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)
- SYSCTRL (Clock Source Control)

The following devices can use this module:

- Atmel | SMART SAM D20/D21
- Atmel | SMART SAM R21
- Atmel | SMART SAM D09/D10/D11
- Atmel | SMART SAM DA1

The outline of this documentation is as follows:

- Prerequisites
- Module Overview
- Special Considerations
- Extra Information
- Examples
- API Overview
7.3.12. Generic Clock Management........................................................................................ 26
7.3.13. Generic Clock Management (Generators).......................................................... 26
7.3.14. Generic Clock Management (Channels).............................................................. 28
7.3.15. Generic Clock Frequency Retrieval........................................................................ 31

7.4. Enumeration Definitions............................................................................................... 31
7.4.1. Enum gclk_generator............................................................................................ 31
7.4.2. Enum system_clock_apb_bus............................................................................... 32
7.4.3. Enum system_clock_dfll_chill_cycle.................................................................... 32
7.4.4. Enum system_clock_dfll_loop_mode..................................................................... 33
7.4.5. Enum system_clock_dfll_quick_lock................................................................... 33
7.4.6. Enum system_clock_dfll_stable_tracking............................................................ 33
7.4.7. Enum system_clock_dfll_wakeup_lock................................................................. 33
7.4.8. Enum system_clock_external............................................................................. 34
7.4.9. Enum system_clock_source................................................................................ 34
7.4.10. Enum system_clock_source_dpll_filter............................................................ 35
7.4.11. Enum system_clock_source_dpll_lock_time...................................................... 35
7.4.12. Enum system_clock_source_dpll_reference_clock.......................................... 35
7.4.13. Enum system_main_clock_div.......................................................................... 35
7.4.14. Enum system_osc32k_startup........................................................................... 36
7.4.15. Enum system_osc8m_div.................................................................................. 36
7.4.16. Enum system_osc8m_frequency_range............................................................. 36
7.4.17. Enum system_xosc32k_startup......................................................................... 37
7.4.18. Enum system_xosc_startup............................................................................... 37

8. Extra Information for SYSTEM CLOCK Driver.............................................................. 39
8.1. Acronyms.................................................................................................................. 39
8.2. Dependencies............................................................................................................ 39
8.3. Errata........................................................................................................................ 39
8.4. Module History........................................................................................................ 39

9. Examples for System Clock Driver............................................................................... 41
9.1. Quick Start Guide for SYSTEM CLOCK - Basic.................................................... 41
9.1.1. Setup.................................................................................................................. 41
9.1.2. Use Case............................................................................................................. 42
9.2. Quick Start Guide for SYSTEM CLOCK - GCLK Configuration.............................. 43
9.2.1. Setup.................................................................................................................. 44
9.2.2. Use Case............................................................................................................. 45

10. Document Revision History....................................................................................... 47
1. **Software License**

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of Atmel may not be used to endorse or promote products derived from this software without specific prior written permission.
4. This software may only be redistributed and used in connection with an Atmel microcontroller product.

THIS SOFTWARE IS PROVIDED BY ATMEL "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY AND SPECIFICALLY DISCLAIMED. IN NO EVENT SHALL ATMEL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
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. **Driver Feature Macro Definition**

<table>
<thead>
<tr>
<th>Driver Feature Macro</th>
<th>Supported devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEATURE_SYSTEM_CLOCK_DPLL</td>
<td>SAM D21, SAM R21, SAM D10, SAM D11, SAM DA1</td>
</tr>
</tbody>
</table>

**Note:** The specific features are only available in the driver when the selected device supports those features.

3.2. **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, internal DFLL modules, as well as external crystal oscillators and/or clock inputs.

3.3. **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 Figure 3-1 **CPU / Bus Clocks** on page 7.
3.4. **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, but will reduce the overall device power consumption.

3.5. **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.
3.5.1. Clock Chain Example

An example setup of a complete clock chain within the device is shown in Figure 3-3 Clock Chain Example on page 8.

3.5.2. 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.

3.5.3. 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_dfll_config

DFLL oscillator configuration structure.

Table 7-1. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum system_clock_dfll_chill_cycle</td>
<td>chill_cycle</td>
<td>Enable Chill Cycle</td>
</tr>
<tr>
<td>uint8_t</td>
<td>coarse_max_step</td>
<td>Coarse adjustment maximum step size (Closed loop mode)</td>
</tr>
<tr>
<td>uint8_t</td>
<td>coarse_value</td>
<td>Coarse calibration value (Open loop mode)</td>
</tr>
<tr>
<td>uint16_t</td>
<td>fine_max_step</td>
<td>Fine adjustment maximum step size (Closed loop mode)</td>
</tr>
<tr>
<td>uint16_t</td>
<td>fine_value</td>
<td>Fine calibration value (Open loop mode)</td>
</tr>
<tr>
<td>enum system_clock_dfll_loop_mode</td>
<td>loop_mode</td>
<td>Loop mode</td>
</tr>
<tr>
<td>uint16_t</td>
<td>multiply_factor</td>
<td>DFLL multiply factor (Closed loop mode)</td>
</tr>
<tr>
<td>bool</td>
<td>on_demand</td>
<td>Run On Demand. If this is set the DFLL won't run until requested by a peripheral.</td>
</tr>
<tr>
<td>enum system_clock_dfll_quick_lock</td>
<td>quick_lock</td>
<td>Enable Quick Lock</td>
</tr>
<tr>
<td>enum system_clock_dfll_stable_tracking</td>
<td>stable_tracking</td>
<td>DFLL tracking after fine lock</td>
</tr>
<tr>
<td>enum system_clock_dfll_wakeup_lock</td>
<td>wakeup_lock</td>
<td>DFLL lock state on wakeup</td>
</tr>
</tbody>
</table>

7.1.2. Struct system_clock_source_dpll_config

DPLL oscillator configuration structure.

Table 7-2. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum system_clock_source_dpll_filter</td>
<td>filter</td>
<td>Filter type of the DPLL module.</td>
</tr>
<tr>
<td>bool</td>
<td>lock_bypass</td>
<td>Bypass lock signal.</td>
</tr>
<tr>
<td>enum system_clock_source_dpll_lock_time</td>
<td>lock_time</td>
<td>Lock time-out value of the DPLL module.</td>
</tr>
<tr>
<td>bool</td>
<td>low_power_enable</td>
<td>Enable low power mode.</td>
</tr>
</tbody>
</table>
### 7.1.3. Struct system_clock_source_dpll_config

Internal 32KHz (nominal) oscillator configuration structure.

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

### 7.1.4. Struct system_clock_source_osc8m_config

Internal 8MHz (nominal) oscillator configuration structure.
### Table 7-4. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>on_demand</td>
<td>Run On Demand. If this is set the OSC8M won't run until requested by a peripheral.</td>
</tr>
<tr>
<td>enum system_osc8m_div</td>
<td>prescaler</td>
<td>Internal 8MHz RC oscillator prescaler</td>
</tr>
<tr>
<td>bool</td>
<td>run_in_standby</td>
<td>Keep the OSC8M enabled in standby sleep mode</td>
</tr>
</tbody>
</table>

#### 7.1.5. Struct system_clock_source_xosc32k_config

External 32KHz oscillator clock configuration structure.

### Table 7-5. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>auto_gain_control</td>
<td>Enable automatic amplitude control</td>
</tr>
<tr>
<td>bool</td>
<td>enable_1khz_output</td>
<td>Enable 1KHz output</td>
</tr>
<tr>
<td>bool</td>
<td>enable_32khz_output</td>
<td>Enable 32KHz output</td>
</tr>
<tr>
<td>enum system_clock_external</td>
<td>external_clock</td>
<td>External clock type</td>
</tr>
<tr>
<td>uint32_t</td>
<td>frequency</td>
<td>External clock/crystal frequency</td>
</tr>
<tr>
<td>bool</td>
<td>on_demand</td>
<td>Run On Demand. If this is set the XOSC32K won't run until requested by a peripheral.</td>
</tr>
<tr>
<td>bool</td>
<td>run_in_standby</td>
<td>Keep the XOSC32K enabled in standby sleep mode</td>
</tr>
<tr>
<td>enum system_xosc32k_startup</td>
<td>startup_time</td>
<td>Crystal oscillator start-up time</td>
</tr>
<tr>
<td>bool</td>
<td>write_once</td>
<td>Lock configuration after it has been written, a device reset will release the lock</td>
</tr>
</tbody>
</table>

#### 7.1.6. Struct system_clock_source_xosc_config

External oscillator clock configuration structure.

### Table 7-6. Members

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>auto_gain_control</td>
<td>Enable automatic amplitude gain control</td>
</tr>
<tr>
<td>enum system_clock_external</td>
<td>external_clock</td>
<td>External clock type</td>
</tr>
<tr>
<td>uint32_t</td>
<td>frequency</td>
<td>External clock/crystal frequency</td>
</tr>
<tr>
<td>bool</td>
<td>on_demand</td>
<td>Run On Demand. If this is set the XOSC won't run until requested by a peripheral.</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>run_in_standby</td>
<td>Keep the XOSC enabled in standby sleep mode</td>
</tr>
<tr>
<td>enum system_xosc_startup</td>
<td>startup_time</td>
<td>Crystal oscillator start-up time</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum gclk_generator</td>
<td>source_generator</td>
<td>Generic Clock Generator source channel</td>
</tr>
<tr>
<td>uint32_t</td>
<td>division_factor</td>
<td>Integer division factor of the clock output compared to the input</td>
</tr>
<tr>
<td>bool</td>
<td>high_when_disabled</td>
<td>If <code>true</code>, the generator output level is high when disabled</td>
</tr>
<tr>
<td>bool</td>
<td>output_enable</td>
<td>If <code>true</code>, enables GCLK generator clock output to a GPIO pin</td>
</tr>
<tr>
<td>bool</td>
<td>run_in_standby</td>
<td>If <code>true</code>, the clock is kept enabled during device standby mode</td>
</tr>
<tr>
<td>uint8_t</td>
<td>source_clock</td>
<td>Source clock input channel index, see the <code>system_clock_source</code></td>
</tr>
</tbody>
</table>

7.2. **Macro Definitions**

7.2.1. **Driver Feature Definition**

Define system clock features set according to different device family.

7.2.1.1. **Macro FEATURE_SYSTEM_CLOCK_DPLL**

```c
#define FEATURE_SYSTEM_CLOCK_DPLL
```

Digital Phase Locked Loop (DPLL) feature support.
7.3. Function Definitions

7.3.1. External Oscillator Management

7.3.1.1. Function `system_clock_source_xosc_get_config_defaults()`

Retrieve the default configuration for XOSC.

```c
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
- Start-up 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)

Table 7-9. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Configuration structure to fill with default values</td>
</tr>
</tbody>
</table>

7.3.1.2. Function `system_clock_source_xosc_set_config()`

Configure the external oscillator clock source.

```c
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

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>config</td>
<td>External oscillator configuration structure containing the new config</td>
</tr>
</tbody>
</table>

7.3.2. External 32KHz Oscillator Management

7.3.2.1. Function `system_clock_source_xosc32k_get_config_defaults()`

Retrieve the default configuration for XOSC32K.

```c
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
- Start-up 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

### Table 7-11. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Configuration structure to fill with default values</td>
</tr>
</tbody>
</table>

#### 7.3.2.2. Function `system_clock_source_xosc32k_set_config()`

Configure the XOSC32K external 32KHz oscillator clock source.

```c
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 7-12. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>config</td>
<td>XOSC32K configuration structure containing the new config</td>
</tr>
</tbody>
</table>

#### 7.3.3. Internal 32KHz Oscillator Management

#### 7.3.3.1. Function `system_clock_source_osc32k_get_config_defaults()`

Retrieve the default configuration for OSC32K.

```c
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 7-13. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Configuration structure to fill with default values</td>
</tr>
</tbody>
</table>

#### 7.3.3.2. Function `system_clock_source_osc32k_set_config()`

Configure the internal OSC32K oscillator clock source.

```c
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.
7.3.4. Internal 8MHz Oscillator Management

7.3.4.1. Function system_clock_source_osc8m_get_config_defaults()

Retrieve the default configuration for OSC8M.

```c
void system_clock_source_osc8m_get_config_defaults(
    struct system_clock_source_osc8m_config *const config)
```

Fills a configuration structure with the default configuration for an internal 8MHz (nominal) oscillator module:
- Clock output frequency divided by a factor of eight
- Don't run in STANDBY sleep mode
- Run only when requested by peripheral (on demand)

7.3.4.2. Function system_clock_source_osc8m_set_config()

Configure the internal OSC8M oscillator clock source.

```c
void system_clock_source_osc8m_set_config(
    struct system_clock_source_osc8m_config *const config)
```

Configures the 8MHz (nominal) internal RC oscillator with the given configuration settings.

7.3.5. Internal DFLL Management

7.3.5.1. Function system_clock_source_dfll_get_config_defaults()

Retrieve the default configuration for DFLL.

```c
void system_clock_source_dfll_get_config_defaults(
    struct system_clock_source_dfll_config *const config)
```

Fills a configuration structure with the default configuration for a DFLL oscillator module:
- Open loop mode
- QuickLock mode enabled
- Chill cycle enabled
- Output frequency lock maintained during device wake-up
- Continuous tracking of the output frequency
• Default tracking values at the mid-points for both coarse and fine tracking parameters
• Don't run in STANDBY sleep mode
• Run only when requested by peripheral (on demand)

### Table 7-17. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Configuration structure to fill with default values</td>
</tr>
</tbody>
</table>

#### 7.3.5.2. Function system\_clock\_source\_dfll\_set\_config()

Configure the DFLL clock source.

```c
void system_clock_source_dfll_set_config(
        struct system_clock_source_dfll_config *const config)
```

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

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

### Table 7-18. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>config</td>
<td>DFLL configuration structure containing the new config</td>
</tr>
</tbody>
</table>

#### 7.3.6. Clock Source Management

#### 7.3.6.1. Function system\_clock\_source\_write\_calibration()

```c
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)
```

#### 7.3.6.2. Function system\_clock\_source\_enable()

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

#### 7.3.6.3. Function system\_clock\_source\_disable()

Disables a clock source.

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

Disables a clock source that was previously enabled.

### Table 7-19. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>clock_source</td>
<td>Clock source to disable</td>
</tr>
</tbody>
</table>
### Table 7-20. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_OK</td>
<td>Clock source was disabled successfully</td>
</tr>
<tr>
<td>STATUS_ERR_INVALID_ARG</td>
<td>An invalid or unavailable clock source was given</td>
</tr>
</tbody>
</table>

### 7.3.6.4. Function `system_clock_source_is_ready()`

Checks if a clock source is ready.

```c
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 7-21. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>clock_source</td>
<td>Clock source to check if ready</td>
</tr>
</tbody>
</table>

**Returns**

Ready state of the given clock source.

### Table 7-22. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>Clock source is enabled and ready</td>
</tr>
<tr>
<td>false</td>
<td>Clock source is disabled or not yet ready</td>
</tr>
</tbody>
</table>

### 7.3.6.5. Function `system_clock_source_get_hz()`

Retrieve the frequency of a clock source.

```c
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 7-23. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>clock_source</td>
<td>Clock source to get the frequency</td>
</tr>
</tbody>
</table>

**Returns**

Frequency of the given clock source, in Hz.
7.3.7. Main Clock Management

7.3.7.1. Function system_cpu_clock_set_divider()

Set main CPU clock divider.

\[
\text{void system_cpu_clock_set_divider(}
\text{\hspace{1em} const enum system_main_clock_div divider)}
\]

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

Table 7-24. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>divider</td>
<td>CPU clock divider to set</td>
</tr>
</tbody>
</table>

7.3.7.2. Function system_cpu_clock_get_hz()

Retrieves the current frequency of the CPU core.

\[
\text{uint32_t system_cpu_clock_get_hz( void )}
\]

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.

7.3.7.3. Function system_apb_clock_set_divider()

Set APBx clock divider.

\[
\text{enum status_code system_apb_clock_set_divider(}
\text{\hspace{1em} const enum system_clock_apb_bus bus,}
\text{\hspace{1em} const enum system_main_clock_div divider)}
\]

Set the clock divider used on the main clock to provide the clock for the given APBx bus.

Table 7-25. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>divider</td>
<td>APBx bus divider to set</td>
</tr>
<tr>
<td>[in]</td>
<td>bus</td>
<td>APBx bus to set divider</td>
</tr>
</tbody>
</table>

Returns

Status of the clock division change operation.

Table 7-26. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_ERR_INVALID_ARG</td>
<td>Invalid bus ID was given</td>
</tr>
<tr>
<td>STATUS_OK</td>
<td>The APBx clock was set successfully</td>
</tr>
</tbody>
</table>
7.3.7.4. Function system_apb_clock_get_hz()

Retrieves the current frequency of a ABPx.

```c
uint32_t system_apb_clock_get_hz(
    const enum system_clock_apb_bus bus)
```

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

Returns
Current APBx bus frequency in Hz.

7.3.8. Bus Clock Masking

7.3.8.1. Function system_ahb_clock_set_mask()

Set bits in the clock mask for the AHB bus.

```c
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, 0 bits in the mask will be ignored.

Table 7-27. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>ahb_mask</td>
<td>AHB clock mask to enable</td>
</tr>
</tbody>
</table>

7.3.8.2. Function system_ahb_clock_clear_mask()

Clear bits in the clock mask for the AHB bus.

```c
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, 0 bits in the mask will be ignored.

Table 7-28. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>ahb_mask</td>
<td>AHB clock mask to disable</td>
</tr>
</tbody>
</table>

7.3.8.3. Function system_apb_clock_set_mask()

Set bits in the clock mask for an APBx bus.

```c
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 7-29. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mask</td>
<td>APBx clock mask, a <code>SYSTEM_CLOCK_APB_APBx</code> constant from the device header files</td>
</tr>
<tr>
<td>[in]</td>
<td>bus</td>
<td>Bus to set clock mask bits for, a mask of <code>PM_APBxMASK_</code> constants from the device header files</td>
</tr>
</tbody>
</table>

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

Table 7-30. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_ERR_INVALID_ARG</td>
<td>Invalid bus given</td>
</tr>
<tr>
<td>STATUS_OK</td>
<td>The clock mask was set successfully</td>
</tr>
</tbody>
</table>

7.3.8.4. Function `system_apb_clock_clear_mask()`

Clear bits in the clock mask for an APBx bus.

```c
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 7-31. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>mask</td>
<td>APBx clock mask, a <code>SYSTEM_CLOCK_APB_APBx</code> constant from the device header files</td>
</tr>
<tr>
<td>[in]</td>
<td>bus</td>
<td>Bus to clear clock mask bits</td>
</tr>
</tbody>
</table>

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

Table 7-32. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS_ERR_INVALID_ARG</td>
<td>Invalid bus ID was given</td>
</tr>
<tr>
<td>STATUS_OK</td>
<td>The clock mask was changed successfully</td>
</tr>
</tbody>
</table>
7.3.9. Internal DPLL Management

7.3.9.1. Function `system_clock_source_dpll_get_config_defaults()`

Retrieve the default configuration for DPLL.

```c
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>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Configuration structure to fill with default values</td>
</tr>
</tbody>
</table>

7.3.9.2. Function `system_clock_source_dpll_set_config()`

Configure the DPLL clock source.

```c
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>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>config</td>
<td>DPLL configuration structure containing the new config</td>
</tr>
</tbody>
</table>

7.3.10. System Clock Initialization

7.3.10.1. Function `system_clock_init()`

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

```c
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: OSC8M is always enabled and if user selects other clocks for GCLK generators, the OSC8M default enable can be disabled after system_clock_init. Make sure the clock switch successfully before disabling OSC8M.

7.3.11. System Flash Wait States

7.3.11.1. Function system_flash_set_waitstates()

Set flash controller wait states.

```c
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 7-35. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>wait_states</td>
<td>Number of wait states to use for internal flash</td>
</tr>
</tbody>
</table>

7.3.12. Generic Clock Management

7.3.12.1. Function system_gclk_init()

Initializes the GCLK driver.

```c
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.

7.3.13. Generic Clock Management (Generators)

7.3.13.1. Function system_gclk_gen_get_config_defaults()

Initializes a Generic Clock Generator configuration structure to defaults.

```c
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:

- Clock is generated undivided from the source frequency
- Clock generator output is low when the generator is disabled
- The input clock is sourced from input clock channel 0
- Clock will be disabled during sleep
- The clock output will not be routed to a physical GPIO pin
Table 7-36. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Configuration structure to initialize to default values</td>
</tr>
</tbody>
</table>

### Function `system_gclk_gen_set_config()`

Writes a Generic Clock Generator configuration to the hardware module.

```c
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 7-37. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>generator</td>
<td>Generic Clock Generator index to configure</td>
</tr>
<tr>
<td>[in]</td>
<td>config</td>
<td>Configuration settings for the generator</td>
</tr>
</tbody>
</table>

### Function `system_gclk_gen_enable()`

Enables a Generic Clock Generator that was previously configured.

```c
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 7-38. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>generator</td>
<td>Generic Clock Generator index to enable</td>
</tr>
</tbody>
</table>

### Function `system_gclk_gen_disable()`

Disables a Generic Clock Generator that was previously enabled.

```c
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 7-39. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>generator</td>
<td>Generic Clock Generator index to disable</td>
</tr>
</tbody>
</table>

7.3.13.5. Function system_gclk_gen_is_enabled()

Determins if the specified Generic Clock Generator is enabled.

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

Table 7-40. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>generator</td>
<td>Generic Clock Generator index to check</td>
</tr>
</tbody>
</table>

Returns

The enabled status.

Table 7-41. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The Generic Clock Generator is enabled</td>
</tr>
<tr>
<td>false</td>
<td>The Generic Clock Generator is disabled</td>
</tr>
</tbody>
</table>

7.3.14. Generic Clock Management (Channels)

7.3.14.1. Function system_gclk_chan_get_config_defaults()

Initializes a Generic Clock configuration structure to defaults.

```c
void system_gclk_chan_get_config_defaults(
    struct system_gclk_chan_conf *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:

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

Table 7-42. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[out]</td>
<td>config</td>
<td>Configuration structure to initialize to default values</td>
</tr>
</tbody>
</table>
7.3.14.2. Function system_gclk_chan_set_config()

Writes a Generic Clock configuration to the hardware module.

```c
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 7-43. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>channel</td>
<td>Generic Clock channel to configure</td>
</tr>
<tr>
<td>[in]</td>
<td>config</td>
<td>Configuration settings for the clock</td>
</tr>
</tbody>
</table>

7.3.14.3. Function system_gclk_chan_enable()

Enables a Generic Clock that was previously configured.

```c
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 7-44. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>channel</td>
<td>Generic Clock channel to enable</td>
</tr>
</tbody>
</table>

7.3.14.4. Function system_gclk_chan_disable()

Disables a Generic Clock that was previously enabled.

```c
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 7-45. Parameters**

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>channel</td>
<td>Generic Clock channel to disable</td>
</tr>
</tbody>
</table>

7.3.14.5. Function system_gclk_chan_is_enabled()

Determins if the specified Generic Clock channel is enabled.

```c
bool system_gclk_chan_is_enabled(
    const uint8_t channel)
```
Table 7-46. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>channel</td>
<td>Generic Clock Channel index</td>
</tr>
</tbody>
</table>

**Returns**
The enabled status.

Table 7-47. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The Generic Clock channel is enabled</td>
</tr>
<tr>
<td>false</td>
<td>The Generic Clock channel is disabled</td>
</tr>
</tbody>
</table>

7.3.14.6. Function system_gclk_chan_lock()

Locks a Generic Clock channel from further configuration writes.

```c
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 7-48. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>channel</td>
<td>Generic Clock channel to enable</td>
</tr>
</tbody>
</table>

7.3.14.7. Function system_gclk_chan_is_locked()

Determins if the specified Generic Clock channel is locked.

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

Table 7-49. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>channel</td>
<td>Generic Clock Channel index</td>
</tr>
</tbody>
</table>

**Returns**
The lock status.

Table 7-50. Return Values

<table>
<thead>
<tr>
<th>Return value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>The Generic Clock channel is locked</td>
</tr>
<tr>
<td>false</td>
<td>The Generic Clock channel is not locked</td>
</tr>
</tbody>
</table>
7.3.15.  Generic Clock Frequency Retrieval

7.3.15.1.  Function system_gclk_gen_get_hz()

Retrieves the clock frequency of a Generic Clock generator.

\[
\text{uint32}_t \text{ system_gclk_gen_get_hz(}
\text{const uint8}_t \text{ generator)}
\]

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

Table 7-51. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>generator</td>
<td>Generic Clock Generator index</td>
</tr>
</tbody>
</table>

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

7.3.15.2.  Function system_gclk_chan_get_hz()

Retrieves the clock frequency of a Generic Clock channel.

\[
\text{uint32}_t \text{ system_gclk_chan_get_hz(}
\text{const uint8}_t \text{ channel)}
\]

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

Table 7-52. Parameters

<table>
<thead>
<tr>
<th>Data direction</th>
<th>Parameter name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>channel</td>
<td>Generic Clock Channel index</td>
</tr>
</tbody>
</table>

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

7.4.  Enumeration Definitions

7.4.1.  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 7-53. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCLK_GENERATOR_0</td>
<td>GCLK generator channel 0</td>
</tr>
<tr>
<td>GCLK_GENERATOR_1</td>
<td>GCLK generator channel 1</td>
</tr>
<tr>
<td>Enum value</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>GCLK_GENERATOR_2</td>
<td>GCLK generator channel 2</td>
</tr>
<tr>
<td>GCLK_GENERATOR_3</td>
<td>GCLK generator channel 3</td>
</tr>
<tr>
<td>GCLK_GENERATOR_4</td>
<td>GCLK generator channel 4</td>
</tr>
<tr>
<td>GCLK_GENERATOR_5</td>
<td>GCLK generator channel 5</td>
</tr>
<tr>
<td>GCLK_GENERATOR_6</td>
<td>GCLK generator channel 6</td>
</tr>
<tr>
<td>GCLK_GENERATOR_7</td>
<td>GCLK generator channel 7</td>
</tr>
<tr>
<td>GCLK_GENERATOR_8</td>
<td>GCLK generator channel 8</td>
</tr>
<tr>
<td>GCLK_GENERATOR_9</td>
<td>GCLK generator channel 9</td>
</tr>
<tr>
<td>GCLK_GENERATOR_10</td>
<td>GCLK generator channel 10</td>
</tr>
<tr>
<td>GCLK_GENERATOR_11</td>
<td>GCLK generator channel 11</td>
</tr>
<tr>
<td>GCLK_GENERATOR_12</td>
<td>GCLK generator channel 12</td>
</tr>
<tr>
<td>GCLK_GENERATOR_13</td>
<td>GCLK generator channel 13</td>
</tr>
<tr>
<td>GCLK_GENERATOR_14</td>
<td>GCLK generator channel 14</td>
</tr>
<tr>
<td>GCLK_GENERATOR_15</td>
<td>GCLK generator channel 15</td>
</tr>
<tr>
<td>GCLK_GENERATOR_16</td>
<td>GCLK generator channel 16</td>
</tr>
</tbody>
</table>

### 7.4.2. Enum system_clock_apb_bus

Available bus clock domains on the APB bus.

Table 7-54. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_APB_APBA</td>
<td>Peripheral bus A on the APB bus</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_APB_APBB</td>
<td>Peripheral bus B on the APB bus</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_APB_APBC</td>
<td>Peripheral bus C on the APB bus</td>
</tr>
</tbody>
</table>

### 7.4.3. Enum system_clock_dfll_chill_cycle

DFLL chill-cycle behavior modes of the DFLL module. A chill cycle is a period of time when the DFLL output frequency is not measured by the unit, to allow the output to stabilize after a change in the input clock source.
Table 7-55. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_DFLL_CHILL_CYCLE_ENABLE</td>
<td>Enable a chill cycle, where the DFLL output frequency is not measured</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_DFLL_CHILL_CYCLE_DISABLE</td>
<td>Disable a chill cycle, where the DFLL output frequency is not measured</td>
</tr>
</tbody>
</table>

7.4.4. Enum system_clock_dfll_loop_mode

Available operating modes of the DFLL clock source module.

Table 7-56. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_DFLL_LOOP_MODE_OPEN</td>
<td>The DFLL is operating in open loop mode with no feedback</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_DFLL_LOOP_MODE_CLOSED</td>
<td>The DFLL is operating in closed loop mode with frequency feedback from a low frequency reference clock</td>
</tr>
</tbody>
</table>

7.4.5. Enum system_clock_dfll_quick_lock

DFLL QuickLock settings for the DFLL module, to allow for a faster lock of the DFLL output frequency at the expense of accuracy.

Table 7-57. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_DFLL_QUICK_LOCK_ENABLE</td>
<td>Enable the QuickLock feature for looser lock requirements on the DFLL</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_DFLL_QUICK_LOCK_DISABLE</td>
<td>Disable the QuickLock feature for strict lock requirements on the DFLL</td>
</tr>
</tbody>
</table>

7.4.6. Enum system_clock_dfll_stable_tracking

DFLL fine tracking behavior modes after a lock has been acquired.

Table 7-58. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_DFLL_STABLE_TRACKING_TRACK_AFTER_LOCK</td>
<td>Keep tracking after the DFLL has gotten a fine lock</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_DFLL_STABLE_TRACKING_FIX_AFTER_LOCK</td>
<td>Stop tracking after the DFLL has gotten a fine lock</td>
</tr>
</tbody>
</table>

7.4.7. Enum system_clock_dfll_wakeup_lock

DFLL lock behavior modes on device wake-up from sleep.
Table 7-59. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_DFLL_WAKEUP_LOCK_KEEP</td>
<td>Keep DFLL lock when the device wakes from sleep</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_DFLL_WAKEUP_LOCK_LOSE</td>
<td>Lose DFLL lock when the device wakes from sleep</td>
</tr>
</tbody>
</table>

7.4.8. Enum system_clock_external

Available external clock source types.

Table 7-60. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_EXTERNAL_CRYSTAL</td>
<td>The external clock source is a crystal oscillator</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_EXTERNAL_CLOCK</td>
<td>The connected clock source is an external logic level clock signal</td>
</tr>
</tbody>
</table>

7.4.9. Enum system_clock_source

Clock sources available to the GCLK generators.

Table 7-61. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_SOURCE_OSC8M</td>
<td>Internal 8MHz RC oscillator</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_OSC32K</td>
<td>Internal 32KHz RC oscillator</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_XOSC</td>
<td>External oscillator</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_XOSC32K</td>
<td>External 32KHz oscillator</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DFLL</td>
<td>Digital Frequency Locked Loop (DFLL)</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_ULP32K</td>
<td>Internal Ultra Low Power 32KHz oscillator</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_GCLKIN</td>
<td>Generator input pad</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_GCLKGEN1</td>
<td>Generic clock generator one output</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL</td>
<td>Digital Phase Locked Loop (DPLL). Check FEATURE_SYSTEM_CLOCK_DPLL for which device support it.</td>
</tr>
</tbody>
</table>
7.4.10. Enum system_clock_source_dpll_filter
Table 7-62. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_FILTER_DEFAULT</td>
<td>Default filter mode.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_FILTER_LOW_BANDWIDTH_FILTER</td>
<td>Low bandwidth filter.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_FILTER_HIGH_BANDWIDTH_FILTER</td>
<td>High bandwidth filter.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_FILTER_HIGH_DAMPING_FILTER</td>
<td>High damping filter.</td>
</tr>
</tbody>
</table>

7.4.11. Enum system_clock_source_dpll_lock_time
Table 7-63. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_DEFAULT</td>
<td>Set no time-out as default.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_8MS</td>
<td>Set time-out if no lock within 8ms.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_9MS</td>
<td>Set time-out if no lock within 9ms.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_10MS</td>
<td>Set time-out if no lock within 10ms.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_LOCK_TIME_11MS</td>
<td>Set time-out if no lock within 11ms.</td>
</tr>
</tbody>
</table>

7.4.12. Enum system_clock_source_dpll_reference_clock
Table 7-64. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_XOSC32K</td>
<td>Select XOSC32K as clock reference.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_XOSC</td>
<td>Select XOSC as clock reference.</td>
</tr>
<tr>
<td>SYSTEM_CLOCK_SOURCE_DPLL_REFERENCE_CLOCK_GCLK</td>
<td>Select GCLK as clock reference.</td>
</tr>
</tbody>
</table>

7.4.13. Enum system_main_clock_div
Available division ratios for the CPU and APB/AHB bus clocks.
Table 7-65. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_MAIN_CLOCK_DIV_1</td>
<td>Divide Main clock by one</td>
</tr>
<tr>
<td>SYSTEM_MAIN_CLOCK_DIV_2</td>
<td>Divide Main clock by two</td>
</tr>
<tr>
<td>SYSTEM_MAIN_CLOCK_DIV_4</td>
<td>Divide Main clock by four</td>
</tr>
</tbody>
</table>
### Enum system_osc32k_startup

Available internal 32KHz oscillator start-up times, as a number of internal OSC32K clock cycles.

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_OSC32K_STARTUP_3</td>
<td>Wait three clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_OSC32K_STARTUP_4</td>
<td>Wait four clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_OSC32K_STARTUP_6</td>
<td>Wait six clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_OSC32K_STARTUP_10</td>
<td>Wait ten clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_OSC32K_STARTUP_18</td>
<td>Wait 18 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_OSC32K_STARTUP_34</td>
<td>Wait 34 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_OSC32K_STARTUP_66</td>
<td>Wait 66 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_OSC32K_STARTUP_130</td>
<td>Wait 130 clock cycles until the clock source is considered stable</td>
</tr>
</tbody>
</table>

### Enum system_osc8m_div

Available prescalers for the internal 8MHz (nominal) system clock.

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_OSC8M_DIV_1</td>
<td>Do not divide the 8MHz RC oscillator output</td>
</tr>
<tr>
<td>SYSTEM_OSC8M_DIV_2</td>
<td>Divide the 8MHz RC oscillator output by two</td>
</tr>
<tr>
<td>SYSTEM_OSC8M_DIV_4</td>
<td>Divide the 8MHz RC oscillator output by four</td>
</tr>
<tr>
<td>SYSTEM_OSC8M_DIV_8</td>
<td>Divide the 8MHz RC oscillator output by eight</td>
</tr>
</tbody>
</table>

### Enum system_osc8m_frequency_range

Internal 8MHz RC oscillator frequency range setting.
Table 7-68. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_OSC8M_FREQUENCY_RANGE_4_TO_6</td>
<td>Frequency range 4MHz to 6MHz</td>
</tr>
<tr>
<td>SYSTEM_OSC8M_FREQUENCY_RANGE_6_TO_8</td>
<td>Frequency range 6MHz to 8MHz</td>
</tr>
<tr>
<td>SYSTEM_OSC8M_FREQUENCY_RANGE_8_TO_11</td>
<td>Frequency range 8MHz to 11MHz</td>
</tr>
<tr>
<td>SYSTEM_OSC8M_FREQUENCY_RANGE_11_TO_15</td>
<td>Frequency range 11MHz to 15MHz</td>
</tr>
</tbody>
</table>

7.4.17. Enum system_xosc32k_startup

Available external 32KHz oscillator start-up times, as a number of external clock cycles.

Table 7-69. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_0</td>
<td>Wait zero clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_32</td>
<td>Wait 32 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_2048</td>
<td>Wait 2048 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_4096</td>
<td>Wait 4096 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_16384</td>
<td>Wait 16384 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_32768</td>
<td>Wait 32768 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_65536</td>
<td>Wait 65536 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC32K_STARTUP_131072</td>
<td>Wait 131072 clock cycles until the clock source is considered stable</td>
</tr>
</tbody>
</table>

7.4.18. Enum system_xosc_startup

Available external oscillator start-up times, as a number of external clock cycles.

Table 7-70. Members

<table>
<thead>
<tr>
<th>Enum value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYSTEM_XOSC_STARTUP_1</td>
<td>Wait one clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_2</td>
<td>Wait two clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_4</td>
<td>Wait four clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>Enum value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_8</td>
<td>Wait eight clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_16</td>
<td>Wait 16 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_32</td>
<td>Wait 32 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_64</td>
<td>Wait 64 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_128</td>
<td>Wait 128 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_256</td>
<td>Wait 256 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_512</td>
<td>Wait 512 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_1024</td>
<td>Wait 1024 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_2048</td>
<td>Wait 2048 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_4096</td>
<td>Wait 4096 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_8192</td>
<td>Wait 8192 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_16384</td>
<td>Wait 16384 clock cycles until the clock source is considered stable</td>
</tr>
<tr>
<td>SYSTEM_XOSC_STARTUP_32768</td>
<td>Wait 32768 clock cycles until the clock source is considered stable</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DFLL</td>
<td>Digital Frequency Locked Loop</td>
</tr>
<tr>
<td>MUX</td>
<td>Multiplexer</td>
</tr>
<tr>
<td>OSC32K</td>
<td>Internal 32KHz Oscillator</td>
</tr>
<tr>
<td>OSC8M</td>
<td>Internal 8MHz Oscillator</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase Locked Loop</td>
</tr>
<tr>
<td>OSC</td>
<td>Oscillator</td>
</tr>
<tr>
<td>XOSC</td>
<td>External Oscillator</td>
</tr>
<tr>
<td>XOSC32K</td>
<td>External 32KHz Oscillator</td>
</tr>
<tr>
<td>AHB</td>
<td>Advanced High-performance Bus</td>
</tr>
<tr>
<td>APB</td>
<td>Advanced Peripheral Bus</td>
</tr>
<tr>
<td>DPLL</td>
<td>Digital Phase Locked Loop</td>
</tr>
</tbody>
</table>

8.2. Dependencies

This driver has the following dependencies:

- None

8.3. Errata

- This driver implements experimental workaround for errata 9905
  "The DFLL clock must be requested before being configured otherwise a write access to a DFLL register can freeze the device." This driver will enable and configure the DFLL before the ONDEMAND bit is set.

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

- Corrected OSC32K startup time definitions
- Support locking of OSC32K and XOSC32K config register (default: false)
- Added DPLL support, functions added:
  - `system_clock_source_dpll_get_config_defaults()` and `system_clock_source_dpll_set_config()`
- Moved gclk channel locking feature out of the config struct functions added:
  - `system_gclk_chan_lock()`, `system_gclk_chan_is_locked()`
  - `system_gclk_chan_is_enabled()` and `system_gclk_gen_is_enabled()`

**Fixed** `system_gclk_chan_disable()` deadlocking if a channel is enabled and configured to a failed/not running clock generator

- Changed default value for CONF_CLOCK_DFLL_ON_DEMAND from `true` to `false`
- Fixed `system_flash_set_waitstates()` failing with an assertion if an odd number of wait states provided

- Updated DFLL configuration function to implement workaround for errata 9905 in the DFLL module
- Updated `system_clock_init()` to reset interrupt flags before they are used
- Fixed `system_clock_source_get_hz()` to return correct DFLL frequency number

- Fixed `system_clock_source_is_ready` not returning the correct state for SYSTEM_CLOCK_SOURCE_OSC8M
- Renamed the various `system_clock_source_*_get_default_config()` functions to `system_clock_source_*_get_config_defaults()` to match the remainder of ASF
- Added OSC8M calibration constant loading from the device signature row when the oscillator is initialized
- Updated default configuration of the XOSC32 to disable Automatic Gain Control due to silicon errata

### 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
- DFL (Digital frequency locked loop)
  - Open loop mode
  - 48MHz frequency
- CPU clock
  - Use two wait states when reading from flash memory
  - Use the DFL, 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:

```c
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
```
9.1.3. Workflow

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

   ```c
   struct system_clock_source_xosc32k_config config_ext32k;
   ```

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

   ```c
   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.

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

4. Write the new configuration to the EXTOSC32K module.

   ```c
   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.

   ```c
   struct system_clock_source_dfll_config config_dfll;
   ```

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

   ```c
   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.

   ```c
   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:

```c
/* 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();
#endif

/* 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 */
}
```
9.1.2.2. Workflow

1. Configure the external 32KHz oscillator source using the previously defined setup function.
   ```c
   configure_extosc32k();
   ```

2. Enable the configured external 32KHz oscillator source.
   ```c
   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.
   ```c
   configure_dfll_open_loop();
   ```

4. Enable the configured DFLL oscillator source.
   ```c
   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.
   ```c
   system_flash_set_waitstates(2);
   ```

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

```c
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()`):

```c
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.

```c
struct system_gclk_gen_config gclock_gen_conf;
```
2. Initialize the generator configuration struct with the module's default values.
   
   ```c
   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.

   ```c
   `gclock_gen_conf.source_clock` = SYSTEM_CLOCK_SOURCE_OSC16M;
   `gclock_gen_conf.division_factor` = 128;
   
   4. Configure the generator using the configuration structure.

   ```c
   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.

   ```c
   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.

   ```c
   `struct` system_gclk_chan_config gclk_chan_conf;
   
   7. Initialize the channel configuration struct with the module's default values.

   ```c
   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.

   ```c
   `gclk_chan_conf.source_generator` = GCLK_GENERATOR_1;
   
   9. Configure the channel using the configuration structure.

   ```c
   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.

   ```c
   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:

   ```c
   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

<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>42119E</td>
<td>12/2015</td>
<td>Added support for SAM DA1 and SAM D09</td>
</tr>
<tr>
<td>42119D</td>
<td>12/2014</td>
<td>Added support for SAM R21 and SAM D10/D11</td>
</tr>
<tr>
<td>42119C</td>
<td>01/2014</td>
<td>Added support for SAM D21</td>
</tr>
<tr>
<td>42119B</td>
<td>06/2013</td>
<td>Corrected documentation typos. Fixed missing steps in the Basic Use Case Quick Start Guide</td>
</tr>
<tr>
<td>42119A</td>
<td>06/2013</td>
<td>Initial release</td>
</tr>
</tbody>
</table>