

AN1492: Clock Manager Migration Guide



This application note provides guidelines and considerations for migrating projects that were created using the previous clock management architecture in the Gecko SDK to the new Clock Manager module in the Simplicity SDK.

KEY POINTS

- Oscillator and clock tree configuration using the Clock Manager module
- CMU emlib API to Clock Manager API replacements
- Step-by-step guide on how to migrate a Series 2 Gecko SDK project to a Simplicity SDK project that uses the Clock Manager module

1. Device Compatibility

This application note supports multiple device families.

EFR32 Wireless Gecko Series 2 consists of:

- EFR32BG21
- EFR32MG21
- EFR32BG22
- EFR32FG22
- EFR32MG22
- EFR32FG23
- EFR32ZG23
- EFR32SG23
- EFR32BG24
- EFR32MG24
- EFR32FG25
- EFR32BG27
- EFR32MG27
- EFR32FG28
- EFR32ZG28
- EFR32SG28

EFM32 Series 2 consists of:

- EFM32PG22
- EFM32PG23
- EFM32PG28

2. Introduction

The Clock Manager module is a new software module introduced in the Simplicity SDK (SiSDK). The Clock Manager module uses Clock Manager APIs and consists of two software components:

- Clock Manager component: handles initialization and configuration of the oscillators and clock tree
- Clock Manager Runtime component: handles runtime processes such as clock configuration, calibration, tuning, and more

In the Gecko SDK (GSDK), clock initialization and configuration is handled by some of the Device Initialization software components. The relevant components consist of:

- Clock initialization component: `sl_device_init_clocks`
- HFRCO initialization component: `sl_device_init_hfrco`
- HFXO initialization component: `sl_device_init_hfxo`
- LFRCO initialization component: `sl_device_init_lfrco`
- LFXO initialization component: `sl_device_init_lfxo`
- RFF PLL initialization component: `sl_device_init_rffpll`
- USB PLL initialization component: `sl_device_init_usbpll`

The clock initialization component automatically configures the clock tree depending on which Device Initialization components are installed. The remaining components configure and initialize the respective oscillators and PLLs. The clocks can further be configured during runtime using the CMU emlib APIs.

When migrating a Series 2 project from the GSDK to the SiSDK:

- The Clock Manager component is not automatically installed. The clock related Device Initialization components will remain installed. These components are mutually exclusive, meaning they cannot be installed at the same time. Users can manually install the Clock Manager component via the Project Configurator in the `*slcp` file. Installing the Clock Manager component will automatically uninstall the clock related Device Initialization components.
- The Clock Manager Runtime component is typically automatically installed as a dependency for other standard project software components. Although the Clock Manager Runtime component is intended to be a replacement for the CMU emlib APIs, Clock Manager APIs can be used alongside CMU emlib APIs. User calls to CMU emlib APIs are not automatically replaced with the Clock Manager APIs when updating the SDK.

This allows users to migrate their Series 2 projects from the GSDK to the SiSDK with minimal changes. Users have the option to manually migrate their project to only use the Clock Manager module. Full adoption of the Clock Manager module is recommended as future devices will not support CMU emlib APIs nor the clock related Device Initialization components.

3. Initialization

The GSDK uses multiple Device Initialization components and configuration files to configure and initialize each oscillator. The Clock Manager module consolidates the oscillator configuration and initialization into a single Clock Manager component. Oscillators are configured via the Project Configurator or the CMSIS annotated configuration file, `sl_clock_manager_oscillator_config.h`.

[Table 3.1 Initialization Substitutions on page 4](#) contains the CMU emlib initialization functions that can be called by the Device Initialization components. The Clock Manager component combines the initialization into a single Clock Manager API.

Table 3.1. Initialization Substitutions

CMU emlib API	Clock Manager API
<pre>void CMU_LFXOInit(const CMU_LFXOInit_TypeDef *lfxoInit)</pre>	<pre>sl_status_t sl_clock_manager_init(void)</pre>
<pre>void CMU_HFXOInit(const CMU_HFXOInit_TypeDef *hfxoInit)</pre>	
<pre>bool CMU_DPLLLock(const CMU_DPLLInit_TypeDef *init)</pre>	
<pre>void CMU_RFFPLLInit(const CMU_RFFPLL_Init_TypeDef *pllInit)</pre>	
<pre>void CMU_USBPLLInit(const CMU_USBPLL_Init_TypeDef *pllInit)</pre>	

4. Configuration

The clock initialization component, `sl_device_init_clocks`, is a Device Initialization component that generates a non-user configurable clock tree during initialization. The Clock Manager component allows for user customization of the device's clock tree during initialization. The clock tree can be configured via the Project Configurator or the CMSIS annotated configuration file, `sl_clock_manager_tree_config.h`.

There are CMU emlib API functions for setting and getting individual clock sources and dividers, but there are no equivalent Clock Manager API functions to read or modify the clock tree during runtime. Instead, the clock source or divider is defined by macros in the configuration file. [Table 4.1 Table 2 on page 5](#) shows the CMU emlib APIs for configuring the clock tree and examples of the corresponding Clock Manager macros, specifically for the PCLK divider and SYSCLK source.

Table 4.1. Configuration Substitutions Example

CMU emlib API	Clock Manager Configuration
<code>CMU_ClkDiv_TypeDef CMU_ClockDivGet(CMU_Clock_TypeDef clock)</code>	<code>SL_CLOCK_MANAGER_PCLK_DIVIDER</code>
<code>void CMU_ClockDivSet(CMU_Clock_TypeDef clock, CMU_ClkDiv_TypeDef div)</code>	<code>#define SL_CLOCK_MANAGER_PCLK_DIVIDER divider</code>
<code>void CMU_ClockSelectGet(CMU_Clock_TypeDef clock)</code>	<code>SL_CLOCK_MANAGER_SYSCLK_SOURCE</code>
<code>void CMU_ClockSelectSet(CMU_Clock_TypeDef clock, CMU_Select_TypeDef ref)</code>	<code>#define SL_CLOCK_MANAGER_SYSCLK_SOURCE source</code>

[Table 4.2 Additional Configuration Examples on page 5](#) shows the remaining CMU emlib APIs that do not have a corresponding Clock Manager API function but instead have a corresponding macro configuration.

Table 4.2. Additional Configuration Examples

CMU emlib API	Clock Manager Configuration
<code>CMU_HFRCODPLLFreq_TypeDef CMU_HFRCODPLLBandGet(void)</code>	<code>SL_CLOCK_MANAGER_HFRCO_Band</code>
<code>void CMU_HFRCODPLLBandSet(CMU_HFRCODPLLFreq_TypeDef freq)</code>	<code>#define SL_CLOCK_MANAGER_HFRCO_BAND freq</code>
<code>void CMU_LFXOPrecisionSet(uint16_t precision)</code>	<code>#define SL_CLOCK_MANAGER_LFXO_PRECISION precision</code>
<code>uint16_t CMU_LFXOPrecisionGet(void)</code>	<code>SL_CLOCK_MANAGER_LFXO_PRECISION</code>
<code>void CMU_HFXOPrecisionSet(uint16_t precision)</code>	<code>#define SL_CLOCK_MANAGER_HFXO_PRECISION precision</code>
<code>uint16_t CMU_HFXOPrecisionGet(void)</code>	<code>SL_CLOCK_MANAGER_HFXO_PRECISION</code>
<code>sl_status_t CMU_HFXOCTuneSet(uint32_t ctune)</code>	<code>#define SL_CLOCK_MANAGER_HFXO_CTUNE ctune</code>
<code>void CMU_LFRCOSetPrecision(CMU_Precision_TypeDef precision)</code>	<code>#define SL_CLOCK_MANAGER_LFRCO_PRECISION precision</code>
<code>void CMU_PCNTClockExternalSet(unsigned int instance, true)</code>	<code>#define SL_CLOCK_MANAGER_PCNT0CLK_SOURCE CMU_PCNT0CLKCTRL_CLKSEL_PCNTS0</code>

CMU emlib API	Clock Manager Configuration
void CMU_PCNTClockExternalSet(unsigned int instance, false)	#define SL_CLOCK_MANAGER_PCNT0CLK_SOURCE CMU_PCNT0CLKCTRL_CLKSEL_EM23GRPACLK
CMU_HFRCOEM23Freq_TypeDef CMU_HFRCOEM23BandGet(void)	SL_CLOCK_MANAGER_HFRCOEM23_BAND
void CMU_HFRCOEM23BandSet(CMU_HFRCOEM23Freq_TypeDef freq)	#define SL_CLOCK_MANAGERHFRCOEM23_BAND freq
void CMU_HFXOStartCrystalSharingLeader(const CMU_BUFOUTLeaderInit_TypeDef *bufoutInit, GPIO_Port_TypeDef port, unsigned int pin)	SL_CLOCK_MANAGER_HFXO_CRYSTAL_SHARING_LEADER_EN
void CMU_HFXOCrystalSharingFollowerInit(CMU_PRS_Status_Output_Select_TypeDef prsStatusSelectOutput, unsigned int prsAsyncCh, GPIO_Port_TypeDef port, unsigned int pin)	SL_CLOCK_MANAGER_HFXO_CRYSTAL_SHARING_FOLLOWER_EN

5. Runtime

Table 5.1 Runtime Substitutions on page 7 contains CMU emlib runtime APIs and their equivalent Clock Manager APIs. Some CMU emlib APIs require calls to multiple Clock Manager APIs to maintain similar functionality. Note that function arguments and return values may be different.

Table 5.1. Runtime Substitutions

CMU emlib API	Clock Manager API	Notable Changes
<pre>uint32_t CMU_Calibrate(uint32_t cycles, CMU_Select_TypeDef ref)</pre>	<pre>sl_status_t sl_clock_manager_configure_rco_calibration(uint32_t cycles, sl_clock_manager_clock_calibration_t down_counter_selection, sl_clock_manager_clock_calibration_t up_counter_selection, bool continuous_calibration)</pre> <pre>sl_status_t sl_clock_manager_start_rco_calibration(void)</pre> <pre>sl_status_t sl_clock_manager_get_rco_calibration_count(uint32_t *count)</pre>	<p><i>CMU_Calibrate</i> calibrates an oscillator using the number of HCLK cycles and a reference clock and returns the number of ticks of the reference clock. Three Clock Manager APIs are required to replicate the behavior of <i>CMU_Calibrate</i>. The Clock Manager API <i>configure_rco_calibration</i> takes in the number of clock cycles, down and up counter selection, and a Boolean type that can enable continuous calibration. Next, <i>start_rco_calibration</i> starts the RCO calibration. Lastly, <i>get_rco_calibration_count</i> retrieves the calibration count value and updates the <i>count</i> variable via call-by-reference. All three of the Clock Manager APIs return the status.</p>

CMU emlib API	Clock Manager API	Notable Changes
<pre>void CMU_CalibrateConfig(uint32_t downCycles, CMU_Select_TypeDef downSel, CMU_Select_TypeDef upSel) void CMU_CalibrateCont(bool enable)</pre>	<pre>sl_status_t sl_clock_manager_configure_rco_calibration(uint32_t cycles, sl_clock_manager_clock_calibration_t down_counter_selection, sl_clock_manager_clock_calibration_t up_counter_selection, bool continuous_calibration)</pre>	<p><i>CMU_CalibrateConfig</i> configures the clock calibration by specifying the number of clock cycles and selecting a down and up counter. <i>CMU_CalibrateCont</i> enables continuous calibration according to the Boolean argument. The Clock Manager API combines the two CMU APIs and takes in the number of clock cycles, down and up counter selection, and a Boolean type that can enable continuous calibration. Lastly, it returns the status.</p>
<pre>uint32_t CMU_CalibrateCountGet(void)</pre>	<pre>void sl_clock_manager_wait_rco_calibration(void) sl_status_t sl_clock_manager_get_rco_calibration_count(uint32_t *count)</pre>	<p><i>CMU_CalibrateCountGet</i> returns the calibration count of the UPSEL clock. Two Clock Manager APIs are required to replicate the behavior of <i>CalibrateCountGet</i>. The first function waits for RCO calibration to complete and the second function updates the <i>count</i> variable via call-by-reference and returns the status.</p>
<pre>void CMU_ClkOutPinConfig(uint32_t clkNo, CMU_Select_TypeDef sel, CMU_ClkDiv_TypeDef clkDiv, GPIO_Port_TypeDef port, unsigned int pin)</pre>	<pre>sl_status_t sl_clock_manager_set_gpio_clock_output(sl_clock_manager_export_clock_source_t export_clock_source, sl_clock_manager_export_clock_output_select_t output_select, uint16_t hfexp_divider, uint32_t port, uint32_t pin)</pre>	<p><i>CMU_ClkOutPinConfig</i> configures the pin by assigning it the clock output number, clock divider, and the port and pin number. The Clock Manager API configures the pin by assigning the clock source, clock output number, divider value, and the port and pin number. The clock sources and output numbers are changed to Clock Manager types and the status is returned.</p>

CMU emlib API	Clock Manager API	Notable Changes
<pre>void CMU_ClockEnable(CMU_Clock_TypeDef clock, bool enable)</pre>	<pre>sl_status_t sl_clock_manager_enable_bus_clock(sl_bus_clock_t module) sl_status_t sl_clock_manager_disable_bus_clock(sl_bus_clock_t module)</pre>	<p><i>CMU_ClockEnable</i> selects a clock and enables it using a Boolean type. The Clock Manager separates the CMU API into two functions. The Clock Manager APIs argument is a bus clock pointer, and it returns the status.</p>
<pre>uint32_t CMU_ClockFreqGet(CMU_Clock_TypeDef clock)</pre>	<pre>sl_status_t sl_clock_manager_get_oscillator_frequency(sl_oscillator_t oscillator, uint32_t *frequency) sl_status_t sl_clock_manager_get_clock_branch_frequency(sl_clock_branch_t clock_branch, uint32_t *frequency)</pre>	<p><i>CMU_ClockFreqGet</i> retrieves the frequency of a specified clock and returns its value. The Clock Manager API separates the CMU API into two functions, which retrieves either the oscillator or the clock branch frequency. The frequency variable is then updated via call-by-reference and the status is returned.</p>
<pre>uint16_t CMU_HF_ClockPrecisionGet(CMU_Clock_TypeDef clock) uint16_t CMU_LF_ClockPrecisionGet(CMU_Clock_TypeDef clock)</pre>	<pre>sl_status_t sl_clock_manager_get_clock_branch_precision(sl_clock_branch_t clock_branch, uint16_t *precision)</pre>	<p>The CMU API retrieves the precision of either an LF or HF clock and returns its value. The Clock Manager API retrieves the precision of a selected clock branch. The precision variable is updated via call-by-reference and the status is returned.</p>

CMU emlib API	Clock Manager API	Notable Changes
<code>uint32_t CMU_OscillatorTuningGet(CMU_Osc_TypeDef osc)</code>	<pre>sl_status_t sl_clock_manager_get_rc_oscillator_calibration(sl_oscillator_t oscillator, uint32_t *val)</pre> <pre>sl_status_t sl_clock_manager_get_hfxo_calibration(uint32_t *val)</pre> <pre>sl_status_t sl_clock_manager_get_lfxo_calibration(uint32_t *val)</pre>	<i>CMU_OscillatorTuningGet</i> retrieves the tuning frequency of the specified oscillator and returns its value. The Clock Manager API separates the CMU API into three functions. The Clock Manager retrieves the tuning frequency of the specified oscillator. The tuning frequency variable is updated via call-by-reference and the status is returned. If the oscillator is the HFXO or the LFXO, the Clock Manager API <i>hfxo_calibration</i> or <i>lfxo_calibration</i> should be used respectively.
<code>void CMU_OscillatorTuningSet(CMU_Osc_TypeDef osc, uint32_t val)</code>	<pre>sl_status_t sl_clock_manager_set_rc_oscillator_calibration(sl_oscillator_t oscillator, uint32_t val)</pre> <pre>sl_status_t sl_clock_manager_set_hfxo_calibration(uint32_t val)</pre> <pre>sl_status_t sl_clock_manager_set_lfxo_calibration(uint32_t val)</pre>	<i>CMU_OscillatorTuningSet</i> sets the tuning frequency of an oscillator according to the argument, <i>val</i> . The Clock Manager API separates the CMU API into three functions. The Clock Manager sets the tuning frequency of the oscillator according to the argument, <i>val</i> . The status is then returned. If the oscillator is the HFXO or the LFXO, the Clock Manager API <i>set_hfxo_calibration</i> or <i>set_lfxo_calibration</i> should be used respectively.
<code>void CMU_CalibrateStart(void)</code>	<pre>void sl_clock_manager_start_rco_calibration(void)</pre>	-
<code>void CMU_CalibrateStop(void)</code>	<pre>void sl_clock_manager_stop_rco_calibration(void)</pre>	-
<code>uint32_t CMU_HFXOCTuneGet(void)</code>	<pre>sl_status_t slx_clock_manager_hfxo_get_ctune(uint32_t *ctune)</pre>	<i>CMU_HFXOCTuneGet</i> returns the crystal tuning capacitance of the HFXO. The Clock Manager API updates the <i>ctune</i> variable via call-by-reference and returns the status.

CMU emlib API	Clock Manager API	Notable Changes
<code>sl_status_t CMU_HFXOCTuneSet(uint32_t ctune)</code>	<code>sl_status_t slx_clock_manager_hfxo_set_ctune(uint32_t ctune)</code>	<i>CMU_HFXOCTuneSet</i> sets the crystal tuning capacitance of the HFXO according to the argument, <i>ctune</i> , and returns the status. The Clock Manager API sets the crystal tuning capacitance of the HFXO according to the argument, <i>ctune</i> , and returns the status.
<code>void CMU_HFXOCoreBiasCurrentCalibrate(void)</code>	<code>sl_status_t slx_clock_manager_hfxo_calibrate_ctune(uint32_t ctune)</code>	The CMU API recalibrates the HFXO's core bias current. A CTUNE value can be passed in the argument <i>ctune</i> to change the value of CTUNE before launching a core bias current calibration. If the current <i>ctune</i> value is given, only a core bias optimization will be performed and CTUNE will remain unchanged.

Table 5.2 CMU emlib APIs with No Substitutions on page 11 contains the CMU emlib APIs that have no equivalent Clock Manager API.

Table 5.2. CMU emlib APIs with No Substitutions

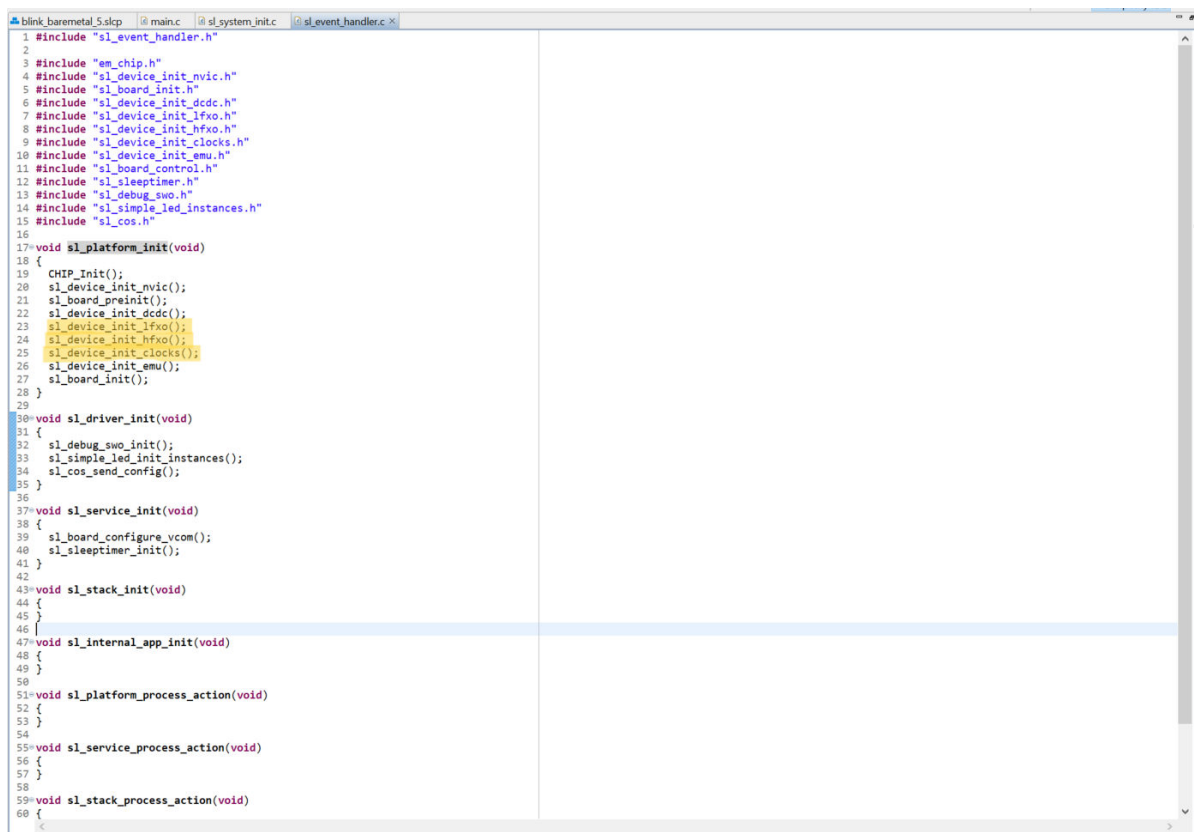
CMU API
<ul style="list-style-type: none"> CMU_DPLLLock CMU_WaitUSBPLLLock CMU_WaitRFFPLLLock CMU_DPLLUnlock CMU_WdogLock CMU_WdogUnlock CMU_HFXOCTuneDeltaSet CMU_HFXOCTuneDeltaGet CMU_UpdateWaitStates CMU_IntClear CMU_IntEnable CMU_IntGet CMU_IntGetEnabled CMU_IntSet CMU_Lock CMU_Unlock

6. Migration Guide

This section provides a step-by-step guide on how to migrate an existing project that uses the GSDK to using the SiSDK and the Clock Manager module.

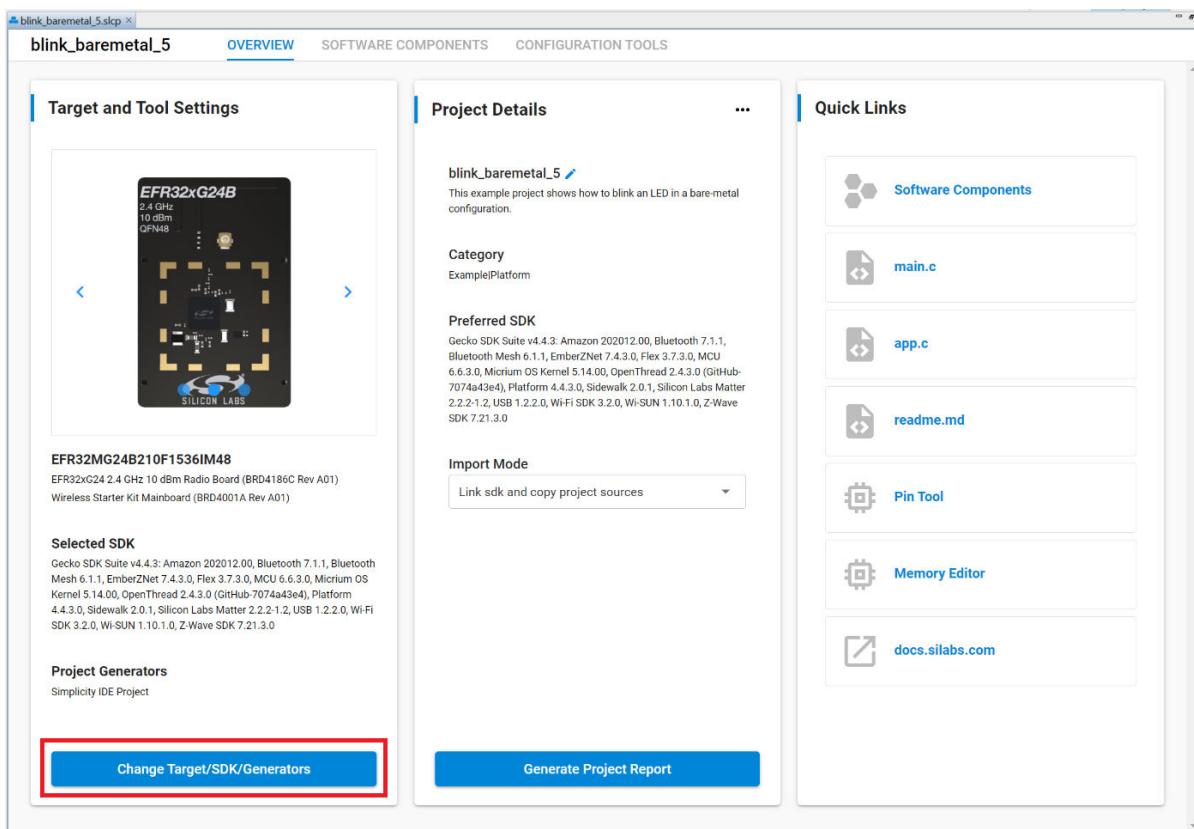
6.1 Installation

1. In the Gecko SDK, the installed Device Initialization software components can be found in the Project Configurator. The Device Initialization components are located inside the `sl_platform_init` function which is embedded within the `sl_system_init` function inside the `main.c` file. `sl_device_init_lfxo`, `sl_device_init_hfxo`, and `sl_device_init_clocks` are installed in this example project.

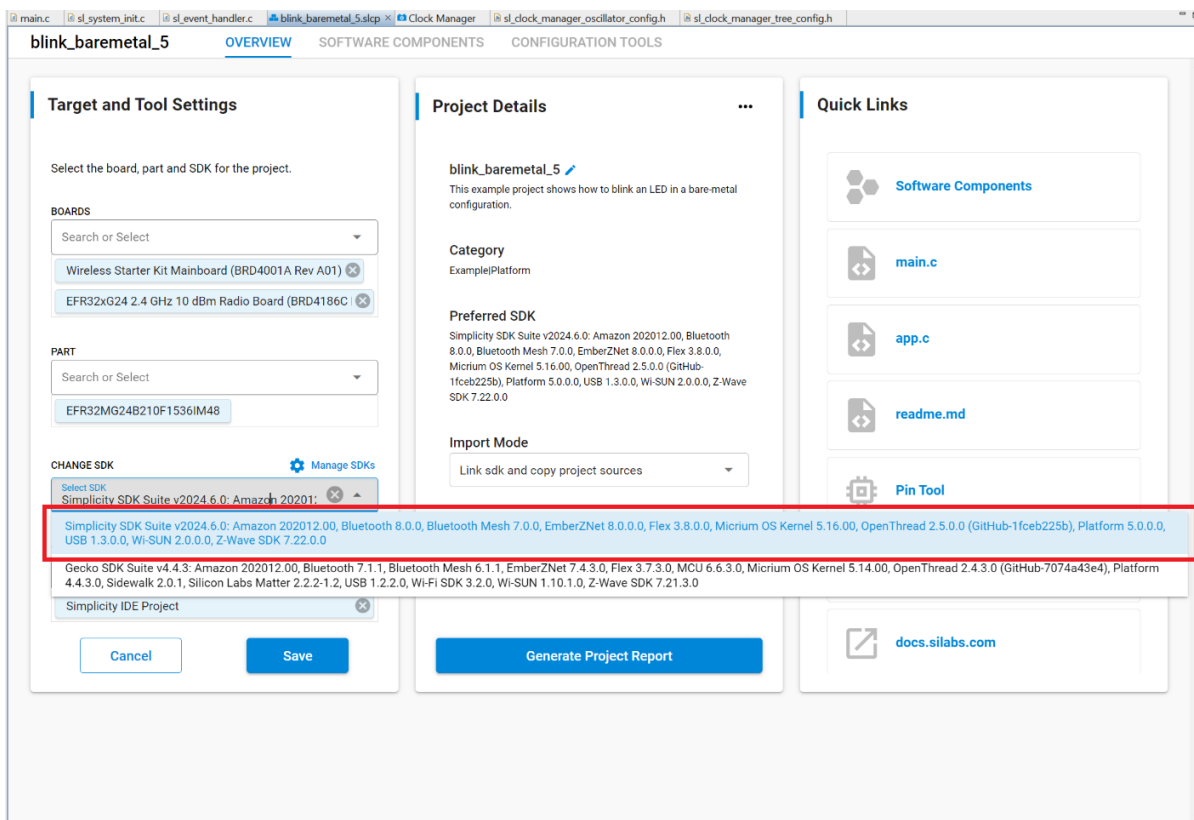


```
1 #include "sl_event_handler.h"
2
3 #include "em_chip.h"
4 #include "sl_device_init_nvic.h"
5 #include "sl_board_init.h"
6 #include "sl_device_init_dcdc.h"
7 #include "sl_device_init_lfxo.h"
8 #include "sl_device_init_hfxo.h"
9 #include "sl_device_init_clocks.h"
10 #include "sl_device_init_emu.h"
11 #include "sl_board_control.h"
12 #include "sl_sleeptimer.h"
13 #include "sl_debug_swo.h"
14 #include "sl_simple_led_instances.h"
15 #include "sl_cos.h"
16
17 void sl_platform_init(void)
18 {
19     CHIP_Init();
20     sl_device_init_nvic();
21     sl_board_preinit();
22     sl_device_init_dcdc();
23     sl_device_init_lfxo();
24     sl_device_init_hfxo();
25     sl_device_init_clocks();
26     sl_device_init_emu();
27     sl_board_init();
28 }
29
30 void sl_driver_init(void)
31 {
32     sl_debug_swo_init();
33     sl_simple_led_init_instances();
34     sl_cos_send_config();
35 }
36
37 void sl_service_init(void)
38 {
39     sl_board_configure_vcom();
40     sl_sleeptimer_init();
41 }
42
43 void sl_stack_init(void)
44 {
45 }
46
47 void sl_internal_app_init(void)
48 {
49 }
50
51 void sl_platform_process_action(void)
52 {
53 }
54
55 void sl_service_process_action(void)
56 {
57 }
58
59 void sl_stack_process_action(void)
60 {
```

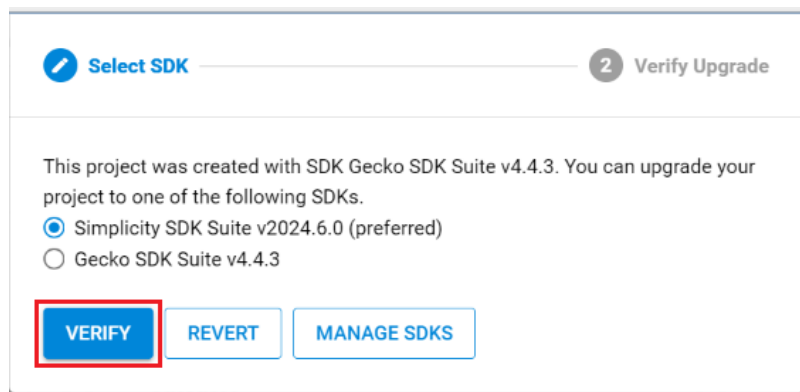
2. To update the SDK, open the *.slcp file and press Change Target/SDK/Generators.



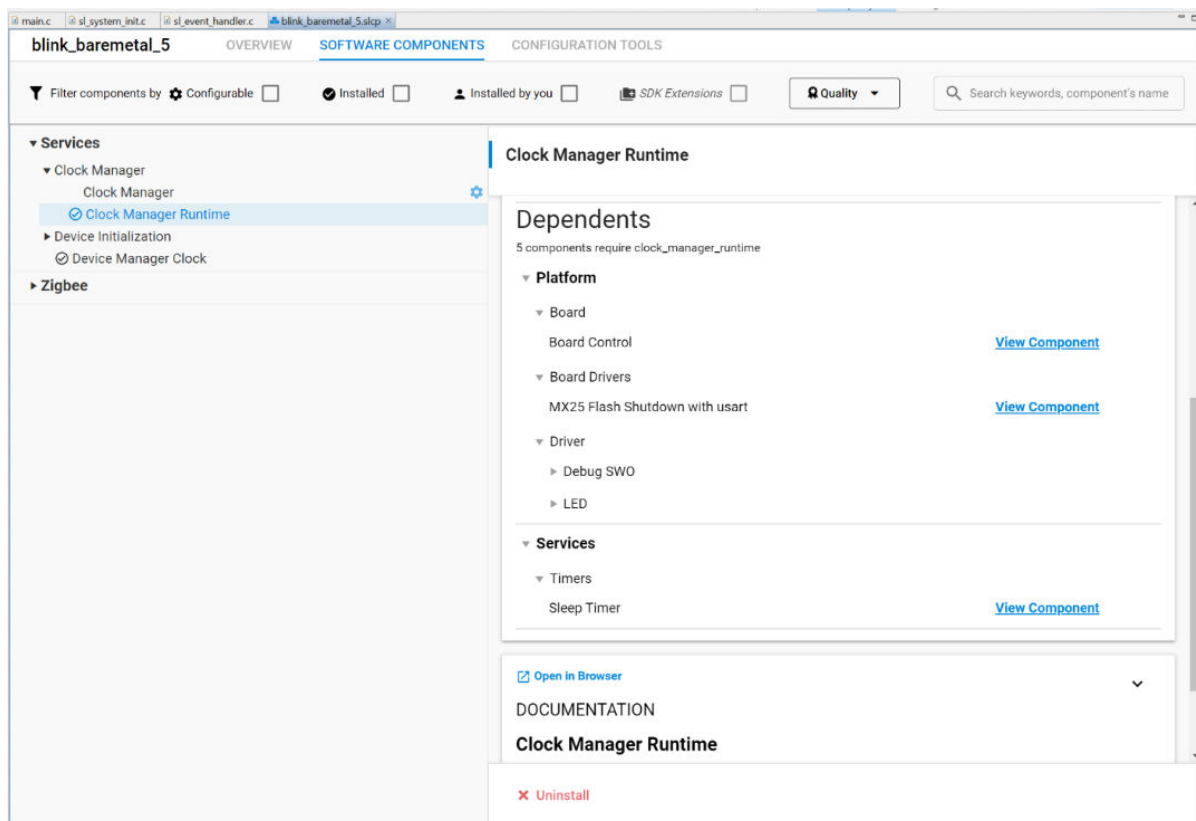
3. Change the SDK to the Simplicity SDK. In this installation example, the SDK is upgraded from the Gecko SDK Suite v4.4.3 to the Simplicity SDK Suite v2024.6.0.



4. Press Verify.



5. After the SDK update, the Clock Manager Runtime component should be automatically installed if there are dependencies in the project. The Clock Manager Runtime component and its dependencies can be found in the *.slcp, under the Services section.



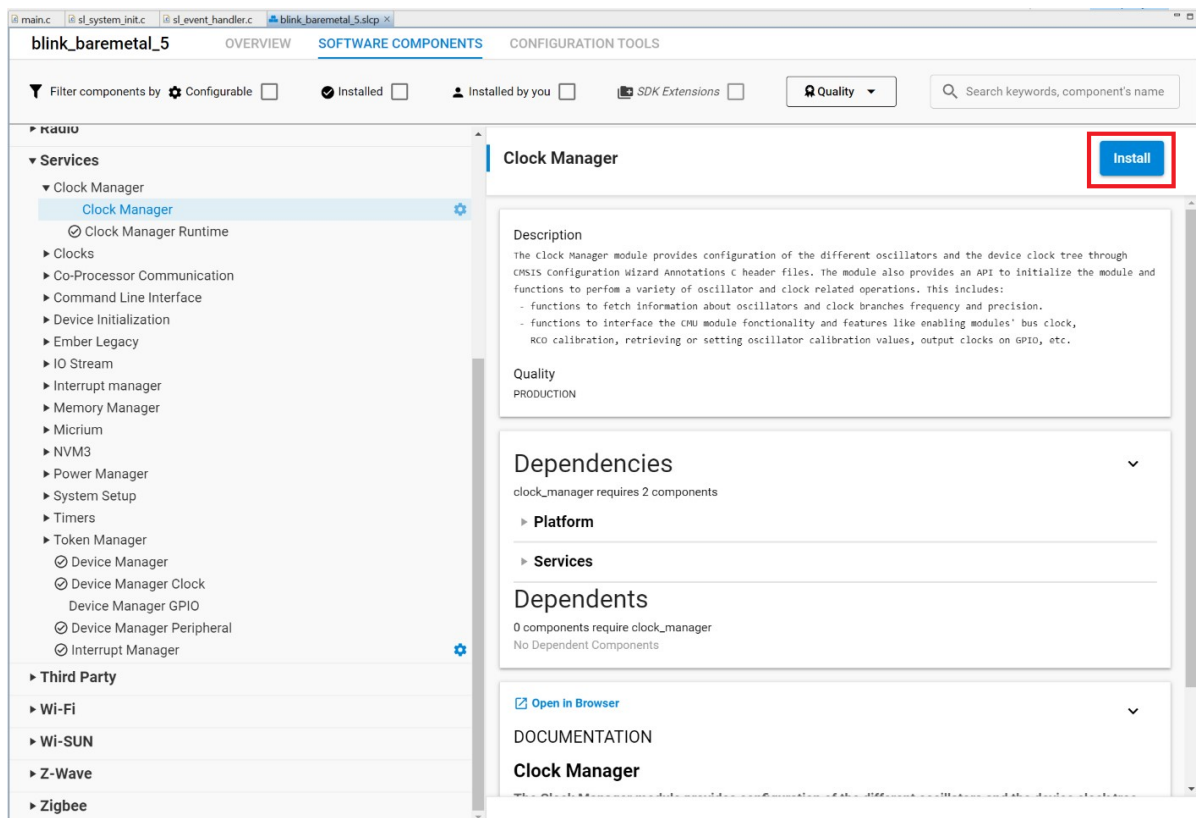
6. After the Clock Manager Runtime component has been installed, the `sl_platform_init` function will be updated as follows.

```

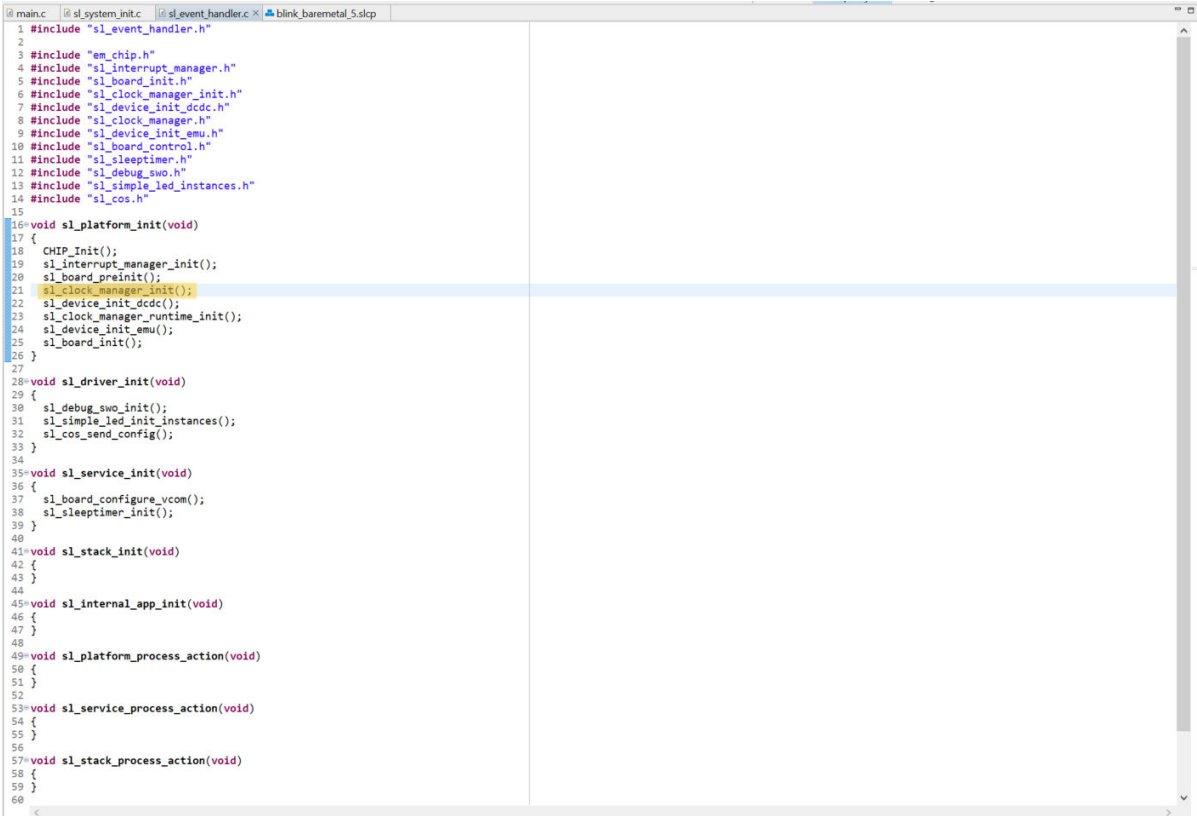
1 #include "sl_event_handler.h"
2
3 #include "em_chip.h"
4 #include "sl_interrupt_manager.h"
5 #include "sl_board_init.h"
6 #include "sl_device_init_dcdc.h"
7 #include "sl_clock_manager.h"
8 #include "sl_device_init_lfxo.h"
9 #include "sl_device_init_hfxo.h"
10 #include "sl_device_init_clocks.h"
11 #include "sl_device_init_emu.h"
12 #include "sl_board_control.h"
13 #include "sl_sleeptimer.h"
14 #include "sl_debug_swo.h"
15 #include "sl_simple_led_instances.h"
16 #include "sl_cos.h"
17
18 void sl_platform_init(void)
19 {
20     CHIP_Init();
21     sl_interrupt_manager_init();
22     sl_board_preinit();
23     sl_device_init_dcdc();
24     sl_clock_manager_runtime_init();
25     sl_device_init_lfxo();
26     sl_device_init_hfxo();
27     sl_device_init_clocks();
28     sl_device_init_emu();
29     sl_board_init();
30 }
31
32 void sl_driver_init(void)
33 {
34     sl_debug_swo_init();
35     sl_simple_led_init_instances();
36     sl_cos_send_config();
37 }
38
39 void sl_service_init(void)
40 {
41     sl_board_configure_vcom();
42     sl_sleeptimer_init();
43 }
44
45 void sl_stack_init(void)
46 {
47 }
48
49 void sl_internal_app_init(void)
50 {
51 }
52
53 void sl_platform_process_action(void)
54 {
55 }
56
57 void sl_service_process_action(void)
58 {
59 }
60

```

7. Users have the option to install the Clock Manager component.



8. After installing the Clock Manager component, `sl_platform_init` now calls the `sl_clock_manager_init` function and no longer calls the Device Initialization components.



```
1 #include "sl_event_handler.h"
2
3 #include "em_chip.h"
4 #include "sl_interrupt_manager.h"
5 #include "sl_board_init.h"
6 #include "sl_clock_manager_init.h"
7 #include "sl_device_init_dcdc.h"
8 #include "sl_clock_manager.h"
9 #include "sl_device_init_emu.h"
10 #include "sl_board_control.h"
11 #include "sl_sleeptimer.h"
12 #include "sl_debug_swo.h"
13 #include "sl_simple_led_instances.h"
14 #include "sl_cos.h"
15
16 void sl_platform_init(void)
17 {
18     CHIP_Init();
19     sl_interrupt_manager_init();
20     sl_board_preinit();
21     sl_clock_manager_init();
22     sl_device_init_dcdc();
23     sl_clock_manager_runtime_init();
24     sl_device_init_emu();
25     sl_board_init();
26 }
27
28 void sl_driver_init(void)
29 {
30     sl_debug_swo_init();
31     sl_simple_led_init_instances();
32     sl_cos_send_config();
33 }
34
35 void sl_service_init(void)
36 {
37     sl_board_configure_vcom();
38     sl_sleeptimer_init();
39 }
40
41 void sl_stack_init(void)
42 {
43 }
44
45 void sl_internal_app_init(void)
46 {
47 }
48
49 void sl_platform_process_action(void)
50 {
51 }
52
53 void sl_service_process_action(void)
54 {
55 }
56
57 void sl_stack_process_action(void)
58 {
59 }
60
```

6.2 Configuration

1. After installation, the Clock Manager component can be configured in the *.slcp. Press Configure to open the oscillator and clock tree settings.

The screenshot shows the 'SOFTWARE COMPONENTS' view in the IDE for the 'blink_baremetal_5' project. The left sidebar lists various components under 'Services', with 'Clock Manager' selected and highlighted. A blue gear icon next to 'Clock Manager' indicates it is configurable. The main panel displays the 'Clock Manager' configuration page, which includes a 'Configure' button (highlighted with a red box), a description of the module, its quality (PRODUCTION), dependencies (clock_manager requires 2 components), and dependents (0 components require clock_manager). The 'Uninstall' button is visible at the bottom.

blink_baremetal_5 OVERVIEW SOFTWARE COMPONENTS CONFIGURATION TOOLS

Filter components by Configurable Installed Installed by you SDK Extensions Quality

Services

- ▼ Clock Manager
 - ⊙ Clock Manager
 - ⊙ Clock Manager Runtime
- ▶ Clocks
- ▶ Co-Processor Communication
- ▶ Command Line Interface
- ▶ Device Initialization
- ▶ Ember Legacy
- ▶ IO Stream
- ▶ Interrupt manager
- ▶ Memory Manager
- ▶ Micrium
- ▶ NVM3
- ▶ Power Manager
- ▶ System Setup
- ▶ Timers
- ▶ Token Manager
 - ⊙ Device Manager
 - ⊙ Device Manager Clock
 - Device Manager GPIO
 - ⊙ Device Manager Peripheral
 - ⊙ Interrupt Manager

▶ Third Party

▶ Wi-Fi

▶ Wi-SUN

▶ Z-Wave

▶ Zigbee

Clock Manager Configure

Description

The Clock Manager module provides configuration of the different oscillators and the device clock tree through CMSIS Configuration wizard Annotations C header files. The module also provides an API to initialize the module and functions to perform a variety of oscillator and clock related operations. This includes:

- functions to fetch information about oscillators and clock branches frequency and precision.
- functions to interface the CPU module functionality and features like enabling modules' bus clock, RCO calibration, retrieving or setting oscillator calibration values, output clocks on GPIO, etc.

Quality

PRODUCTION

Dependencies

clock_manager requires 2 components

- ▶ Platform
- ▶ Services

Dependents

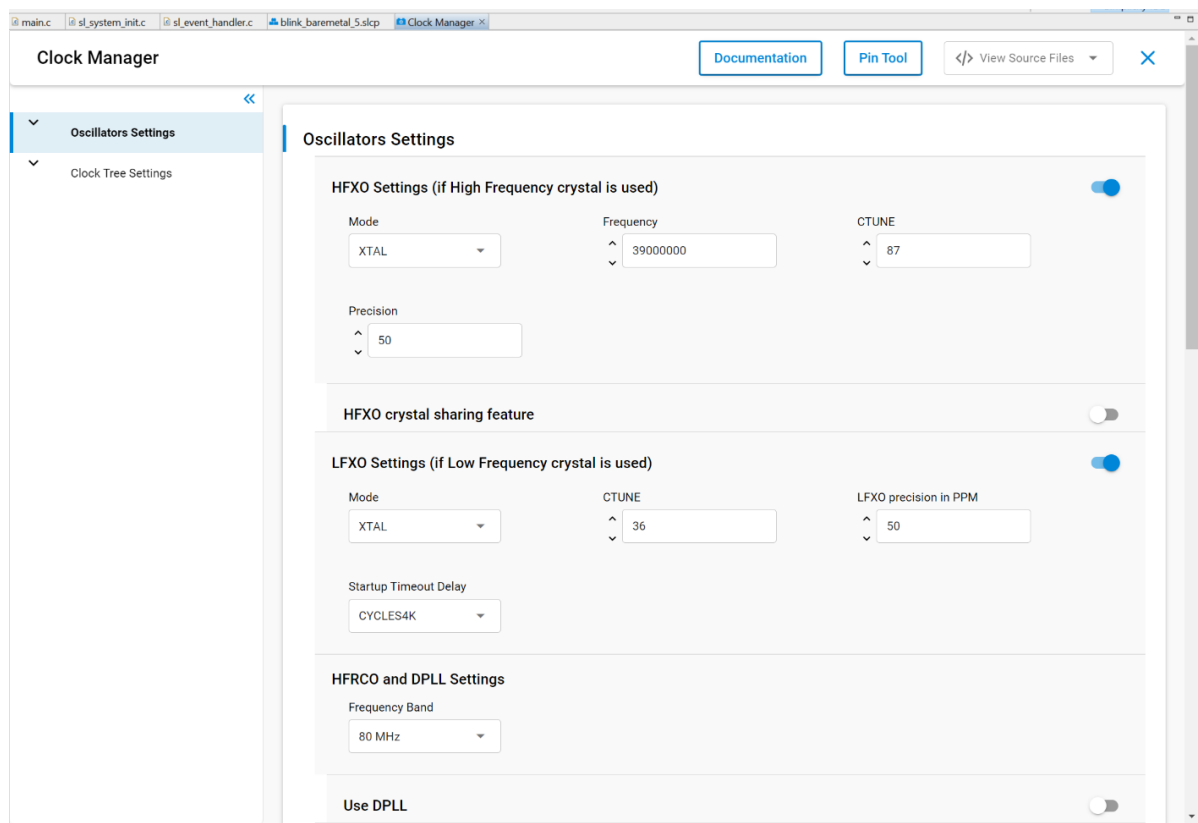
0 components require clock_manager
No Dependent Components

[Open in Browser](#)

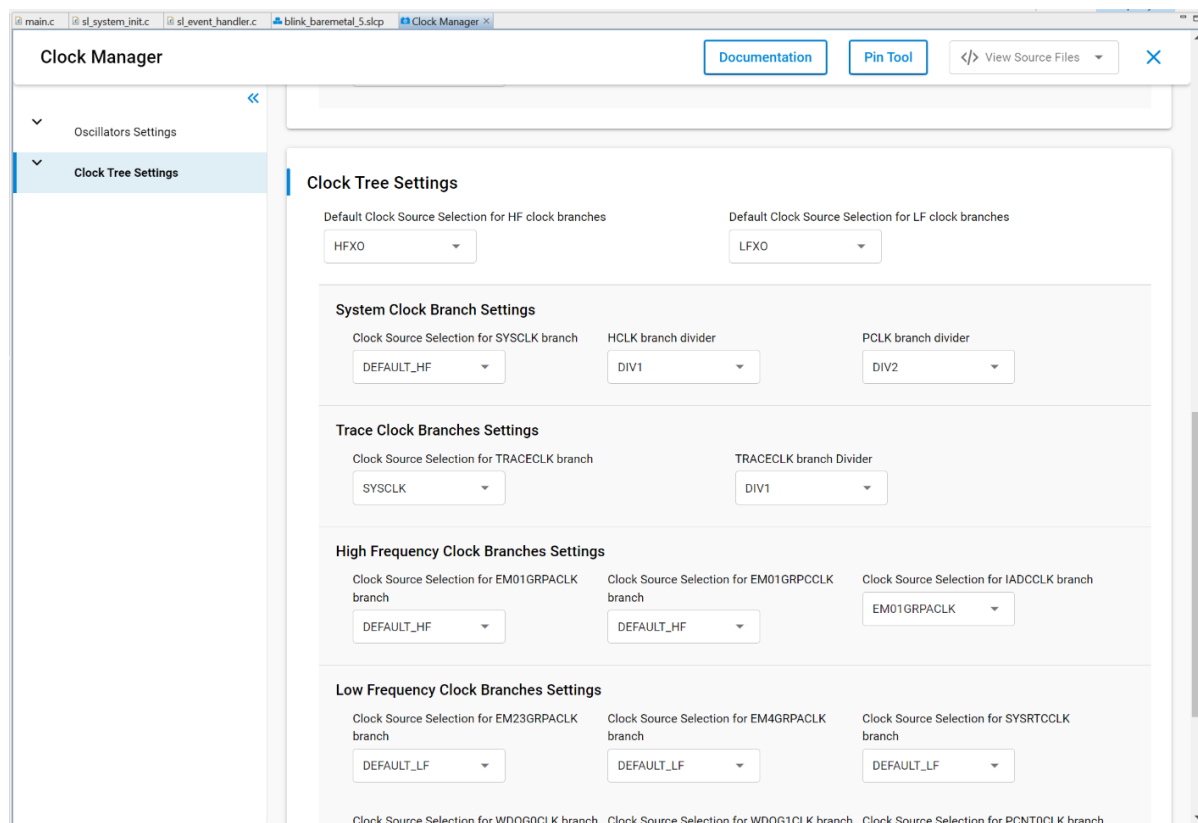
DOCUMENTATION

Uninstall

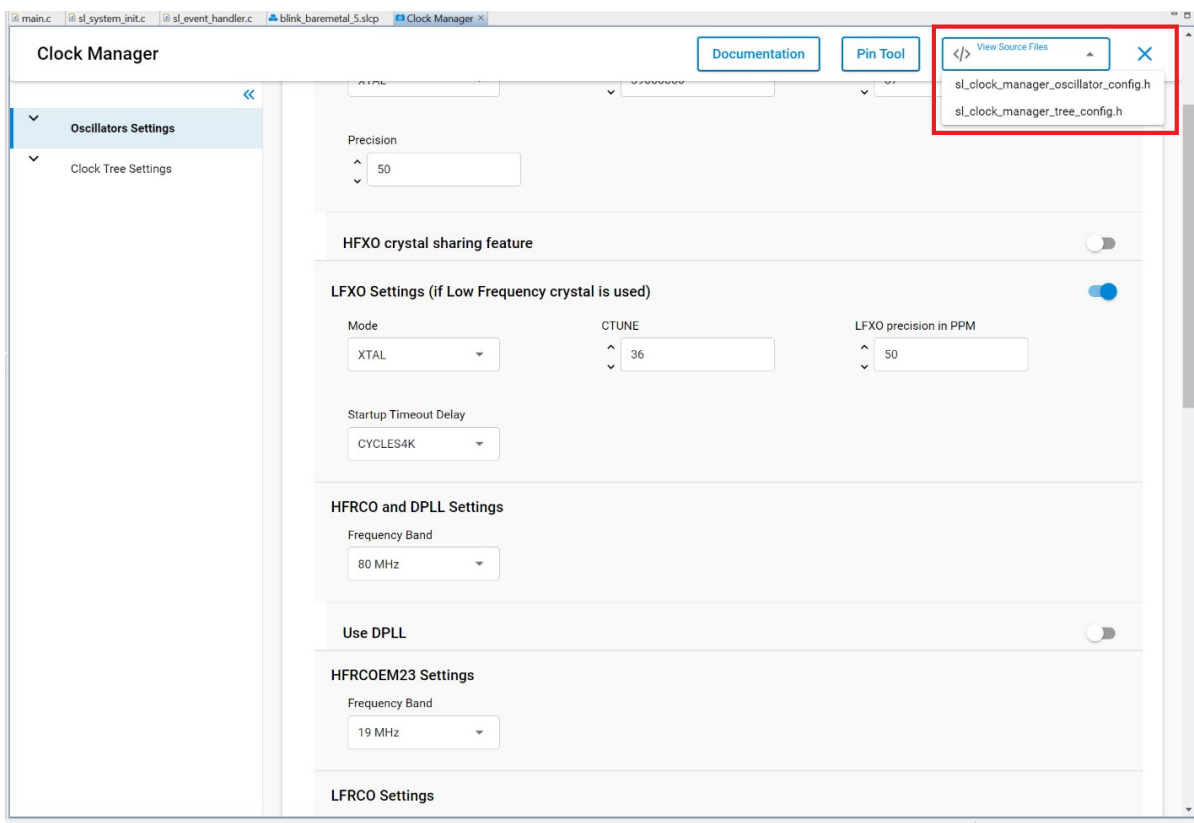
2. Both the oscillators and the clock tree settings are configurable in the *.slcp.



The clock sources and dividers are configurable in the clock tree settings.



3. The CMSIS annotated configuration files can be accessed in the *.slcp by pressing View Source Files.



4. The configuration macros can be viewed and modified in the configuration files. Oscillator configuration macros can be found in `sl_clock_manager_config.h`.

```

1  //*****
2  * @file
3  * @brief Clock Manager - Oscillators configuration file.
4  //*****
5  * License
6  * <b>Copyright 2024 Silicon Laboratories Inc. www.silabs.com</b>
7  //*****
8  *
9  * SPDX-License-Identifier: Zlib
10 *
11 * The licensor of this software is Silicon Laboratories Inc.
12 *
13 * This software is provided 'as-is', without any express or implied
14 * warranty. In no event will the authors be held liable for any damages
15 * arising from the use of this software.
16 *
17 * Permission is granted to anyone to use this software for any purpose,
18 * including commercial applications, and to alter it and redistribute it
19 * freely, subject to the following restrictions:
20 *
21 * 1. The origin of this software must not be misrepresented; you must not
22 * claim that you wrote the original software. If you use this software
23 * in a product, an acknowledgment in the product documentation would be
24 * appreciated but is not required.
25 * 2. Altered source versions must be plainly marked as such, and must not be
26 * misrepresented as being the original software.
27 * 3. This notice may not be removed or altered from any source distribution.
28 *
29 //*****
30
31 // <<< Use Configuration Wizard in Context Menu >>>
32
33 #ifndef SL_CLOCK_MANAGER_OSCILLATOR_CONFIG_H
34 #define SL_CLOCK_MANAGER_OSCILLATOR_CONFIG_H
35
36 // <h> Oscillators Settings
37
38 // <e SL_CLOCK_MANAGER_HFXO_EN> HFXO Settings (if High Frequency crystal is used)
39 // <i> Enable to configure HFXO
40 #ifndef SL_CLOCK_MANAGER_HFXO_EN
41 #define SL_CLOCK_MANAGER_HFXO_EN 1
42 #endif
43
44 // <o SL_CLOCK_MANAGER_HFXO_MODE> Mode
45 // <i>
46 // <HFXO_CFG_MODE_XTAL> XTAL
47 // <HFXO_CFG_MODE_EXTCLK> EXTCLK
48 // <HFXO_CFG_MODE_EXTCLKPKDET> EXTCLKPKDET
49 // <d> HFXO_CFG_MODE_XTAL
50 #ifndef SL_CLOCK_MANAGER_HFXO_MODE
51 #define SL_CLOCK_MANAGER_HFXO_MODE HFXO_CFG_MODE_XTAL
52 #endif
53
54 // <o SL_CLOCK_MANAGER_HFXO_FREQ> Frequency <38000000-40000000>
55 // <d> 39000000
56 #ifndef SL_CLOCK_MANAGER_HFXO_FREQ
57 #define SL_CLOCK_MANAGER_HFXO_FREQ 39000000
58 #endif
59
60 // <o SL_CLOCK_MANAGER_HFXO_CTUNE> CTUNE <0-255>

```

Clock tree configuration macros can be found in `sl_clock_manager_tree_config.h`.

```

1  //*****
2  * @file
3  * @brief Clock Manager - Clock Tree configuration file.
4  //*****
5  * License
6  * <b>Copyright 2024 Silicon Laboratories Inc. www.silabs.com</b>
7  //*****
8  *
9  * SPDX-License-Identifier: Zlib
10 *
11 * The licensor of this software is Silicon Laboratories Inc.
12 *
13 * This software is provided 'as-is', without any express or implied
14 * warranty. In no event will the authors be held liable for any damages
15 * arising from the use of this software.
16 *
17 * Permission is granted to anyone to use this software for any purpose,
18 * including commercial applications, and to alter it and redistribute it
19 * freely, subject to the following restrictions:
20 *
21 * 1. The origin of this software must not be misrepresented; you must not
22 * claim that you wrote the original software. If you use this software
23 * in a product, an acknowledgment in the product documentation would be
24 * appreciated but is not required.
25 * 2. Altered source versions must be plainly marked as such, and must not be
26 * misrepresented as being the original software.
27 * 3. This notice may not be removed or altered from any source distribution.
28 *
29 //*****
30
31 // <<< Use Configuration Wizard in Context Menu >>>
32
33 #ifndef SL_CLOCK_MANAGER_TREE_CONFIG_H
34 #define SL_CLOCK_MANAGER_TREE_CONFIG_H
35
36 // Internal Defines: DO NOT MODIFY
37 // Those defines are used internally to help converting the DEFAULT_HF_CLOCK_SOURCE and DEFAULT_LF_CLOCK_SOURCE
38 // selection of each clock branch to the right HW register value.
39 #define SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_HFRCODPLL 0xFF
40 #define SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_HFXO 0xFE
41 #define SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_FSRCO 0xFD
42 #define SL_CLOCK_MANAGER_DEFAULT_LF_CLOCK_SOURCE_LFRCO 0xFC
43 #define SL_CLOCK_MANAGER_DEFAULT_LF_CLOCK_SOURCE_LFXO 0xFB
44 #define SL_CLOCK_MANAGER_DEFAULT_LF_CLOCK_SOURCE_ULFRCO 0xFA
45
46 // <h> Clock Tree Settings
47
48 // <o SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE> Default Clock Source Selection for HF clock branches
49 // <SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_HFRCODPLL> HFRCODPLL
50 // <SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_HFXO> HFXO
51 // <SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_FSRCO> FSRCO
52 // <i> Selection of the high frequency clock source. HF clock branches can select this value by choosing the DEFAULT_HF value.
53 // <d> SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_HFRCODPLL
54 #ifndef SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE
55 #define SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE SL_CLOCK_MANAGER_DEFAULT_HF_CLOCK_SOURCE_HFXO
56 #endif
57
58 // <o SL_CLOCK_MANAGER_DEFAULT_LF_CLOCK_SOURCE> Default Clock Source Selection for LF clock branches
59 // <SL_CLOCK_MANAGER_DEFAULT_LF_CLOCK_SOURCE_LFRCO> LFRCO
60 // <SL_CLOCK_MANAGER_DEFAULT_LF_CLOCK_SOURCE_LFXO> LFXO

```

7. Revision History

Revision 0.1

August, 2024

- Initial revision.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!



IoT Portfolio
www.silabs.com/IoT



SW/HW
www.silabs.com/simplicity



Quality
www.silabs.com/quality



Support & Community
www.silabs.com/community

Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Trademark Information

Silicon Laboratories Inc.[®], Silicon Laboratories[®], Silicon Labs[®], SiLabs[®] and the Silicon Labs logo[®], Bluegiga[®], Bluegiga Logo[®], EFM[®], EFM32[®], EFR, Ember[®], Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals[®], WiSeConnect, n-Link, EZLink[®], EZRadio[®], EZRadioPRO[®], Gecko[®], Gecko OS, Gecko OS Studio, Precision32[®], Simplicity Studio[®], Telegesis, the Telegesis Logo[®], USBXpress[®], Zentri, the Zentri logo and Zentri DMS, Z-Wave[®], and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.



Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com