FLL target frequency	100000000ul (100 MHz)
CLK_FREQ_ECO	Source clock frequency	16000000ul (16 MHz)
PATH_SOURCE_CLOCK_FREQ	Source clock frequency	CLK_FREQ_ECO
CY_SYSCLK_FLLPLL_OUTPUT_AUTO	FLL output mode CY_SYSCLK_FLLPLL_OUTPUT_AUTO: Automatic using lock indicator. CY_SYSCLK_FLLPLL_OUTPUT_LOCKED_OR_NOthing: Similar to AUTO, except the clock is gated off when unlocked. CY_SYSCLK_FLLPLL_OUTPUT_INPUT: Select FLL reference input (bypass mode) CY_SYSCLK_FLLPLL_OUTPUT_OUTPUT: Select FLL output. Ignores lock indicator. See SRSS_CLK_FLL_CONFIG3 in registers reference manual for more details.	0ul

Table 6 List of FLL settings functions

Functions	Description	Value
AllClockConfiguration()	Clock configuration	–
Cy_SysClk_FllConfigureStandard(inputFreq,outputFreq,outputMode)	inputFreq: Input frequency outputFreq: Output frequency outputMode: FLL output mode	inputFreq = PATH_SOURCE_CLOCK_FREQ, outputFreq = FLL_TARGET_FREQ, outputMode = CY_SYSCLK_FLLPLL_OUTPUT_AUTO
Cy_SysClk_FllEnable(Timeout value)	Set FLL enable and timeout value	Timeout value = WAIT_FOR_STABILIZATION
Cy_SysLib_DelayUs(Wait Time)	Delays by the specified number of microseconds	Wait time = 1u (1us)

4.1.4 Sample code

There is a sample code as shown [Code Listing 14](#) to [Code Listing 18](#).

4 Configuration of FLL and PLL

Code Listing 14 General configuration of FLL settings

```

/** Wait time definition */
#define WAIT_FOR_STABILIZATION (10000ul) /* Define TIMEOUT Variable. */
:
#define FLL_TARGET_FREQ (100000000ul) /* Define FLL Target Frequency. */
#define CLK_FREQ_ECO (16000000ul) /* Define FLL Input Frequency. */
#define PATH_SOURCE_CLOCK_FREQ CLK_FREQ_ECO
:
#define FLL_PATH_NO (0ul) /* Define FLL number. */
:

int main(void)
{
:
    /* Enable interrupt */
    __enable_irq();
:
    /* Set Clock Configuring registers */
    AllClockConfiguration(); /* FLL setting. See Code Listing 15. */
:
    /* Please check clock output using oscilloscope after CPU reached here. */
    for(;;);
}

```

Code Listing 15 AllClockConfiguration() function

```

static void AllClockConfiguration(void)
{
:
    /****** FLL(PATH0) source setting *****/
    {
:
        fllStatus = Cy_SysClk_FllConfigureStandard(PATH_SOURCE_CLOCK_FREQ, FLL_TARGET_FREQ,
        CY_SYSClk_FLLPLL_OUTPUT_AUTO); /* FLL Configuration. See Code Listing 16. */
        CY_ASSERT(fllStatus == CY_SYSClk_SUCCESS);

        fllStatus = Cy_SysClk_FllEnable(WAIT_FOR_STABILIZATION); /* FLL Enable. See Code
        Listing 18. */
        CY_ASSERT((fllStatus == CY_SYSClk_SUCCESS) || (fllStatus == CY_SYSClk_TIMEOUT));
:
    }
    return;
}

```


4 Configuration of FLL and PLL

Code Listing 16 Cy_SysClk_FllConfigureStandard() function

```

cy_en_sysclk_status_t Cy_SysClk_FllConfigureStandard(uint32_t inputFreq, uint32_t outputFreq,
cy_en_fll_pll_output_mode_t outputMode)
{
    /* check for errors */
    if (SRSS->unCLK_FLL_CONFIG.stcField.u1FLL_ENABLE != 0u1) /* 1 = enabled */ /* (1) Check if
FLL is already enabled. */
    {
        return(CY_SYSCLK_INVALID_STATE);
    }
    else if ((outputFreq < CY_SYSCLK_MIN_FLL_OUTPUT_FREQ) || (CY_SYSCLK_MAX_FLL_OUTPUT_FREQ <
outputFreq)) /* invalid output frequency */ /* Check the FLL output range. */
    {
        return(CY_SYSCLK_INVALID_STATE);
    }
    else if (((float32_t)outputFreq / (float32_t)inputFreq) < 2.2f) /* check output/input
frequency ratio */ /* Check if FLL frequency ratio. */
    {
        return(CY_SYSCLK_INVALID_STATE);
    }

    /* no error */

    /* If output mode is bypass (input routed directly to output), then done.
    The output frequency equals the input frequency regardless of the frequency parameters.
    */
    if (outputMode == CY_SYSCLK_FLLPLL_OUTPUT_INPUT)
    {
        /* bypass mode */
        /* update CLK_FLL_CONFIG3 register with divide by 2 parameter */
        SRSS->unCLK_FLL_CONFIG3.stcField.u2BYPASS_SEL = (uint32_t)outputMode;
        return(CY_SYSCLK_SUCCESS);
    }

    cy_stc_fll_manual_config_t config = { 0u1 };

    config.outputMode = outputMode;

    /* 1. Output division is not required for standard accuracy. */ /* FLL parameter
    calculation. */
    config.enableOutputDiv = false;

    /* 2. Compute the target CCO frequency from the target output frequency and output
    division. */
    uint32_t ccoFreq;
    ccoFreq = outputFreq * ((uint32_t)(config.enableOutputDiv) + 1u1);

    /* 3. Compute the CCO range value from the CCO frequency */
    if(ccoFreq >= CY_SYSCLK_FLL_CCO_BOUNDARY4_FREQ)
    {
        config.ccoRange = CY_SYSCLK_FLL_CCO_RANGE4;
    }
}

```

4 Configuration of FLL and PLL

```

else if(ccoFreq >= CY_SYCLK_FLL_CCO_BOUNDARY3_FREQ)
{
    config.ccoRange = CY_SYCLK_FLL_CCO_RANGE3;
}
else if(ccoFreq >= CY_SYCLK_FLL_CCO_BOUNDARY2_FREQ)
{
    config.ccoRange = CY_SYCLK_FLL_CCO_RANGE2;
}
else if(ccoFreq >= CY_SYCLK_FLL_CCO_BOUNDARY1_FREQ)
{
    config.ccoRange = CY_SYCLK_FLL_CCO_RANGE1;
}
else
{
    config.ccoRange = CY_SYCLK_FLL_CCO_RANGE0;
}

/* 4. Compute the FLL reference divider value. */
config.refDiv = CY_SYCLK_DIV_ROUNDUP(inputFreq * 250ul, outputFreq);

/* 5. Compute the FLL multiplier value.
   Formula is fllMult = (ccoFreq * refDiv) / fref */
config.fllMult = CY_SYCLK_DIV_ROUND((uint64_t)ccoFreq * (uint64_t)config.refDiv,
(uint64_t)inputFreq);

/* 6. Compute the lock tolerance.
   Recommendation: ROUNDUP((refDiv / fref ) * ccoFreq * 3 * CCO_Trim_Step) + 2 */
config.updateTolerance = CY_SYCLK_DIV_ROUNDUP(config.fllMult, 100ul /* Reciprocal number
of Ratio */ );
config.lockTolerance = config.updateTolerance + 20ul /*Threshold*/;
// TODO: Need to check the recommend formula to calculate the value.

/* 7. Compute the CCO igain and pgain. */
/* intermediate parameters */
float32_t kcco = trimSteps_RefArray[config.ccoRange] *
fMargin_MHz_RefArray[config.ccoRange];
float32_t ki_p = (0.85f * (float32_t)inputFreq) / (kcco * (float32_t)(config.refDiv)) /
1000.0f;
/* find the largest IGAIN value that is less than or equal to ki_p */
for(config.igain = CY_SYCLK_N_ELMTS(fll_gains_RefArray) - 1ul; config.igain > 0ul;
config.igain--)
{
    if(fll_gains_RefArray[config.igain] < ki_p)
    {
        break;
    }
}

/* then find the largest PGAIN value that is less than or equal to ki_p - gains[igain] */
for(config.pgain = CY_SYCLK_N_ELMTS(fll_gains_RefArray) - 1ul; config.pgain > 0ul;
config.pgain--)
{
    if(fll_gains_RefArray[config.pgain] < (ki_p - fll_gains_RefArray[config.igain]))

```

4 Configuration of FLL and PLL

```
        {  
            break;  
        }  
    }  
  
    /* 8. Compute the CCO_FREQ bits will be set by HW */  
    config.ccoHwUpdateDisable = 0ul;  
  
    /* 9. Compute the settling count, using a 1-usec settling time. */  
    config.settlingCount = (uint16_t)((float32_t)inputFreq / 1000000.0f);  
  
    /* configure FLL based on calculated values */  
    cy_en_sysclk_status_t returnStatus;  
    returnStatus = Cy_SysClk_FllManualConfigure(&config); /* Set FLL registers. See Code  
Listing 17. */  
  
    return (returnStatus);  
}
```

4 Configuration of FLL and PLL

Code Listing 17 Cy_SysClk_FllManualConfigure() function

```

cy_en_sysclk_status_t Cy_SysClk_FllManualConfigure(const cy_stc_fll_manual_config_t *config)
{
    cy_en_sysclk_status_t returnStatus = CY_SYSCLOCK_SUCCESS;

    /* check for errors */
    if (SRSS->unCLK_FLL_CONFIG.stcField.u1FLL_ENABLE != 0u1) /* 1 = enabled */ /* (1) Check if
FLL is already enabled.*/
    {
        returnStatus = CY_SYSCLOCK_INVALID_STATE;
    }
    else
    { /* return status is OK */
    }

    /* no error */
    if (returnStatus == CY_SYSCLOCK_SUCCESS) /* no errors */ /* (2) FLL Configuration */
    {
        /* update CLK_FLL_CONFIG register with 2 parameters; FLL_ENABLE is already 0 */
        un_CLK_FLL_CONFIG_t tempConfig;
        tempConfig.u32Register = SRSS->unCLK_FLL_CONFIG.u32Register; /* Set
CLK_FLL_CONFIG register. */
        tempConfig.stcField.u18FLL_MULT = config->fllMult; /* Set CLK_FLL_CONFIG register.
*/
        tempConfig.stcField.u1FLL_OUTPUT_DIV = (uint32_t)(config->enableOutputDiv); /* Set
CLK_FLL_CONFIG register. */
        SRSS->unCLK_FLL_CONFIG.u32Register = tempConfig.u32Register;

        /* update CLK_FLL_CONFIG2 register with 2 parameters */
        un_CLK_FLL_CONFIG2_t tempConfig2;
        tempConfig2.u32Register = SRSS->unCLK_FLL_CONFIG2.u32Register; /* Set
CLK_FLL_CONFIG2 register. */
        tempConfig2.stcField.u13FLL_REF_DIV = config->refDiv; /* Set CLK_FLL_CONFIG2 register.
*/
        tempConfig2.stcField.u8LOCK_TOL = config->lockTolerance; /* Set CLK_FLL_CONFIG2
register. */
        tempConfig2.stcField.u8UPDATE_TOL = config->updateTolerance; /* Set CLK_FLL_CONFIG2
register. */
        SRSS->unCLK_FLL_CONFIG2.u32Register = tempConfig2.u32Register;

        /* update CLK_FLL_CONFIG3 register with 4 parameters */
        un_CLK_FLL_CONFIG3_t tempConfig3;
        tempConfig3.u32Register = SRSS->unCLK_FLL_CONFIG3.u32Register; /* Set
CLK_FLL_CONFIG3 register. */
        tempConfig3.stcField.u4FLL_LF_IGAIN = config->igain; /* Set CLK_FLL_CONFIG3 register.
*/
        tempConfig3.stcField.u4FLL_LF_PGAIN = config->pgain; /* Set CLK_FLL_CONFIG3 register.
*/
        tempConfig3.stcField.u13SETTLING_COUNT = config->settleCount; /* Set CLK_FLL_CONFIG3
register. */
        tempConfig3.stcField.u2BYPASS_SEL = (uint32_t)(config->outputMode); /* Set
CLK_FLL_CONFIG3 register. */
    }
}

```

4 Configuration of FLL and PLL

```

        SRSS->unCLK_FLL_CONFIG3.u32Register    = tempConfig3.u32Register;

        /* update CLK_FLL_CONFIG4 register with 1 parameter; preserve other bits */
        un_CLK_FLL_CONFIG4_t tempConfig4;
        tempConfig4.u32Register                = SRSS->unCLK_FLL_CONFIG4.u32Register; /* Set
CLK_FLL_CONFIG4 register. */
        tempConfig4.stcField.u3CCO_RANGE       = (uint32_t)(config->ccoRange); /* Set
CLK_FLL_CONFIG4 register. */
        tempConfig4.stcField.u9CCO_FREQ       = (uint32_t)(config->cco_Freq); /* Set
CLK_FLL_CONFIG4 register. */
        tempConfig4.stcField.u1CCO_HW_UPDATE_DIS = (uint32_t)(config->ccoHwUpdateDisable); /*
Set CLK_FLL_CONFIG4 register. */
        SRSS->unCLK_FLL_CONFIG4.u32Register    = tempConfig4.u32Register; /* Set
CLK_FLL_CONFIG4 register. */
    } /* if no error */

    return (returnStatus);
}
    
```

4 Configuration of FLL and PLL

Code Listing 18 Cy_SysClk_FllEnable() function

```
cy_en_sysclk_status_t Cy_SysClk_FllEnable(uint32_t timeoutus)
{
    /* first set the CCO enable bit */
    SRSS->unCLK_FLL_CONFIG4.stcField.u1CCO_ENABLE = 1ul; /* (3) Enable CCO. */

    /* Wait until CCO is ready */
    while(SRSS->unCLK_FLL_STATUS.stcField.u1CCO_READY == 0ul) /* (4) Wait until CCO is
available. */
    {
        if(timeoutus == 0ul) /* (5) Check Timeout. */
        {
            /* If cco ready doesn't occur, FLL is stopped. */
            Cy_SysClk_FllDisable(); /* FLL Disabled if timeout occur. */
            return(CY_SYSCLK_TIMEOUT);
        }
        Cy_SysLib_DelayUs(1u); /* Wait for 1 us. */
        timeoutus--;
    }

    /* Set the FLL bypass mode to 2 */
    SRSS->unCLK_FLL_CONFIG3.stcField.u2BYPASS_SEL = (uint32_t)CY_SYSCLK_FLLPLL_OUTPUT_INPUT;

    /* Set the FLL enable bit, if CCO is ready */
    SRSS->unCLK_FLL_CONFIG.stcField.u1FLL_ENABLE = 1ul; /* (6) Enable FLL */

    /* now do the timeout wait for FLL_STATUS, bit LOCKED */
    while(SRSS->unCLK_FLL_STATUS.stcField.u1LOCKED == 0ul) /* (7) Wait until FLL is locked. */
    {
        if(timeoutus == 0ul) /* (8) Check Timeout. */
        {
            /* If lock doesn't occur, FLL is stopped. */
            Cy_SysClk_FllDisable(); /* FLL Disabled if timeout occurs. */
            return(CY_SYSCLK_TIMEOUT);
        }
        Cy_SysLib_DelayUs(1u); /* Wait for 1 us. */
        timeoutus--;
    }

    /* Lock occurred; we need to clear the unlock occurred bit.
    Do so by writing a 1 to it. */
    SRSS->unCLK_FLL_STATUS.stcField.u1UNLOCK_OCCURRED = 1ul;
    /* Set the FLL bypass mode to 3 */
    SRSS->unCLK_FLL_CONFIG3.stcField.u2BYPASS_SEL = (uint32_t)CY_SYSCLK_FLLPLL_OUTPUT_OUTPUT;

    return(CY_SYSCLK_SUCCESS);
}
```

4 Configuration of FLL and PLL

4.2 Setting PLL

4.2.1 Operation overview

To use PLL, it is necessary to set PLL. [Figure 12](#) shows the steps to set PLL400 and PLL200. For details about PLL400 and PLL200, see the [architecture reference manual](#) and [registers reference manual](#).

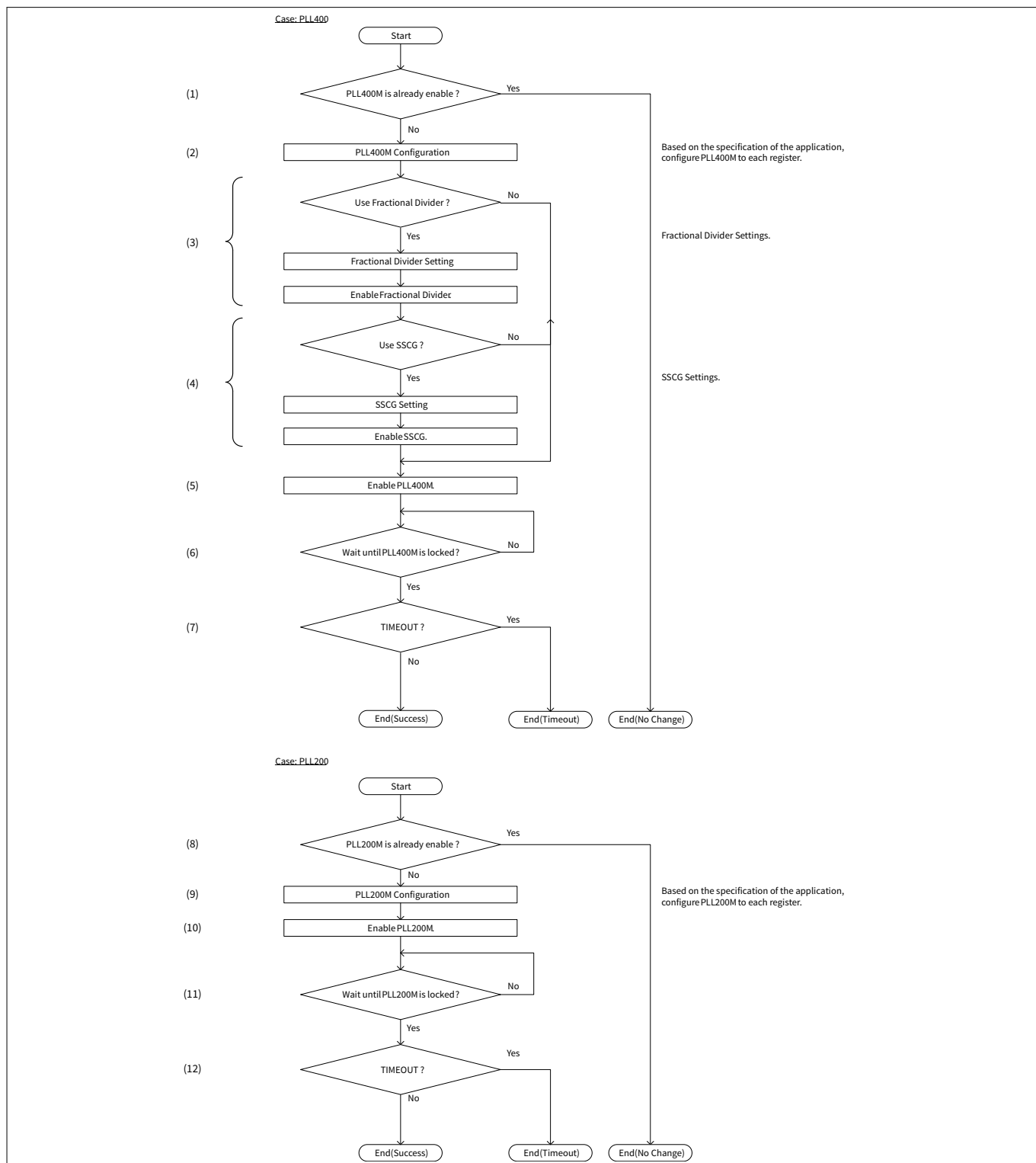


Figure 12 Procedure for configuring PLL

4 Configuration of FLL and PLL

4.2.2 Use case

- Input clock frequency: 16.000 MHz
- Output clock frequency:
 - 340.000 MHz (PLL400 #0)
 - 196.608 MHz (PLL400 #1)
 - 160.000 MHz (PLL200 #0)
 - 80.000 MHz (PLL200 #1)
- Fractional divider:
 - Disable (PLL400 #0)
 - Enable (PLL400 #1)
- SSCG:
 - Enable (PLL400 #0)
 - Disable (PLL400 #1)
- SSCG dithering:
 - Enable (PLL400 #0)
 - Disable (PLL400 #1)
- SSCG modulation depth: -2.0% (PLL400)
- SSCG modulation rate: Divide 512 (PLL400)
- LF mode: 200 MHz to 400 MHz (PLL200)

4.2.3 Configuration

Table 7 and Table 9 lists each parameter of PLL (400/200) and Table 8 and Table 10 lists each function of PLL (400/200) of the configuration part of in SDL for PLL (400/200) settings.

Table 7 List of PLL 400 settings parameters

Parameters	Description	Value
PLL400_0_TARGET_FREQ	PLL400 #0 target frequency	340 MHz (340000000ul)
PLL400_1_TARGET_FREQ	PLL400 #1 target frequency	196.608 MHz (196608000ul)
WAIT_FOR_STABILIZATION	Waiting for stabilization	10000ul
PLL400_0_PATH_NO	PLL400 #0 number	1u
PLL400_1_PATH_NO	PLL400 #1 number	2u
CLK_FREQ_ECO	ECO clock frequency	16000000ul (16 MHz)
PATH_SOURCE_CLOCK_FREQ	PATH source clock frequency	CLK_FREQ_ECO

(table continues...)

4 Configuration of FLL and PLL

Table 7 (continued) List of PLL 400 settings parameters

Parameters	Description	Value
CY_SYSCLK_FLLPLL_OUTPUT_AUTO	FLL output mode CY_SYSCLK_FLLPLL_OUTPUT_AUTO: Automatic using lock indicator. CY_SYSCLK_FLLPLL_OUTPUT_LOCKED_OR_NOHING: Similar to AUTO, except the clock is gated off when unlocked. CY_SYSCLK_FLLPLL_OUTPUT_INPUT: Select FLL reference input (bypass mode) CY_SYSCLK_FLLPLL_OUTPUT_OUTPUT: Select FLL output. Ignores lock indicator. See SRSS_CLK_FLL_CONFIG3 in registers reference manual for more details.	0ul
pllConfig.inputFreq	Input PLL frequency	PATH_SOURCE_CLOCK_FREQ
pllConfig.outputFreq	Output PLL frequency (PLL400 #0)	PLL400_0_TARGET_FREQ
	Output PLL frequency (PLL400 #1)	PLL400_1_TARGET_FREQ
pllConfig.outputMode	Output mode 0: CY_SYSCLK_FLLPLL_OUTPUT_AUTO 1: CY_SYSCLK_FLLPLL_OUTPUT_LOCKED_OR_NOHING 2: CY_SYSCLK_FLLPLL_OUTPUT_INPUT 3: CY_SYSCLK_FLLPLL_OUTPUT_OUTPUT	CY_SYSCLK_FLLPLL_OUTPUT_A UTO
pllConfig.fracEn	Enable fractional divider (PLL400 #0)	false
	Enable fractional divider (PLL400 #1)	true
pllConfig.fracDitherEn	Enable dithering operation (PLL400 #0)	false
	Enable dithering operation (PLL400 #1)	true
pllConfig.sscgEn	Enable SSCG (PLL400 #0)	true
	Enable SSCG (PLL400 #1)	false
pllConfig.sscgDitherEn	Enable SSCG dithering operation (PLL400 #0)	true
	Enable SSCG dithering operation (PLL400 #1)	false
pllConfig.sscgDepth	Set SSCG modulation depth	CY_SYSCLK_SSCG_DEPTH_MINU S_2_0
pllConfig.sscgRate	Set SSCG modulation rate	CY_SYSCLK_SSCG_RATE_DIV_51 2
manualConfig.feedbackDiv	Control bits for feedback divider.	p (Calculated value)

(table continues...)

4 Configuration of FLL and PLL

Table 7 (continued) List of PLL 400 settings parameters

Parameters	Description	Value
manualConfig.referenceDiv	Control bits for reference divider.	q (Calculated value)
manualConfig.outputDiv	Control bits for output divider. 0: illegal (undefined behavior) 1: illegal (undefined behavior) 2: divide by 2. Suitable for direct usage as HFCLK source. ... 16: divide by 16. Suitable for direct usage as HFCLK source. >16: illegal (undefined behavior)	out (Calculated value)
manualConfig.lfMode	VCO frequency range selection. 0: VCO frequency is [200 MHz, 400 MHz] 1: VCO frequency is [170 MHz, 200 MHz)	config->lfMode (Calculated value)
manualConfig.outputMode	Bypass mux is located just after PLL output. 0: AUTO 1: LOCKED_OR_NOTHING 2: PLL_REF 3: PLL_OUT	config->outputMode (Calculated value)

Table 8 List of PLL 400 settings functions

Functions	Description	Value
AllClockConfiguration()	Clock configuration	–
Cy_SysClk_Pll400M Configure(PLL Number, PLL Configure)	Set PLL path No. and PLL configure (PLL400 #0)	PLL number = PLL400_0_PATH_NO, PLL configure = g_pll400_0_Config
	Set PLL path No. and PLL configure (PLL400 #1)	PLL number = PLL400_1_PATH_NO, PLL configure = g_pll400_1_Config
Cy_SysClk_Pll400M Enable(PLL Number, Timeout value)	Set PLL path No. and monitor PLL configure (PLL400 #0)	PLL number = PLL400_0_PATH_NO, Timeout value = WAIT_FOR_STABILIZATION
	Set PLL path No. and monitor PLL configure (PLL400 #1)	PLL number = PLL400_1_PATH_NO, Timeout value = WAIT_FOR_STABILIZATION
Cy_SysLib_DelayUs(Wait Time)	Delays by the specified number of microseconds	Wait time = 1u (1us)

(table continues...)

4 Configuration of FLL and PLL

Table 8 (continued) List of PLL 400 settings functions

Functions	Description	Value
Cy_SysClk_PllManualConfigure(PLL Number, PLL Manual Configure)	Set PLL path No. and PLL manual configure (PLL400 #0)	PLL number = PLL400_0_PATH_NO,
	Set PLL path No. and PLL manual configure (PLL400 #1)	PLL manual configure = manualConfig PLL number = PLL400_1_PATH_NO, PLL manual configure = manualConfig
Cy_SysClk_GetPll400MNo(Clkpath, PllNo)	Return PLL number according to input PATH number (PLL400 #0)	Clkpath = 1u PllNo = 0u
	Return PLL number according to input PATH number (PLL400 #1)	Clkpath = 2u PllNo = 1u
Cy_SysClk_PllCalucDividers()	Calculates appropriate divider settings according to PLL input/output frequency.	
Cy_SysClk_Pll400MEnable(PLL Number, Timeout value)	Set PLL path No. and monitor PLL configure (PLL400 #0)	PLL number = PLL400_0_PATH_NO, Timeout value = WAIT_FOR_STABILIZATION
	Set PLL path No. and monitor PLL configure (PLL400 #1)	PLL number = PLL400_1_PATH_NO, Timeout value = WAIT_FOR_STABILIZATION

Table 9 List of PLL 200 settings parameters

Parameters	Description	Value
PLL200_0_TARGET_FREQ	PLL200 #0 target frequency	160 MHz (160000000ul)
PLL200_1_TARGET_FREQ	PLL200 #1 target frequency	80 MHz (80000000ul)
WAIT_FOR_STABILIZATION	Waiting for stabilization	10000ul
PLL200_0_PATH_NO	PLL200 #0 number	3u
PLL200_1_PATH_NO	PLL200 #1 number	4u
PATH_SOURCE_CLOCK_FREQ	PATH source clock frequency	16000000ul (16 MHz)
pllConfig.inputFreq	Input PLL frequency	PATH_SOURCE_CLOCK_FREQ
pllConfig.outputFreq	Output PLL frequency (PLL200 #0)	PLL200_0_TARGET_FREQ
	Output PLL frequency (PLL200 #1)	PLL200_1_TARGET_FREQ
pllConfig.lfMode	PLL LF mode 0: VCO frequency is [200 MHz, 400 MHz] 1: VCO frequency is [170 MHz, 200 MHz]	0u (VCO frequency is 320 MHz)

(table continues...)

4 Configuration of FLL and PLL

Table 9 (continued) List of PLL 200 settings parameters

Parameters	Description	Value
pllConfig.outputMode	Output mode 0: CY_SYSCLK_FLLPLL_OUTPUT_AUTO 1: CY_SYSCLK_FLLPLL_OUTPUT_LOCKED_OR_NOTHING 2: CY_SYSCLK_FLLPLL_OUTPUT_INPUT 3: CY_SYSCLK_FLLPLL_OUTPUT_OUTPUT	CY_SYSCLK_FLLPLL_OUTPUT_AUTO
manualConfig.feedbackDiv	Control bits for feedback divider.	p (Calculated value)
manualConfig.referenceDiv	Control bits for reference divider.	q (Calculated value)
manualConfig.outputDiv	Control bits for output divider. 0: illegal (undefined behavior) 1: illegal (undefined behavior) 2: divide by 2. Suitable for direct usage as HFCLK source. ... 16: divide by 16. Suitable for direct usage as HFCLK source. >16: illegal (undefined behavior)	out (Calculated value)
manualConfig.lfMode	VCO frequency range selection. 0: VCO frequency is [200 MHz, 400 MHz] 1: VCO frequency is [170 MHz, 200 MHz]	config->lfMode (Calculated value)
manualConfig.outputMode	Bypass mux is located just after PLL output. 0: AUTO 1: LOCKED_OR_NOTHING 2: PLL_REF 3: PLL_OUT	config->outputMode (Calculated value)
manualConfig.fracDiv	Fractional divider value	config->fracDiv (Calculated value)

Table 10 List of PLL 200 settings functions

Functions	Description	Value
AllClockConfiguration()	Clock configuration	–
Cy_SysClk_PllConfigure (PLL Number, PLL Configure)	Set PLL path No. and PLL configure (PLL200 #0)	PLL number = PLL200_0_PATH_NO, PLL configure = g_pll200_0_Config
	Set PLL path No. and PLL configure (PLL200 #1)	PLL number = PLL200_1_PATH_NO, PLL configure = g_pll200_1_Config
Cy_SysLib_DelayUs(Wait Time)	Delays by the specified number of microseconds	Wait time = 1u (1us)

(table continues...)

4 Configuration of FLL and PLL

Table 10 (continued) List of PLL 200 settings functions

Functions	Description	Value
Cy_SysClk_PllManualConfigure(PLL Number, PLL Manual Configure)	Set PLL path No. and PLL manual configure (PLL200 #0)	PLL number = PLL200_0_PATH_NO, PLL manual configure = manualConfig
	Set PLL path No. and PLL manual configure (PLL200 #1)	PLL number = PLL200_1_PATH_NO, PLL manual configure = manualConfig
Cy_SysClk_GetPllNo(Clkpath, PllNo)	Return PLL number according to input PATH number (PLL200 #0)	Clkpath = 3u PllNo = 0u
	Return PLL number according to input PATH number (PLL200 #1)	Clkpath = 4u PllNo = 1u
Cy_SysClk_PllCalucDividers(InputFreq, OutputFreq, PLLlimit, FracBitNum, RefDiv, OutputDiv, FeedbackFracDiv)	Calculates appropriate divider settings according to PLL input/output frequency	InputFreq = PATH_SOURCE_CLOCK_FREQ OutputFreq = PLL400_0_TARGET_FREQ (PLL 400 #0), PLL400_1_TARGET_FREQ (PLL 400 #1), PLL200_0_TARGET_FREQ (PLL 200 #0), PLL200_1_TARGET_FREQ (PLL 200 #1) PLLlimit = g_limPll400Mfrac (PLL 400 #1 only), g_limPll400M (Other) FracBitNum = 24ul (PLL 400 #1 only), 0ul (Other) FeedbackDiv = manualConfig.feedbackDiv RefDiv = manualConfig.referenceDiv OutputDiv = manualConfig.outputDiv FeedbackFracDiv = manualConfig.fracDiv
Cy_SysClk_PllEnable(PLL Number, Timeout value)	Set PLL path No. and monitor PLL configure (PLL200 #0)	PLL number = PLL200_0_PATH_NO, Timeout value = WAIT_FOR_STABILIZATION
	Set PLL path No. and monitor PLL configure (PLL200 #1)	PLL number = PLL200_1_PATH_NO, Timeout value = WAIT_FOR_STABILIZATION

4.2.4 Sample code

The sample code for PLL400 #0 is shown in [Code Listing 19](#) to [Code Listing 25](#), and for PLL200 #0 is shown in [Code Listing 26](#) to [Code Listing 32](#).

4 Configuration of FLL and PLL

Code Listing 19 General configuration of PLL 400 #0 settings

```

:
#define PLL400_0_TARGET_FREQ      (340000000ul) /* PLL Target frequency. */
#define PLL400_1_TARGET_FREQ      (196608000ul) /* PLL Target frequency. */
:
/** Wait time definition */
#define WAIT_FOR_STABILIZATION (10000ul) /* Define TIMEOUT variable. */
:
#define PLL_400M_0_PATH_NO        (1ul) /* Define PLL number. */
#define PLL_400M_1_PATH_NO        (2ul) /* Define PLL number. */
#define PLL_200M_0_PATH_NO        (3ul) /* Define PLL number. */
#define PLL_200M_1_PATH_NO        (4ul) /* Define PLL number. */
#define BYPASSED_PATH_NO          (5ul) /* Define PLL number. */
:
/** Parameters for Clock Configuration */
cy_stc_pll_400M_config_t g_pll400_0_Config = /* PLL400 #0 Configuration. */
{
    .inputFreq      = PATH_SOURCE_CLOCK_FREQ,
    .outputFreq      = PLL400_0_TARGET_FREQ,
    .outputMode      = CY_SYSCLK_FLLPLL_OUTPUT_AUTO,
    .fracEn          = false,
    .fracDitherEn    = false,
    .sscEn           = true,
    .sscDitherEn     = true,
    .sscDepth        = CY_SYSCLK_SSCG_DEPTH_MINUS_2_0,
    .sscRate         = CY_SYSCLK_SSCG_RATE_DIV_512,
};
:
int main(void)
{
:
    /* Enable interrupt */
    __enable_irq();

    /* Set Clock Configuring registers */
    AllClockConfiguration(); /* PLL400 #0 setting. See Code Listing 20. */
:
    /* Please check clock output using oscilloscope after CPU reached here. */
    for(;;);
}

```

4 Configuration of FLL and PLL

Code Listing 20 AllClockConfiguration() function

```
static void AllClockConfiguration(void)
{
    :
    /**** PLL400M#0(PATH1) source setting ***/
    {
        :
        status = Cy_SysClk_Pll400MConfigure(PLL_400M_0_PATH_NO, &g_pll400_0_Config); /* PLL400
Configuration. See Code Listing 21. */
        CY_ASSERT(status == CY_SYSCLK_SUCCESS);

        status = Cy_SysClk_Pll400MEnable(PLL_400M_0_PATH_NO, WAIT_FOR_STABILIZATION); /* PLL400
Enable. See Code Listing 25. */
        CY_ASSERT(status == CY_SYSCLK_SUCCESS);
    :
    }
    return;
}
```

4 Configuration of FLL and PLL

Code Listing 21 Cy_SysClk_Pll400MConfigure() function

```

cy_en_sysclk_status_t Cy_SysClk_Pll400MConfigure(uint32_t clkPath, const
cy_stc_pll_400M_config_t *config)
{
    /* check for error */
    uint32_t pllNo;
    cy_en_sysclk_status_t status = Cy_SysClk_GetPll400MNo(clkPath, &pllNo); /* Check if the
valid clock path and PLL400 Number. See Code Listing 23. */
    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    if (SRSS->CLK_PLL400M[pllNo].unCONFIG.stcField.u1ENABLE != 0ul) /* 1 = enabled */ /* (1)
Check if PLL400 is already enabled. */
    {
        return (CY_SYSCLK_INVALID_STATE);
    }

    cy_stc_pll_400M_manual_config_t manualConfig = {0ul};
    const cy_stc_pll_limitation_t* pllLim;
    uint32_t fracBitNum;
    if(config->fracEn == true)
    {
        pllLim = &g_limPll400MFrac;
        fracBitNum = 24ul;
    }
    else
    {
        pllLim = &g_limPll400M;
        fracBitNum = 0ul;
    }
    status = Cy_SysClk_PllCalucDividers(config->inputFreq,
                                        config->outputFreq,
                                        pllLim,
                                        fracBitNum,
                                        &manualConfig.feedbackDiv,
                                        &manualConfig.referenceDiv,
                                        &manualConfig.outputDiv,
                                        &manualConfig.fracDiv
                                        );

    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    manualConfig.outputMode = config->outputMode;
    manualConfig.fracEn = config->fracEn;
    manualConfig.fracDitherEn = config->fracDitherEn;
    manualConfig.sscgEn = config->sscgEn;
    manualConfig.sscgDitherEn = config->sscgDitherEn;

```


4 Configuration of FLL and PLL

```
manualConfig.sscgDepth    = config->sscDepth;  
manualConfig.sscgRate     = config->sscRate;  
  
status = Cy_SysClk_Pll400MManualConfigure(clkPath, &manualConfig); /* PLL400 Manual  
Configure. See Code Listing 22. */  
return (status);  
}
```

4 Configuration of FLL and PLL

Code Listing 22 Cy_SysClk_PLL400MManualConfigure() function

```

cy_en_sysclk_status_t Cy_SysClk_PLL400MManualConfigure(uint32_t clkPath, const
cy_stc_pll_400M_manual_config_t *config)
{
    /* check for error */
    uint32_t pllNo;
    cy_en_sysclk_status_t status = Cy_SysClk_GetPLL400MNo(clkPath, &pllNo);/* Getting PLL400
PATH Number. See Code Listing 23. */
    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    /* valid divider bitfield values */
    if((config->outputDiv < PLL_400M_MIN_OUTPUT_DIV) || (PLL_400M_MAX_OUTPUT_DIV < config-
>outputDiv))
    {
        return(CY_SYSCLK_BAD_PARAM);
    }

    if((config->referenceDiv < PLL_400M_MIN_REF_DIV) || (PLL_400M_MAX_REF_DIV < config-
>referenceDiv))
    {
        return(CY_SYSCLK_BAD_PARAM);
    }

    if((config->feedbackDiv < PLL_400M_MIN_FB_DIV) || (PLL_400M_MAX_FB_DIV < config-
>feedbackDiv))
    {
        return(CY_SYSCLK_BAD_PARAM);
    }

    un_CLK_PLL400M_CONFIG_t tempClkPLL400MConfigReg;
    tempClkPLL400MConfigReg.u32Register = SRSS->CLK_PLL400M[pllNo].unCONFIG.u32Register;
    if (tempClkPLL400MConfigReg.stcField.u1ENABLE != 0u1) /* 1 = enabled */
    {
        return(CY_SYSCLK_INVALID_STATE);
    }

    /* no errors */
    /* If output mode is bypass (input routed directly to output), then done.
    The output frequency equals the input frequency regardless of the frequency parameters.
    */
    if (config->outputMode != CY_SYSCLK_FLLPLL_OUTPUT_INPUT) /* (2) PLL400 Configuration */
    {
        tempClkPLL400MConfigReg.stcField.u8FEEDBACK_DIV      = (uint32_t)config->feedbackDiv;
        tempClkPLL400MConfigReg.stcField.u5REFERENCE_DIV     = (uint32_t)config->referenceDiv;
        tempClkPLL400MConfigReg.stcField.u5OUTPUT_DIV        = (uint32_t)config->outputDiv;
    }
    tempClkPLL400MConfigReg.stcField.u2BYPASS_SEL            = (uint32_t)config->outputMode;
    SRSS->CLK_PLL400M[pllNo].unCONFIG.u32Register            =
tempClkPLL400MConfigReg.u32Register;

```

4 Configuration of FLL and PLL

```

    un_CLK_PLL400M_CONFIG2_t tempClkPLL400MConfig2Reg;
    tempClkPLL400MConfig2Reg.u32Register = SRSS-
>CLK_PLL400M[p1lNo].unCONFIG2.u32Register;
    tempClkPLL400MConfig2Reg.stcField.u24FRAC_DIV = config->fracDiv; /* (3) Fractional
Divider Settings */
    tempClkPLL400MConfig2Reg.stcField.u3FRAC_DITHER_EN = config->fracDitherEn;
    tempClkPLL400MConfig2Reg.stcField.u1FRAC_EN = config->fracEn;
    SRSS->CLK_PLL400M[p1lNo].unCONFIG2.u32Register = tempClkPLL400MConfig2Reg.u32Register;

    un_CLK_PLL400M_CONFIG3_t tempClkPLL400MConfig3Reg;
    tempClkPLL400MConfig3Reg.u32Register = SRSS-
>CLK_PLL400M[p1lNo].unCONFIG3.u32Register;
    tempClkPLL400MConfig3Reg.stcField.u10SSCG_DEPTH = (uint32_t)config->sscgDepth; /* (4)
SSCG Settings */
    tempClkPLL400MConfig3Reg.stcField.u3SSCG_RATE = (uint32_t)config->sscgRate;
    tempClkPLL400MConfig3Reg.stcField.u1SSCG_DITHER_EN = (uint32_t)config->sscgDitherEn;
    tempClkPLL400MConfig3Reg.stcField.u1SSCG_EN = (uint32_t)config->sscgEn;
    SRSS->CLK_PLL400M[p1lNo].unCONFIG3.u32Register = tempClkPLL400MConfig3Reg.u32Register;

    return (CY_SYCLK_SUCCESS);
}

```

Code Listing 23 Cy_SysClk_GetPLL400MNo() function

```

__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_GetP11400MNo(uint32_t pathNo, uint32_t* p1lNo)
{
    /* check for error */
    if ((pathNo <= 0u1) || (pathNo > SRSS_NUM_PLL400M))
    {
        /* invalid clock path number */
        return(CY_SYCLK_BAD_PARAM);
    }

    *p1lNo = pathNo - 1u1;
    return(CY_SYCLK_SUCCESS);
}

```

4 Configuration of FLL and PLL

Code Listing 24 Cy_SysClk_PllCalucDividers() function

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PllCalucDividers(uint32_t inFreq,
                                                                uint32_t targetOutFreq,
                                                                const cy_stc_pll_limitation_t*
lim,
                                                                uint32_t fracBitNum,
                                                                uint32_t* feedBackDiv,
                                                                uint32_t* refDiv,
                                                                uint32_t* outputDiv,
                                                                uint32_t* feedBackFracDiv)
{
    uint64_t errorMin = 0xFFFFFFFFFFFFFFFFull;

    if(feedBackDiv == NULL)
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    if((feedBackFracDiv == NULL) && (fracBitNum != 0ul))
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    if(refDiv == NULL)
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    if(outputDiv == NULL)
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    if ((targetOutFreq < lim->minFoutput) || (lim->maxFoutput < targetOutFreq))
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    /* REFERENCE_DIV selection */
    for (uint32_t i_refDiv = lim->minRefDiv; i_refDiv <= lim->maxRefDiv; i_refDiv++)
    {
        uint32_t fpd_roundDown = inFreq / i_refDiv;
        if (fpd_roundDown < lim->minFpd)
        {
            break;
        }

        uint32_t fpd_roundUp = CY_SYSCLOCK_DIV_ROUNDUP(inFreq, i_refDiv);
        if (lim->maxFpd < fpd_roundUp)
        {
            continue;
        }
    }
}
```

4 Configuration of FLL and PLL

```

/* OUTPUT_DIV selection */
for (uint32_t i_outDiv = lim->minOutputDiv; i_outDiv <= lim->maxOutputDiv; i_outDiv++)
{
    uint64_t tempVco = i_outDiv * targetOutFreq;

    if(tempVco < lim->minFvco)
    {
        continue;
    }
    else if(lim->maxFvco < tempVco)
    {
        break;
    }

    // (inFreq / refDiv) * feedBackDiv = Fvco
    // feedBackDiv = Fvco * refDiv / inFreq
    uint64_t tempFeedBackDivLeftShifted = ((tempVco << (uint64_t)fracBitNum) *
(uint64_t)i_refDiv) / (uint64_t)inFreq;
    uint64_t error = abs(((uint64_t)targetOutFreq << (uint64_t)fracBitNum) -
((uint64_t)inFreq * tempFeedBackDivLeftShifted / ((uint64_t)i_refDiv * (uint64_t)i_outDiv)));

    if (error < errorMin)
    {
        *feedBackDiv      = (uint32_t)(tempFeedBackDivLeftShifted >>
(uint64_t)fracBitNum);
        if(feedBackFracDiv != NULL)
        {
            if(fracBitNum == 0ul)
            {
                *feedBackFracDiv = 0ul;
            }
            else
            {
                *feedBackFracDiv = (uint32_t)(tempFeedBackDivLeftShifted & ((1ull <<
(uint64_t)fracBitNum) - 1ull));
            }
        }
        *refDiv          = i_refDiv;
        *outputDiv        = i_outDiv;
        errorMin          = error;
        if(errorMin == 0ull){break;}
    }
}
if(errorMin == 0ull){break;}

if(errorMin == 0xFFFFFFFFFFFFFFFFull)
{
    return (CY_SYSCLOCK_BAD_PARAM);
}
else
{

```

4 Configuration of FLL and PLL

```
        return (CY_SYSCLK_SUCCESS);
    }
}
```

Code Listing 25 Cy_SysClk_Pll400MEnable() function

```
cy_en_sysclk_status_t Cy_SysClk_Pll400MEnable(uint32_t clkPath, uint32_t timeoutus)
{
    uint32_t pllNo;
    cy_en_sysclk_status_t status = Cy_SysClk_GetPll400MNo(clkPath, &pllNo);
    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    /* first set the PLL enable bit */
    SRSS->CLK_PLL400M[pllNo].unCONFIG.stcField.u1ENABLE = 1ul; /* (5) Enable PLL400. */

    /* now do the timeout wait for PLL_STATUS, bit LOCKED */
    for (; (SRSS->CLK_PLL400M[pllNo].unSTATUS.stcField.u1LOCKED == 0ul) && /* (6) Wait until
PLL400 is locked. */
        (timeoutus != 0ul); /* (7) Check Timeout. */
        timeoutus--)
    {
        Cy_SysLib_DelayUs(1u); /* Wait for 1 us. */
    }

    status = ((timeoutus == 0ul) ? CY_SYSCLK_TIMEOUT : CY_SYSCLK_SUCCESS);

    return (status);
}
```

4 Configuration of FLL and PLL

Code Listing 26 General configuration of PLL 200 #0 settings

```

:
#define PLL200_0_TARGET_FREQ      (16000000ul) /* PLL Target Frequency. */
#define PLL200_1_TARGET_FREQ      (8000000ul) /* PLL Target Frequency. */
:
/** Wait time definition */
#define WAIT_FOR_STABILIZATION (10000ul) /* Define TIMEOUT Variable */
:
#define PLL_400M_0_PATH_NO        (1ul) /* Define PLL number.*/
#define PLL_400M_1_PATH_NO        (2ul) /* Define PLL number. */
#define PLL_200M_0_PATH_NO        (3ul) /* Define PLL number. */
#define PLL_200M_1_PATH_NO        (4ul) /* Define PLL number. */
#define BYPASSED_PATH_NO          (5ul) /* Define PLL number. */
:
/** Parameters for Clock Configuration */
cy_stc_pll_config_t g_pll200_0_Config = /* PLL200 #0 Configuration. */
{
    .inputFreq  = PATH_SOURCE_CLOCK_FREQ,          // ECO: 16MHz
    .outputFreq  = PLL200_0_TARGET_FREQ,           // target PLL output
    .lfMode      = false,                          // VCO frequency is [200MHz, 400MHz]
    .outputMode  = CY_SYSClk_FLLPLL_OUTPUT_AUTO,
};
:
int main(void)
{
:
    /* Enable interrupt */
    __enable_irq();

    /* Set Clock Configuring registers */
    AllClockConfiguration(); /* PLL200 #0 setting. See Code Listing 27. */
:
    /* Please check clock output using oscilloscope after CPU reached here. */
    for(;;);
}

```

4 Configuration of FLL and PLL

Code Listing 27 AllClockConfiguration() function

```
static void AllClockConfiguration(void)
{
:
    /**** PLL200M#0(PATH3) source setting ***/
    {
:
        status = Cy_SysClk_PllConfigure(PLL_200M_0_PATH_NO , &g_pll200_0_Config); /* PLL200
Configuration. See Code Listing 28. */
        CY_ASSERT(status == CY_SYSCLK_SUCCESS);

        status = Cy_SysClk_PllEnable(PLL_200M_0_PATH_NO, WAIT_FOR_STABILIZATION); /* PLL200
Enable. See Code Listing 32. */
        CY_ASSERT(status == CY_SYSCLK_SUCCESS);
:
    }
    return;
}
```


4 Configuration of FLL and PLL

Code Listing 28 Cy_SysClk_PllConfigure() function

```

cy_en_sysclk_status_t Cy_SysClk_PllConfigure(uint32_t clkPath, const cy_stc_pll_config_t
*config)
{
    /* check for error */
    uint32_t pllNo;
    cy_en_sysclk_status_t status = Cy_SysClk_GetPllNo(clkPath, &pllNo);
    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    if (SRSS->unCLK_PLL_CONFIG[pllNo].stcField.u1ENABLE != 0ul) /* 1 = enabled */ /* (8) Check if
PLL200 is already enabled. */
    {
        return (CY_SYSCLK_INVALID_STATE);
    }

    /* invalid output frequency */
    cy_stc_pll_manual_config_t manualConfig = {0ul};
    const cy_stc_pll_limitation_t* pllLim = (config->lfMode) ? &g_limPllLF : &g_limPllNORM;
    status = Cy_SysClk_PllCalucDividers(config->inputFreq,
                                        config->outputFreq, /* PLL200 Calculating Dividers
Settings. See Code Listing 31. */
                                        pllLim,
                                        0ul, // Frac bit num
                                        &manualConfig.feedbackDiv,
                                        &manualConfig.referenceDiv,
                                        &manualConfig.outputDiv,
                                        NULL
                                        );

    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    /* configure PLL based on calculated values */
    manualConfig.lfMode = config->lfMode;
    manualConfig.outputMode = config->outputMode;

    status = Cy_SysClk_PllManualConfigure(clkPath, &manualConfig); /* PLL200 Manual Configuring
Settings. See Code Listing 29. */
    return (status);
}

```

4 Configuration of FLL and PLL

Code Listing 29 Cy_SysClk_PllManualConfigure() function

```

cy_en_sysclk_status_t Cy_SysClk_PllManualConfigure(uint32_t clkPath, const
cy_stc_pll_manual_config_t *config)
{
    /* check for error */
    uint32_t pllNo;
    cy_en_sysclk_status_t status = Cy_SysClk_GetPllNo(clkPath, &pllNo);
    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    /* valid divider bitfield values */
    if((config->outputDiv < MIN_OUTPUT_DIV) || (MAX_OUTPUT_DIV < config->outputDiv))
    {
        return(CY_SYSCLK_BAD_PARAM);
    }

    if((config->referenceDiv < MIN_REF_DIV) || (MAX_REF_DIV < config->referenceDiv))
    {
        return(CY_SYSCLK_BAD_PARAM);
    }

    if((config->feedbackDiv < (config->lfMode ? MIN_FB_DIV_LF : MIN_FB_DIV_NORM)) ||
        ((config->lfMode ? MAX_FB_DIV_LF : MAX_FB_DIV_NORM) < config->feedbackDiv))
    {
        return(CY_SYSCLK_BAD_PARAM);
    }

    un_CLK_PLL_CONFIG_t tempClkPLLConfigReg;
    tempClkPLLConfigReg.u32Register = SRSS->unCLK_PLL_CONFIG[pllNo].u32Register;
    if (tempClkPLLConfigReg.stcField.u1ENABLE != 0u1) /* 1 = enabled */
    {
        return(CY_SYSCLK_INVALID_STATE);
    }

    /* no errors */
    /* If output mode is bypass (input routed directly to output), then done.
    The output frequency equals the input frequency regardless of the frequency parameters.
    */
    if (config->outputMode != CY_SYSCLK_FLLPLL_OUTPUT_INPUT) /* (9) PLL200 Configuration. */
    {
        tempClkPLLConfigReg.stcField.u7FEEDBACK_DIV = (uint32_t)config->feedbackDiv;
        tempClkPLLConfigReg.stcField.u5REFERENCE_DIV = (uint32_t)config->referenceDiv;
        tempClkPLLConfigReg.stcField.u5OUTPUT_DIV = (uint32_t)config->outputDiv;
        tempClkPLLConfigReg.stcField.u1PLL_LF_MODE = (uint32_t)config->lfMode;
    }
    tempClkPLLConfigReg.stcField.u2BYPASS_SEL = (uint32_t)config->outputMode;

    SRSS->unCLK_PLL_CONFIG[pllNo].u32Register = tempClkPLLConfigReg.u32Register;

```

4 Configuration of FLL and PLL

```
    return (CY_SYSCLK_SUCCESS);
}
```

Code Listing 30 Cy_SysClk_GetPllNo() function

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_GetPllNo(uint32_t pathNo, uint32_t* pllNo)
{
    /* check for error */
    if ((pathNo <= SRSS_NUM_PLL400M) || (pathNo > (SRSS_NUM_PLL400M + SRSS_NUM_PLL)))
    {
        /* invalid clock path number */
        return(CY_SYSCLK_BAD_PARAM);
    }

    *pllNo = pathNo - (uint32_t)(SRSS_NUM_PLL400M + 1u);
    return(CY_SYSCLK_SUCCESS);
}
```

4 Configuration of FLL and PLL

Code Listing 31 Cy_SysClk_PllCalucDividers() function

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PllCalucDividers(uint32_t inFreq,
                                                                uint32_t targetOutFreq,
                                                                const cy_stc_pll_limitation_t*
lim,
                                                                uint32_t fracBitNum,
                                                                uint32_t* feedBackDiv,
                                                                uint32_t* refDiv,
                                                                uint32_t* outputDiv,
                                                                uint32_t* feedBackFracDiv)
{
    uint64_t errorMin = 0xFFFFFFFFFFFFFFFFull;

    if(feedBackDiv == NULL)
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    if((feedBackFracDiv == NULL) && (fracBitNum != 0ul))
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    if(refDiv == NULL)
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }
    if(outputDiv == NULL)
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    if ((targetOutFreq < lim->minFoutput) || (lim->maxFoutput < targetOutFreq))
    {
        return (CY_SYSCLOCK_BAD_PARAM);
    }

    /* REFERENCE_DIV selection */
    for (uint32_t i_refDiv = lim->minRefDiv; i_refDiv <= lim->maxRefDiv; i_refDiv++)
    {
        uint32_t fpd_roundDown = inFreq / i_refDiv;
        if (fpd_roundDown < lim->minFpd)
        {
            break;
        }

        uint32_t fpd_roundUp = CY_SYSCLOCK_DIV_ROUNDUP(inFreq, i_refDiv);
        if (lim->maxFpd < fpd_roundUp)
        {
            continue;
        }
    }
}
```

4 Configuration of FLL and PLL

```

/* OUTPUT_DIV selection */
for (uint32_t i_outDiv = lim->minOutputDiv; i_outDiv <= lim->maxOutputDiv; i_outDiv++)
{
    uint64_t tempVco = i_outDiv * targetOutFreq;

    if(tempVco < lim->minFvco)
    {
        continue;
    }
    else if(lim->maxFvco < tempVco)
    {
        break;
    }

    // (inFreq / refDiv) * feedBackDiv = Fvco
    // feedBackDiv = Fvco * refDiv / inFreq
    uint64_t tempFeedBackDivLeftShifted = ((tempVco << (uint64_t)fracBitNum) *
(uint64_t)i_refDiv) / (uint64_t)inFreq;
    uint64_t error = abs(((uint64_t)targetOutFreq << (uint64_t)fracBitNum) -
((uint64_t)inFreq * tempFeedBackDivLeftShifted / ((uint64_t)i_refDiv * (uint64_t)i_outDiv)));

    if (error < errorMin)
    {
        *feedBackDiv      = (uint32_t)(tempFeedBackDivLeftShifted >>
(uint64_t)fracBitNum);
        if(feedBackFracDiv != NULL)
        {
            if(fracBitNum == 0ul)
            {
                *feedBackFracDiv = 0ul;
            }
            else
            {
                *feedBackFracDiv = (uint32_t)(tempFeedBackDivLeftShifted & ((1ull <<
(uint64_t)fracBitNum) - 1ull));
            }
        }
        *refDiv          = i_refDiv;
        *outputDiv        = i_outDiv;
        errorMin          = error;
        if(errorMin == 0ull){break;}
    }
}
if(errorMin == 0ull){break;}

if(errorMin == 0xFFFFFFFFFFFFFFFFull)
{
    return (CY_SYSCLK_BAD_PARAM);
}
else
{
    return (CY_SYSCLK_SUCCESS);
}

```

4 Configuration of FLL and PLL

```
}
}
```

Code Listing 32 Cy_SysClk_PllEnable() function

```
cy_en_sysclk_status_t Cy_SysClk_PllEnable(uint32_t clkPath, uint32_t timeoutus)
{
    uint32_t pllNo;
    cy_en_sysclk_status_t status = Cy_SysClk_GetPllNo(clkPath, &pllNo);
    if(status != CY_SYSCLK_SUCCESS)
    {
        return(status);
    }

    /* first set the PLL enable bit */
    SRSS->unCLK_PLL_CONFIG[pllNo].stcField.u1ENABLE = 1ul; /* (10) Enable PLL200. */

    /* now do the timeout wait for PLL_STATUS, bit LOCKED */
    for (; (SRSS->unCLK_PLL_STATUS[pllNo].stcField.u1LOCKED == 0ul) && /* (11) Wait until
PLL200 is locked. */
        (timeoutus != 0ul); /* (12) Check Timeout. */
        timeoutus--)
    {
        Cy_SysLib_DelayUs(1u); /* Wait for 1 us. */
    }

    status = ((timeoutus == 0ul) ? CY_SYSCLK_TIMEOUT : CY_SYSCLK_SUCCESS);

    return (status);
}
```

5 Configuration of the internal clock

5 Configuration of the internal clock

This section explains how to configure the internal clocks as part of the clock system.

5.1 Configuring the CLK_PATHx

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

Figure 13 shows the generation diagram for CLK_PATHx.

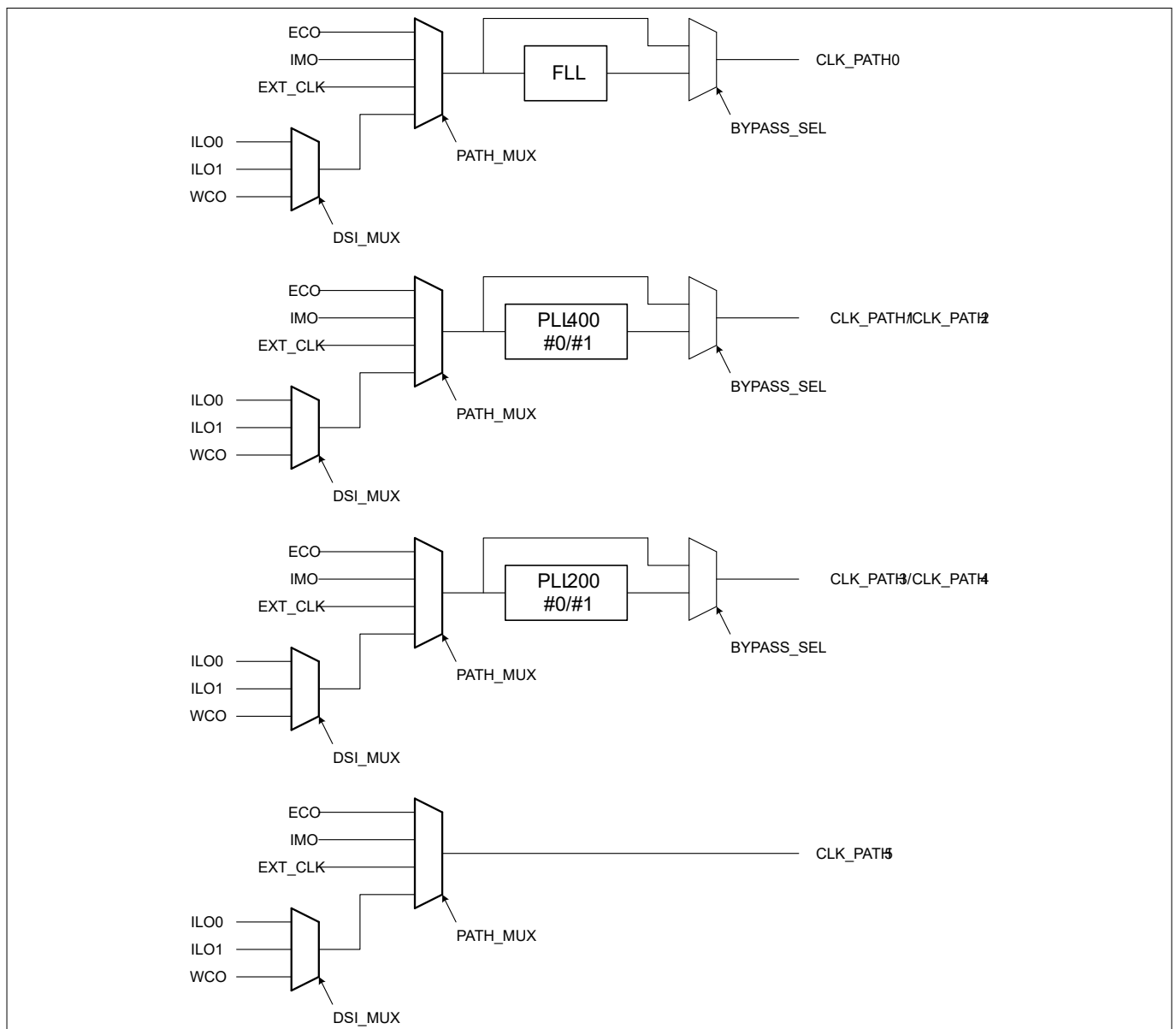


Figure 13 Generation diagram for CLK_PATHx

To configure CLK_PATHx, it is necessary to configure DSI_MUX and PATH_MUX. BYPASS_MUX is also required for CLK_PATHx. Table 11 shows the registers necessary for configuring CLK_PATHx. See the [architecture reference manual](#) for more details.

5 Configuration of the internal clock

Table 11 Configuring CLK_PATHx

Register name	Bit name	Value	Selected clock and item
CLK_PATH_SELECT	PATH_MUX[2:0]	0 (Default)	IMO
		1	EXT_CLK
		2	ECO
		4	DSI_MUX
		other	Reserved. Do not use
CLK_DSI_SELECT	DSI_MUX[4:0]	16	ILO0
		17	WCO
		20	ILO1
		Other	Reserved. Do not use
CLK_FLL_CONFIG3	BYPASS_SEL[29:28]	0 (Default)	AUTO ¹⁾
		1	LOCKED_OR_NOTHING ²⁾
		2	FLL_REF (bypass mode) ³⁾
		3	FLL_OUT ³⁾
CLK_PLL_CONFIG	BYPASS_SEL[29:28]	0 (Default)	AUTO ¹⁾
		1	LOCKED_OR_NOTHING ²⁾
		2	PLL_REF (bypass mode) ³⁾
		3	PLL_OUT ³⁾

5.2 Configuring the CLK_HF_x

CLK_HF_x (x=0,1,2,3,4,5,6,7) can be selected from any CLK_PATH_y (y=0,1,2,3,4,5). A predivider is available to divide the selected CLK_PATH_x. CLK_HF0 is always enabled because it is the source clock for the CPU cores. It is possible to disable CLK_HF_x.

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

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

Figure 14 shows the details of ROOT_MUX and the predivider.

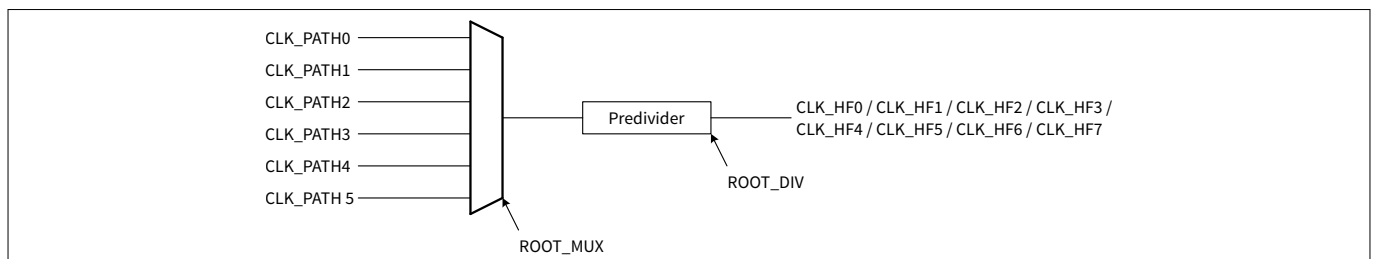


Figure 14 ROOT_MUX and predivider

- ¹ Switching automatically according to locked state.
- ² The clock is gated off, when unlocked.
- ³ In this mode, lock state is ignored.

5 Configuration of the internal clock

Table 12 shows the registers necessary for CLK_HF_x. See the [architecture reference manual](#) for more details.

Table 12 Configuring of CLK_HF_x (x = 0, 1, 2, 3, 4, 5, 6, 7)

Register name	Bit name	Value	Selected item
CLK_ROOT_SELECT	ROOT_MUX[3:0]	0	CLK_PATH0
		1	CLK_PATH1
		2	CLK_PATH2
		3	CLK_PATH3
		4	CLK_PATH4
		5	CLK_PATH5
		Other	Reserved. Do not use.
CLK_ROOT_SELECT	ROOT_DIV[1:0]	0	No division
		1	Divide clock by 2
		2	Divide clock by 4
		3	Divide clock by 8

5.3 Configuring the 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 WDT.

Figure 15 shows the details of LFCLK_SEL that CLK_LF is configured.

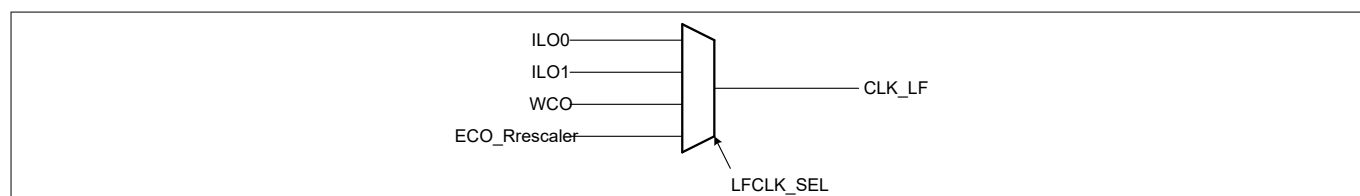


Figure 15 LFCLK_SEL

Table 13 shows the registers necessary for CLK_LF. See the [architecture reference manual](#) for more details.

Table 13 Configuring of CLK_LF

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

5 Configuration of the internal clock

5.4 Configuring the 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 the 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 the 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 the 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 the CLK_GR

The clock source of CLK_GP is CLK_PERI in Group 3, 4, 8 and CLK_HF2 in Group 5, 6, 9. Groups 3, 4, 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 the 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 16](#) shows the flow to set peripheral clock dividers. See the [architecture reference manual](#) for more details.

5 Configuration of the internal clock

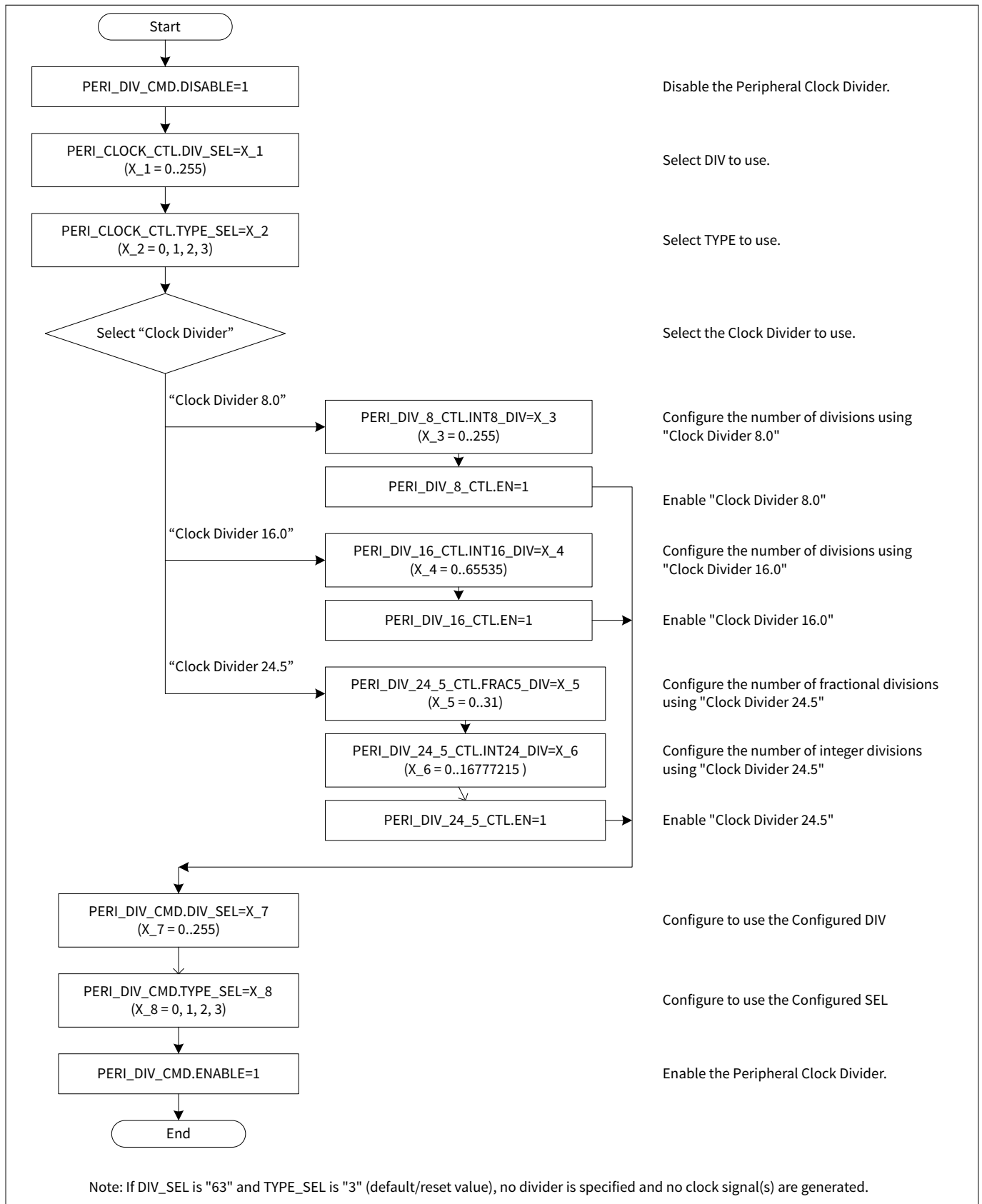


Figure 16 Procedure for setting to generate PCLK

5 Configuration of the internal clock

5.9.1 Example of PCLK setting

5.9.1.1 Use case

- 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: 38 (TCPWM0, Group#0, Counter#0)

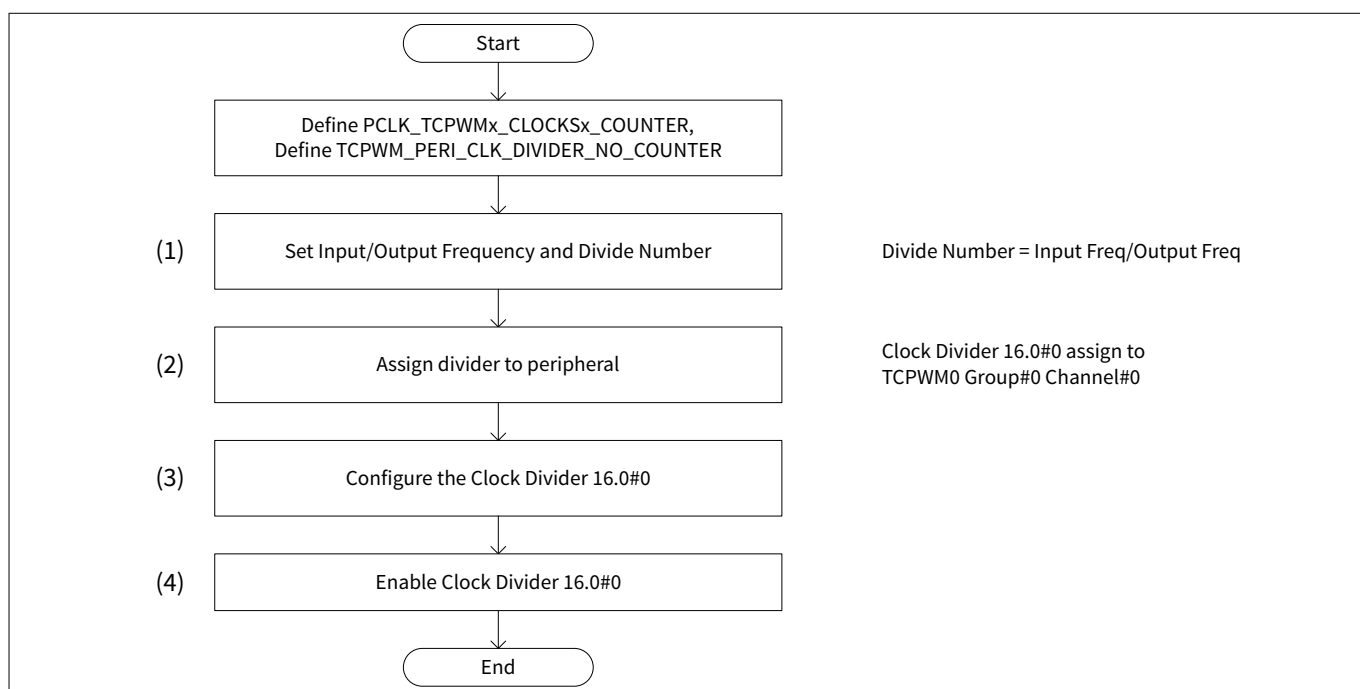


Figure 17 Example procedure for setting PCLK

5.9.1.2 Configuration

Table 14 lists the parameters and Table 15 lists the functions of the configuration part of in SDL for PCLK (Example of the TCPWM timer) settings.

Table 14 List of PCLK (Example of the TCPWM timer) settings parameters

Parameters	Description	Value
PCLK_TCPWMx_CLOCKSx_COUNTER	PCLK of TCPWM0	PCLK_TCPWM0_CLOCKS0 = 38ul
TCPWM_PERI_CLK_DIVIDER_NO_COUNTER	Number of dividers to be used	0ul

(table continues...)

5 Configuration of the internal clock

Table 14 (continued) List of PCLK (Example of the TCPWM timer) settings parameters

Parameters	Description	Value
CY_SYSCLK_DIV_16_BIT	Divider type CY_SYSCLK_DIV_8_BIT = 0u, 8-bit divider CY_SYSCLK_DIV_16_BIT = 1u, 16-bit divider CY_SYSCLK_DIV_16_5_BIT = 2u, 16.5-bit fractional divider CY_SYSCLK_DIV_24_5_BIT = 3u, 24.5-bit fractional divider	1ul
periFreq	Peripheral clock frequency	80000000ul (80 MHz)
targetFreq	Target clock frequency	2000000ul (2 MHz)
divNum	Divide number	periFreq/targetFreq

Table 15 List of PCLK (Example of the TCPWM timer) settings functions

Functions	Description	Value
Cy_SysClk_PeriphAssignDivider(IPblock, dividerType, dividerNum)	Assigns a programmable divider to a selected IP block (such as a TCPWM).	IPblock = PCLK_TCPWMx_CLOCKSx_COUNTER dividerType = CY_SYSCLK_DIV_16_BIT dividerNum = TCPWM_PERI_CLK_DIVIDER_NO_COUNTER
Cy_SysClk_PeriphSetDivider(dividerType, dividerNum, dividerValue)	Set peripheral divider	dividerType, = CY_SYSCLK_DIV_16_BIT dividerNum = TCPWM_PERI_CLK_DIVIDER_NO_COUNTER dividerValue = divNum-1ul
Cy_SysClk_PeriphEnableDivider(dividerType, dividerNum)	Enable peripheral divider	dividerType, = CY_SYSCLK_DIV_16_BIT dividerNum = TCPWM_PERI_CLK_DIVIDER_NO_COUNTER

5.9.2 Sample code (Example of the TCPWM timer)

There is a sample code as shown [Code Listing 33](#) to [Code Listing 36](#).

5 Configuration of the internal clock

Code Listing 33 General configuration of PCLK (Example of the TCPWM timer) settings

```

:
#define PCLK_TCPWMx_CLOCKSx_COUNTER      PCLK_TCPWM0_CLOCKS0      /* Define
PCLK_TCPWMx_CLOCKSx_COUNTER, Define TCPWM_PERI_CLK_DIVIDER_NO_COUNTER. */
#define TCPWM_PERI_CLK_DIVIDER_NO_COUNTER 0u
:
int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

    uint32_t periFreq = 8000000ul; /* (1) Set Input/Output Frequency and Divide Number. */
    uint32_t targetFreq = 2000000ul;
    uint32_t divNum = (periFreq / targetFreq); /* Calculation of division. */

    CY_ASSERT((periFreq % targetFreq) == 0ul); // inaccurate target clock
    Cy_SysClk_PeriphAssignDivider(PCLK_TCPWMx_CLOCKSx_COUNTER, CY_SYSCLK_DIV_16_BIT,
TCPWM_PERI_CLK_DIVIDER_NO_COUNTER); /* Peripheral Divider Assign setting. See Code Listing 34.
*/
    /* Sets the 16-bit divider */
    Cy_SysClk_PeriphSetDivider(CY_SYSCLK_DIV_16_BIT, TCPWM_PERI_CLK_DIVIDER_NO_COUNTER,
(divNum-1ul)); /* Peripheral Divider setting. See Code Listing 35. */
    Cy_SysClk_PeriphEnableDivider(CY_SYSCLK_DIV_16_BIT, TCPWM_PERI_CLK_DIVIDER_NO_COUNTER); /*
Peripheral Divider Enable setting. See Code Listing 36. */

    for(;;);
}

```

Code Listing 34 Cy_SysClk_PeriphAssignDivider() function

```

__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphAssignDivider(en_clk_dst_t ipBlock,
cy_en_divider_types_t dividerType, uint32_t dividerNum)
{
:

    un_PERI_CLOCK_CTL_t tempCLOCK_CTL_RegValue;
    tempCLOCK_CTL_RegValue.u32Register      = PERI->unCLOCK_CTL[ipBlock].u32Register; /* (2)
Assign Divider to Peripheral. */
    tempCLOCK_CTL_RegValue.stcField.u2TYPE_SEL = dividerType; /* (2) Assign Divider to
Peripheral. */
    tempCLOCK_CTL_RegValue.stcField.u8DIV_SEL = dividerNum; /* (2) Assign Divider to
Peripheral. */
    PERI->unCLOCK_CTL[ipBlock].u32Register      = tempCLOCK_CTL_RegValue.u32Register; /* (2)
Assign Divider to Peripheral. */

    return CY_SYSCLK_SUCCESS;
}

```

5 Configuration of the internal clock

Code Listing 35 Cy_SysClk_PeriphSetDivider() function

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphSetDivider(cy_en_divider_types_t
dividerType,
                                uint32_t dividerNum, uint32_t dividerValue)
{
    :
    if (dividerType == CY_SYSCLOCK_DIV_8_BIT)
    {
        :
    }
    else if (dividerType == CY_SYSCLOCK_DIV_16_BIT)
    {
        :
        PERI->unDIV_16_CTL[dividerNum].stcField.u16INT16_DIV = dividerValue; /* (3)
Division Setting to Clock Divider 16.0#0. */
        :
    }
    else
    { /* return bad parameter */
        return CY_SYSCLOCK_BAD_PARAM;
    }

    return CY_SYSCLOCK_SUCCESS;
}
```

Code Listing 36 Cy_SysClk_PeriphEnableDivider() function

```
__STATIC_INLINE cy_en_sysclk_status_t Cy_SysClk_PeriphEnableDivider(cy_en_divider_types_t
dividerType, uint32_t dividerNum) /* (4) Enable Clock Divider 16#0. */
{
    :
    /* specify the divider, make the reference = clk_peri, and enable the divider */
    un_PERI_DIV_CMD_t tempDIV_CMD_RegValue;
    tempDIV_CMD_RegValue.u32Register = PERI->unDIV_CMD.u32Register;
    tempDIV_CMD_RegValue.stcField.u1ENABLE = 1ul;
    tempDIV_CMD_RegValue.stcField.u2PA_TYPE_SEL = 3ul;
    tempDIV_CMD_RegValue.stcField.u8PA_DIV_SEL = 0xFFul;
    tempDIV_CMD_RegValue.stcField.u2TYPE_SEL = dividerType; /* Set divider Type Select. */
    tempDIV_CMD_RegValue.stcField.u8DIV_SEL = dividerNum; /* Set divider number. */
    PERI->unDIV_CMD.u32Register = tempDIV_CMD_RegValue.u32Register;

    (void)PERI->unDIV_CMD; /* dummy read to handle buffered writes */

    return CY_SYSCLOCK_SUCCESS;
}
```

5 Configuration of the internal clock

5.10 Setting ECO_Prescaler

5.10.1 Operation overview

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 18 shows the steps to enable ECO_Prescaler as follows. For details on ECO_Prescaler, see [architecture reference manual](#) and [registers reference manual](#).

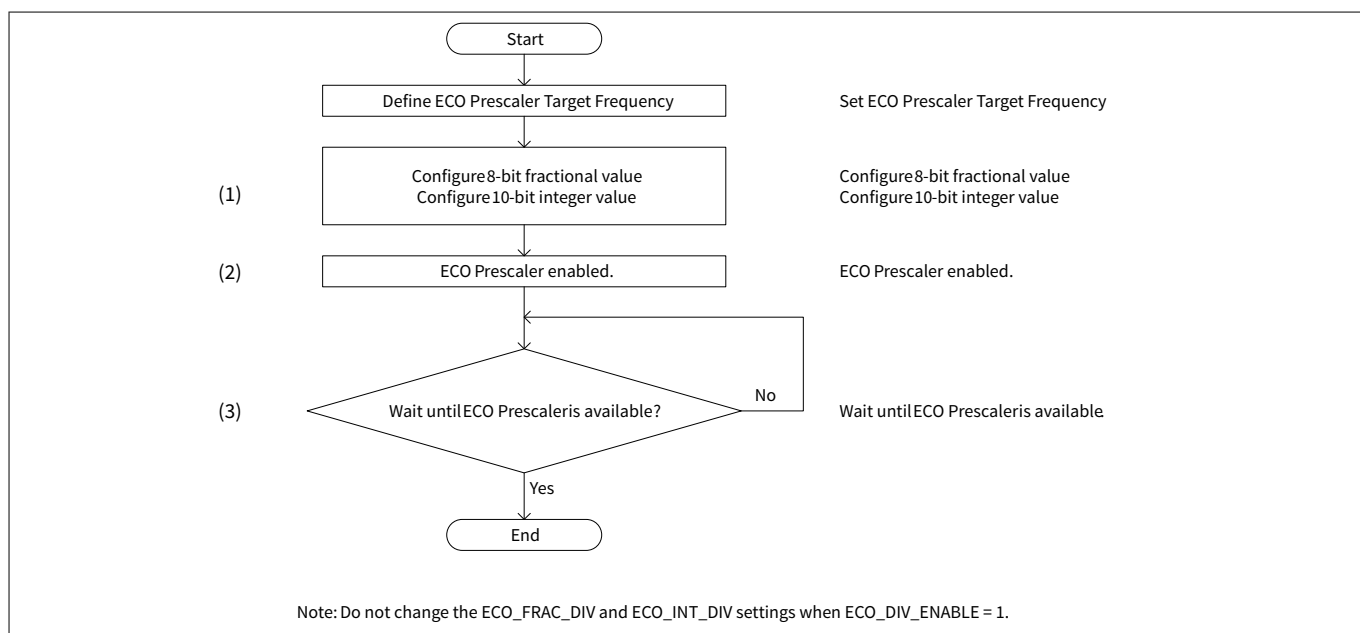


Figure 18 Enabling ECO_Prescaler

Figure 19 shows the steps to disable ECO_Prescaler. For details on ECO_Prescaler, see [architecture reference manual](#) and [registers reference manual](#).

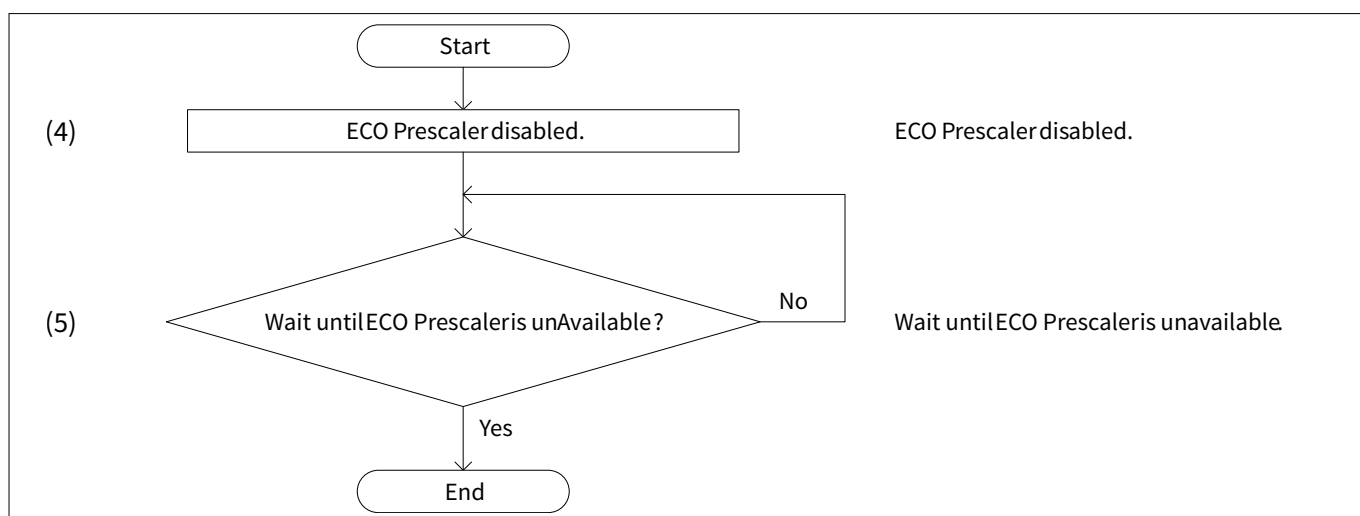


Figure 19 Disabling ECO_Prescaler

5.10.2 Use case

- Input clock frequency: 16 MHz
- ECO prescaler target frequency: 1.234567 MHz

5 Configuration of the internal clock

5.10.3 Configuration

Table 16 lists the parameters and Table 17 lists the functions of the configuration part of in SDL for ECO prescaler settings.

Table 16 List of ECO prescaler settings parameters

Parameters	Description	Value
ECO_PRESCALER_TARGET_FREQ	ECO prescaler target frequency	1234567ul
WAIT_FOR_STABILIZATION	Waiting for stabilization	10000ul
CLK_FREQ_ECO	ECO clock frequency	16000000ul (16 MHz)
PATH_SOURCE_CLOCK_FREQ	PATH source clock frequency	CLK_FREQ_ECO

Table 17 List of ECO prescaler settings functions

Functions	Description	Value
AllClockConfiguration()	Clock configuration	–
Cy_SysClk_SetEco Prescale(Inclk, Targetclk)	Set ECO frequency and target frequency	Inclk = PATH_SOURCE_CLOCK_FREQ, Targetclk = ECO_PRESCALER_TARGET_FREQ
Cy_SysClk_EcoPrescale Enable(Timeout value)	Set ECO prescaler enable and timeout value	Timeout value = WAIT_FOR_STABILIZATION
Cy_SysClk_SetEco PrescaleManual (divInt, divFact)	divInt: 10-bit integer value allows for ECO frequencies. divFrac: 8-bit fractional value	–
Cy_SysClk_GetEco PrescaleStatus	Check prescaler status	–

5.10.4 Sample code

There is a sample code as shown Code Listing 37 to Code Listing 43.

5 Configuration of the internal clock

Code Listing 37 General configuration of ECO prescaler settings

```
#define ECO_PRESCALER_TARGET_FREQ (1234567ul) /* Define ECO Prescaler Target Frequency */
#define CLK_FREQ_ECO (16000000ul) /* Define ECO Clock Frequency */
#define PATH_SOURCE_CLOCK_FREQ CLK_FREQ_ECO

/** Wait time definition */
#define WAIT_FOR_STABILIZATION (10000ul) /* Define TIMEOUT Variable */

int main(void)
{
:
    /* Set Clock Configuring registers */
    AllClockConfiguration(); /* ECO Prescaler setting. See Code Listing 38. */
:
    /* Please check clock output using oscilloscope after CPU reached here. */
    for(;;);
}
```

Code Listing 38 AllClockConfiguration() function

```
static void AllClockConfiguration(void)
{
    /*** ECO prescaler setting ***/
    {
:
        cy_en_sysclk_status_t ecoPreStatus;

        ecoPreStatus = Cy_SysClk_SetEcoPrescale(CLK_FREQ_ECO, ECO_PRESCALER_TARGET_FREQ); /*
ECO Prescaler setting. See Code Listing 39. */
        CY_ASSERT(ecoPreStatus == CY_SYSCLK_SUCCESS);

        ecoPreStatus = Cy_SysClk_EcoPrescaleEnable(WAIT_FOR_STABILIZATION); /* ECO Prescaler
enable. See Code Listing 41. */
        CY_ASSERT(ecoPreStatus == CY_SYSCLK_SUCCESS);
    }

    return;
}
```

5 Configuration of the internal clock

Code Listing 39 Cy_SysClk_SetEcoPrescale() function

```
cy_en_sysclk_status_t Cy_SysClk_SetEcoPrescale(uint32_t ecoFreq, uint32_t targetFreq)
{
    // Frequency of ECO (4MHz ~ 33.33MHz) might exceed 32bit value if shifted 8 bit.
    // So, it uses 64 bit data for fixed point operation.
    // Lowest 8 bit are fractional value. Next 10 bit are integer value.
    uint64_t fixedPointEcoFreq = ((uint64_t)ecoFreq << 8u11);
    uint64_t fixedPointDivNum64;
    uint32_t fixedPointDivNum;

    // Calculate divider number
    fixedPointDivNum64 = fixedPointEcoFreq / (uint64_t)targetFreq;

    // Dividing num should be larger 1.0, and smaller than maximum of 10bit number.
    if((fixedPointDivNum64 < 0x100u11) && (fixedPointDivNum64 > 0x40000u11))
    {
        return CY_SYSCLK_BAD_PARAM;
    }

    fixedPointDivNum = (uint32_t)fixedPointDivNum64;

    Cy_SysClk_SetEcoPrescaleManual(
        (((fixedPointDivNum & 0x0003FF00u1) >> 8u1) - 1u1), /*
Configure ECO Prescaler. See Code Listing 40. */
        (fixedPointDivNum & 0x000000FFu1)
    );

    return CY_SYSCLK_SUCCESS;
}
```

Code Listing 40 Cy_SysClk_SetEcoPrescaleManual() function

```
__STATIC_INLINE void Cy_SysClk_SetEcoPrescaleManual(uint16_t divInt, uint8_t divFract)
{
    un_CLK_ECO_PRESCALE_t tempRegEcoPrescale;
    tempRegEcoPrescale.u32Register = SRSS->unCLK_ECO_PRESCALE.u32Register; /* (1)
Configure ECO Prescaler. */
    tempRegEcoPrescale.stcField.u10ECO_INT_DIV = divInt;
    tempRegEcoPrescale.stcField.u8ECO_FRAC_DIV = divFract;
    SRSS->unCLK_ECO_PRESCALE.u32Register = tempRegEcoPrescale.u32Register;

    return;
}
```

5 Configuration of the internal clock

Code Listing 41 Cy_SysClk_EcoPrescaleEnable() function

```
cy_en_sysclk_status_t Cy_SysClk_EcoPrescaleEnable(uint32_t timeoutus)
{
    // Send enable command
    SRSS->unCLK_ECO_CONFIG.stcField.u1ECO_DIV_ENABLE = 1u1; /* (2) Enable ECO Prescaler */

    // Wait eco prescaler get enabled
    while(CY_SYSCLOCK_ECO_PRESCALE_ENABLE != Cy_SysClk_GetEcoPrescaleStatus()) /* (3) Wait until
    ECO Prescaler is available. */
    {
        if(0u1 == timeoutus)
        {
            return CY_SYSCLOCK_TIMEOUT;
        }

        Cy_SysLib_DelayUs(1u);

        timeoutus--;
    }

    return CY_SYSCLOCK_SUCCESS;
}
```

Code Listing 42 Cy_SysClk_GetEcoPrescaleStatus() function

```
__STATIC_INLINE cy_en_eco_prescale_enable_t Cy_SysClk_GetEcoPrescaleStatus(void)
{
    return (cy_en_eco_prescale_enable_t)(SRSS->unCLK_ECO_PRESCALE.stcField.u1ECO_DIV_ENABLED); /* Check prescaler status. */
}
```

If you want to disable the ECO prescaler, set the wait time in the same way as the function above, then call the next function.

5 Configuration of the internal clock

Code Listing 43 Cy_SysClk_EcoPrescaleDisable() function

```
cy_en_sysclk_status_t Cy_SysClk_EcoPrescaleDisable(uint32_t timeoutus)
{
    // Send disable command
    SRSS->unCLK_ECO_CONFIG.stcField.u1ECO_DIV_DISABLE = 1ul; /* (4) Disable ECO Prescaler. */

    // Wait eco prescaler actually get disabled
    while(CY_SYSCLK_ECO_PRESCALE_DISABLE != Cy_SysClk_GetEcoPrescaleStatus()) /* (5) Wait until
    ECO Prescaler is unavailable. */
    {
        if(0ul == timeoutus)
        {
            return CY_SYSCLK_TIMEOUT;
        }

        Cy_SysLib_DelayUs(1u);

        timeoutus--;
    }

    return CY_SYSCLK_SUCCESS;
}
```

6 Supplementary information

6 Supplementary information

6.1 Input clocks in peripheral functions

Table 18 to Table 27 lists the clock input to each peripheral function. For detailed values of PCLK, see the “Peripheral clocks” section of the [datasheet](#).

Table 18 Clock input to TCPWM[0]

Peripheral function	Operation clock	Channel clock
TCPWM[0]	CLK_GR3 (Group 3)	PCLK (PCLK_TCPWM0_CLOCKSx, x = 0–2)
		PCLK (PCLK_TCPWM0_CLOCKSy, y = 256–268)
		PCLK (PCLK_TCPWM0_CLOCKSz, z = 512–514)

Table 19 Clock input to CAN FD

Peripheral function	Operation clock (clk_sys (hclk))	Channel clock (clk_can (cclk))
CAN FD0	CLK_GR5 (Group 5)	Ch0: PCLK (PCLK_CANFD0_CLOCK_CANFD0)
		Ch1: PCLK (PCLK_CANFD0_CLOCK_CANFD1)
		Ch2: PCLK (PCLK_CANFD0_CLOCK_CANFD2)
		Ch3: PCLK (PCLK_CANFD0_CLOCK_CANFD3)
		Ch4: PCLK (PCLK_CANFD0_CLOCK_CANFD4)
CAN FD1		Ch0: PCLK (PCLK_CANFD1_CLOCK_CANFD0)
		Ch1: PCLK (PCLK_CANFD1_CLOCK_CANFD1)
		Ch2: PCLK (PCLK_CANFD1_CLOCK_CANFD2)
		Ch3: PCLK (PCLK_CANFD1_CLOCK_CANFD3)
		Ch4: PCLK (PCLK_CANFD1_CLOCK_CANFD4)

Table 20 Clock input to LIN

Peripheral function	Operation clock	Channel clock (clk_lin_ch)
LIN	CLK_GR5 (Group 5)	Ch0: PCLK (PCLK_LIN_CLOCK_CH_EN0)
		Ch1: PCLK (PCLK_LIN_CLOCK_CH_EN1)
		Ch2: PCLK (PCLK_LIN_CLOCK_CH_EN2)
		Ch3: PCLK (PCLK_LIN_CLOCK_CH_EN3)
		Ch4: PCLK (PCLK_LIN_CLOCK_CH_EN4)
		Ch5: PCLK (PCLK_LIN_CLOCK_CH_EN5)
		Ch6: PCLK (PCLK_LIN_CLOCK_CH_EN6)
		Ch7: PCLK (PCLK_LIN_CLOCK_CH_EN7)
		Ch8: PCLK (PCLK_LIN_CLOCK_CH_EN8)
		Ch9: PCLK (PCLK_LIN_CLOCK_CH_EN9)

(table continues...)

6 Supplementary information

Table 20 (continued) Clock input to LIN

Peripheral function	Operation clock	Channel clock (clk_lin_ch)
		Ch10: PCLK (PCLK_LIN_CLOCK_CH_EN10)
		Ch11: PCLK (PCLK_LIN_CLOCK_CH_EN11)
		Ch12: PCLK (PCLK_LIN_CLOCK_CH_EN12)
		Ch13: PCLK (PCLK_LIN_CLOCK_CH_EN13)
		Ch14: PCLK (PCLK_LIN_CLOCK_CH_EN14)
		Ch15: PCLK (PCLK_LIN_CLOCK_CH_EN15)
		Ch16: PCLK (PCLK_LIN_CLOCK_CH_EN16)
		Ch17: PCLK (PCLK_LIN_CLOCK_CH_EN17)
		Ch18: PCLK (PCLK_LIN_CLOCK_CH_EN18)
		Ch19: PCLK (PCLK_LIN_CLOCK_CH_EN19)

Table 21 Clock input to SCB

Peripheral function	Operation clock	Channel clock
SCB0	CLK_GR6 (Group 6)	PCLK (PCLK_SCB0_CLOCK)
SCB1		PCLK (PCLK_SCB1_CLOCK)
SCB2		PCLK (PCLK_SCB2_CLOCK)
SCB3		PCLK (PCLK_SCB3_CLOCK)
SCB4		PCLK (PCLK_SCB4_CLOCK)
SCB5		PCLK (PCLK_SCB5_CLOCK)
SCB6		PCLK (PCLK_SCB6_CLOCK)
SCB7		PCLK (PCLK_SCB7_CLOCK)
SCB8		PCLK (PCLK_SCB8_CLOCK)
SCB9		PCLK (PCLK_SCB9_CLOCK)
SCB10		PCLK (PCLK_SCB10_CLOCK)

Table 22 Clock input to SAR ADC

Peripheral function	Operation clock	Unit clock
SAR ADC	CLK_GR9 (Group 9)	Unit0: PCLK (PCLK_PASS_CLOCK_SAR0)
		Unit1: PCLK (PCLK_PASS_CLOCK_SAR1)
		Unit2: PCLK (PCLK_PASS_CLOCK_SAR2)

Table 23 Clock input to TCPWM[1]

Peripheral function	Operation clock	Channel clock
TCPWM[1]	CLK_GR3 (Group 3)	PCLK (PCLK_TCPWM1_CLOCKSx, x = 0–83)
		PCLK (PCLK_TCPWM1_CLOCKSy, y = 256–267)
		PCLK (PCLK_TCPWM1_CLOCKSz, z = 512–524)

6 Supplementary information

Table 24 Clock input to FLEX-RAY

Peripheral function	Operation clock	Channel clock
FLEX-RAY	CLK_GR5 (Group 5)	PCLK (PCLK_FLEXRAY0_CLK_FLEXRAY)

Table 25 Clock input to SMIF

Peripheral function	“clk_slow” domain(XIP AHB-Lite Interface0)	“clk_mem” domain(XIP AHB interface)	“clk_sys” domain(MMIO AHB-Lite interface)	“clk_if” domain
SMIF	clk_slow	clk_mem	CLK_GR4	CLK_HF6

Table 26 Clock input to SDHC

Peripheral function	CLK_SLOW	CLK_SYS	CLK_HF[i]
SDHC	clk_slow	CLK_GR4	CLK_HF6

Table 27 Clock input to AUDIOSS

Peripheral function	clk_sys_i2s	clk_audio_i2s
AUDIOSS	CLK_GR8	CLK_HF5

Note: For the clocks of Ethernet, see the [architecture reference manual](#).

6.2 Use case of clock calibration counter function

6.2.1 How to use the clock calibration counter

6.2.1.1 Operation 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 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:

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

Equation 1 Example equation

Figure 20 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 [Setting ILO0/ILO1](#) and [Setting the ECO](#).

6 Supplementary information

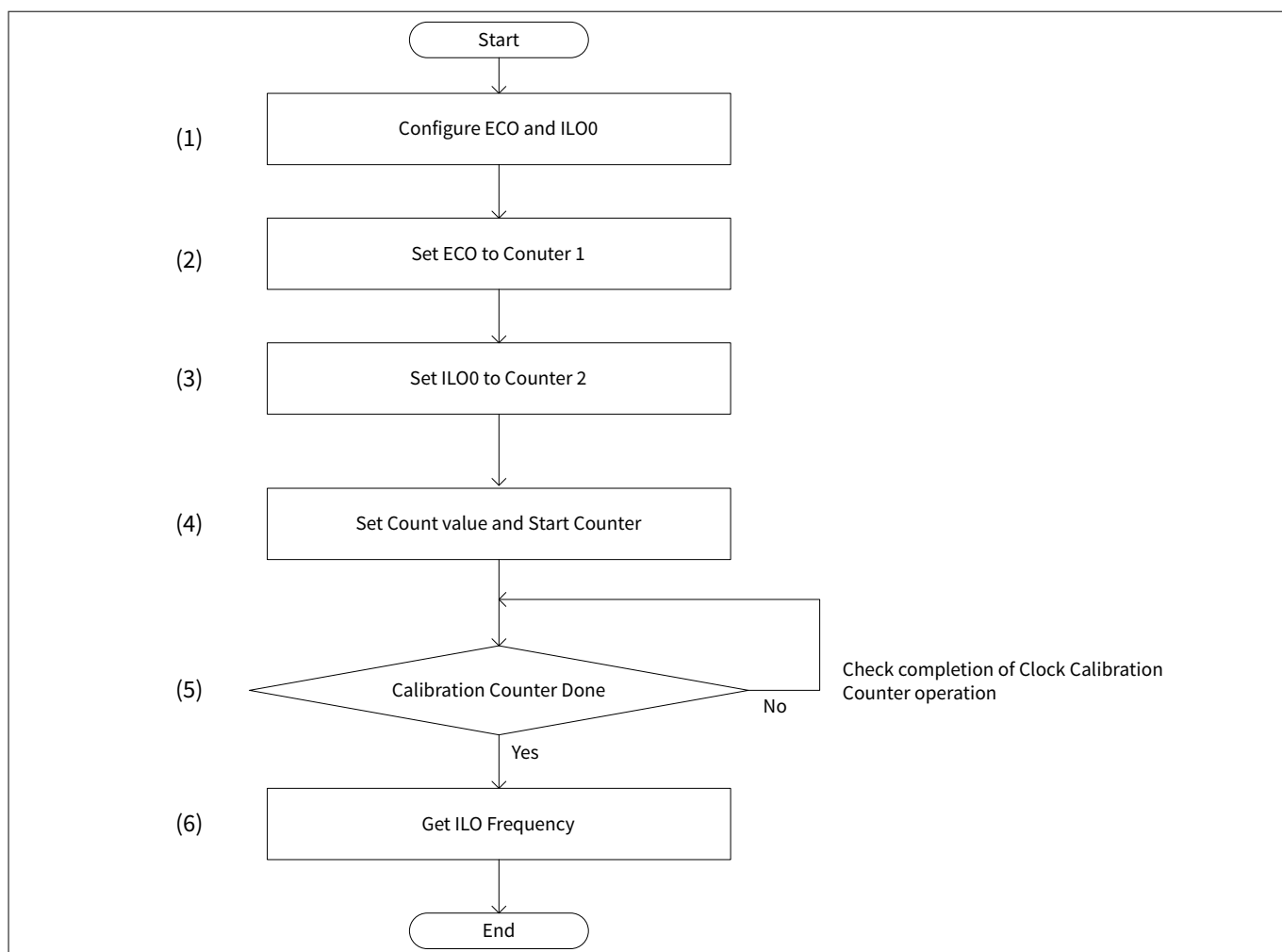


Figure 20 Example of clock calibration counter with ILO0 and ECO

6.2.1.2 Use case

- Measurement clock: ILO0 clock frequency 32.768 kHz
- Reference clock: ECO clock frequency 16 MHz
- Reference clock count value: 40000ul

6.2.1.3 Configuration

Table 28 lists the parameters and Table 29 lists the functions of the configuration part of in SDL for clock calibration counter with ILO0 and ECO settings.

Table 28 List of clock calibration counter with ILO0 and ECO settings parameters

Parameters	Description	Value
ILO_0	Define ILO_0 setting parameter	0ul
ILO_1	Define ILO_1 setting parameter	1ul
ILONo	Define measurement clock	ILO_0

(table continues...)

6 Supplementary information

Table 28 (continued) List of clock calibration counter with ILO0 and ECO settings parameters

Parameters	Description	Value
clockMeasuredInfo[].name	Measurement clock	CY_SYSCLK_MEAS_CLK_ILO0 = 1ul
clockMeasuredInfo[].measuredFreq	Store measurement clock frequency	–
counter1	Reference clock count value	40000ul
CLK_FREQ_ECO	ECO clock frequency	16000000ul (16 MHz)

Table 29 List of clock calibration counter with ILO0 and ECO settings functions

Functions	Description	Value
GetILOClockFreq()	Get ILO 0 frequency	–
Cy_SysClk_StartClk MeasurementCounters (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_ClkMeasurement CountersDone()	Check if the counter measurement is done	–
Cy_SysClk_ClkMeasurement CountersGetFreq(MesauredFreq, refClkFreq)	Get measurement clock frequency MesauredFreq - Stored measurement clock frequency refClkFreq - Reference clock frequency	MesauredFreq = clockMeasuredInfo[].measuredFreq refClkFreq = CLK_FREQ_ECO

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

There is a sample code as shown [Code Listing 44](#).

6 Supplementary information

Code Listing 44 General configuration of clock calibration counter with ILO0 and ECO settings

```
#define CY_SYSClk_DIV_ROUND(a, b) (((a) + ((b) / 2u1)) / (b)) /* Define CY_SYSClk_DIV_ROUND
function. */

#define ILO_0    0u1
#define ILO_1    1u1 /* Define measurement clock (ILO0). */
#define ILONo    ILO_0
#define CLK_FREQ_ECO          (16000000u1)

int32_t ILOFreq;

stc_clock_measure clockMeasuredInfo[] =
{
    #if(ILONo == ILO_0)
        { .name = CY_SYSClk_MEAS_CLK_ILO0,      .measuredFreq= 0u1},
    #else
        { .name = CY_SYSClk_MEAS_CLK_ILO1,      .measuredFreq= 0u1},
    #endif
};

int main(void)
{
    :

    /* Enable interrupt */
    __enable_irq();

    /* Set Clock Configuring registers */
    AllClockConfiguration(); /* (1) ECO and ILO0 setting. See Setting the ECO and Setting ILO0/
ILO1 */

    /* return: Frequency of ILO */
    ILOFreq = GetILOClockFreq(); /* Get Clock Frequency. See Code Listing 45 */

    /* Please check clock output using oscilloscope after CPU reached here. */
    for(;;);
}
```

6 Supplementary information

Code Listing 45 GetILOClockFreq() function

```
uint32_t GetILOClockFreq(void)
{
    uint32_t counter1 = 40000ul;

    if((SRSS->unCLK_ECO_STATUS.stcField.u1ECO_OK == 0ul) || (SRSS->unCLK_ECO_STATUS.stcField.u1ECO_READY == 0ul)) /* Check ECO status. */
    {
        while(1);
    }

    cy_en_sysclk_status_t status;
    status = Cy_SysClk_StartClkMeasurementCounters(CY_SYSClk_MEAS_CLK_ECO, counter1,
clockMeasuredInfo[0].name); /* Start Clock Measurement Counter. See Code Listing 46. */
    CY_ASSERT(status == CY_SYSClk_SUCCESS);

    while(Cy_SysClk_ClkMeasurementCountersDone() == false); /* Check if the Counter Measurement
is done. See Code Listing 47. */

    status = Cy_SysClk_ClkMeasurementCountersGetFreq(&clockMeasuredInfo[0].measuredFreq,
CLK_FREQ_ECO); /* Get ILO frequency. See Code Listing 48. */
    CY_ASSERT(status == CY_SYSClk_SUCCESS);

:

    uint32_t Frequency = clockMeasuredInfo[0].measuredFreq;
    return (Frequency);
}
```

6 Supplementary information

Code Listing 46 Cy_SysClk_StartClkMeasurementCounters() function

```
cy_en_sysclk_status_t Cy_SysClk_StartClkMeasurementCounters(cy_en_meas_clks_t clock1, uint32_t
count1, cy_en_meas_clks_t clock2)
{
    cy_en_sysclk_status_t rtnval = CY_SYSClk_INVALID_STATE;

:
    if (!preventCounting /* don't start a measurement if about to enter DeepSleep mode */ ||
        SRSS->unCLK_CAL_CNT1.stcField.u1CAL_COUNTER_DONE != 0u/*1 = done*/)
    {
:
        SRSS->unCLK_OUTPUT_FAST.stcField.u4FAST_SEL0 = (uint32_t)clock1; /* (2) Setting the
reference clock (ECO). */
:
        SRSS->unCLK_OUTPUT_SLOW.stcField.u4SLOW_SEL1 = (uint32_t)clock2; /* (3) Setting the measurement
clock (ILO0). */
        SRSS->unCLK_OUTPUT_FAST.stcField.u4FAST_SEL1 = 7u; /*slow_sel1 output*/;
:
        rtnval = CY_SYSClk_SUCCESS;

        /* Save this input parameter for use later, in other functions.
        No error checking is done on this parameter.*/
        clk1Count1 = count1;

        /* Counting starts when counter1 is written with a nonzero value. */
        SRSS->unCLK_CAL_CNT1.stcField.u24CAL_COUNTER1 = clk1Count1; /* (4) Set Count value and
Start Counter. */
:
        return (rtnval);
    }
}
```

Code Listing 47 Cy_SysClk_ClkMeasurementCountersDone() function

```
__STATIC_INLINE bool Cy_SysClk_ClkMeasurementCountersDone(void)
{
    return (bool)(SRSS->unCLK_CAL_CNT1.stcField.u1CAL_COUNTER_DONE); /* 1 = done */ /* (5) Check
completion of Clock Calibration Counter Operation. */
}
```

6 Supplementary information

Code Listing 48 Cy_SysClk_ClkMeasurementCountersGetFreq() function

```

cy_en_sysclk_status_t Cy_SysClk_ClkMeasurementCountersGetFreq(uint32_t *measuredFreq, uint32_t
refClkFreq)
{
    if(SRSS->unCLK_CAL_CNT1.stcField.u1CAL_COUNTER_DONE != 1u1)
    {
        return(CY_SYSClk_INVALID_STATE);
    }

    if(clk1Count1 == 0u1)
    {
        return(CY_SYSClk_INVALID_STATE);
    }

    volatile uint64_t counter2Value = (uint64_t)SRSS->unCLK_CAL_CNT2.stcField.u24CAL_COUNTER2;
    /* Get ILO 0 Count value. */

    /* Done counting; allow entry into DeepSleep mode. */
    clkCounting = false;

    *measuredFreq = CY_SYSClk_DIV_ROUND(counter2Value * (uint64_t)refClkFreq,
(uint64_t)clk1Count1 ); /* (6) Get ILO 0 Frequency. */

    return(CY_SYSClk_SUCCESS);
}

```

6.2.2 ILO0 calibration using clock calibration counter function

6.2.2.1 Operation overview

The ILO frequency is determined during manufacturing; however, the ILO frequency can be updated on the field according to the voltage and temperature conditions.

The ILO frequency trim can be updated using the ILOx_FTRIM bit of the CLK_TRIM_ILOx_CTL register. The initial value of the ILOx_FTRIM bit is 0x2C. Increasing the value of this bit by 0x01 increases the frequency by 1.5% (typical); decreasing this bit value by 0x01 decreases the frequency by 1.5% (typical). The CLK_TRIM_ILO0_CTL register is protected by WDT_CTL.ENABLE. For the specification of the WDT_CTL register, see the “Watchdog timer” section of the TRAVEO™ T2G [architecture reference manual](#).

[Figure 21](#) shows an example flow of ILO0 calibration using clock calibration counter and the CLK_TRIM_ILOx_CTL register.

6 Supplementary information

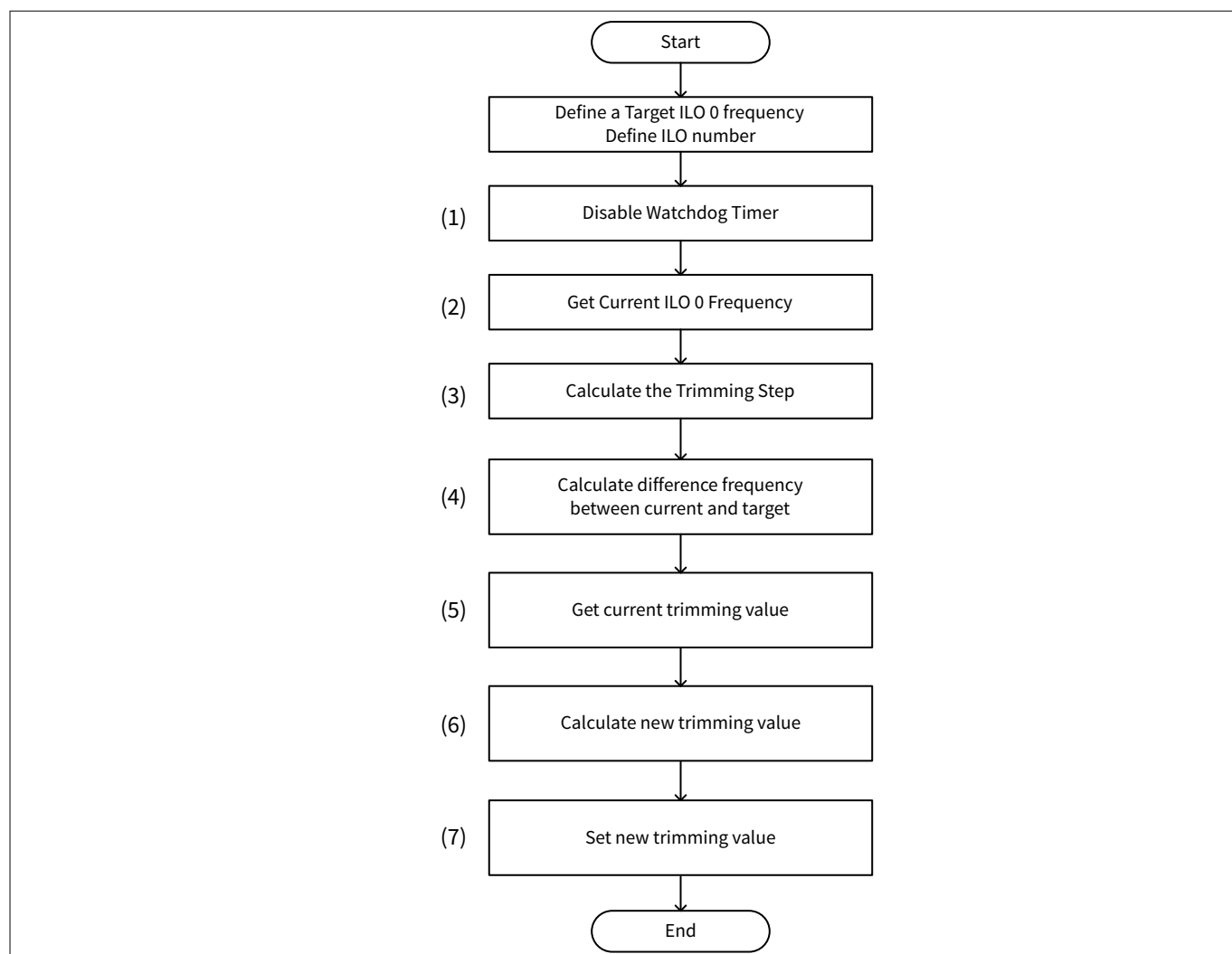


Figure 21 ILO0 calibration

6.2.2.2 Configuration

Table 30 lists the parameters and Table 31 lists the functions of the configuration part of in SDL for ILO0 calibration using clock calibration counter settings.

Table 30 List of ILO0 calibration using clock calibration counter settings parameters

Parameters	Description	Value
CY_SYSCLK_ILO_TARGET_FREQ	ILO target frequency	32768ul (32.768 kHz)
ILO_0	Define ILO_0 setting parameter	0ul
ILO_1	Define ILO_1 setting parameter	1ul
ILONo	Define measurement clock	ILO_0
iloFreq	Current ILO 0 frequency stored	–

6 Supplementary information

Table 31 List of ILO0 calibration using clock calibration counter settings functions

Functions	Description	Value
Cy_WDT_Disable ()	WDT disable	–
Cy_WDT_Unlock()	Unlocks the watchdog timer	–
GetILOClockFreq()	Get current ILO 0 frequency	–
Cy_SysClk_IloTrim (iloFreq, iloNo)	Set trim iloFreq: Current ILO 0 frequency iloNo: Trimming ILO number	iloFreq: iloFreq iloNo: ILONo

6.2.2.3 Sample code for initial configuration of ILO0 calibration using clock calibration counter settings

Code Listing 49 shows a sample code.

Code Listing 49 General configuration of ILO0 calibration using clock calibration counter settings

```
#define CY_SYSClk_DIV_ROUND(a, b) (((a) + ((b) / 2u11)) / (b)) /* Define CY_SYSClk_DIV_ROUND
function. */

#define CY_SYSClk_ILO_TARGET_FREQ 32768u1 /* Define Target ILO 0 frequency. */

#define ILO_0 0
#define ILO_1 1
#define ILONo ILO_0 /* Define ILO 0 number. */

int32_t iloFreq;

int main(void)
{
    /* Enable global interrupts. */
    __enable_irq();

    Cy_WDT_Disable(); /* (1) Watchdog Timer disable. */
:
    /* return: Frequency of ILO */
    ILOFreq = GetILOClockFreq(); /* (2) Get Current ILO 0 frequency. See Code Listing 45 */
:
    /* Must unlock WDT before update Trim */
    Cy_WDT_Unlock(); /* Watchdog timer unlock. */
    Trim_diff = Cy_SysClk_IloTrim(ILOFreq, ILONo); /* Trimming the ILO 0. See Code Listing 50 */
:
    for(;;);
}
```


6 Supplementary information

Code Listing 50 Cy_SysClk_IloTrim() function

```
int32_t Cy_SysClk_IloTrim(uint32_t iloFreq, uint8_t iloNo)
{
    /* Nominal trim step size is 1.5% of "the frequency". Using the target frequency. */
    const uint32_t trimStep = CY_SYSClk_DIV_ROUND((uint32_t)CY_SYSClk_ILO_TARGET_FREQ * 15uL,
    1000uL); /* (3) Calculate Trimming Step. */

    uint32_t newTrim = 0uL;
    uint32_t curTrim = 0uL;

    /* Do nothing if iloFreq is already within one trim step from the target */
    uint32_t diff = (uint32_t)abs((int32_t)iloFreq - (int32_t)CY_SYSClk_ILO_TARGET_FREQ); /*
    (4) Calculate Diff between current and target Frequency. */
    if (diff >= trimStep) /* Check if diff is greater than trimming step. */
    {
        if(iloNo == 0u)
        {
            curTrim = SRSS->unCLK_TRIM_ILO0_CTL.stcField.u6ILO0_FTRIM; /* (5) Read current
            trimming value. */
        } /* (5) Read current trimming value. */
        else /* (5) Read current trimming value. */
        { /* (5) Read current trimming value. */
            curTrim = SRSS->unCLK_TRIM_ILO1_CTL.stcField.u6ILO1_FTRIM; /* (5) Read current
            trimming value. */
        }

        if (iloFreq > CY_SYSClk_ILO_TARGET_FREQ) /* Check if current frequency is smaller than
        target frequency. */
        { /* iloFreq is too high. Reduce the trim value */
            newTrim = curTrim - CY_SYSClk_DIV_ROUND(iloFreq - CY_SYSClk_ILO_TARGET_FREQ,
            trimStep); /* (6) Calculate new trim value. */
        } /* (6) Calculate new trim value. */
        else /* (6) Calculate new trim value. */
        { /* iloFreq too low. Increase the trim value. */ /* (6) Calculate new trim value. */
            newTrim = curTrim + CY_SYSClk_DIV_ROUND(CY_SYSClk_ILO_TARGET_FREQ - iloFreq,
            trimStep); /* (6) Calculate new trim value. */
        }

        /* Update the trim value */
        if(iloNo == 0u)
        {
            if(WDT->unLOCK.stcField.u2WDT_LOCK != 0uL) /* WDT registers are disabled */ /* Check
            if Watchdog timer disabled. */

            {
                return(CY_SYSClk_INVALID_STATE);
            }
            SRSS->unCLK_TRIM_ILO0_CTL.stcField.u6ILO0_FTRIM = newTrim; /* (7) Set New trimming
            value */
        }
    }
}
```

6 Supplementary information

```

    } /* (7) Set New trimming value */
    else /* (7) Set New trimming value */
    { /* (7) Set New trimming value */
        SRSS->unCLK_TRIM_ILO1_CTL.stcField.u6ILO1_FTRIM = newTrim; /* (7) Set New trimming
value */
    }
}
return (int32_t)(curTrim - newTrim);
}

```

6.3 CSV diagram and relationship of monitored clock and reference clock

Figure 22 shows the clock diagram with monitored clock and reference clock for CSV. Table 32 shows the relationship between monitored clock and reference clock.

6 Supplementary information

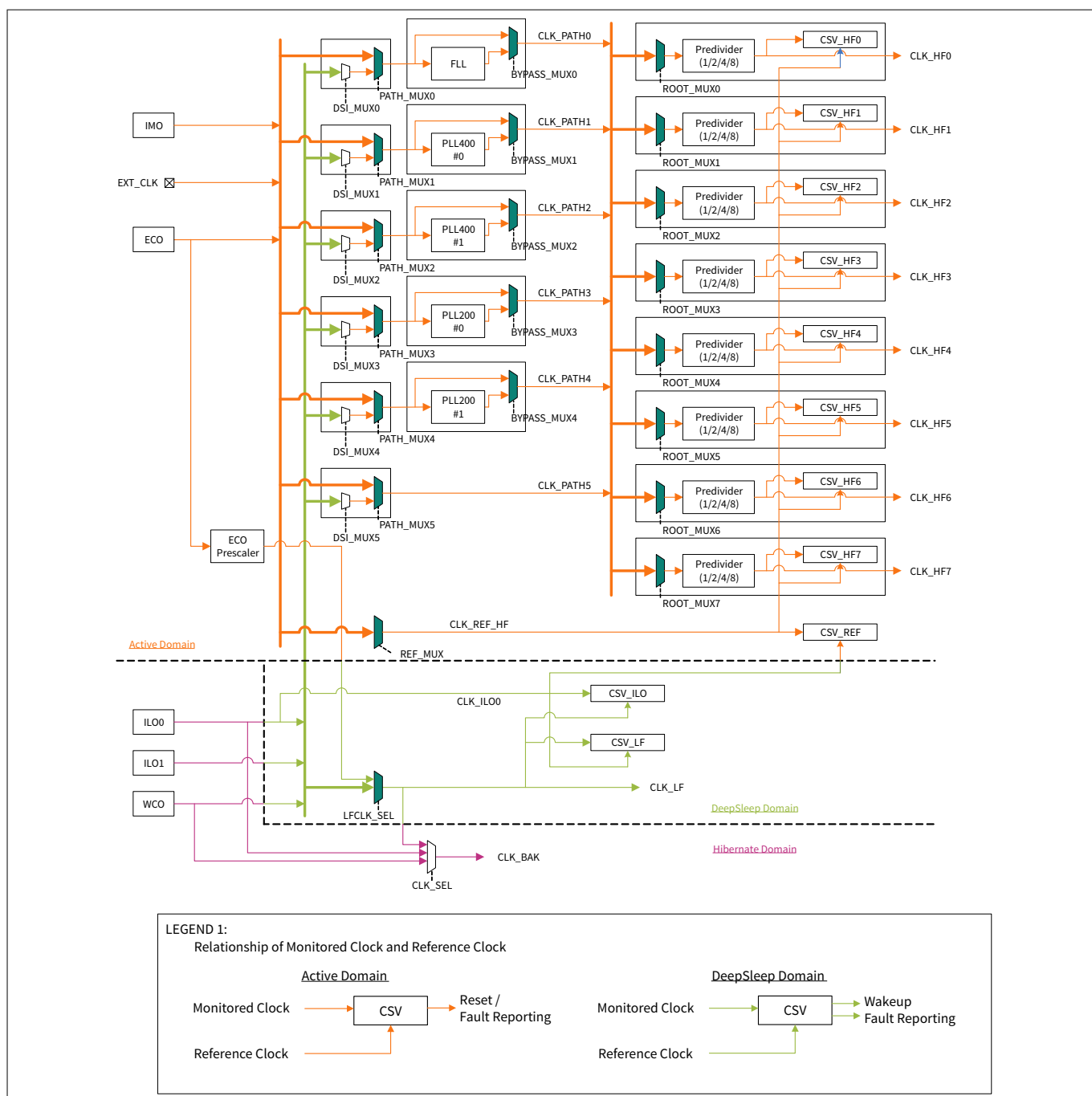


Figure 22 CSV diagram

Table 32 Monitored clock and reference clock

CSV components	Monitor clock	Reference clock	Notes
CSV_HF0	CSV_HF0	CLK_REF_HF	CLK_REF_HF is selected CLK_IMO, EXT_CLK, or CLK_ECO
CSV_HF1	CLK_HF1	CLK_REF_HF	CLK_REF_HF is selected CLK_IMO, EXT_CLK, or CLK_ECO

(table continues...)

6 Supplementary information

Table 32 (continued) Monitored clock and reference clock

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

Glossary

Glossary

Table 33 **Glossary**

Terms	Description
AUDIOSS	Audio subsystem. See the “Audio subsystem” chapter of TRAVEO™ T2G 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 TRAVEO™ T2G architecture reference manual for details
CLK_FAST_0	Fast clock. The CLK_FAST is used for the CM7 and CPUSS fast infrastructure
CLK_FAST_1	Fast clock. The CLK_FAST is used for the CM7 and CPUSS fast infrastructure
CLK_FAST_2	Fast clock. The CLK_FAST is used for the CM7 and CPUSS fast infrastructure (only CYT6BJ series)
CLK_FAST_3	Fast clock. The CLK_FAST is used for the CM7 and CPUSS fast infrastructure (only CYT6BJ series)
CLK_HF	High-frequency clock. The CLK_HF derive both CLK_FAST and CLK_SLOW. CLK_HF, CLK_FAST, and CLK_SLOW are synchronous to each other
CLK_GR	Group clock. The CLK_GR is the clock input to peripheral functions
CLK_MEM	Memory clock. CLK_MEM clocks the CPUSS fast infrastructure
CLK_PERI	Peripheral clock. The CLK_PERI is the clock source for CLK_SLOW, CLK_GR, and peripheral clock divider
CLK_SLOW	Slow clock. The CLK_FAST is used for the CM0+ and CPUSS slow infrastructure
Clock calibration counter	Clock calibration counter has a function to calibrate clock using two clocks
CSV	Clock supervision
ECO	External crystal oscillator
EXT_CLK	External clock
FLL	Frequency locked loop
ILO	Internal low-speed oscillators
IMO	Internal main oscillator
LIN	Local Interconnect Network. See the “Local Interconnect Network (LIN)” chapter of TRAVEO™ T2G architecture reference manual for details
Peripheral clock divider	Peripheral clock divider derive a clock to use of each peripheral function
PLL200	Phase locked loop. This PLL is not implemented with SSCG and fractional operation
PLL400	Phase locked loop. This PLL is implemented with SSCG and fractional operation
SAR ADC	Successive approximation register analog-to-digital converter. See the “SAR ADC” chapter of TRAVEO™ T2G architecture reference manual for details
SCB	Serial communications block. See the “Serial communications block (SCB)” chapter of TRAVEO™ T2G architecture reference manual for details

(table continues...)

Glossary

Table 33 (continued) Glossary

Terms	Description
SDHC	The secure digital high-capacity host controller. See the “SDHC host controller” chapter of TRAVEO™ T2G architecture reference manual for details
SMIF	Serial memory interface. See the “Serial memory interface” chapter of TRAVEO™ T2G architecture reference manual for details
TCPWM	Timer, counter, and pulse width modulator. See the “Timer, counter, and PWM” chapter of TRAVEO™ T2G architecture reference manual for details
WCO	Watch crystal oscillator

References

References

The following are the TRAVEO™ T2G family series datasheets and technical reference manuals. Contact [Technical Support](#) to obtain these documents.

- [1] Device datasheet
 - [CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family](#)
 - [CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family](#)
 - [CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family \(002-33466\)](#)
- [2] Body controller high family reference manual
 - [TRAVEO™ T2G automotive body controller high family architecture reference manual](#)
 - [TRAVEO™ T2G automotive body controller high registers reference manual for CYT4BF](#)
 - [TRAVEO™ T2G automotive body controller high registers reference manual for CYT3BB/4BB](#)
 - [TRAVEO™ T2G automotive body controller high registers reference manual for CYT6BJ \(002-36068\)](#)
- [3] User guide
 - [Setting ECO parameters in TRAVEO™ T2G family user guide](#)

A Sample Driver Library (SDL) including startup is provided to access various peripherals. The SDL also serves as a reference, to customers, for drivers that are not covered by the official AUTOSAR products. The SDL cannot be used for production purposes as it does not qualify to any automotive standards. The code snippets in this application note are part of the SDL. Contact [Technical Support](#) to obtain the SDL.

Revision history

Revision history

Document revision	Date	Description of changes
**	2019-09-24	New application note
*A	2020-12-03	Updated Configuration of the Clock Resources: Added flowchart and example codes in all instances Updated Configuration of FLL and PLL: Added flowchart and example codes in all instances Updated Configuration of the Internal Clock: Added flowchart and example codes in all instances Removed “Example for Configuring Internal Clock”
*B	2021-05-25	Updated to Infineon template
*C	2021-11-29	Corrected “Section 3.4. Setting ILO0/ILO1”
*D	2024-02-19	Added specifications for CYT6BJ series in Updated series name from CYT4B series to CYT4B/6B series
*E	2024-04-23	Template update; no content update
*F	2025-02-20	Fixed peripheral clock output number in Example of PCLK setting Fixed AUDIOSS function values in Table 27

Trademarks

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

Edition 2025-02-20

Published by

Infineon Technologies AG
81726 Munich, Germany

© 2025 Infineon Technologies AG
All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

IFX-xoo1708964629966

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.