Renesas Synergy™ Platform

Mastering Stack Usage

R11AN0195EU0100
Rev.1.00
Jul 11, 2017

Introduction

This application note explains how to setup and use stacks in IAR Embedded Workbench® for Renesas Synergy™, and how to monitor and analyze stack usage at runtime.

The stack is a fixed block of continuous memory and must be allocated statically by the developer. It contains local data for C/C++ functions and other:

- Local variables which are not stored in registers
- Function parameters which are not stored in registers
- Temporary result of expressions
- The return value of a function (unless it is passed through registers)
- Interrupt contexts
- Processor registers that should be restored before the function returns

A stack can be divided into two parts. The first part contains allocated memory used by functions and the second part contains free memory that can be allocated. The border between them is called the top of stack and is represented by the which is a dedicated processor register in usual. Memory is allocated from the stack by moving stack pointer (SP). The memory allocated on the stack is released when the function returns, so it is impossible to store data which is supposed to live thereafter.

The main advantage of stack is that functions in different parts of the application can share the same memory space to store their data. Unlike a heap, the stack will never become fragmented or suffer from memory leaks.

A proper configuration of the stack is essential to your system stability and reliability. If the stack size is too small, SP might be moved out of the stack area thus an overflow situation occurs. In this case, the executing code could write to the area allocated below the stack (in case the stack grows downward) and lead to a serious runtime failure like overwritten variables, wild pointers, corrupted return address, and others. On the other hand, setting the stack size too large means a waste of RAM resource which could be very limited in MCU-based embedded systems.

Contents

1. Static stack usage analysis

Under the right circumstances, the linker can accurately calculate the maximum stack usage for each call graph root (a function that is not called from any another functions). A stack usage chapter will be added into the linker map file (`.map`), listing the depth of the deepest call chain for each **call graph root**, as well as the sum of deepest call chain depths for each **call graph root** category. The calculation is only accurate if there is enough stack usage information for each function in the application.

Usually, the compiler will generate this information for each function. But in some cases, additional directives must be provided by the developer to inform the compiler about indirect calls (calls using function pointers) or the maximum number of iteration for recursive functions. This can be achieved by either using `#pragma directives` in the source code or specifying a separate stack usage control file in the project options dialog.

2. Enable stack usage analysis

In the **Advanced** tab of **Linker** options, check **Enable stack usage analysis**:

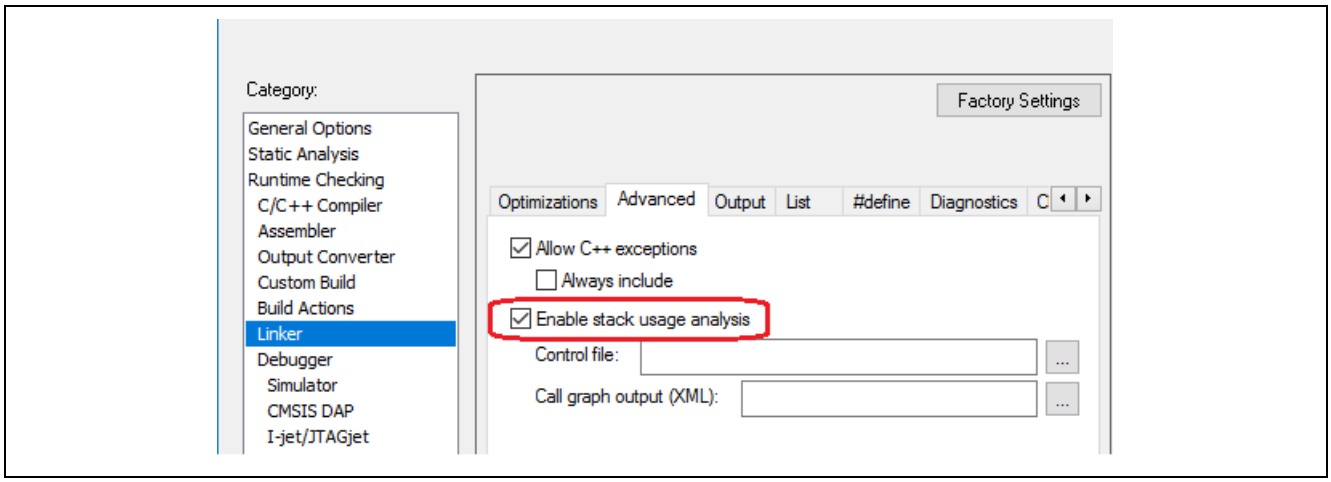

Figure 1 Enable stack usage analysis

Generate a **linker map** file, since it contains the result of stack usage analysis. It can be **enabled** in the **Linker** option under the **Category** window in the **List** tab:

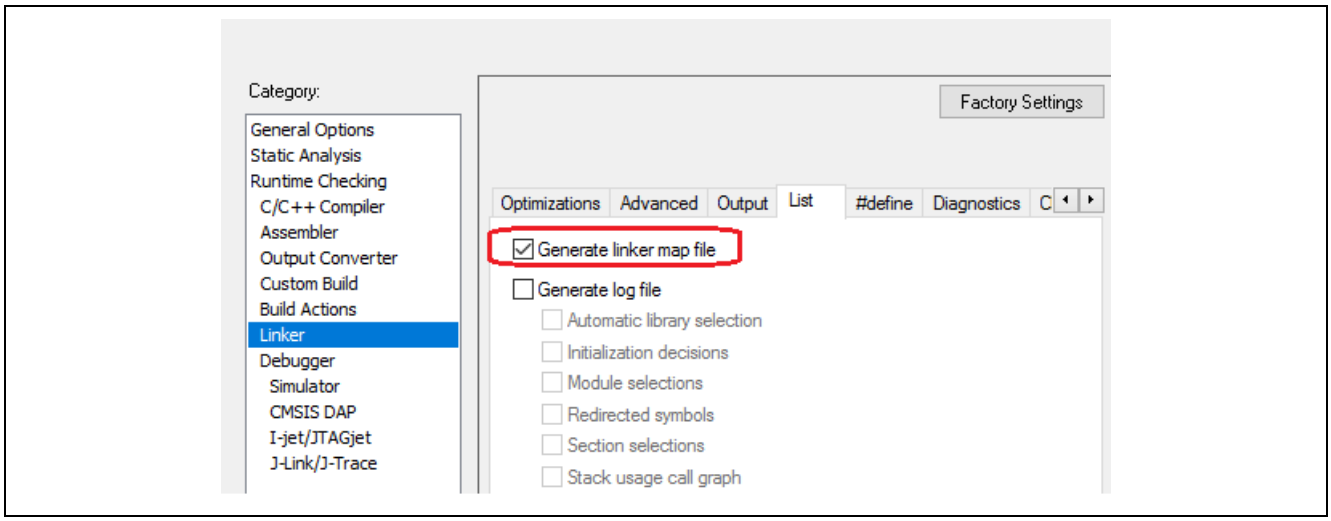

Figure 2 Enable Generate linker map file

For simple applications, the result of stack usage analysis is easy to understand. The program entry and interrupt handlers would be regarded as **call graph root** since they are not called by any other functions. In the example below from a simple **Blinky** project generated by the Synergy configurator for S7G2-SK Synergy MCU Group, the maximum stack depth is 72 bytes for the program entry **call graph root** category (`Reset_Handler`) and totally 64 bytes for the

Uncalled function call graph root category (`NMI_Handler, R_CGC_BusClockOutCfgand,…`). See the example below.

```
*******************************************************************************
*** STACK USAGE
***

  Call Graph Root Category  Max Use  Total Use
  ------------------------  -------  ---------
  Program entry                  72         72
  Uncalled function              64      1 340


Program entry
  "Reset_Handler": 0x00003fcd

  Maximum call chain                              *?* 72  bytes
    (** call graph contains indirect calls (example: "SystemInit") **)

    "Reset_Handler"                                    8
    "main"                                             8
    "hal_entry"                                       32
    "R_BSP_SoftwareDelay"                             16
    "bsp_cpu_clock_get"                                8

Uncalled function
  "NMI_Handler": 0x00002213

  Maximum call chain                              *?* 16  bytes
    (** call graph contains indirect calls (example: "bsp_group_irq_call") **)

    "NMI_Handler"                                      8
    "bsp_group_irq_call"                               8

Uncalled function
  "R_CGC_BusClockOutCfg": 0x00001129

  Maximum call chain                                 32  bytes

    "R_CGC_BusClockOutCfg"                             8
    "HW_CGC_BusClockOutCfg"                           16
    "HW_CGC_HardwareLock"                              8
    "R_BSP_RegisterProtectEnable"                      0
```

Figure 3 Call Graph Root Category example

3. Specify indirect calls

An indirect call means calling a function through a function pointer. Since the **callee** function is unknown at building time, the linker cannot automatically retrieve the stack usage information for indirect calls. A warning message will be generated by the linker, for example:

```
Warning[Ls016]: [stack usage analysis] the program contains at least one
indirect call. Example: from "SystemInit". A complete list of such functions is
in the map file.
```

At the end of *Stack Usage* chapter of the linker map file, there is the description:

```
The following functions perform unknown indirect calls:

  "R_CGC_ClocksCfg": 0x000005bb
```

```
  "R_CGC_Init": 0x00000529

  "R_ELC_Init": 0x00003927

  "R_IOPORT_Init": 0x000026d9

  "SystemInit": 0x00003b55

  "bsp_clock_init": 0x00001fd1

  "bsp_cpu_clock_get": 0x0000208d

  "bsp_group_irq_call": 0x000021c9
```

To solve this problem, the developer should use the *#pragma calls* directive to list the functions that could be indirectly called by a statement. This directive should be inserted just before the indirect call statement and specify the list of all possible callee functions. For example, the following code specifies that the function `UartRxHandler(), UartTxHandler()`and `UartFaultHandler()` could be indirectly called through the function pointer `isr():`

```
void BSP_IntHandler (int int_id) {
void (*isr)(void);
……
    if (int_id < BSP_INT_SRC_NBR) {
        isr = BSP_IntVectTbl[int_id];
#pragma calls=UartRxHandler,UartTxHandler,UartFaultHandler
        isr();
}
……
}
```

4. Provide call graph root information

In a multi-task environment using RTOS, the root function of each task is also a call graph root. Sometimes they are not able to be automatically identified by the linker. The linker will generate warning messages instead because it seems that they are not called by any other functions:

```
Warning[Lo008]: [stack usage analysis] at least one function appears
to be uncalled. Example: "blinky_thread_func" in blinky_thread.c [1].
A complete list of uncalled functions is in the map file.
```

In the *Stack Usage* chapter of the linker map file, there is the description:

```
Uncalled function
  "blinky_thread_func" in blinky_thread.o [1]: 0x00004641
……
Uncalled function
  "NMI_Handler": 0x00002243
```

To solve this problem, use the `#pragma call_graph_root` directive to identify the function as a call graph root. For example:

```
#pragma call_graph_root="task"                          // task category
static void blinky_thread_func (ULONG thread_input) {
{ …… }
```

```
#pragma call_graph_root="interrupt"         // interrupt category
void NMI_Handler (void)
{ …… }
```

It is possible to use any string other than **task** or **interrupt** to be the name of call graph root categories. The compiler will automatically assign a call graph root category to **interrupt** and **task** functions.

5. Use a stack usage control file

The **#pragma directives** must be inserted into source files, that are not allowed in some cases. Without changing the source code, a separate stack usage control file can alternatively provide the same stack usage information to the compiler and linker.

The stack usage control file is a text file which has `*.suc` as its suffix. The path of the stack usage control file can be set in the **Advanced** tab of the **Linker** options:

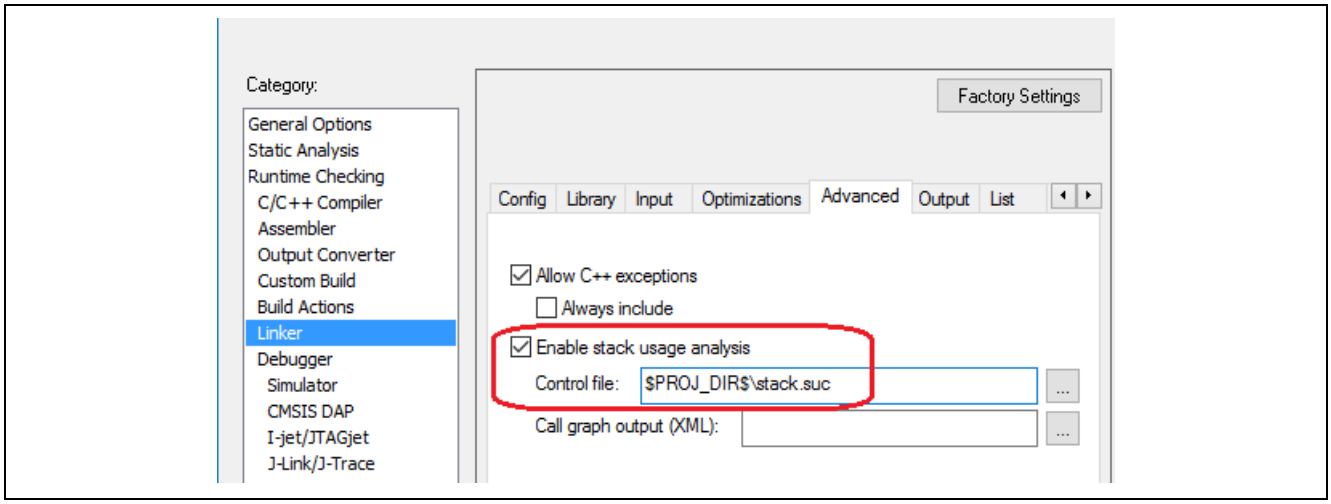

Figure 4 Stack usage control file

There are several types of directive that can be used in the stack usage control file, such as **function, exclude, possible calls, call graph root, max recursion depth, no calls from** and others. The **possible calls** directive has the similar effect as **#pragma calls**, which specifies the possible **callee** functions of an indirect call. The **call graph root** directive has the similar effect as **#pragma call_graph_root,** that identifies a group of none-called functions as **call graph root**.

Replacing the **#pragma directives** used in previous examples, is the content of a stack usage control file is shown below.

```
call graph root [task] : blinky_thread_func [blinky_thread.o];
call graph root [interrupt] : NMI_Handler;
```

6. Specify the iteration of recursive functions

A recursive function calls itself either directly or indirectly. Each invocation can store its own data on the stack. If it is not properly designed to return after several iterations, there is a high risk to cause stack overflow.

Since the actual number of iteration is unknown at building time, the linker cannot automatically retrieve the stack usage information for recursive functions. A warning message will be generated by the linker, for example:

```
Warning[Lo010]: [stack usage analysis] the program contains at least
one instance of recursion for which stack usage analysis has not been
able to calculate a maximum stack depth. One function involved is
"_GLCD_SendCmd. A complete list of all recursion nests is in the map file.
```

In the *Stack Usage* chapter of the linker map file, there is the description:

```
The following functions make up recursion nest 0, which has no
maximum recursion depth specified:
  "_GLCD_SendCmd": 0xffff8aac
```

To solve this problem, use the **max recursion depth** directive in a **stack usage control file** to specify the maximum recursion depth for each recursive function. **Stack usage analysis** will base its result on the maximum number of iteration multiplied by the stack usage of the deepest cycle in the recursion nest. The example below sets the maximum recursion depth to 3 for the function `GLCD_SendCmd()`:

```
max recursion depth _GLCD_SendCmd : 3;
```

7. Redefining the stack size in the Synergy configurator

The estimated stack from the static stack usage analysis after the fine adjustments can be used in the SSP settings. The stack size is defined under the BSP settings from the Synergy configurator. Always regenerate the project after changing the stack size settings.

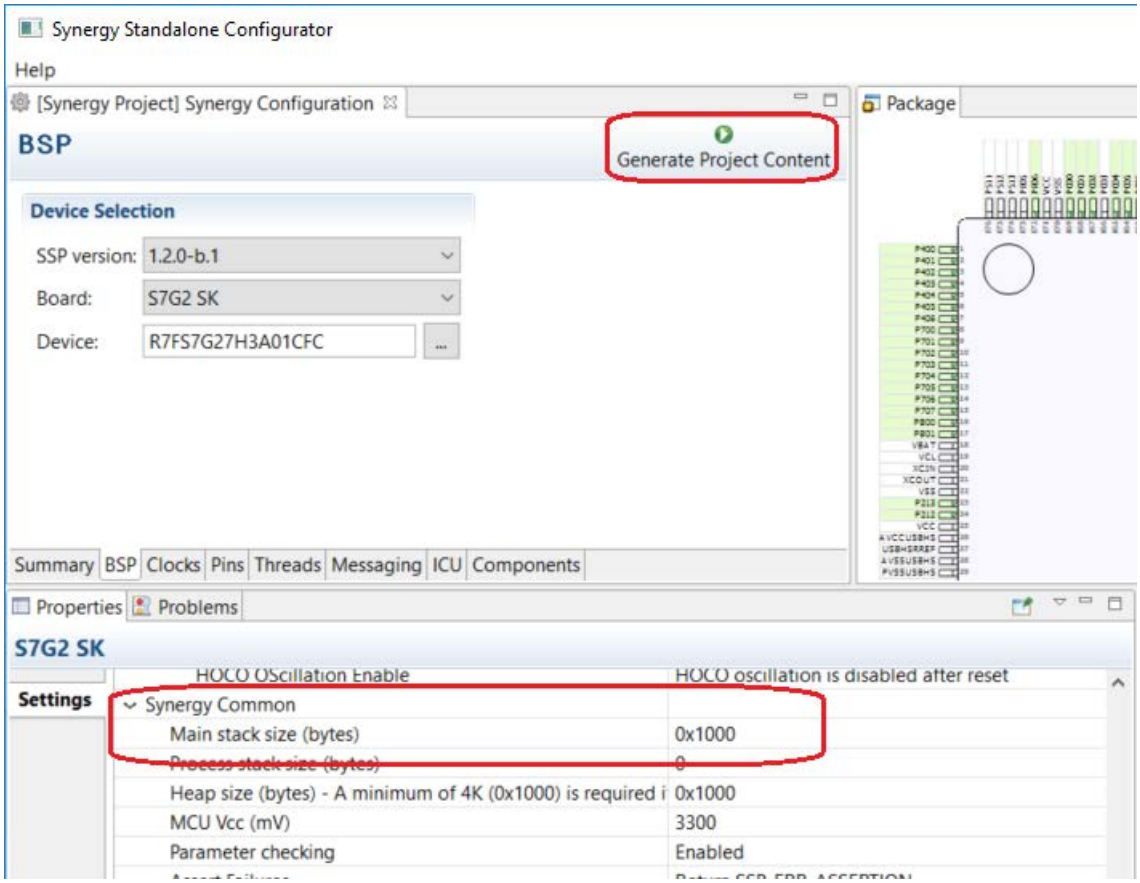

Figure 5 Redefining the stack size

8. Runtime stack usage monitoring

Static stack usage analysis calculates the theoretical maximum stack requirement at building time. The actual stack consumption can vary during execution. IAR EW for Synergy provides another approach to track the stack usage at runtime, implemented by the C-SPY debugger. C-SPY can fill the entire stack area with a magic data pattern. For example, 0xCD before the application starts to execute and after the program has been running for a while, preferably under certain test conditions, the stack memory can be checked upwards from its end until finding a value that is different from 0xCD. It is assumed to be the utmost location where SP has ever reached. The part of stack memory that still contains 0xCD has never been overwritten, so that it is safe to reduce the stack size by that amount. It could be wise to reserve a little extra space just in case your test didn't last long enough or didn't accurately reflect all possible runtime scenarios.

To enable the graphical stack analysis during debugging with the SSP package in the IAR EW for Synergy you must enable the following extra options for the debugger in **Project->Options->Debugger->Extra Options->** Use command line options: `--proc_stack_main=g_main_stack,g_main_stack+sizeof(g_main_stack)`

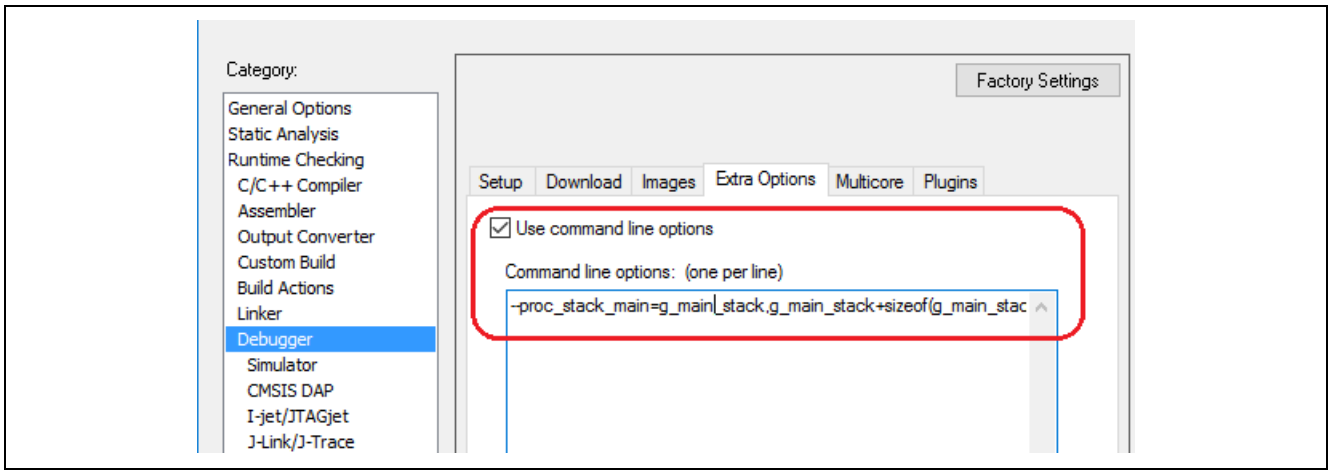

Figure 6 Runtime stack usage

Enable graphical stack display and stack usage tracking in the **Stack** category of the **IDE Options** dialog to enable the runtime stack usage tracking:

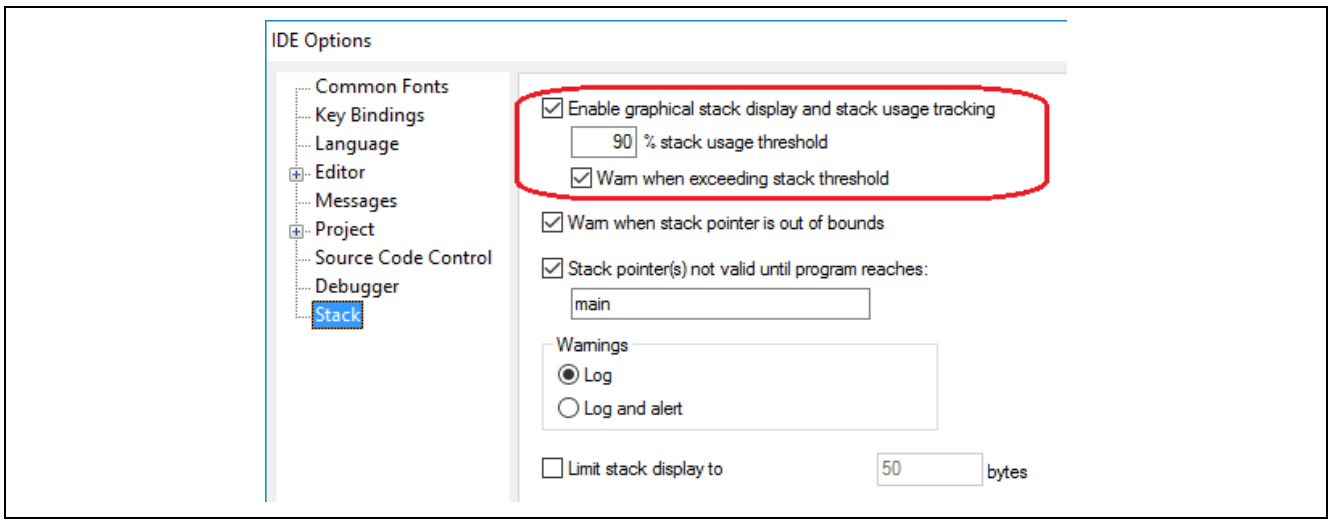

Figure 7 Runtime stack usage tracking

The **Stack** window is available from the **View** menu. Whenever the execution stops, C-SPY can update the graphical representation of stack usage in the window below.

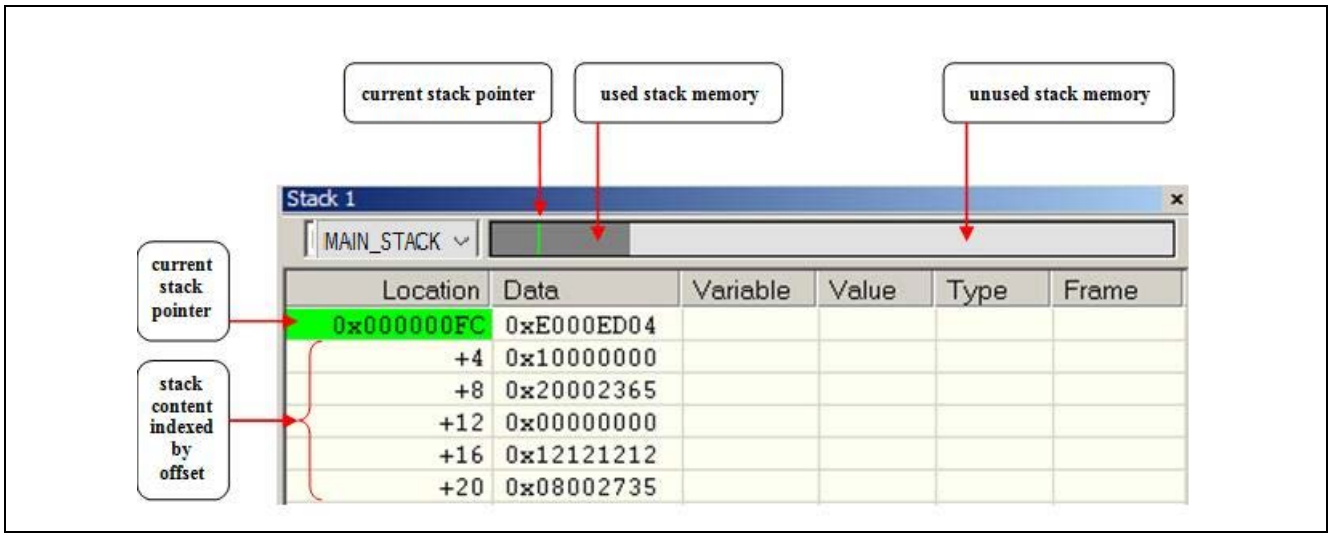

Figure 8 Graphical representation of stack usage

The left end of the graphical stack bar represents the bottom of stack, the position of SP when the stack is empty. The right end represents the end of memory space reserved for the stack. The dark grey area represents the used stack memory and the light grey area represents the unused stack memory. The graphical stack bar turns red when the stack usage exceeds a threshold which you can set in the **IDE Options** dialog.

Note: This functionality cannot detect a stack overflow when it happens, but can only detect the signs it leaves behind. Although this is a reliable way to track the stack usage, there is no guarantee that a stack overflow can be detected. For example, a stack can incorrectly grow outside its bound and even modify memory outside the stack area, without modifying any bytes near the border. For monitoring the stack usage, it is recommended the use of a data breakpoint. The data breakpoint can monitor any read or write access to the last bytes of stack and halt the application for further analysis.

Reference Information

SSP User Manual: Available in html format in the SSP distribution package and as a pdf from the Synergy Gallery.

To find the most up-to-date reference materials and their locations, visit the Synergy Knowledge Base and do a search for the module name and include **module guide references** in the search.

For example, if you are looking for the References for the `r_doc` module, visit https://en-us.knowledgebase.renesas.com/English_Content/Renesas_Synergy%E2%84%A2_Platform/Renesas_Synergy_Knowledge_Base and enter **r_doc module guide references** in the search bar. The search will bring up a list of results, and the top one will be the References Page for that Module Guide. The following URL will take you directly to the search results for the example.

https://en-us.knowledgebase.renesas.com/Special:Search?fpid=230&search=r_doc%20module%20guide&path=&limit=55&page=1&q=r_doc%20module%20guide%20references&tags

Website and Support

Support: https://synergygallery.renesas.com/support

Technical Contact Details:

- America: https://renesas.zendesk.com/anonymous_requests/new
- Europe: https://www.renesas.com/en-eu/support/contact.html
- Japan: https://www.renesas.com/ja-jp/support/contact.html

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Jul 11, 2017	-	Initial Release

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.3.0-1 November 2016)

RENESAS

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
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

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 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

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 1207, 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 India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 6.0