SMART ARM-based Microcontrollers

AT13486: SAM C System Clock Management (SYSTEM CLOCK) Driver

APPLICATION NOTE

Introduction

This driver for Atmel® | SMART ARM®-based microcontrollers provides an interface for the configuration and management of the device's clocking related functions. This includes the various clock sources, bus clocks, and generic clocks within the device, with functions to manage the enabling, disabling, source selection, and prescaling of clocks to various internal peripherals.

The following peripherals are used by this module:

- GCLK (Generic Clock Management)
- PM (Power Management)
- OSCCTRL (Oscillators Controller)
- OSC32KCTRL (32K Oscillators Controller)
- MCLK (Main Clock)

The following devices can use this module:

- Atmel | SMART SAM C20/C21

The outline of this documentation is as follows:

Table of Contents

1. Software License

2. Prerequisites

There are no prerequisites for this module.

3. Module Overview

The SAM devices contain a sophisticated clocking system, which is designed to give the maximum flexibility to the user application. This system allows a system designer to tune the performance and power consumption of the device in a dynamic manner, to achieve the best trade-off between the two for a particular application.

This driver provides a set of functions for the configuration and management of the various clock related functionality within the device.

3.1. Clock Sources

The SAM devices have a number of master clock source modules, each of which being capable of producing a stabilized output frequency, which can then be fed into the various peripherals and modules within the device.

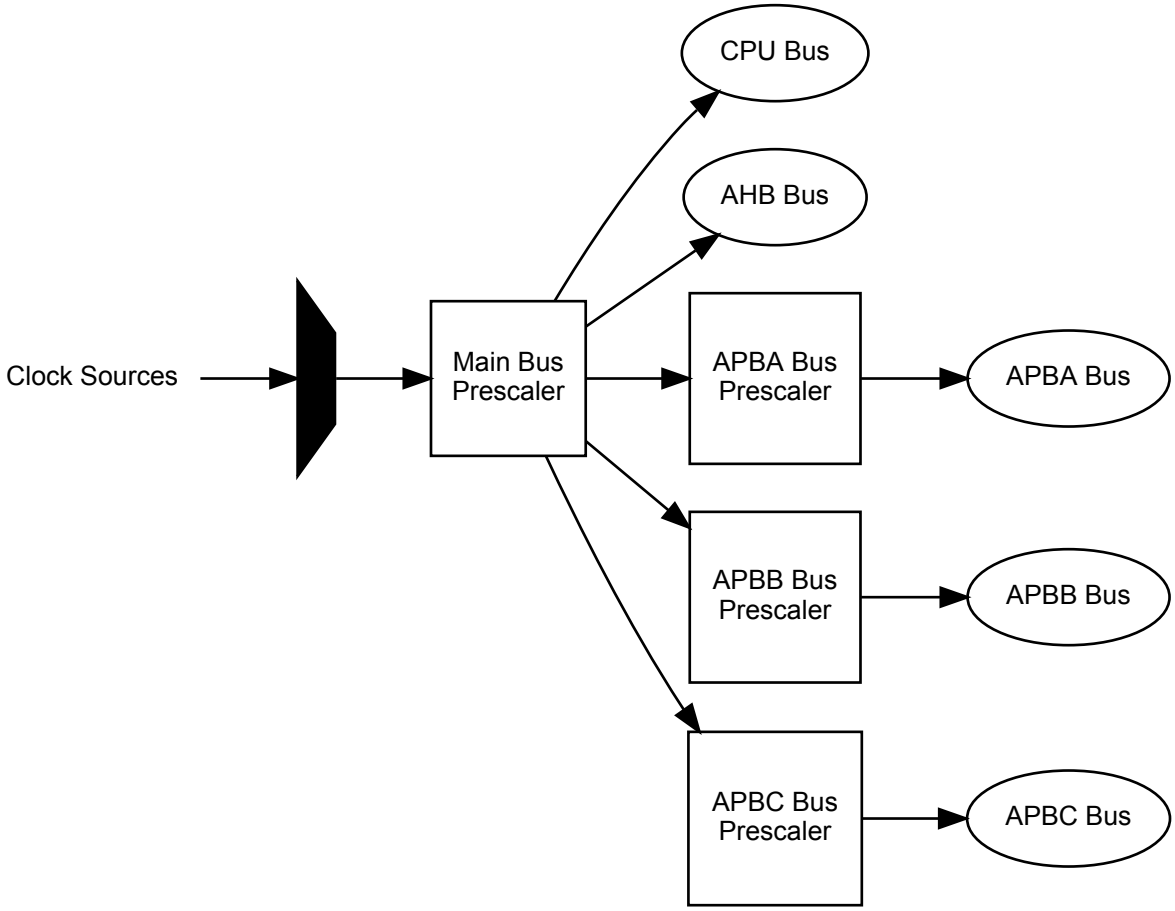
Possible clock source modules include internal R/C oscillators as well as external crystal oscillators and/or clock inputs.

3.2. CPU / Bus Clocks

The CPU and AHB/APBx buses are clocked by the same physical clock source (referred in this module as the Main Clock), however the APBx buses may have additional prescaler division ratios set to give each peripheral bus a different clock speed.

The general main clock tree for the CPU and associated buses is shown in the figure below.

Figure 3-1. CPU / Bus Clocks



3.3. Clock Masking

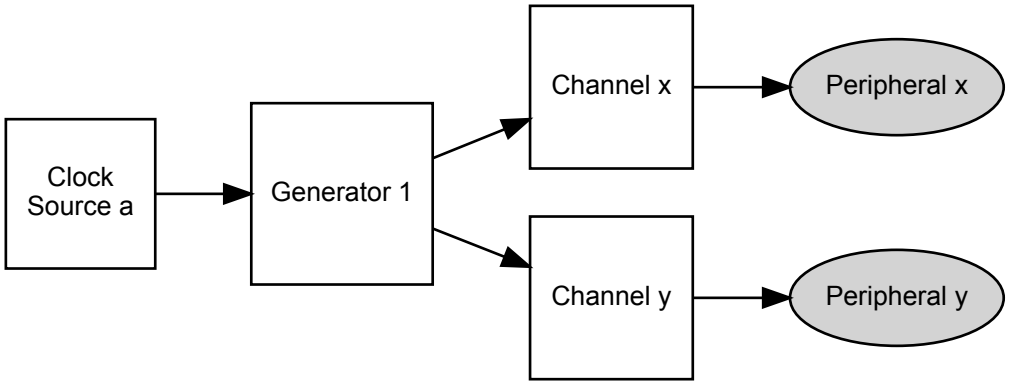
To save power, the input clock to one or more peripherals on the AHB and APBx buses can be masked away - when masked, no clock is passed into the module. Disabling of clocks of unused modules will prevent all access to the masked module, and will reduce the overall device power consumption.

3.4. Generic Clocks

Within the SAM devices there are a number of Generic Clocks; these are used to provide clocks to the various peripheral clock domains in the device in a standardized manner. One or more master source clocks can be selected as the input clock to a Generic Clock Generator, which can prescale down the input frequency to a slower rate for use in a peripheral.

Additionally, a number of individually selectable Generic Clock Channels are provided, which multiplex and gate the various generator outputs for one or more peripherals within the device. This setup allows for a single common generator to feed one or more channels, which can then be enabled or disabled individually as required.

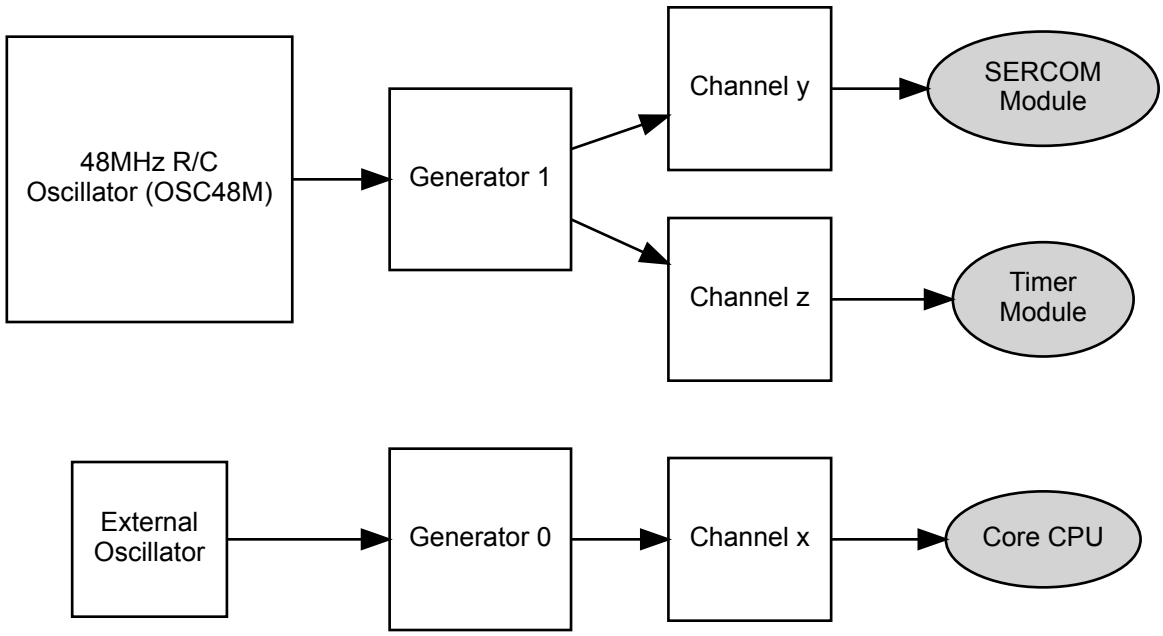
Figure 3-2. Generic Clocks



3.4.1. Clock Chain Example

An example setup of a complete clock chain within the device is shown in the figure below.

Figure 3-3. Clock Chain Example



3.4.2. Generic Clock Generators

Each Generic Clock generator within the device can source its input clock from one of the provided Source Clocks, and prescale the output for one or more Generic Clock Channels in a one-to-many relationship. The generators thus allow for several clocks to be generated of different frequencies, power usages, and accuracies, which can be turned on and off individually to disable the clocks to multiple peripherals as a group.

3.4.3. Generic Clock Channels

To connect a Generic Clock Generator to a peripheral within the device, a Generic Clock Channel is used. Each peripheral or peripheral group has an associated Generic Clock Channel, which serves as the clock input for the peripheral(s). To supply a clock to the peripheral module(s), the associated channel must be connected to a running Generic Clock Generator and the channel enabled.

4. Special Considerations

There are no special considerations for this module.

5. Extra Information

For extra information, see Extra Information for SYSTEM CLOCK Driver. This includes:

- Acronyms
- Dependencies
- Errata
- Module History

6. Examples

For a list of examples related to this driver, see Examples for System Clock Driver.

7. API Overview

7.1. Structure Definitions

7.1.1. Struct system_clock_source_dpll_config

DPLL oscillator configuration structure.

Table 7-1. Members

Type	Name	Description
enum system_clock_source_dpll_filter	filter	Filter type of the DPLL module
bool	lock_bypass	Bypass lock signal
enum system_clock_source_dpll_lock_time	lock_time	Lock time-out value of the DPLL module
bool	low_power_enable	Enable low power mode
bool	on_demand	Run On Demand. If this is set the DPLL won't run until requested by a peripheral
uint32_t	output_frequency	Output frequency of the clock
enum system_clock_source_dpll_prescaler	prescaler	DPLL prescaler
enum system_clock_source_dpll_reference_clock	reference_clock	Reference clock source of the DPLL module
uint16_t	reference_divider	Devider of reference clock
uint32_t	reference_frequency	Reference frequency of the clock
bool	run_in_standby	Keep the DPLL enabled in standby sleep mode
bool	wake_up_fast	Wake up fast. If this is set DPLL output clock is enabled after the startup time

7.1.2. Struct system_clock_source_osc32k_config

Internal 32KHz oscillator configuration structure.

Table 7-2. Members

Type	Name	Description
bool	enable_1khz_output	Enable 1KHz output
bool	enable_32khz_output	Enable 32KHz output
bool	on_demand	Run On Demand. If this is set the OSC32K won't run until requested by a peripheral
bool	run_in_standby	Keep the OSC32K enabled in standby sleep mode
enum system_osc32k_startup	startup_time	Startup time
bool	write_once	Lock configuration after it has been written, a device reset will release the lock

7.1.3. Struct system_clock_source_osc48m_config

Internal 48MHz (nominal) oscillator configuration structure.

Table 7-3. Members

Type	Name	Description
enum system_osc48m_div	div	Internal 48MHz RC oscillator divider
bool	on_demand	Run On Demand. If this is set the OSC48M won't run until requested by a peripheral
bool	run_in_standby	Keep the OSC48M enabled in standby sleep mode
enum system_osc48m_startup	startup_time	Crystal oscillator startup time

7.1.4. Struct system_clock_source_osculp32k_config

Internal 32KHz Ultra Low Power oscillator configuration structure.

Table 7-4. Members

Type	Name	Description
bool	write_once	Lock configuration after it has been written, a device reset will release the lock

7.1.5. Struct system_clock_source_xosc32k_config

External 32KHz oscillator clock configuration structure.

Table 7-5. Members

Type	Name	Description
enum system_clock_xosc32k_failure_detector_prescaler	clock_failure_detector_prescaler	Clock failure detector prescaler from OSCULP32K
bool	enable_1khz_output	Enable 1KHz output
bool	enable_32khz_output	Enable 32KHz output
bool	enable_clock_failure_detector	Enable clock failure detector
bool	enable_clock_failure_detector_event_outut	Enable clock failure detector event output
bool	enable_clock_switch_back	Enable clock switch back
enum system_clock_external	external_clock	External clock type
uint32_t	frequency	External clock/crystal frequency
bool	on_demand	Run On Demand. If this is set the XOSC32K won't run until requested by a peripheral
bool	run_in_standby	Keep the XOSC32K enabled in standby sleep mode

Type	Name	Description
enum system_xosc32k_startup	startup_time	Crystal oscillator startup time
bool	write_once	Lock configuration after it has been written. A device reset will release the lock

7.1.6. Struct system_clock_source_xosc_config

External oscillator clock configuration structure.

Table 7-6. Members

Type	Name	Description
bool	auto_gain_control	Enable automatic amplitude gain control
enum system_clock_xosc_failure_detector_prescaler	clock_failure_detector_prescaler	Clock failure detector prescaler from OSC48M for clock the CFD prescaler
bool	enable_clock_failure_detector	Enable clock failure detector
bool	enable_clock_failure_detector_event_outut	Enable clock failure detector event output
bool	enable_clock_switch_back	Enable clock switch back

Type	Name	Description
enum system_clock_external	external_clock	External clock type
uint32_t	frequency	External clock/crystal frequency
bool	on_demand	Run On Demand. If this is set the XOSC won't run until requested by a peripheral
bool	run_in_standby	Keep the XOSC enabled in standby sleep mode
enum system_xosc_startup	startup_time	Crystal oscillator startup time

7.1.7. Struct system_gclk_chan_config

Configuration structure for a Generic Clock channel. This structure should be initialized by the system_gclk_chan_get_config_defaults() function before being modified by the user application.

Table 7-7. Members

Type	Name	Description
enum gclk_generator	source_generator	Generic Clock Generator source channel

7.1.8. Struct system_gclk_gen_config

Configuration structure for a Generic Clock Generator channel. This structure should be initialized by the system_gclk_gen_get_config_defaults() function before being modified by the user application.

Table 7-8. Members

Type	Name	Description
uint32_t	division_factor	Integer division factor of the clock output compared to the input
bool	high_when_disabled	If `true`, the generator output level is high when disabled
bool	output_enable	If `true`, enables GCLK generator clock output to a GPIO pin

Type	Name	Description
bool	run_in_standby	If `true`, the clock is kept enabled during device standby mode
uint8_t	source_clock	Source clock input channel index, see the system_clock_source

7.2. Function Definitions

7.2.1. External Oscillator Management

7.2.1.1. Function system_clock_source_xosc_get_config_defaults()

Retrieve the default configuration for XOSC.

```
void system_clock_source_xosc_get_config_defaults(
        struct system_clock_source_xosc_config *const config)
```

Fills a configuration structure with the default configuration for an external oscillator module:

- External Crystal
- Startup time of 16384 external clock cycles
- Automatic crystal gain control mode enabled
- Frequency of 12MHz
- Don't run in STANDBY sleep mode
- Run only when requested by peripheral (on demand)
- Clock failure detector prescaler is 1
- Clock failure detector event outut is disabled
- Clock failure detector is disabled
- Clock switch back is disabled

Table 7-9. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to fill with default values

7.2.1.2. Function system_clock_source_xosc_set_config()

Configure the external oscillator clock source.

```
void system_clock_source_xosc_set_config(
        struct system_clock_source_xosc_config *const config)
```

Configures the external oscillator clock source with the given configuration settings.

Table 7-10. Parameters

Data direction	Parameter name	Description
[in]	config	External oscillator configuration structure containing the new config

7.2.1.3. Function system_clock_xosc_clock_failure_detected()

Checks if XOSC clock failure detected.

```
bool system_clock_xosc_clock_failure_detected( void )
```

Checks if XOSC clock detected.

Returns
XOSC clock failure detected state.

Table 7-11. Return Values

Return value	Description
true	XOSC clock failure detected
false	XOSC clock failure is not detected

7.2.1.4. Function system_clock_xosc_is_switched()

Checks if XOSC is switched and provides the safe clock.

```
bool system_clock_xosc_is_switched( void )
```

Checks if XOSC is switched and provides the safe clock.

Returns
XOSC clock switch state.

Table 7-12. Return Values

Return value	Description
true	XOSC clock is switched and provides the safe clock
false	XOSC clock is not switched and provides the external clock or crystal oscillator clock

7.2.2. External 32KHz Oscillator Management

7.2.2.1. Function system_clock_source_xosc32k_get_config_defaults()

Retrieve the default configuration for XOSC32K.

```
void system_clock_source_xosc32k_get_config_defaults(
        struct system_clock_source_xosc32k_config *const config)
```

Fills a configuration structure with the default configuration for an external 32KHz oscillator module:

- External Crystal
- Startup time of 16384 external clock cycles
- Automatic crystal gain control mode disabled
- Frequency of 32.768KHz
- 1KHz clock output disabled
- 32KHz clock output enabled
- Don't run in STANDBY sleep mode

- Run only when requested by peripheral (on demand)
- Don't lock registers after configuration has been written
- Clock failure detector prescaler is 1
- Clock failure detector event outut is disabled
- Clock failure detector is disabled
- Clock switch back is disabled

Table 7-13. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to fill with default values

7.2.2.2. Function system_clock_source_xosc32k_set_config()

Configure the XOSC32K external 32KHz oscillator clock source.

```
void system_clock_source_xosc32k_set_config(
        struct system_clock_source_xosc32k_config *const config)
```

Configures the external 32KHz oscillator clock source with the given configuration settings.

Table 7-14. Parameters

Data direction	Parameter name	Description
[in]	config	XOSC32K configuration structure containing the new config

7.2.2.3. Function system_clock_xosc32k_clock_failure_detected()

Checks if XOSC32K clock failure detected.

```
bool system_clock_xosc32k_clock_failure_detected( void )
```

Checks if XOSC32K clock detected.

Returns
XOSC32K clock failure detected state.

Table 7-15. Return Values

Return value	Description
true	XOSC32K clock failure detected
false	XOSC32K clock failure is not detected

7.2.2.4. Function system_clock_xosc32k_is_switched()

Checks if XOSC32K is switched and provides the safe clock.

```
bool system_clock_xosc32k_is_switched( void )
```

Checks if XOSC32K is switched and provides the safe clock.

Returns
XOSC32K clock switch state.

Table 7-16. Return Values

Return value	Description
true	XOSC32K clock is switched and provides the safe clock
false	XOSC32K clock is not switched and provides the external clock or crystal oscillator clock

7.2.3. Internal 32KHz Oscillator Management

7.2.3.1. Function system_clock_source_osc32k_get_config_defaults()

Retrieve the default configuration for OSC32K.

```
void system_clock_source_osc32k_get_config_defaults(
        struct system_clock_source_osc32k_config *const config)
```

Fills a configuration structure with the default configuration for an internal 32KHz oscillator module:

- 1KHz clock output enabled
- 32KHz clock output enabled
- Don't run in STANDBY sleep mode
- Run only when requested by peripheral (on demand)
- Set startup time to 130 cycles
- Don't lock registers after configuration has been written

Table 7-17. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to fill with default values

7.2.3.2. Function system_clock_source_osc32k_set_config()

Configure the internal OSC32K oscillator clock source.

```
void system_clock_source_osc32k_set_config(
        struct system_clock_source_osc32k_config *const config)
```

Configures the 32KHz (nominal) internal RC oscillator with the given configuration settings.

Table 7-18. Parameters

Data direction	Parameter name	Description
[in]	config	OSC32K configuration structure containing the new config

7.2.4. Internal Ultra Low Power 32KHz Oscillator management

7.2.4.1. Function system_clock_source_osculp32k_get_config_defaults()

Retrieve the default configuration for OSCULP32K.

```
void system_clock_source_osculp32k_get_config_defaults(
        struct system_clock_source_osculp32k_config *const config)
```

Fills a configuration structure with the default configuration for an internal Ultra Low Power 32KHz oscillator module:

- 1KHz clock output enabled
- 32KHz clock output enabled

Table 7-19. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to fill with default values

7.2.4.2. Function system_clock_source_osculp32k_set_config()

Configure the internal OSCULP32K oscillator clock source.

```
void system_clock_source_osculp32k_set_config(
        struct system_clock_source_osculp32k_config *const config)
```

Configures the Ultra Low Power 32KHz internal RC oscillator with the given configuration settings.

Table 7-20. Parameters

Data direction	Parameter name	Description
[in]	config	OSCULP32K configuration structure containing the new config

7.2.5. Internal 48MHz Oscillator Management

7.2.5.1. Function system_clock_source_osc48m_get_config_defaults()

Retrieve the default configuration for OSC48M.

```
void system_clock_source_osc48m_get_config_defaults(
        struct system_clock_source_osc48m_config *const config)
```

Fills a configuration structure with the default configuration for an internal 48MHz (nominal) oscillator module:

- Clock divider by 12 and output frequency 4MHz
- Don't run in STANDBY sleep mode
- Run only when requested by peripheral (on demand)
- Startup time of eight clock cycles

Table 7-21. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to fill with default values

7.2.5.2. Function system_clock_source_osc48m_set_config()

Configure the internal OSC48M oscillator clock source.

```
void system_clock_source_osc48m_set_config(
        struct system_clock_source_osc48m_config *const config)
```

Configures the 48MHz (nominal) internal RC oscillator with the given configuration settings.

Note: Frequency selection can be done only when OSC48M is disabled.

Table 7-22. Parameters

Data direction	Parameter name	Description
[in]	config	OSC48M configuration structure containing the new config

7.2.6. Clock Source Management

7.2.6.1. Function system_clock_source_write_calibration()

```
enum status_code system_clock_source_write_calibration(
        const enum system_clock_source system_clock_source,
        const uint16_t calibration_value,
        const uint8_t freq_range)
```

7.2.6.2. Function system_clock_source_enable()

```
enum status_code system_clock_source_enable(
        const enum system_clock_source system_clock_source)
```

7.2.6.3. Function system_clock_source_disable()

Disables a clock source.

```
enum status_code system_clock_source_disable(
        const enum system_clock_source clk_source)
```

Disables a clock source that was previously enabled.

Table 7-23. Parameters

Data direction	Parameter name	Description
[in]	clock_source	Clock source to disable

Table 7-24. Return Values

Return value	Description
STATUS_OK	Clock source was disabled successfully
STATUS_ERR_INVALID_ARG	An invalid or unavailable clock source was given

7.2.6.4. Function system_clock_source_is_ready()

Checks if a clock source is ready.

```
bool system_clock_source_is_ready(
        const enum system_clock_source clk_source)
```

Checks if a given clock source is ready to be used.

Table 7-25. Parameters

Data direction	Parameter name	Description
[in]	clock_source	Clock source to check if ready

Returns
Ready state of the given clock source.

Table 7-26. Return Values

Return value	Description
true	Clock source is enabled and ready
false	Clock source is disabled or not yet ready

7.2.6.5. Function system_clock_source_get_hz()

Retrieve the frequency of a clock source.

```
uint32_t system_clock_source_get_hz(
        const enum system_clock_source clk_source)
```

Determines the current operating frequency of a given clock source.

Table 7-27. Parameters

Data direction	Parameter name	Description
[in]	clock_source	Clock source to get the frequency of

Returns
Frequency of the given clock source, in Hz.

7.2.7. Main Clock Management

7.2.7.1. Function system_cpu_clock_set_divider()

Set main CPU clock divider.

```
void system_cpu_clock_set_divider(
        const enum system_main_clock_div divider)
```

Sets the clock divider used on the main clock to provide the CPU clock.

Table 7-28. Parameters

Data direction	Parameter name	Description
[in]	divider	CPU clock divider to set

7.2.7.2. Function system_cpu_clock_get_hz()

Retrieves the current frequency of the CPU core.

```
uint32_t system_cpu_clock_get_hz( void )
```

Retrieves the operating frequency of the CPU core, obtained from the main generic clock and the set CPU bus divider.

Returns
Current CPU frequency in Hz.

7.2.8. Bus Clock Masking

7.2.8.1. Function system_ahb_clock_set_mask()

Set bits in the clock mask for the AHB bus.

```
void system_ahb_clock_set_mask(
        const uint32_t ahb_mask)
```

This function will set bits in the clock mask for the AHB bus. Any bits set to 1 will enable that clock, zero bits in the mask will be ignored

Table 7-29. Parameters

Data direction	Parameter name	Description
[in]	ahb_mask	AHB clock mask to enable

7.2.8.2. Function system_ahb_clock_clear_mask()

Clear bits in the clock mask for the AHB bus.

```
void system_ahb_clock_clear_mask(
        const uint32_t ahb_mask)
```

This function will clear bits in the clock mask for the AHB bus. Any bits set to 1 will disable that clock, zero bits in the mask will be ignored.

Table 7-30. Parameters

Data direction	Parameter name	Description
[in]	ahb_mask	AHB clock mask to disable

7.2.8.3. Function system_apb_clock_set_mask()

Set bits in the clock mask for an APBx bus.

```
enum status_code system_apb_clock_set_mask(
        const enum system_clock_apb_bus bus,
        const uint32_t mask)
```

This function will set bits in the clock mask for an APBx bus. Any bits set to 1 will enable the corresponding module clock, zero bits in the mask will be ignored.

Table 7-31. Parameters

Data direction	Parameter name	Description
[in]	mask	APBx clock mask, a `SYSTEM_CLOCK_APB_APBx` constant from the device header files
[in]	bus	Bus to set clock mask bits for, a mask of `PM_APBxMASK_*` constants from the device header files

Returns
Status indicating the result of the clock mask change operation.

Table 7-32. Return Values

Return value	Description
STATUS_ERR_INVALID_ARG	Invalid bus given
STATUS_OK	The clock mask was set successfully

7.2.8.4. Function system_apb_clock_clear_mask()

Clear bits in the clock mask for an APBx bus.

```
enum status_code system_apb_clock_clear_mask(
        const enum system_clock_apb_bus bus,
        const uint32_t mask)
```

This function will clear bits in the clock mask for an APBx bus. Any bits set to 1 will disable the corresponding module clock, zero bits in the mask will be ignored.

Table 7-33. Parameters

Data direction	Parameter name	Description
[in]	mask	APBx clock mask, a `SYSTEM_CLOCK_APB_APBx` constant from the device header files
[in]	bus	Bus to clear clock mask bits

Returns
Status indicating the result of the clock mask change operation.

Table 7-34. Return Values

Return value	Description
STATUS_ERR_INVALID_ARG	Invalid bus ID was given
STATUS_OK	The clock mask was changed successfully

7.2.9. Internal DPLL Management

7.2.9.1. Function system_clock_source_dpll_get_config_defaults()

Retrieve the default configuration for DPLL.

```
void system_clock_source_dpll_get_config_defaults(
        struct system_clock_source_dpll_config *const config)
```

Fills a configuration structure with the default configuration for a DPLL oscillator module:

- Run only when requested by peripheral (on demand)
- Don't run in STANDBY sleep mode
- Lock bypass disabled
- Fast wake up disabled
- Low power mode disabled
- Output frequency is 48MHz
- Reference clock frequency is 32768Hz

- Not divide reference clock
- Select REF0 as reference clock
- Set lock time to default mode
- Use default filter

Table 7-35. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to fill with default values

7.2.9.2. Function system_clock_source_dpll_set_config()

Configure the DPLL clock source.

```
void system_clock_source_dpll_set_config(
        struct system_clock_source_dpll_config *const config)
```

Configures the Digital Phase-Locked Loop clock source with the given configuration settings.

Note: The DPLL will be running when this function returns, as the DPLL module needs to be enabled in order to perform the module configuration.

Table 7-36. Parameters

Data direction	Parameter name	Description
[in]	config	DPLL configuration structure containing the new config

7.2.10. System Clock Initialization

7.2.10.1. Function system_clock_init()

Initialize clock system based on the configuration in conf_clocks.h.

```
void system_clock_init( void )
```

This function will apply the settings in conf_clocks.h when run from the user application. All clock sources and GCLK generators are running when this function returns.

Note: OSC48M is always enabled and if the user selects other clocks for GCLK generators, the OSC48M default enable can be disabled after system_clock_init. Make sure the clock switches successfully before disabling OSC48M.

7.2.11. System Flash Wait States

7.2.11.1. Function system_flash_set_waitstates()

Set flash controller wait states.

```
void system_flash_set_waitstates(
        uint8_t wait_states)
```

Will set the number of wait states that are used by the onboard flash memory. The number of wait states depend on both device supply voltage and CPU speed. The required number of wait states can be found in the electrical characteristics of the device.

Table 7-37. Parameters

Data direction	Parameter name	Description
[in]	wait_states	Number of wait states to use for internal flash

7.2.12. Generic Clock Management

7.2.12.1. Function system_gclk_init()

Initializes the GCLK driver.

```
void system_gclk_init( void )
```

Initializes the Generic Clock module, disabling and resetting all active Generic Clock Generators and Channels to their power-on default values.

7.2.13. Generic Clock Management (Generators)

7.2.13.1. Function system_gclk_gen_get_config_defaults()

Initializes a Generic Clock Generator configuration structure to defaults.

```
void system_gclk_gen_get_config_defaults(
        struct system_gclk_gen_config *const config)
```

Initializes a given Generic Clock Generator configuration structure to a set of known default values. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is:

- The clock is generated undivided from the source frequency
- The clock generator output is low when the generator is disabled
- The input clock is sourced from input clock channel 0
- The clock will be disabled during sleep
- The clock output will not be routed to a physical GPIO pin

Table 7-38. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to initialize to default values

7.2.13.2. Function system_gclk_gen_set_config()

Writes a Generic Clock Generator configuration to the hardware module.

```
void system_gclk_gen_set_config(
        const uint8_t generator,
        struct system_gclk_gen_config *const config)
```

Writes out a given configuration of a Generic Clock Generator configuration to the hardware module.

Note: Changing the clock source on the fly (on a running generator) can take additional time if the clock source is configured to only run on-demand (ONDEMAND bit is set) and it is not currently running (no peripheral is requesting the clock source). In this case the GCLK will request the new clock while still keeping a request to the old clock source until the new clock source is ready.

Note: This function will not start a generator that is not already running; to start the generator, call system_gclk_gen_enable() after configuring a generator.

Table 7-39. Parameters

Data direction	Parameter name	Description
[in]	generator	Generic Clock Generator index to configure
[in]	config	Configuration settings for the generator

7.2.13.3. Function system_gclk_gen_enable()

Enables a Generic Clock Generator that was previously configured.

```
void system_gclk_gen_enable(
        const uint8_t generator)
```

Starts the clock generation of a Generic Clock Generator that was previously configured via a call to system_gclk_gen_set_config().

Table 7-40. Parameters

Data direction	Parameter name	Description
[in]	generator	Generic Clock Generator index to enable

7.2.13.4. Function system_gclk_gen_disable()

Disables a Generic Clock Generator that was previously enabled.

```
void system_gclk_gen_disable(
        const uint8_t generator)
```

Stops the clock generation of a Generic Clock Generator that was previously started via a call to system_gclk_gen_enable().

Table 7-41. Parameters

Data direction	Parameter name	Description
[in]	generator	Generic Clock Generator index to disable

7.2.13.5. Function system_gclk_gen_is_enabled()

Determins if the specified Generic Clock Generator is enabled.

```
bool system_gclk_gen_is_enabled(
        const uint8_t generator)
```

Table 7-42. Parameters

Data direction	Parameter name	Description
[in]	generator	Generic Clock Generator index to check

Returns
The enabled status.

Table 7-43. Return Values

Return value	Description
true	The Generic Clock Generator is enabled
false	The Generic Clock Generator is disabled

7.2.14. Generic Clock Management (Channels)

7.2.14.1. Function system_gclk_chan_get_config_defaults()

Initializes a Generic Clock configuration structure to defaults.

```
void system_gclk_chan_get_config_defaults(
        struct system_gclk_chan_config *const config)
```

Initializes a given Generic Clock configuration structure to a set of known default values. This function should be called on all new instances of these configuration structures before being modified by the user application.

The default configuration is as follows:

- The clock is sourced from the Generic Clock Generator channel 0
- The clock configuration will not be write-locked when set

Table 7-44. Parameters

Data direction	Parameter name	Description
[out]	config	Configuration structure to initialize to default values

7.2.14.2. Function system_gclk_chan_set_config()

Writes a Generic Clock configuration to the hardware module.

```
void system_gclk_chan_set_config(
        const uint8_t channel,
        struct system_gclk_chan_config *const config)
```

Writes out a given configuration of a Generic Clock configuration to the hardware module. If the clock is currently running, it will be stopped.

Note: Once called the clock will not be running. To start the clock, call system_gclk_chan_enable() after configuring a clock channel.

Table 7-45. Parameters

Data direction	Parameter name	Description
[in]	channel	Generic Clock channel to configure
[in]	config	Configuration settings for the clock

7.2.14.3. Function system_gclk_chan_enable()

Enables a Generic Clock that was previously configured.

```
void system_gclk_chan_enable(
        const uint8_t channel)
```

Starts the clock generation of a Generic Clock that was previously configured via a call to system_gclk_chan_set_config().

Table 7-46. Parameters

Data direction	Parameter name	Description
[in]	channel	Generic Clock channel to enable

7.2.14.4. Function system_gclk_chan_disable()

Disables a Generic Clock that was previously enabled.

```
void system_gclk_chan_disable(
        const uint8_t channel)
```

Stops the clock generation of a Generic Clock that was previously started via a call to system_gclk_chan_enable().

Table 7-47. Parameters

Data direction	Parameter name	Description
[in]	channel	Generic Clock channel to disable

7.2.14.5. Function system_gclk_chan_is_enabled()

Determins if the specified Generic Clock channel is enabled.

```
bool system_gclk_chan_is_enabled(
        const uint8_t channel)
```

Table 7-48. Parameters

Data direction	Parameter name	Description
[in]	channel	Generic Clock Channel index

Returns
The enabled status.

Table 7-49. Return Values

Return value	Description
true	The Generic Clock channel is enabled
false	The Generic Clock channel is disabled

7.2.14.6. Function system_gclk_chan_lock()

Locks a Generic Clock channel from further configuration writes.

```
void system_gclk_chan_lock(
        const uint8_t channel)
```

Locks a generic clock channel from further configuration writes. It is only possible to unlock the channel configuration through a power on reset.

Table 7-50. Parameters

Data direction	Parameter name	Description
[in]	channel	Generic Clock channel to enable

7.2.14.7. Function system_gclk_chan_is_locked()

Determins if the specified Generic Clock channel is locked.

```
bool system_gclk_chan_is_locked(
        const uint8_t channel)
```

Table 7-51. Parameters

Data direction	Parameter name	Description
[in]	channel	Generic Clock Channel index

Returns
The lock status.

Table 7-52. Return Values

Return value	Description
true	The Generic Clock channel is locked
false	The Generic Clock channel is not locked

7.2.15. Generic Clock Frequency Retrieval

7.2.15.1. Function system_gclk_gen_get_hz()

Retrieves the clock frequency of a Generic Clock generator.

```
uint32_t system_gclk_gen_get_hz(
        const uint8_t generator)
```

Determines the clock frequency (in Hz) of a specified Generic Clock generator, used as a source to a Generic Clock Channel module.

Table 7-53. Parameters

Data direction	Parameter name	Description
[in]	generator	Generic Clock Generator index

Returns
The frequency of the generic clock generator, in Hz.

7.2.15.2. Function system_gclk_chan_get_hz()

Retrieves the clock frequency of a Generic Clock channel.

```
uint32_t system_gclk_chan_get_hz(
        const uint8_t channel)
```

Determines the clock frequency (in Hz) of a specified Generic Clock channel, used as a source to a device peripheral module.

Table 7-54. Parameters

Data direction	Parameter name	Description
[in]	channel	Generic Clock Channel index

Returns
The frequency of the generic clock channel, in Hz.

7.3. Enumeration Definitions

7.3.1. Enum gclk_generator

List of Available GCLK generators. This enum is used in the peripheral device drivers to select the GCLK generator to be used for its operation.

The number of GCLK generators available is device dependent.

Table 7-55. Members

Enum value	Description
GCLK_GENERATOR_0	GCLK generator channel 0
GCLK_GENERATOR_1	GCLK generator channel 1
GCLK_GENERATOR_2	GCLK generator channel 2
GCLK_GENERATOR_3	GCLK generator channel 3
GCLK_GENERATOR_4	GCLK generator channel 4
GCLK_GENERATOR_5	GCLK generator channel 5
GCLK_GENERATOR_6	GCLK generator channel 6
GCLK_GENERATOR_7	GCLK generator channel 7
GCLK_GENERATOR_8	GCLK generator channel 8
GCLK_GENERATOR_9	GCLK generator channel 9
GCLK_GENERATOR_10	GCLK generator channel 10
GCLK_GENERATOR_11	GCLK generator channel 11
GCLK_GENERATOR_12	GCLK generator channel 12
GCLK_GENERATOR_13	GCLK generator channel 13
GCLK_GENERATOR_14	GCLK generator channel 14

Enum value	Description
GCLK_GENERATOR_15	GCLK generator channel 15
GCLK_GENERATOR_16	GCLK generator channel 16

7.3.2. Enum system_clock_apb_bus

Available bus clock domains on the APB bus.

Table 7-56. Members

Enum value	Description
SYSTEM_CLOCK_APB_APBA	Peripheral bus A on the APB bus
SYSTEM_CLOCK_APB_APBB	Peripheral bus B on the APB bus
SYSTEM_CLOCK_APB_APBC	Peripheral bus C on the APB bus

7.3.3. Enum system_clock_external

Available external clock source types.

Table 7-57. Members

Enum value	Description
SYSTEM_CLOCK_EXTERNAL_CRYSTAL	The external clock source is a crystal oscillator
SYSTEM_CLOCK_EXTERNAL_CLOCK	The connected clock source is an external logic level clock signal

7.3.4. Enum system_clock_source

Clock sources available to the GCLK generators

Table 7-58. Members

Enum value	Description
SYSTEM_CLOCK_SOURCE_OSC48M	Internal 48MHz RC oscillator
SYSTEM_CLOCK_SOURCE_OSC32K	Internal 32KHz RC oscillator
SYSTEM_CLOCK_SOURCE_XOSC	External oscillator
SYSTEM_CLOCK_SOURCE_XOSC32K	External 32KHz oscillator
SYSTEM_CLOCK_SOURCE_ULP32K	Internal Ultra Low Power 32KHz oscillator
SYSTEM_CLOCK_SOURCE_GCLKIN	Generator input pad
SYSTEM_CLOCK_SOURCE_GCLKGEN1	Generic clock generator one output
SYSTEM_CLOCK_SOURCE_DPLL	Digital Phase Locked Loop (DPLL)

7.3.5. Enum system_clock_source_dpll_filter

Table 7-59. Members

Enum value	Description
SYSTEM_CLOCK_SOURCE_DPLL_FILTER_DEFAULT	Default filter mode
SYSTEM_CLOCK_SOURCE_DPLL_FILTER_LOW_BANDWIDTH_FILTER	Low bandwidth filter
SYSTEM_CLOCK_SOURCE_DPLL_FILTER_HIGH_BANDWIDTH_FILTER	High bandwidth filter
SYSTEM_CLOCK_SOURCE_DPLL_FILTER_HIGH_DAMPING_FILTER	High damping filter

7.3.6. Enum system_clock_source_dpll_lock_time

Table 7-60. Members

Enum value	Description
SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_DEFAULT	Set no time-out as default
SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_8MS	Set time-out if no lock within 8ms
SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_9MS	Set time-out if no lock within 9ms
SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_10MS	Set time-out if no lock within 10ms
SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_11MS	Set time-out if no lock within 11ms

7.3.7. Enum system_clock_source_dpll_prescaler

Table 7-61. Members

Enum value	Description
SYSTEM_CLOCK_SOURCE_DPLL_DIV_1	DPLL output is divided by 1
SYSTEM_CLOCK_SOURCE_DPLL_DIV_2	DPLL output is divided by 2
SYSTEM_CLOCK_SOURCE_DPLL_DIV_4	DPLL output is divided by 4

7.3.8. Enum system_clock_source_dpll_reference_clock

Table 7-62. Members

Enum value	Description
SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_XOSC32K	Select XOSC32K as clock reference
SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_XOSC	Select XOSC as clock reference
SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_GCLK	Select GCLK as clock reference

7.3.9. Enum system_clock_xosc32k_failure_detector_prescaler

Available clock failure detector prescaler.

Table 7-63. Members

Enum value	Description
SYSTEM_CLOCK_XOSC32K_FAILURE_DETECTOR_PRESCALER_1	Divide OSCULP32K frequency by 1 for clock failure detector clock
SYSTEM_CLOCK_XOSC32K_FAILURE_DETECTOR_PRESCALER_2	Divide OSCULP32K frequency by 2 for clock failure detector clock

7.3.10. Enum system_clock_xosc_failure_detector_prescaler

Available clock failure detector prescaler.

Table 7-64. Members

Enum value	Description
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_1	Divide OSC48M frequency by 1 for clock failure detector clock
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_2	Divide OSC48M frequency by 2 for clock failure detector clock
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_4	Divide OSC48M frequency by 4 for clock failure detector clock
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_8	Divide OSC48M frequency by 8 for clock failure detector clock
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_16	Divide OSC48M frequency by 16 for clock failure detector clock
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_32	Divide OSC48M frequency by 32 for clock failure detector clock
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_64	Divide OSC48M frequency by 64 for clock failure detector clock
SYSTEM_CLOCK_XOSC_FAILURE_DETECTOR_PRESCALER_128	Divide OSC48M frequency by 128 for clock failure detector clock

7.3.11. Enum system_main_clock_div

Available division ratios for the CPU clocks.

Table 7-65. Members

Enum value	Description
SYSTEM_MAIN_CLOCK_DIV_1	Divide Main clock by one
SYSTEM_MAIN_CLOCK_DIV_2	Divide Main clock by two
SYSTEM_MAIN_CLOCK_DIV_4	Divide Main clock by four
SYSTEM_MAIN_CLOCK_DIV_8	Divide Main clock by eight
SYSTEM_MAIN_CLOCK_DIV_16	Divide Main clock by 16
SYSTEM_MAIN_CLOCK_DIV_32	Divide Main clock by 32
SYSTEM_MAIN_CLOCK_DIV_64	Divide Main clock by 64
SYSTEM_MAIN_CLOCK_DIV_128	Divide Main clock by 128

7.3.12. Enum system_osc32k_startup

Available internal 32KHz oscillator startup times, as a number of internal OSC32K clock cycles.

Table 7-66. Members

Enum value	Description
SYSTEM_OSC32K_STARTUP_3	Wait three clock cycles until the clock source is considered stable
SYSTEM_OSC32K_STARTUP_4	Wait four clock cycles until the clock source is considered stable
SYSTEM_OSC32K_STARTUP_6	Wait six clock cycles until the clock source is considered stable
SYSTEM_OSC32K_STARTUP_10	Wait ten clock cycles until the clock source is considered stable
SYSTEM_OSC32K_STARTUP_18	Wait 18 clock cycles until the clock source is considered stable
SYSTEM_OSC32K_STARTUP_34	Wait 34 clock cycles until the clock source is considered stable
SYSTEM_OSC32K_STARTUP_66	Wait 66 clock cycles until the clock source is considered stable
SYSTEM_OSC32K_STARTUP_130	Wait 130 clock cycles until the clock source is considered stable

7.3.13. Enum system_osc48m_div

Available prescalers for the internal 48MHz (nominal) system clock.

Table 7-67. Members

Enum value	Description
SYSTEM_OSC48M_DIV_1	Divide the 48MHz RC oscillator output by one
SYSTEM_OSC48M_DIV_2	Divide the 48MHz RC oscillator output by two
SYSTEM_OSC48M_DIV_3	Divide the 48MHz RC oscillator output by three
SYSTEM_OSC48M_DIV_4	Divide the 48MHz RC oscillator output by four

Enum value	Description
SYSTEM_OSC48M_DIV_5	Divide the 48MHz RC oscillator output by five
SYSTEM_OSC48M_DIV_6	Divide the 48MHz RC oscillator output by six
SYSTEM_OSC48M_DIV_7	Divide the 48MHz RC oscillator output by seven
SYSTEM_OSC48M_DIV_8	Divide the 48MHz RC oscillator output by eight
SYSTEM_OSC48M_DIV_9	Divide the 48MHz RC oscillator output by nine
SYSTEM_OSC48M_DIV_10	Divide the 48MHz RC oscillator output by ten
SYSTEM_OSC48M_DIV_11	Divide the 48MHz RC oscillator output by eleven
SYSTEM_OSC48M_DIV_12	Divide the 48MHz RC oscillator output by twelve
SYSTEM_OSC48M_DIV_13	Divide the 48MHz RC oscillator output by thirteen
SYSTEM_OSC48M_DIV_14	Divide the 48MHz RC oscillator output by fourteen
SYSTEM_OSC48M_DIV_15	Divide the 48MHz RC oscillator output by fifteen
SYSTEM_OSC48M_DIV_16	Divide the 48MHz RC oscillator output by sixteen

7.3.14. Enum system_osc48m_startup

Available internal 48MHz oscillator startup times, as a number of internal clock cycles.

Table 7-68. Members

Enum value	Description
SYSTEM_OSC48M_STARTUP_8	Wait 8 clock cycles until the clock source is considered stable
SYSTEM_OSC48M_STARTUP_16	Wait 16 clock cycles until the clock source is considered stable
SYSTEM_OSC48M_STARTUP_32	Wait 32 clock cycles until the clock source is considered stable
SYSTEM_OSC48M_STARTUP_64	Wait 64 clock cycles until the clock source is considered stable
SYSTEM_OSC48M_STARTUP_128	Wait 128 clock cycles until the clock source is considered stable
SYSTEM_OSC48M_STARTUP_256	Wait 256 clock cycles until the clock source is considered stable
SYSTEM_OSC48M_STARTUP_512	Wait 512 clock cycles until the clock source is considered stable
SYSTEM_OSC48M_STARTUP_1024	Wait 1024 clock cycles until the clock source is considered stable

7.3.15. Enum system_xosc32k_startup

Available external 32KHz oscillator startup times, as a number of external clock cycles.

Table 7-69. Members

Enum value	Description
SYSTEM_XOSC32K_STARTUP_2048	Wait 2048 clock cycles until the clock source is considered stable
SYSTEM_XOSC32K_STARTUP_4096	Wait 4096 clock cycles until the clock source is considered stable
SYSTEM_XOSC32K_STARTUP_16384	Wait 16384 clock cycles until the clock source is considered stable
SYSTEM_XOSC32K_STARTUP_32768	Wait 32768 clock cycles until the clock source is considered stable
SYSTEM_XOSC32K_STARTUP_65536	Wait 65536 clock cycles until the clock source is considered stable
SYSTEM_XOSC32K_STARTUP_131072	Wait 131072 clock cycles until the clock source is considered stable
SYSTEM_XOSC32K_STARTUP_262144	Wait 262144 clock cycles until the clock source is considered stable

7.3.16. Enum system_xosc_startup

Available external oscillator startup times, as a number of external clock cycles.

Table 7-70. Members

Enum value	Description
SYSTEM_XOSC_STARTUP_1	Wait one clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_2	Wait two clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_4	Wait four clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_8	Wait eight clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_16	Wait 16 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_32	Wait 32 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_64	Wait 64 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_128	Wait 128 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_256	Wait 256 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_512	Wait 512 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_1024	Wait 1024 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_2048	Wait 2048 clock cycles until the clock source is considered stable

Enum value	Description
SYSTEM_XOSC_STARTUP_4096	Wait 4096 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_8192	Wait 8192 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_16384	Wait 16384 clock cycles until the clock source is considered stable
SYSTEM_XOSC_STARTUP_32768	Wait 32768 clock cycles until the clock source is considered stable

8. Extra Information for SYSTEM CLOCK Driver

8.1. Acronyms

Below is a table listing the acronyms used in this module, along with their intended meanings.

Acronym	Description
DFLL	Digital Frequency Locked Loop
MUX	Multiplexer
MCLK	Main Clock
OSC32K	Internal 32KHz Oscillator
OSC48M	Internal 48MHz Oscillator
PLL	Phase Locked Loop
OSC	Oscillator
XOSC	External Oscillator
XOSC32K	External 32KHz Oscillator
AHB	Advanced High-performance Bus
APB	Advanced Peripheral Bus
DPLL	Digital Phase Locked Loop

8.2. Dependencies

This driver has the following dependencies:

- None

8.3. Errata

There are no errata related to this driver.

8.4. Module History

An overview of the module history is presented in the table below, with details on the enhancements and fixes made to the module since its first release. The current version of this corresponds to the newest version in the table.

Changelog
Initial Release

9. Examples for System Clock Driver

This is a list of the available Quick Start guides (QSGs) and example applications for SAM System Clock Management (SYSTEM CLOCK) Driver. QSGs are simple examples with step-by-step instructions to configure and use this driver in a selection of use cases. Note that a QSG can be compiled as a standalone application or be added to the user application.

- Quick Start Guide for SYSTEM CLOCK - Basic
- Quick Start Guide for SYSTEM CLOCK - GCLK Configuration

9.1. Quick Start Guide for SYSTEM CLOCK - Basic

In this case we apply the following configuration:

- RC8MHz (internal 8MHz RC oscillator)
  - Divide by four, giving a frequency of 2MHz
- DFLL (Digital frequency locked loop)
  - Open loop mode
  - 48MHz frequency
- CPU clock
  - Use two wait states when reading from flash memory
  - Use the DFLL, configured to 48MHz

9.1.1. Setup

9.1.1.1. Prerequisites

There are no special setup requirements for this use-case.

9.1.1.2. Code

Copy-paste the following setup code to your application:

```
void configure_extosc32k(void)
{
    struct system_clock_source_xosc32k_config config_ext32k;
    system_clock_source_xosc32k_get_config_defaults(&config_ext32k);

    config_ext32k.startup_time = SYSTEM_XOSC32K_STARTUP_4096;

    system_clock_source_xosc32k_set_config(&config_ext32k);
}

#if (!SAMC21)
void configure_dfll_open_loop(void)
{
    struct system_clock_source_dfll_config config_dfll;
    system_clock_source_dfll_get_config_defaults(&config_dfll);

    system_clock_source_dfll_set_config(&config_dfll);
}
#endif
```

9.1.1.3. Workflow

1. Create a EXTOSC32K module configuration struct, which can be filled out to adjust the configuration of the external 32KHz oscillator channel.

```
struct system_clock_source_xosc32k_config config_ext32k;
```

2. Initialize the oscillator configuration struct with the module's default values.

```
system_clock_source_xosc32k_get_config_defaults(&config_ext32k);
```

Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

3. Alter the EXTOSC32K module configuration struct to require a start-up time of 4096 clock cycles.

```
config_ext32k.startup_time = SYSTEM_XOSC32K_STARTUP_4096;
```

4. Write the new configuration to the EXTOSC32K module.

```
system_clock_source_xosc32k_set_config(&config_ext32k);
```

5. Create a DFLL module configuration struct, which can be filled out to adjust the configuration of the external 32KHz oscillator channel.

```
struct system_clock_source_dfll_config config_dfll;
```

6. Initialize the DFLL oscillator configuration struct with the module's default values.

```
system_clock_source_dfll_get_config_defaults(&config_dfll);
```

Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

7. Write the new configuration to the DFLL module.

```
system_clock_source_dfll_set_config(&config_dfll);
```

9.1.2. Use Case

9.1.2.1. Code

Copy-paste the following code to your user application:

```
/* Configure the external 32KHz oscillator */
configure_extosc32k();

/* Enable the external 32KHz oscillator */
enum status_code osc32k_status =
        system_clock_source_enable(SYSTEM_CLOCK_SOURCE_XOSC32K);

if (osc32k_status != STATUS_OK) {
    /* Error enabling the clock source */
}

#if (!SAMC21)
    /* Configure the DFLL in open loop mode using default values */
configure_dfll_open_loop();

/* Enable the DFLL oscillator */
enum status_code dfll_status =
        system_clock_source_enable(SYSTEM_CLOCK_SOURCE_DFLL);

if (dfll_status != STATUS_OK) {
    /* Error enabling the clock source */
```

```
}

/* Configure flash wait states before switching to high frequency clock */
system_flash_set_waitstates(2);

/* Change system clock to DFLL */
struct system_gclk_gen_config config_gclock_gen;
system_gclk_gen_get_config_defaults(&config_gclock_gen);
config_gclock_gen.source_clock    = SYSTEM_CLOCK_SOURCE_DFLL;
config_gclock_gen.division_factor = 1;
system_gclk_gen_set_config(GCLK_GENERATOR_0, &config_gclock_gen);
#endif
```

9.1.2.2. Workflow

1. Configure the external 32KHz oscillator source using the previously defined setup function.

```
configure_extosc32k();
```

2. Enable the configured external 32KHz oscillator source.

```
enum status_code osc32k_status =
        system_clock_source_enable(SYSTEM_CLOCK_SOURCE_XOSC32K);

if (osc32k_status != STATUS_OK) {
    /* Error enabling the clock source */
}
```

3. Configure the DFLL oscillator source using the previously defined setup function.

```
configure_dfll_open_loop();
```

4. Enable the configured DFLL oscillator source.

```
enum status_code dfll_status =
        system_clock_source_enable(SYSTEM_CLOCK_SOURCE_DFLL);

if (dfll_status != STATUS_OK) {
    /* Error enabling the clock source */
}
```

5. Configure the flash wait states to have two wait states per read, as the high speed DFLL will be used as the system clock. If insufficient wait states are used, the device may crash randomly due to misread instructions.

```
system_flash_set_waitstates(2);
```

6. Switch the system clock source to the DFLL, by reconfiguring the main clock generator.

```
struct system_gclk_gen_config config_gclock_gen;
system_gclk_gen_get_config_defaults(&config_gclock_gen);
config_gclock_gen.source_clock    = SYSTEM_CLOCK_SOURCE_DFLL;
config_gclock_gen.division_factor = 1;
system_gclk_gen_set_config(GCLK_GENERATOR_0, &config_gclock_gen);
```

9.2. Quick Start Guide for SYSTEM CLOCK - GCLK Configuration

In this use-case, the GCLK module is configured for:

- One generator attached to the internal 8MHz RC oscillator clock source
- Generator output equal to input frequency divided by a factor of 128
- One channel (connected to the TC0 module) enabled with the enabled generator selected

This use-case configures a clock channel to output a clock for a peripheral within the device, by first setting up a clock generator from a master clock source, and then linking the generator to the desired channel. This clock can then be used to clock a module within the device.

9.2.1. Setup

9.2.1.1. Prerequisites

There are no special setup requirements for this use-case.

9.2.1.2. Code

Copy-paste the following setup code to your user application:

```
void configure_gclock_generator(void)
{
    struct system_gclk_gen_config gclock_gen_conf;
    system_gclk_gen_get_config_defaults(&gclock_gen_conf);

#if (SAML21) || (SAML22)
    gclock_gen_conf.source_clock    = SYSTEM_CLOCK_SOURCE_OSC16M;
    gclock_gen_conf.division_factor = 128;
#elif (SAMC21)
    gclock_gen_conf.source_clock    = SYSTEM_CLOCK_SOURCE_OSC48M;
    gclock_gen_conf.division_factor = 128;
#else
    gclock_gen_conf.source_clock    = SYSTEM_CLOCK_SOURCE_OSC8M;
    gclock_gen_conf.division_factor = 128;
#endif

    system_gclk_gen_set_config(GCLK_GENERATOR_1, &gclock_gen_conf);

    system_gclk_gen_enable(GCLK_GENERATOR_1);
}

void configure_gclock_channel(void)
{
    struct system_gclk_chan_config gclk_chan_conf;
    system_gclk_chan_get_config_defaults(&gclk_chan_conf);

    gclk_chan_conf.source_generator = GCLK_GENERATOR_1;
#if (SAMD10) || (SAMD11)
    system_gclk_chan_set_config(TC1_GCLK_ID, &gclk_chan_conf);

    system_gclk_chan_enable(TC1_GCLK_ID);
#else
    system_gclk_chan_set_config(TC3_GCLK_ID, &gclk_chan_conf);

    system_gclk_chan_enable(TC3_GCLK_ID);
#endif
}
```

Add to user application initialization (typically the start of `main()`):

```
configure_gclock_generator();
configure_gclock_channel();
```

9.2.1.3. Workflow

1. Create a GCLK generator configuration struct, which can be filled out to adjust the configuration of a single clock generator.

   ```
   struct system_gclk_gen_config gclock_gen_conf;
   ```

2. Initialize the generator configuration struct with the module's default values.

```
system_gclk_gen_get_config_defaults(&gclock_gen_conf);
```

Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

3. Adjust the configuration struct to request the master clock source channel 0 is used as the source of the generator, and set the generator output prescaler to divide the input clock by a factor of 128.

```
gclock_gen_conf.source_clock    = SYSTEM_CLOCK_SOURCE_OSC16M;
gclock_gen_conf.division_factor = 128;
```

4. Configure the generator using the configuration structure.

```
system_gclk_gen_set_config(GCLK_GENERATOR_1, &gclock_gen_conf);
```

Note: The existing configuration struct may be re-used, as long as any values that have been altered from the default settings are taken into account by the user application.

5. Enable the generator once it has been properly configured, to begin clock generation.

```
system_gclk_gen_enable(GCLK_GENERATOR_1);
```

6. Create a GCLK channel configuration struct, which can be filled out to adjust the configuration of a single generic clock channel.

```
struct system_gclk_chan_config gclk_chan_conf;
```

7. Initialize the channel configuration struct with the module's default values.

```
system_gclk_chan_get_config_defaults(&gclk_chan_conf);
```

Note: This should always be performed before using the configuration struct to ensure that all values are initialized to known default settings.

8. Adjust the configuration struct to request the previously configured and enabled clock generator is used as the clock source for the channel.

```
gclk_chan_conf.source_generator = GCLK_GENERATOR_1;
```

9. Configure the channel using the configuration structure.

```
system_gclk_chan_set_config(TC1_GCLK_ID, &gclk_chan_conf);
```

Note: The existing configuration struct may be re-used, as long as any values that have been altered from the default settings are taken into account by the user application.

10. Enable the channel once it has been properly configured, to output the clock to the channel's peripheral module consumers.

```
system_gclk_chan_enable(TC1_GCLK_ID);
```

9.2.2. Use-case

9.2.2.1. Code

Copy-paste the following code to your user application:

```
while (true) {
    /* Nothing to do */
}
```

9.2.2.2. Workflow

1. As the clock is generated asynchronously to the system core, no special extra application code is required.

10. Document Revision History

Doc. Rev.	Date	Comments
42647A	01/2016	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**