Renesas Synergy™ Platform

CRC HAL Module Guide

Introduction

This module guide will enable you to effectively use a module in your own design. Upon completion of this guide, you will be able to add this module to your own design, configure it correctly for the target application and write code, using the included application project code as a reference and an efficient starting point. References to more detailed API descriptions and suggestions of other application projects that illustrate more advanced uses of the module are included in the document and should be valuable resources for creating more complex designs.

The CRC HAL module is a high-level API used to calculate 8, 16 and 32-bit CRC values on a block of data in memory or a stream of data over a Serial Communication Interface (SCI) channel using various types of industry-standard polynomials. The CRC HAL module is implemented on r_crc and uses the CRC peripheral on the Synergy MCU.

Contents

1. CRC HAL Module Features 2

2. CRC APIs Overview 2

3. CRC HAL Module Operational Overview 3

3.1 CRC HAL Module Important Operational Notes and Limitations 3

3.1.1 CRC HAL Module Operational Notes 3

3.2 CRC HAL Module Limitations 3

4. Including the CRC HAL Module in an Application 3

5. Configuring the CRC HAL Module 4

5.1 CRC HAL Module Clock Configuration 5

5.2 CRC HAL Module Pin Configuration 5

6. Using the CRC HAL Module in an Application 5

7. The CRC HAL Module Application Project 6

8. Customizing the CRC HAL Module for a Target Application 7

9. Running the CRC HAL Module Application Project 7

10. CRC HAL Module Conclusion 8

11. CRC HAL Module Next Steps 8

12. CRC HAL Module Reference Information 8

1. CRC HAL Module Features

- CRC HAL module can calculate CRC on a block of data in memory.
- CRC HAL module can calculate CRC on a stream of data being transmitted or received over a serial communication Interface (SCI) channel (snoop mode).
- CRC HAL module supports the following 8-bit and 16-bit CRC polynomials which operates on 8-bit data in parallel
  — $X^8 + X^2 + X + 1$ (CRC-8)
  — $X^{16} + X^{15} + X^2 + 1$ (CRC-16)
  — $X^{16} + X^{12} + X^5 + 1$ (CRC-CCITT)
- CRC HAL module supports the following 32-bit CRC polynomials which operates on 32-bit data in parallel
  — $X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$ (CRC-32)
  — $X^{32} + X^{28} + X^{27} + X^{26} + X^{25} + X^{23} + X^{22} + X^{20} + X^{19} + X^{18} + X^{14} + X^{13} + X^{11} + X^{10} + X^9 + X^8 + X^6 + 1$ (CRC-32C)
- CRC HAL module can calculate CRC with LSB first or MSB first bit order.

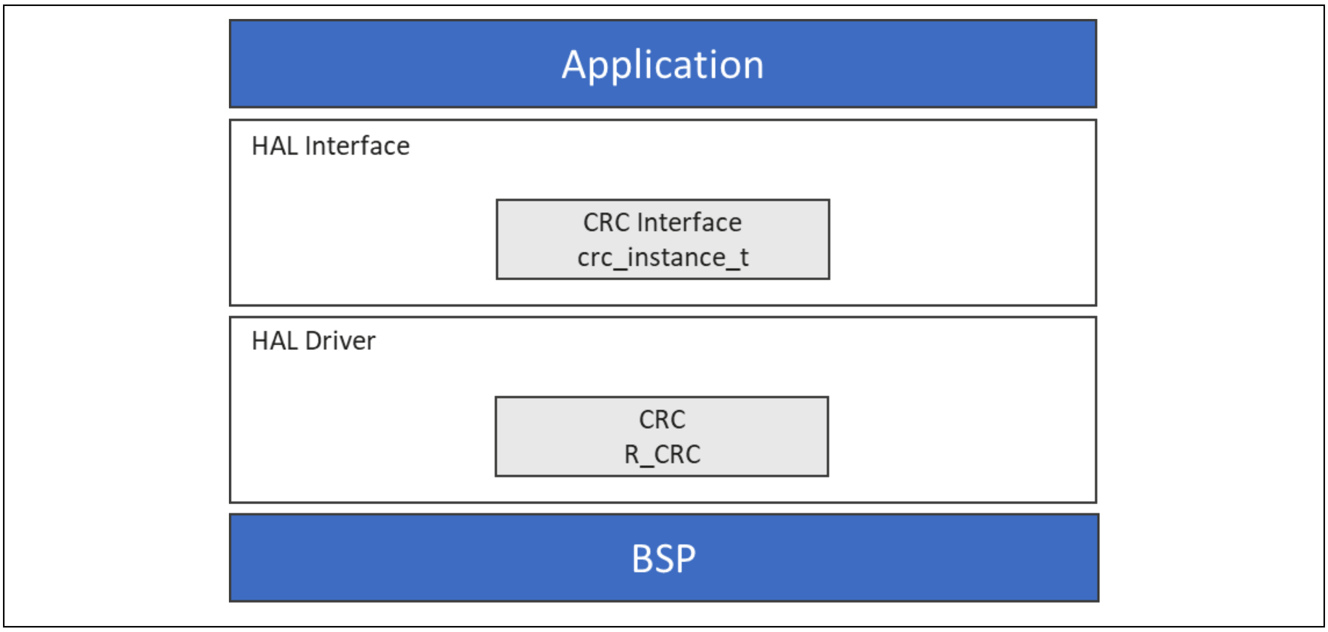

Figure 1. CRC HAL Module Block Diagram

2. CRC APIs Overview

The CRC HAL module defines APIs for opening, closing, enabling and calculating. A complete list of the available APIs, an example API call and a short description of each can be found in the following table. A table of status return values follows the API summary table.

Table 1. CRC HAL Module API Summary

Function Name	Example API Call and Description
`.open`	`g_crc.p_api->open(g_crc.p_ctrl, g_crc.p_cfg);` Open the CRC driver module.
`.close`	`g_crc.p_api->close(g_crc.p_ctrl);` Close the CRC module driver.
`.crcResultGet`	`g_crc.p_api->crcResultGet(g_crc.p_ctrl, &result);` Return the current calculated value.
`.snoopEnable`	`g_crc.p_api->snoopEnable(g_crc.p_ctrl, seed);` Enable snooping.
`.snoopDisable`	`g_crc.p_api->snoopDisable(g_crc.p_ctrl);` Disable snooping.

Function Name	Example API Call and Description
.snoopCfg	`g_crc.p_api->snoopCfg(g_crc.p_ctrl, g_crc.p_cfg);` Configure the snoop channel and direction.
.calculate	`g_crc.p_api->calculate(g_crc.p_ctrl, &input_buffer, num_bytes, crc_seed, &crc_result);` Perform a CRC calculation on a block of data.
.versionGet	`g_crc.p_api->versionGet(&version);` Retrieve the API version with the version pointer.

Note: For details on operation and definitions for the function data structures, typedefs, defines, API data, API structures, and function variables, review the SSP User's Manual API References for the associated module.

Table 2. Status Return Values

Name	Description
SSP_SUCCESS	Configuration was successful.
SSP_ERR_ASSERTION	Assertion error.
SSP_ERR_INVALID_ARGUMENT	Invalid argument error.
SSP_ERR_NOT_OPEN	The driver is not opened.
SSP_ERR_IN_USE	If driver is already open.

Note: Lower-level drivers may return common error codes. Refer to the *SSP User's Manual* API References for the associated module for a definition of all relevant status return values.

3. CRC HAL Module Operational Overview

When the CRC HAL module is used to calculate the CRC value for a block of data in memory, the crc_api_t::calculate API can be used which takes the input buffer pointer, length and the CRC seed value as input and outputs the calculated CRC value.

When the CRC HAL module is used to calculate CRC on a stream of data being transmitted or received over a serial communication Interface (SCI) channel (snoop mode), the module should be configured to be in snoop mode by calling the crc_api_t::snoopCfg followed by the crc_api_t::snoopEnable APIs. After the requested number of data is transmitted or received on the SCI channel, the calculated CRC value can be polled from the module using the crc_api_t::crcResultGet API.

3.1 CRC HAL Module Important Operational Notes and Limitations

3.1.1 CRC HAL Module Operational Notes

- The CRC block does not use any interrupts.
- There is no clock configuration for the CRC module.
- There are no callbacks for the CRC module.
- When using 32-bit CRC polynomials for calculating CRC values of data block in memory, the data block is interpreted using little-endian byte order.

3.2 CRC HAL Module Limitations

Refer to the most recent SSP Release Notes for any additional operational limitations for this module.

4. Including the CRC HAL Module in an Application

This section describes how to include the CRC HAL module in an application using the SSP configurator.

Note: This section assumes you are familiar with creating a project, adding threads, adding a stack to a thread and configuring a block within the stack. If you are unfamiliar with any of these items, refer to the first few chapters of the *SSP User's Manual* to learn how to manage each of these important steps in creating SSP-based applications.

To add the CRC Driver to an application, simply add it to a thread using the stacks selection sequence provided in the following table. (The default name for the CRC Driver is `g_crc0`. This name can be changed in the associated **Properties** window.)

Table 3. CRC Driver Selection Sequence

Resource	ISDE Tab	Stacks Selection Sequence
`r_crc0` CRC Driver on `r_crc`	**Threads**	**New Stack> Driver> Monitoring> CRC Driver on `r_crc`**

When the CRC Driver on `r_crc` is added to the thread stack as shown in the following figure, the configurator automatically adds any needed lower-level modules. Any drivers that need additional configuration information will be box text highlighted in Red. Modules with a Gray band are individual modules that stand alone.

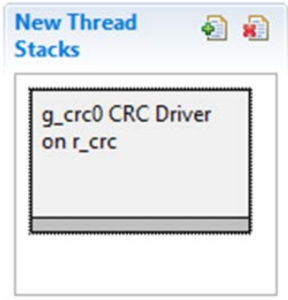

Figure 2. CRC HAL Module Stack

5. Configuring the CRC HAL Module

The CRC HAL module must be configured by the user for the desired operation. The SSP configuration window will automatically identify (by highlighting the block in red) any required configuration selections, such as interrupts or operating modes, which must be configured for lower-level modules for successful operation. Furthermore, only those properties that can be changed without causing conflicts are available for modification. Other properties are locked and are not available for changes and are identified with a lock icon for the locked property in the Properties window in the ISDE. This approach simplifies the configuration process and makes it much less error prone than previous manual approaches to configuration. The available configuration settings and defaults for all the user-accessible properties are given in the Properties tab within the SSP configurator and are shown in the following tables for easy reference.

One of the properties most often identified as requiring a change is the interrupt priority; this configuration setting is available within the Properties window of the associated module. Simply select the indicated module and then view the Properties window; the interrupt settings are often toward the bottom of the properties list, so scroll down until they become available. Note that the interrupt priorities listed in the Properties window in the ISDE will include an indication as to the validity of the setting based on the targeted MCU (CM4 or CM0+). This level of detail is not included in the following configuration properties tables but is easily visible with the ISDE when configuring interrupt-priority levels.

Note: You may want to open your ISDE, create the module and explore the Property settings in parallel with looking over the following configuration table settings. This will help orient you and can be a useful hands-on approach to learning the ins and outs of developing with SSP.

Table 4. Configuration Settings for the CRC HAL Module on r_crc

ISDE Property	Value	Description
Parameter Checking	BSP, Enabled, Disabled Default: BSP	Enable or disable the parameter error checking.
Name	g_crc0	Module name.
CRC Polynomial	CRC-8, CRC-16, CRC-CCITT, CRC-32, CRC-32C Default: CRC-32C	Specify the polynomial to use for calculation.
Bit Order	LSB, MSB Default: MSB	Specify the bit order of the calculation.

Note: The example values and defaults are for a project using the S7G2 Synergy MSU Group. Other MCUs may have different default values and available configuration settings.

In some cases, settings other than the defaults for a module can be desirable. For example, it might be useful to select a different clock source than the default. The configurable properties for the lower-level stack modules are given in the following sections for completeness and as a reference.

5.1 CRC HAL Module Clock Configuration

The CRC HAL module is clocked via the Peripheral Clock A (PCLKA.)

The CRC HAL module does not support any APIs for setting the frequency at which it operates.

5.2 CRC HAL Module Pin Configuration

The CRC HAL module doesn't have any configurable pins.

6. Using the CRC HAL Module in an Application

The typical steps in using the CRC HAL module for computing CRC of data block in memory are:

1. Initialize the CRC HAL module using the crc_api_t::open API
2. Compute the CRC HAL module using the crc_api_t::calculate API
3. Close the CRC HAL module using the crc_api_t::close API

The following diagram shows these common steps in a typical operational flow:

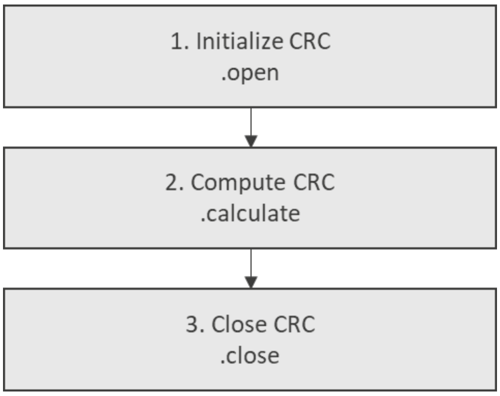

Figure 3. Flow Diagram of a Typical CRC HAL Module Application

The typical steps in using the CRC HAL module for computing CRC in snoop mode are:

1. Initialize the CRC HAL module using the open API.
2. Configure CRC module to snoop an SCI channel (and its direction) using the crc_api_t::snoopCfg API.
3. Enable snooping of the SCI channel using the crc_api_t::snoopEnable API.
4. Once the required number of bytes are transmitted or received on the SCI channel, get the calculated CRC value using the crc_api_t::crcResultGet API.
5. Disable the snooping operation using the crc_api_t::snoopDisable API.
6. Close the CRC HAL module using the crc_api_t::close API.

The following diagram shows these common steps in a typical operational flow:

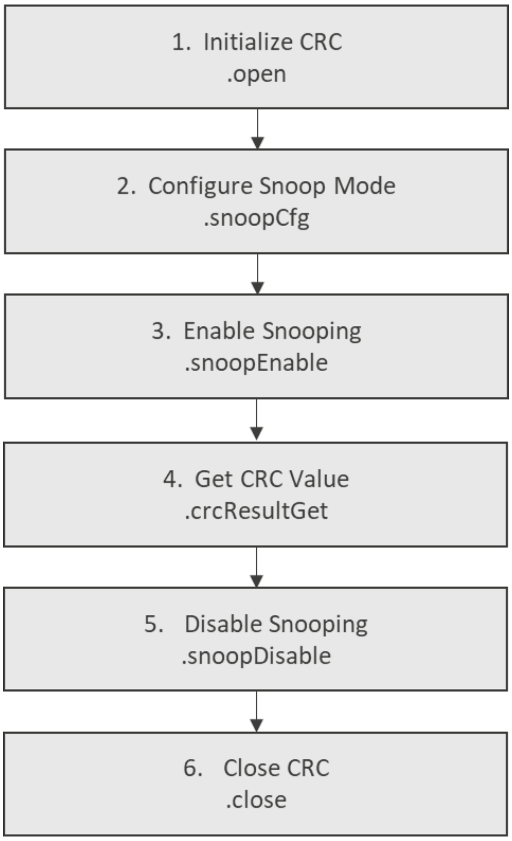

Figure 4. Flow Diagram of a Typical CRC HAL Module Snoop Mode Application

7. The CRC HAL Module Application Project

The application project associated with this module guide demonstrates these steps in a full design. The project can be found using the link provided in the References section at the end of this document. You may want to import and open the application project within the ISDE and view the configuration settings for the CRC HAL module. You can also read over the code (in `crc_hal.c`) which is used to illustrate the CRC HAL module APIs in a complete design.

The application project demonstrates the typical use of the CRC HAL module APIs. The application project main-thread entry initializes the CRC HAL module and periodically computes the CRC. The CRC is initially computed for a data buffer that is filled with generated bytes, then the data is copied to the so-called receive buffer, (the application simulates data transmission) and once again the CRC is computed. Both CRCs are compared, and based on the result, the green (CRCs are equal) or red (CRCs are different) LED is lit. The simulation of data transmission can introduce errors which result in lighting the red LED. The following table identifies the target versions for the associated software and hardware used by the application project.

Table 5. Software and Hardware Resources Used by the Application Project

Resource	Revision	Description
e^2 studio	5.3.1 or later	Integrated Solution Development Environment (ISDE)
SSP	1.2.0 or later	Synergy Software Platform
IAR EW for Synergy	7.71.2 or later	IAR Embedded Workbench® for Renesas Synergy™
SSC	5.3.1 or later	Synergy Standalone Configurator
SK-S7G2	v3.0 to v3.1	Starter Kit

The following flow diagram shows a simple application project:

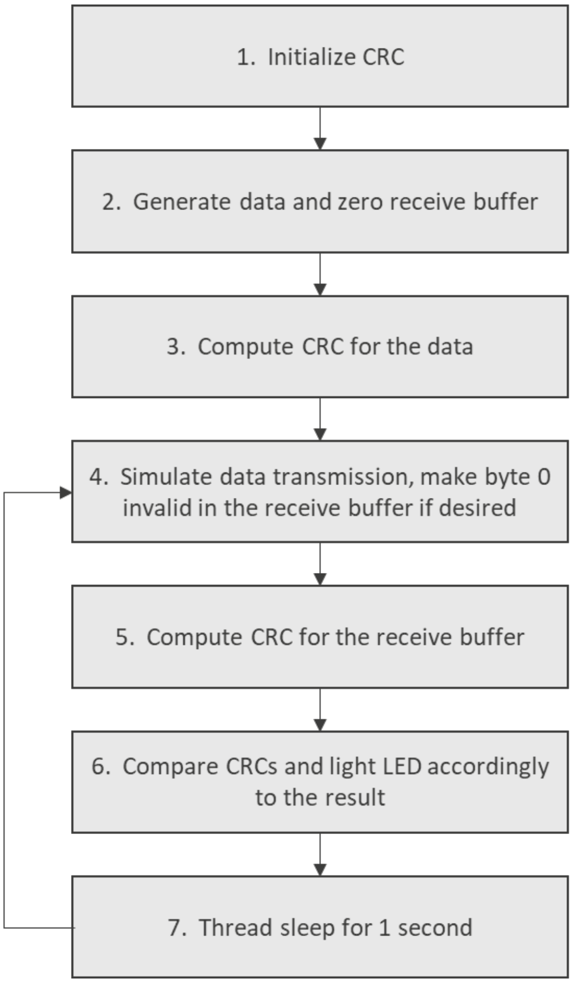

Figure 5. CRC Application Project Flow Diagram

The complete application project can be found using the link provided in the References section at the end of this document. The `crc_hal.c` file is located in the project once it has been imported into the ISDE. You can open this file within the ISDE and follow along with the description provided to help identify key uses of APIs.

In the application project, main-thread entry, the CRC HAL module is initialized using the open API right after the data array and receive buffer are set up. The data array contains growing numbers, as bytes, starting with 0; the receive buffer is initialized with zeros. When the data is ready, and the CRC HAL module is initialized, a CRC is computed for the data. Next, the application enters a loop which simulates data transmission and fills the receive buffer, computes the CRC for the buffer, and lights the green (CRCs are equal) or red (CRCs are different) LED, according to the CRC comparison result. The application is initially configured to introduce data-transmission errors. After copying bytes from the data array to the receive buffer, the byte at index 0 in the buffer is altered. This results in a different CRC and lights the red LED. This error can be removed by setting the `makeError` variable to false. The CRC will then match, lighting the green LED.

The CRC application project uses the CRC HAL module default property-values; no extra activities must be performed by the user.

8. Customizing the CRC HAL Module for a Target Application

The application project uses the default setting of the CRC-32C Polynomial on MSB data. The developer may wish to change the polynomial value and bit order to suit their target application.

9. Running the CRC HAL Module Application Project

To run the CRC HAL module application project and to see it executed on a target kit, you can simply import it into your ISDE, compile and run debug.

To implement the CRC HAL module application in a new project, follow the steps for defining, configuring, auto-generating files, adding code, compiling and debugging on the target kit. Following these steps is a hands-on approach that can help make the development process with SSP more practical, while just reading over this guide will tend to be more theoretical.

Note: The following steps are sufficient for someone experienced with the basic flow through the Renesas Synergy™ Platform development process. If these steps are unfamiliar, refer to the first few chapters in the *SSP User's Manual* listed in the References section at the end of this document.

To create and run the CRC HAL module application project, simply follow these steps:

1. Create a new project for the S7G2 SK Synergy MCU Group called `CRC_HAL_MG_AP`.
2. Select the **BSP** tab and select **RTOS being used** as **No RTOS**.
3. Select the **Threads** tab.
4. Add the CRC HAL module to the HAL thread.
5. Click on the **Generate Project Content** button.
6. Add the code from the supplied project file `crc_hal.c` or copy over the generated `crc_hal.c` file.
7. Connect to the host PC via a micro USB cable to J19 on the SK-S7G2 Kit.
8. Start to debug the application
9. Watch the green or red led being lit on the board.

10. CRC HAL Module Conclusion

This module guide has provided all the background information needed to select, add, configure and use the module in an example project. Many of these steps were time consuming and error-prone activities in previous generations of embedded systems. The Renesas Synergy Platform makes these steps much less time consuming and removes the common errors, like conflicting configuration settings or the incorrect selection of lower-level drivers. The use of high-level APIs (as demonstrated in the application project) illustrates additional development time savings by allowing work to begin at a high level and avoiding the time required in older development environments to use or, in some cases, create, lower-level drivers.

11. CRC HAL Module Next Steps

After you have mastered a simple CRC HAL module project, you may want to review a more complex example. Other application projects and application notes that demonstrate CRC HAL-use can be found as described in the References section at the end of this document.

12. CRC HAL Module Reference Information

SSP User Manual: Available from the Renesas Synergy Software Package website here: https://www.renesas.com/us/en/products/synergy/software/ssp.html

Links to all the most up-to-date r_crc module reference materials and resources are available on the Synergy Knowledge Base here: https://en-support.renesas.com/knowledgeBase/16977474

Website and Support

Visit the following vanity URLs to learn about key elements of the Synergy Platform, download components and related documentation, and get support.

Synergy Software	www.renesas.com/synergy/software
Synergy Software Package	www.renesas.com/synergy/ssp
Software add-ons	www.renesas.com/synergy/addons
Software glossary	www.renesas.com/synergy/softwareglossary
Development tools	www.renesas.com/synergy/tools
Synergy Hardware	www.renesas.com/synergy/hardware
Microcontrollers	www.renesas.com/synergy/mcus
MCU glossary	www.renesas.com/synergy/mcuglossary
Parametric search	www.renesas.com/synergy/parametric
Kits	www.renesas.com/synergy/kits
Synergy Solutions Gallery	www.renesas.com/synergy/solutionsgallery
Partner projects	www.renesas.com/synergy/partnerprojects
Application projects	www.renesas.com/synergy/applicationprojects
Self-service support resources:	
Documentation	www.renesas.com/synergy/docs
Knowledgebase	www.renesas.com/synergy/knowledgebase
Forums	www.renesas.com/synergy/forum
Training	www.renesas.com/synergy/training
Videos	www.renesas.com/synergy/videos
Chat and web ticket	www.renesas.com/synergy/resourcelibrary

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jul.31.17	—	Initial Release
1.01	Aug.30.17	7	Update to Hardware and Software Resources Table
1.02	Feb.01.19	7	Update to program operational description

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.