RX210

CGC Module Using Firmware Integration Technology

R01AN1968EU0100
Rev.1.00
Feb 21, 2014

Introduction

This module allows the user to configure any of the available clocks (HOCO, LOCO, Main Clock, PLL, Sub-Oscillator) as the system clock source, configure the internal clocks (ICLK, PCLK etc), configure the BCLK Clock Out feature and the Oscillation Stop Detection feature..

Target Device

The following is a list of devices that are currently supported by this API:

- **RX210**

Contents

1. Overview

This software provides a simple interface to the RX CGC (Clock Generator Circuit) module. The CGC module on the RX210 includes a High Speed On-Chip-Oscillator (HOCO) configurable for 32,36.864,40 or 50 MHz, a Low Speed On-Chip-Oscillator (LOCO) running at 125 KHz, supports an external oscillator input (Main Osc) up to a maximum of 20 MHz, a PLL circuit (50-100MHz) and a 32 KHz SubClock. The API allows the user to configure each of these clock sources including turning them on and off, configuring operating parameters where supported and setting them as the system clock.

The RX210 also has four internal clock domains with independent divisors for each. These are :

a. ICLK, which is the core clock (max 50 MHz).
b. PCLKB, which is the peripheral clock (max 32 MHz)
c. PCLKD, which is the clock source for the S12AD (max 50 MHz)
d. FCLK which is the clock source for the Flash memory (max 32 MHz)
e. BCLK which is the external bus clock (max 25 MHz)

The CGC module also supports configuration of these divisors.

1.1 Using the FIT CGC module

The primary function of the CGC module is to allow configuration and modification of the clocks at runtime. The ability to modify clock frequencies, turn off clocks and change divisors at runtime is important when trying to maximize MCU performance while minimizing current draw.

For details on adding and configuring the module for your project, refer to the section on Adding Middleware to Your Project.

1.2 API Overview

The following functions are included in this API:

Function	Description
R_CGC_ClockStart()	This function starts the specified clock. In case of the PLL, an extra argument that configures the divider and multiplier is used.
R_CGC_ClockStop()	This function stops the specified clock. If the specified clock is currently used as the system clock, the function returns an error.
R_CGC_SystemClockSet()	This function sets the specified clock as the system clock and configures the dividers for all the internal clocks based on the arguments passed. If the specified clock source is inactive or if the divider combination violates the hardware specifications, an error is returned.
R_CGC_ClockHzGet()	This function returns the specified internal clock source frequency (ICLK, PCLK etc) in Hz.
R_CGC_ClockCheck()	This function check if the specified clock is active and if so, checks for stability.
R_CGC_OscStopDetect()	This function configures the oscillation stop detection function and assigns a callback to the Non-maskable interrupt event.
R_CGC_OscStopStatusClear()	This function clears the Oscillation Stop Detection Status register.
R_CGC_Control ()	This function allows command specific control and configuration of the CLKOUT clock as well as the RX113 specific LCD and USB clocks.

2. API Information

This Middleware API follows the Renesas API naming standards.

2.1 Hardware Requirements

This middleware requires an RX210 MCU.

2.2 Hardware Resource Requirements

This driver does not require any resources other than the CGC module.

2.3 Software Requirements

This middleware is dependent upon the following software:

- FIT compliant BSP module r_bsp (v 2.30 or newer)

2.4 Limitations

This driver does not support ELC linking

2.5 Supported Toolchains

This middleware is tested and working with the following toolchains:

- Renesas RX Toolchain v1.02

2.6 Header Files

All API calls and their supporting interface definitions are located in r_cgc_rx210_if.h. Compile time configurable options are located in r_cgc_rx210_config.h. Both of these files should be included by the User's application.

2.7 Integer Types

This project uses ANSI C99 "Exact width integer types" in order to make the code clearer and more portable. These types are defined in *stdint.h*.

2.8 Configuration Overview

All configurable options that can be set at build time are located in the file "r_cgc_rx210_config.h". A summary of these settings are provided in the following table:

Configuration options in *r_cgc_rx210_config.h*	
`#define CGC_CFG_PARAM_CHECKING_ENABLE    1`	This compile time option will remove any code that checks for valid arguments supplied to functions.
`#define CGC_CFG_OSC_STOP_DET_USED        1`	This compile time function, when enabled, will implement Oscillation stop detection. When disabled, the respective code is not included and therefore not compiled.
`#define CGC_CFG_MAIN_CLOCK_OSC_SOURCE    0`	This define is used to indicate if a resonator (set to 0) or external oscillator input (set to 1) is used
`#define CGC_CFG_RESONATOR_TYPE`	This define is used to specify the type of resonator used : lead type or non-lead type ceramic resonator
`#define CGC_CFG_MAIN_OSC_WAIT`	This define is used to set the wait time to be used with the main clock oscillator. This should be set at a minimum to the wait time specified by the oscillator manufacturer. If an external oscillator input is used, this delay will be set to 0.
`#define CGC_CFG_PLL_WAIT`	This define is used to configure the PLL stabilization wait time. Refer to the hardware manual for appropriate values.
`#define CGC_CFG_HOCO_FREQUENCY`	This define is used to set the frequency of the HOCO. The RX210 HOCO can be set to 32, 36.864, 40 and 50 MHz
`#define CGC_CFG_SUBCLOCK_DRIVE`	This define is used to set the drive capacity of the subclock. Configure this according to the characteristics of the subclock.

2.9 API Data Structures

This section details the data structures that are used with the middleware's API functions.

```
struct _cgc_system_clock_config{
      cgc_system_clock_dividers_t pclkd_div;
      cgc_system_clock_dividers_t pclkb_div;
      cgc_system_clock_dividers_t bclk_div;
      cgc_system_clock_dividers_t iclk_div;
      cgc_system_clock_dividers_t fclk_div;
}cgc_system_clock_config_t;


struct _cgc_clock_config{
      cgc_pll_divider_t divider;
      cgc_pll_multiplier_t multiplier;
}cgc_clock_config_t;
```

2.9.1 Special Data Types

To provide strong type checking and reduce errors, many parameters used in API functions require arguments to be passed using the provided type definitions. Allowable values are defined in the public interface file r_cgc_rx210_if.h and r_cgc_rx210.h

2.10 Return Values

This shows the different values API functions can return. This enum is found in r_cgc_rx210_if.h along with the API function declarations.

```
typedef enum _cgc_err       // CGC API error codes
{
    CGC_SUCCESS = 0,
    CGC_ERR_CLOCK_INACTIVE,       // inactive clock specified as system clock
    CGC_ERR_CLOCK_ACTIVE,         // active clock source cannot be modified
without
                                  // stopping first.
    CGC_ERR_STABILIZED,           // Clock has stabilized after its been turned
on/off
    CGC_ERR_NOT_STABILIZED,       // clock has not stabilized after its been
turned
                                  // on/off
    CGC_ERR_MAIN_OSC_INACTIVE,    // PLL initialization attempted when main osc
is
                                  // turned off
    CGC_ERR_NULL_PTR,             // Null pointer passed as argument
    CGC_ERR_OSC_STOP_DET_ENABLED, // Illegal attempt to stop LOCO when
Oscillation
                                  // stop is enabled
    CGC_ERR_OSC_STOP_DETECTED,    // The Oscillation stop detection status flag
is set
    CGC_ERR_OSC_STOP_CLOCK_ACTIVE,// Attempt to clear Oscillation Stop Detect
Status
                                  // with PLL/MAIN_OSC active
    CGC_ERR_BCLK_EXCEEDED,        // Output on BCLK pin exceeds maximum
supported limit
    CGC_ERR_ILL_PLL_MODE,         // PLL used in Low Speed 1,2 mode
    CGC_ERR_PLL_EXCEEDED,         // Output of PLL divider or multiplier is
outside
                                  // allowable range
    CGC_ERR_NOT_OPEN,             // CGC function called before calling
R_CGC_Open()
} cgc_err_t;
```

2.11 Adding Middleware to Your Project

The driver must be added to an existing e2Studio project. It is best to use the e2Studio FIT plugin to add the driver to your project as that will automatically update the include file paths for you. Alternatively, the driver can be imported from the archive that accompanies this application note and manually added by following these steps:

1. This application note is distributed with a zip file package that includes the FIT RX210 CGC support module in its own folder r_cgc_rx210.
2. Unzip the package into the location of your choice.
3. In a file browser window, browse to the directory where you unzipped the distribution package and locate the r_cgc_rx210 folder.
4. Open your e2Studio workspace.
5. In the e2Studio project explorer window, select the project that you want to add the CGC module to.
6. Drag and drop the r_cgc_rx210 folder from the browser window (or copy/paste) into your e2Studio project at the top level of the project.
7. Update the source search/include paths for your project by adding the paths to the module files:
    a. Navigate to the "Add directory path" control:
        i. 'project name'->properties->C/C++ Build->Settings->Compiler->Source -Add (green + icon)

   b. Add the following paths:
      i. "${workspace_loc:/${ProjName}/r_cgc_rx210}"\
      ii. "${workspace_loc:/${ProjName}/r_cgc_rx210/src}"

Whether you used the plug-in or manually added the package to your project, it is necessary to configure the driver for your application.

8. Locate the r_cgc_rx210_config_reference.h file in the r_cgc_rx210/ref/ source folder in your project and copy it to your project's r_config folder.
9. Change the name of the copy in the r_config folder to r_cgc_rx210_config.h.
10. Make the required configuration settings by editing the copied r_cgc_rx210_config.h file. See Configuration Overview.

The CGC module uses the r_bsp package for certain MCU information. The r_bsp package is easily configured through the platform.h header file which is located in the r_bsp folder. To configure the r_bsp package, open up platform.h and uncomment the #include for the board you are using. For example, to run the project on a RSKRX210 board, the user would uncomment the #include for './board/rskrx210/r_bsp.h' macro and make sure all other board #includes are commented out.

3. API Functions

3.1 Summary

The following functions are included in this module:

Function	Description
R_CGC_ClockStart()	This function starts the specified clock. In case of the PLL, an extra argument that configures the divider and multiplier is used.
R_CGC_ClockStop()	This function stops the specified clock. If the specified clock is currently used as the system clock, the function returns an error.
R_CGC_SystemClockSet()	This function sets the specified clock as the system clock and configures the dividers for all the internal clocks based on the arguments passed. If the specified clock source is inactive or if the divider combination violates the hardware specifications, an error is returned.
R_CGC_ClockHzGet()	This function returns the specified internal clock source frequency (ICLK, PCLK etc) in Hz.
R_CGC_ClockCheck()	This function check if the specified clock is active and if so, checks for stability.
R_CGC_OscStopDetect()	This function configures the oscillation stop detection function and assigns a callback to the Non-maskable interrupt event.
R_CGC_OscStopStatusClear()	This function clears the Oscillation Stop Detection Status register.
R_CGC_Control ()	This function allows command specific control and configuration of the BCLK output.

3.2 R_CGC_Open()

This function performs initialization of certain clock registers that need to be set once after reset. Most of these configurations are dependent on external characteristics like crystal frequency, drive capacity and stabilization time and so are configurable by the user at compile time. This function must be executed once at startup. No other CGC functions are available until this function has been called.

Format

cgc_err_t R_CGC_Open(void)

Parameters

none

Return Values

CGC_SUCCESS:	*Successful; registers initialized*

Properties

Prototyped in file "r_cgc_rx210_if.h"

Description

This function performs initialization of certain clock registers that need to be set once after reset. Most of these configurations are dependent on external characteristics like crystal frequency, drive capacity and stabilization time and so are configurable by the user at compile time.
The function configures the following aspects of the clock generation module:

- HOCO stabilization time
- SubClock drive capacity
- Main Clock stabilization time
- Main Clock drive capacity

The Main Clock drive capacity is configured based on the frequency of the external crystal which is defined by *BSP_CFG_XTAL_HZ.*
The HOCO stabilization time is configured by setting the *CGC_CFG_HOCO_WAIT* definition.
The SubClock drive capacity is configured by the *CGC_CFG_SUBCLOCK_DRIVE* definition and the Main Clock stabilization time is configured by the *CGC_CFG_MAIN_OSC_WAIT* definition.

This function **must** be executed before any of the CGC functions are called and before the clock source or dividers are changed out of reset.

Reentrant

Function is re-entrant.

Example

```
cgc_err_t err;
err = R_CGC_Open();
```

Special Notes:

This function uses a software delay (~153 usec) for proper initialization of the SubClock registers.

3.3 R_CGC_ClockStart()

This function starts the specified clock source. The clock sources supported by this function on the RX210 are LOCO, HOCO, SubClock, Main Oscillator, and the PLL.

Format

```
cgc_err_t   R_CGC_ClockStart(cgc_clock_t clock_source,
                              cgc_clock_config_t* p_clock_config)
```

Parameters

clock_source
Clock source to start (see enumeration below)

p_clock_config
Pointer to configuration structure (see below) of type cgc_clock_config_t

The following enum, used as the first argument for this function (and others functions in the CGC module), lists the supported clock sources on the RX210.

```
typedef enum _cgc_clocks    // Available system clock sources
{
   CGC_LOCO      = 0x00,
   CGC_HOCO      = 0x01,
   CGC_MAIN_OSC = 0x02,
   CGC_SUBCLOCK = 0x03,
   CGC_PLL       = 0x04
} cgc_clocks_t;
```

The configuration for clock sources that have divider and multipliers (only the PLL for the RX210) are configured via the second argument. This argument is a pointer to a structure of type cgc_clock_config_t is shown below. Since this argument is only required when configuring the PLL, it can be a NULL pointer when starting other clock sources like the LOCO, HOCO and the SubClock which do not have any configuration options.

```
typedef struct _cgc_clock_config
{
   cgc_pll_divider_t    divider;        // specify clock divider
   cgc_pll_multiplier_t multiplier;     // specify clock multiplier
}
```

The divider and multiplier fields have to be populated with entries from the cgc_pll_divider_t and cgc_pll_multiplier_t enums which are shown below.

```
typedef enum _cgc_pll_divider
{
   CGC_PLL_DIV_1 = 0x00,
   CGC_PLL_DIV_2 = 0x01,
   CGC_PLL_DIV_4 = 0x02
}cgc_pll_divider_t;

typedef enum _cgc_pll_multiplier{
            CGC_PLL_MUL_8  = 0x07,
            CGC_PLL_MUL_10 = 0x09,
            CGC_PLL_MUL_12 = 0x0B,
            CGC_PLL_MUL_16 = 0x0F,
            CGC_PLL_MUL_20 = 0x13,
            CGC_PLL_MUL_24 = 0x17,
            CGC_PLL_MUL_25 = 0x18,
}cgc_pll_multiplier_t;
```

Return Values

CGC_SUCCESS:	*Successful; clock started*
CGC_ERR_NOT_STABILIZED:	*The clock source is not stabilized after being turned off*
CGC_ERR_CLOCK_ACTIVE:	*The clock source is already oscillating*
CGC_ERR_MAIN_OCO_INACTIVE:	*PLL Initialization attempted with Main OCO turned off*
CGC_ERR_ILL_PLL_MODE:	*Attempt to start PLL in Low Speed 1 or 2 mode.*
CGC_ERR_PLL_EXCEEDED:	*Output of PLL divider or multiplier is outside allowable range.*
CGC_ERR_ILL_PARAM:	*One of the parameters is invalid*
CGC_ERR_NULL_PTR:	*pClock_config pointer is NULL*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*

Properties

Prototyped in file "r_cgc_rx210_if.h"

Description

This is a blocking call that initializes the specified clock source. This function uses a software delay to wait until the specified clock source has stabilized. The function first checks to see if the specified clock source is already active and returns CGC_ERR_CLOCK_ACTIVE. The function returns with CGC_SUCCESS after the clock is turned on. When using this function make sure that a duration of at least 5 clock cycles of the clock being restarted have elapsed before the R_CGC_Clock_Start() function is called.

Reentrant

Function is re-entrant.

Example

```
 cgc_clock_config_t clock_config;
 cgc_err_t err;
 err = R_CGC_ClockStart(CGC_HOCO, NULL); //second argument can be NULL for HOCO
 err = R_CGC_ClockStart(CGC_LOCO, NULL); //second argument can be NULL for LOCO
 err = R_CGC_ClockStart(CGC_MAIN_OSC, NULL); //second argument can be NULL for
MAIN_OSC
 err = R_CGC_ClockStart(CGC_SUBCLOCK, NULL); //second argument can be NULL for
SUBCLOCK
 clock_config.divider = CGC_PLL_DIV_4;
 clock_config.multiplier = CGC_PLL_MUL_10;
 err = R_CGC_ClockStart(CGC_LOCO, clock_config);//second argument can't be NULL
for PLL
```

Special Notes:

The PLL cannot be turned ON if the main oscillator is not active. The function will return *CGC_ERR_MAIN_OCO_INACTIVE* in this case. The PLL also requires that the output of the divider should be between 4-12 MHz and that of the multiplier be between 50 and 100 MHz. Violation of these conditions or passing undefined arguments will return *CGC_ERR_ILL_PARAM.*

3.4 R_CGC_ClockStop()

This function turns off the specified clock source.

Format

cgc_err_t R_CGC_ClockStop(cgc_clocks_t clock_source)

Parameters

Clock_source

The clock source to be stopped. This is of type cgc_clocks_t which was expanded in R_CGC_ClockStart().

Return Values

CGC_SUCCESS:	*Successful; clock stopped*
CGC_ERR_CLOCK_ACTIVE:	*The clock source is currently used as the system clock and cannot be stopped.*
CGC_ERR_NOT_STABILIZED:	*Clock cannot be stopped if it has not stabilized after being turned on*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*
CGC_ERR_OSC_DET_ENABLED:	*LOCO cannot be stopped if oscillation stop detection is enabled*
CGC_ERR_ILL_PARAM:	*One of the parameters is invalid*

Properties

Prototyped in file “r_cgc_rx210_if.h”

Description

This is a non blocking function that turns off the specified clock source. The stabilization state for the HOCO, Main OCO and PLL are checked by reading the state of the software overflow flags. Any clock source that is used as the current system clock cannot be turned off. The main OCO cannot be turned off if the PLL is the current system clock. Attempting to stop the clock under either of these two conditions will return a CGC_ERR_CLOCK_ACTIVE error.

Reentrant

Function is re-entrant.

Example

```
cgc_err_t err;
err = R_CGC_ClockStop(CGC_HOCO);
```

Special Notes:

When the PLL and HOCO are stopped, the power supply to those clocks are stopped as well.
There is a finite stabilization time after a clock is turned ON. Only after this time has elapsed can the clock be turned off. This time is normally duration of about 5 clock cycles. After the clock is stopped it must not be restarted until this duration has elapsed
The LOCO cannot be stopped if the Oscillation Stop Detection function is enabled. Attempting to top the LOCO by calling this function with the Oscillation Stop Detection function enabled will return CGC_ERR_OSC_DET_ENABLED.

3.5 R_CGC_SystemClockSet()

This function sets the specified clock source as the system clock and configures the internal clock dividers.

Format

```
cgc_err_t   R_CGC_SystemClockSet(cgc_clocks_t clock_source
                                 cgc_system_clock_config_t* p_clock_config)
```

Parameters

clock_source

The clock source to be used as the system clock. This is of type cgc_clocks_t which was expanded in R_CGC_ClockStart().

p_clock_config

This is a pointer to a structure of type cgc_system_clock_config_t (see below) which specifies the dividers to be used for the internal clocks ICLK, PCLKB, PCLKD, BCLK and FCLK.

```
typedef struct _cgc_system_clock_config
{
    cgc_system_clock_dividers_t pclkd;
    cgc_system_clock_dividers_t pclkb;
    cgc_system_clock_dividers_t t bclk;
    cgc_system_clock_dividers_t t iclk;
    cgc_system_clock_dividers_t fclk;
}cgc_system_clock_config_t;
```

The dividers to be used for each internal clock are defined in the enum cgc_system_clock_dividers_t (see below).

```
typedef enum _cgc_system_clock_dividers{
      CGC_SYS_CLOCK_DIV_1  = 0x00,
      CGC_SYS_CLOCK_DIV_2  = 0x01,
      CGC_SYS_CLOCK_DIV_4  = 0x02,
      CGC_SYS_CLOCK_DIV_8  = 0x03,
      CGC_SYS_CLOCK_DIV_16 = 0x04,
      CGC_SYS_CLOCK_DIV_32 = 0x05,
      CGC_SYS_CLOCK_DIV_64 = 0x06
}cgc_system_clock_dividers_t
```

Return Values

CGC_SUCCESS:	*Successful; system clock and dividers successfully configured.*
CGC_ERR_NOT_STABILIZED:	*The clock source is not stabilized after being turned on*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*
CGC_ERR_CLOCK_INACTIVE:	*Illegal attempt to set an inactive clock as the clock source*
CGC_ERR_ILL_PARAM:	*One of the parameters is invalid, ICLK is not the fastest clock*
CGC_ERR_NULL_PTR:	*pClock_config pointer is NULL*

Properties

Prototyped in file “r_cgc_rx210_if.h”

Description

This function sets the specified clock as the system clock and configures the dividers for the internal clocks based on the passed parameters. The argument is checked to make sure that ICLK will be the highest clock frequency.

Reentrant

Function is re-entrant

Example

```
cgc_err_t err;
cgc_system_clock_config_t clock;

clock.pclkb_div = CGC_SYS_CLOCK_DIV_4;
clock.pclkd_div = CGC_SYS_CLOCK_DIV_4;
clock.bclk_div  = CGC_SYS_CLOCK_DIV_4;
clock.iclk_div  = CGC_SYS_CLOCK_DIV_1;
clock.fclk_div  = CGC_SYS_CLOCK_DIV_2;

err = R_CGC_SystemClockSet(CGC_HOCO, &clock);
```

Special Notes:

This function does not check to see if setting a clock divider violates the limits of the current operating mode. For example if the current operating mode is Low Speed 1 mode which only allows for a maximum 8 MHz frequency for ICLK, PCLKB and D, and FCLK, using this function to set those clocks above 8 MHz should not be done without first modifying the operating mode to High Speed mode. The API to modify the operating mode is provided in the R_LPC_RX module.
This function also does not check to see if the BCLK divider exceeds the maximum output on the BCLK pin.
For proper frequency limits depending on the current operating mode and Vcc, refer to table 11.4 (Relationship between the Operating Power Control Modes and the Operating Frequency and Voltage Ranges) in the RX210 hardware manual.
The function also checks for invalid clock division ratios and returns an error if an illegal ratio is selected. If the selected clock source is valid and active, but not stabilized, then the function will return CGC_ERR_NOT_STABILIZED.

3.6 R_CGC_SystemClockRead()

This function returns the current system clock source and the value of the internal clock dividers.

Format

cgc_err_t R_CGC_SystemClockRead(cgc_clocks_t* clock_source
cgc_system_clock_config_t* p_clock_config)

Parameters

clock_source

The clock source that is currently used as the system clock. This is a pointer of type cgc_clocks_t which was expanded in R_CGC_ClockStart().

p_clock_config

This is a pointer to a structure of type cgc_system_clock_config_t which this function populates with the current state of the dividers used for the internal clocks ICLK, PCLKB, PCLKD,BCLK and FCLK.

Return Values

CGC_SUCCESS:	*Successful; current system clock and dividers successfully read.*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*

Properties

Prototyped in file “r_cgc_rx210_if.h”

Description

This function returns the current system clock and the dividers for the internal clocks.

Reentrant

Function is not re-entrant.

Example

```
cgc_err_t err;
cgc_system_clock_config_t clock;
cgc_clocks_t clock_source;


err = R_CGC_SystemClockRead(&clock_source, &clock);
```

Special Notes:

3.7 R_CGC_ClockHzGet()

This function returns the frequency of the selected internal clock in Hz.

Format

uint32_t R_CGC_ClockHzGet(cgc_system_clocks_t clock)

Parameters

clock

Specify the internal clock source. This argument is of type cgc_system_clocks_t which is listed below:

```
typedef enum _cgc_system_clocks{
      CGC_PCLKD,
      CGC_PCLKB,
      CGC_BCLK,
      CGC_ICLK,
      CGC_FCLK
}cgc_system_clocks_t;
```

Return Values

Uint32_t:	*clock frequency in Hz*
CGC_ERR_ILL_PARAM:	*One of the parameters specifies an invalid clock*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*

Properties

Prototyped in file "r_cgc_rx210_if.h"

Description

Returns the clock frequency of the specified internal clock

Reentrant

Function is re-entrant.

Example

```
uint32_t pclkb_freq;
pclkb_freq = R_CGC_ClockHzGet(CGC_PCLKB);
```

Special Notes:

This function calculates the clock frequency of the requested clock at runtime by reading the clock registers so that even if any automatic system clock modifications made without the knowledge of the CGC module (by an oscillation stop detect event for example) will be reflected in the return value.

3.8 R_CGC_ClockCheck()

This function checks to see if a particular clock source is stabilized and returns an appropriate status.

Format

cgc_err_t   R_CGC_ClockCheck(cgc_clocks_t clock_source)

Parameters

Clock_source
Specify clock source to check stabilization status.

Return Values

CGC_ERR_STABILIZED:	*The clock source has stabilized*
CGC_ERR_NOT_STABILIZED:	*The clock source has not stabilized after being turned on*
CGC_ERR_CLOCK_INACTIVE:	*The clock source is not turned on*
CGC_ERR_ILL_PARAM:	*Stabilization check is not supported for this clock source*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*

Properties

Prototyped in file “r_cgc_rx210_if.h”

Description

This function will check the specified clock for stabilization by looking at the state of the software overflow flag. This function will return CGC_ERR_CLOCK_INACTIVE if the clock is not active and CGC_ERR_ILL_PARAM for illegal arguments.

Reentrant

Function is re-entrant.

Example

```
cgc_err_t err;
err = R_CGC_ClockCheck (CGC_HOCO);
```

Special Notes:

3.9 R_CGC_Control()

This function allows the configuration and control of the RX210 BCLK output.

Format

```
cgc_err_t       R_CGC_Control(cgc_cmd_t const  cmd,   void   *p_cfg)
```

Parameters

cmd

Command to run (see enumeration below)

p_cfg

Pointer to sometimes optional configuration structure (see below). FIT_NO_PTR/NULL for enable/disable commands.

```
typedef enum e_cgc_cmd
{
   CGC_CMD_BCLK_CONFIGURE,     // configure BCLK divider
   CGC_CMD_BCLK_ENABLE,        // enable BCLK
   CGC_CMD_BCLK_DISABLE,       // disable BCLK
} cgc_cmd_t;
```

p_cfg points to a pointer to a cgc_bclk_settings_t structure when paired with a CGC_CMD_BCLK_CONFIGURE command.

ENABLE/DISABLE commands do not require a p_cfg paramter and FIT_NO_PTR/NULL may be provided.

The enums and typedefs for the RX210 are shown below:

```
typedef enum _cgc_bclk_dividers{
   CGC_BCLK_DIV_1 = 0x00,
   CGC_BCLK_DIV_2 = 0x01
}cgc_bclk_dividers_t;

typedef struct _cgc_bclk_settings{
   cgc_bclk_dividers_t   divider;
   bool enable;
}cgc_bclk_settings_t;

typedef struct _cgc_config_t{
   cgc_bclk_settings_t bclk;
}cgc_config_t;
```

Return Values

CGC_SUCCESS:	*Request successfully executed*
CGC_ERR_ILL_PARAM:	*Illegal parameter passed*
CGC_ERR_BCLK_EXCEEDED:	*Request results in an BCLK output clock exceeding allowable limit*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*

Properties

Prototyped in file “r_cgc_rx210_if.h”

Description

This function implements the functionality defined for the supplied command and argument(if any). This is either a configuration command which configures the BCLK to use the divider values provided, or a clock enable/disable command. In addition the structure enable parameter if true, will enable the BCLK output subsequent to configuring it. While the external bus frequency (BCLK) is set by the R_CGC_SystemClockSet() function, this function can be used to further divide it by 1 or 2 and set that as the output on the BCLK pin.

Reentrant
Function is re-entrant.

Examples
```
cgc_bclk_settings_t BCLKConfig;
cgc_err_t err;

// Configure but don’t yet enable the BCLK output
BCLKConfig.divider = CGC_BCLK_DIV_1;
BCLKConfig.enable = false;
cgc_request_status = R_CGC_Control(CGC_CMD_BCLK_CONFIGURE, (void *)&BCLKConfig);

// Enable the already configured BCLK.
cgc_request_status = R_CGC_Control(CGC_CMD_BCLK_ENABLE, FIT_NO_PTR);
```

Special Notes:
The max clock that can be output on the BCLK pin is dependent on the current operating mode and Vcc. The function will check to see if the limits are exceeded.

3.10 R_CGC_OscStopDetect()

This function enables or disables the oscillation stop detection function for the Main Clock (External Oscillator) and assigns the interrupt callback.

Format

cgc_err_t R_CGC_OscStopDetect(void (*pcallback)(void* pdata),
bool enable)

Parameters

*void (*pcallback)(void* pdata),*
callback function pointer for oscillation stop detection interrupt. This can be null if the next argument is "*false*".
bool enable
Enables or disables the oscillation stop detection function and interrupt.

Return Values

CGC_SUCCESS:	*Successful; Oscillation stop detection is configured*
CGC_ERR_OSC_STOP_DETECTED:	*Operation cannot be disabled if an oscillation stop event was detected and not cleared.*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*
CGC_ERR_NULL_PTR:	*a null pointer as passed for the call back function when attempting to enable Oscillation Stop Detection*

Properties

Prototyped in file "r_cgc_rx210_if.h"

Description

This function configures the oscillation stop detection function for the external oscillator. If the "*enable*" argument is "*false*", then passing a null for the first argument is accepted. In this case, the function will attempt to disable the oscillation stop detection function and interrupt. If however, the status register indicates that an oscillation stop has occurred, then the function will return with *CGC_ERR_OSC_STOP_DETECTED* without modifying any of the registers. This is because the oscillation stop detection function cannot be disabled if the status register indicates that an oscillation stop event has been detected. In this case, it is necessary to call the R_CGC_OscStopStatusClear() function to clear the register after handling the oscillation stop detection event.
To enable the Oscillation Stop Detection function and interrupt, pass "*true*" to the "*enable*" argument and pass the callback function pointer as well.

Reentrant

Function is re-entrant

Example

```
cgc_err_t err;
err = R_CGC_OscStopDetect(&Osc_int_handler, true);
```

Special Notes:

After disabling the Oscillation Stop Detection function, at least two PCLKB cycles have to pass before re-enabling. This function does not account for the wait period.
If the Main Clock is selected as the system clock when the oscillation stop is detected, the MCU will automatically switch the system clock to the LOCO. If the PLL is selected as the system clock when the oscillation stop is detected, then the clock source is not switched and the PLL free running frequency (approximately 1MHz) is the system clock frequency.

3.11 R_CGC_OscStopStatusClear()

This function clears the status of the Oscillation Stop Detection Unit. It is important to call this function once an oscillation stop detection event occurs.

Format

cgc_err_t R_CGC_OscStopStatusClear(void)

Parameters

None

Return Values

CGC_SUCCESS:	*Successful; Status register cleared*
CGC_ERR_NOT_OPEN:	*R_CGC_Open() has not yet been called.*
CGC_ERR_OSC_STOP_CLOCK_ACTIVE:	*The Oscillation Detect Status flag cannot be cleared if the Main OCO or PLL is set as the system clock.*

Properties

Prototyped in file “r_cgc_rx210_if.h”

Description

This function will clear the Oscillation Stop Detection Status register if an oscillation stop condition was detected. This register is not cleared automatically if the stopped clock is restarted. Since it takes a minimum of 3 ICLK cycles until the status register is cleared, nops are inserted to account for this time. The function will return an error code (CGC_ERR_OSC_STOP_CLOCK_ACTIVE) if the PLL or Main Oscillator is set as the system clock while this function is called. Change the system clock before attempting to clear this bit.

Reentrant

Function is re-entrant.

Example

```
cgc_err_t err;
err = R_CGC_OscStopStatusClear();
```

Special Notes:

3.12 R_CGC_GetVersion()

This function returns the driver version number at runtime.

Format

uint32_t R_CGC_GetVersion(void)

Parameters

None

Return Values

Version number.

Properties

Prototyped in file “r_cgc_rx210_if.h”

Description

Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number.

Reentrant

Yes

Example

```
uint32_t   version;
version = R_CGC_GetVersion();
```

Special Notes:

This function is inlined using the “#pragma inline” directive

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Feb 19.14		Initial release
		•	
		•	

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141