To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of 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. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may 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.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “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.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its 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 be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be 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, or if you have any other inquiries.

(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.

R8C Family

How to Compute MR8C/4 User Stack and System Stack

Introduction

Computing the required stack size is a critical step for embedded developers to avoid wasting any additional, unnecessary memory and ensuring stack overflows does not occur. Deciding how much memory to allocate for the stack has been a trial and error process.

This document explains the method of computing the user/task stack and system stack for application written in either C or assembly language.

Target Device

Applicable MCU: R8C Family

Contents

1. Guide in using this Document

This document aims to equip users with the technique of determining the required user/task stack and system stack for MR8C/4.

Table 1 Explanation of Document Topics

Topic	Objective	Pre-requisite
Introduction to MR8C/4	Provides explanation of user and system stack	Knowledge in MR8C/4
Settings for STK Viewer Utility	Explains the settings for STK viewer	Knowledge in STK viewer utility
Computing User Stack Size	Explains methods of computing user stack size to be allocated	Knowledge in MR8C/4
Computing System Stack Size	Explains methods of computing system stack size to be allocated	Knowledge in MR8C/4
Reference Documents	Listing of documents that equip users with knowledge in the pre-requisite requirements	None

2. Introduction to MR8C/4 Stack

The MR8C/4 provides two kinds of stacks: the user stack and the system stack.

User stack refers to the memory block assigned for each single task. This block of memory denotes the worst-case memory consumption of the individual task during run-time. The allocation of the user stack size can be done in the configurator file as shown in Figure 1.

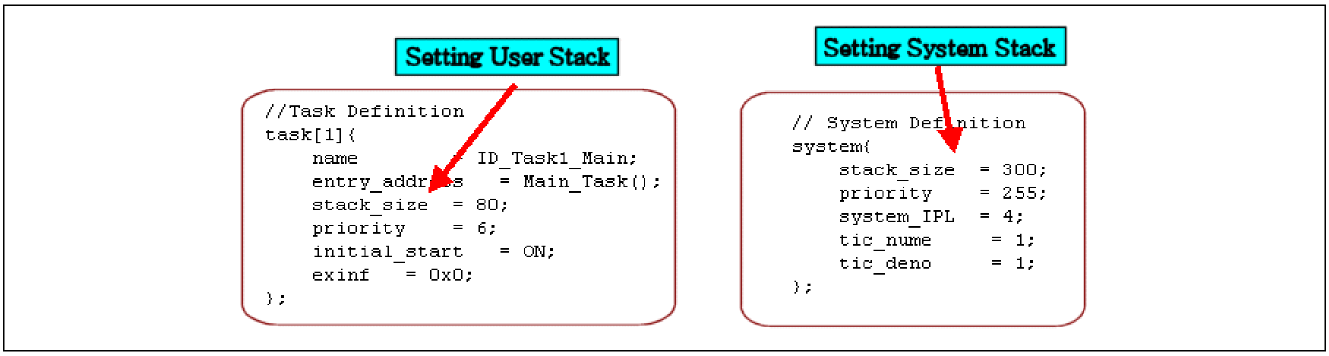


Figure 1 Setting User Stack and System Stack

System stack refers to the memory block assigned for MR8C/4 during execution of the service call. It denotes the worst-case memory consumption of MR8C/4 during execution of the handler. Size allocation of system stack is also done in the configurator file shown in Figure 1.

Caution will need to be exercised when allocating the sizes of the user stack and system stack. Allocating too much memory will result in wasting of memory. Allocating too less will result in stack overflows, which can corrupt other memory areas and typically trigger a program crash.

Both the user stack and system stack reside in the STACK section within the internal RAM segment of the device. Refer to Figure 2 for the stacks layout in the R8C/23 device. Figure 3 provides an example of the user stack and system stack layout using Map Viewer utility.

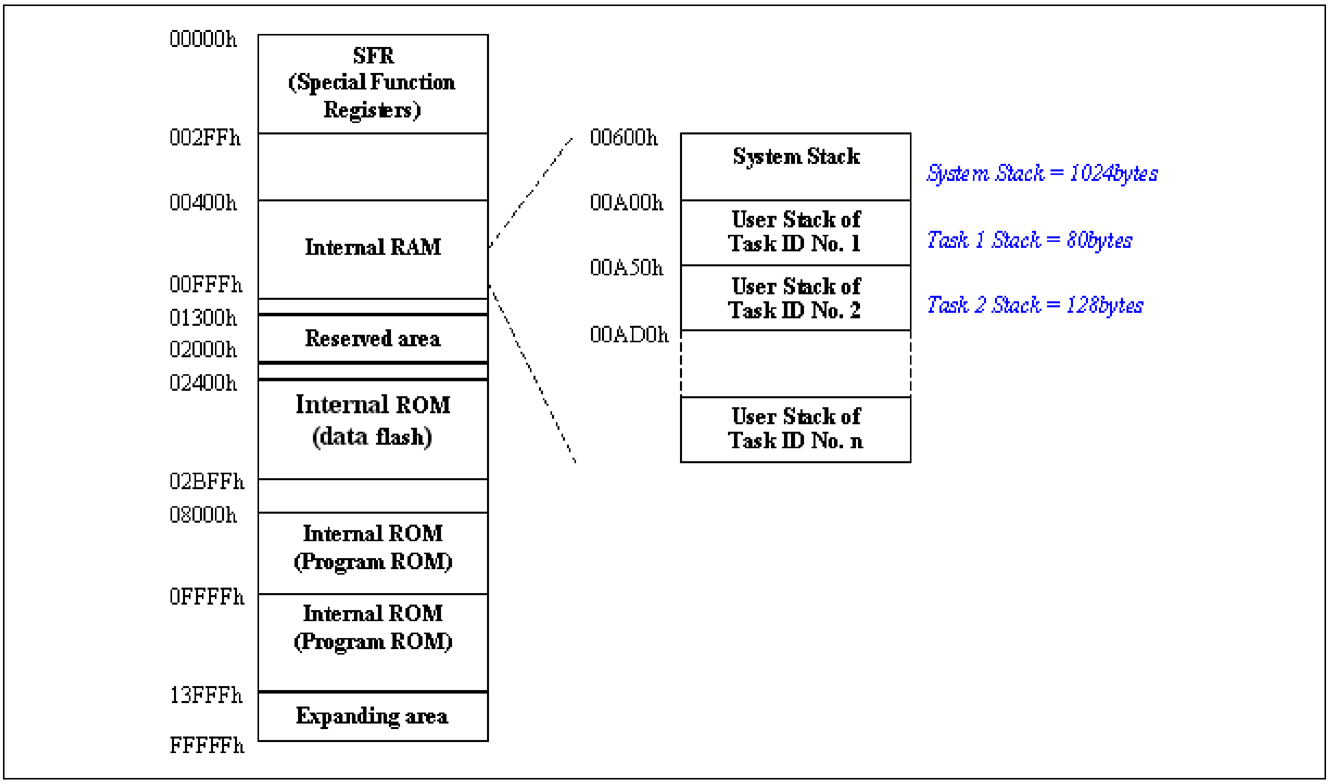


Figure 2 User Stack and System Stack Layout

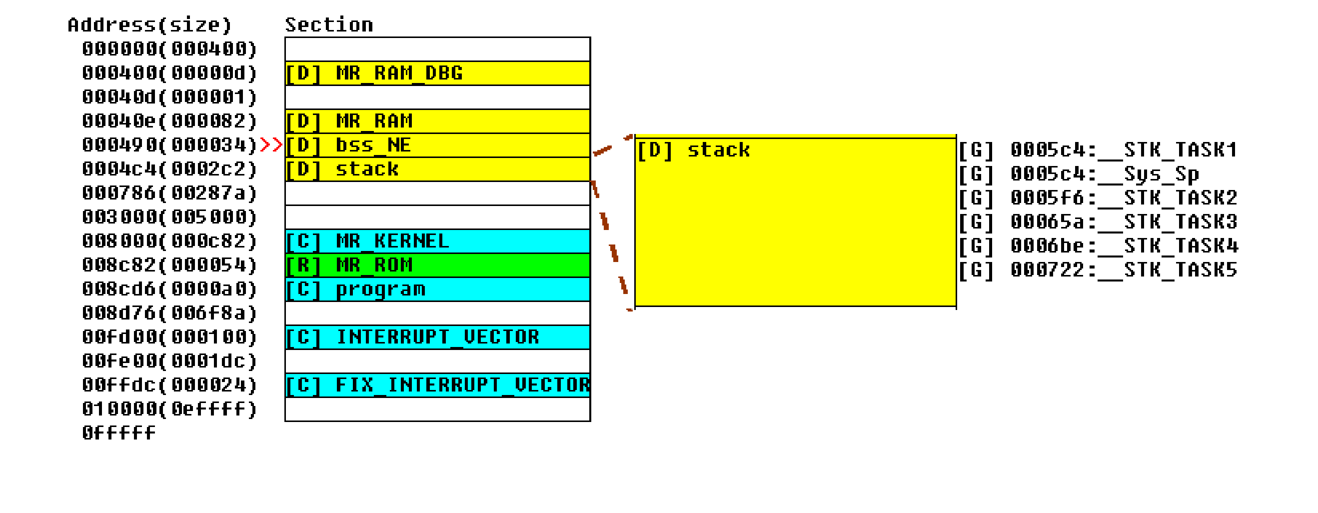


Figure 3 A Snapshot Example of User Stack and System Stack Layout using Map Viewer Utility

The stack size calculation method differs between the stacks and the language an application is written in. It can be summarized as shown in Figures 4 and 5.

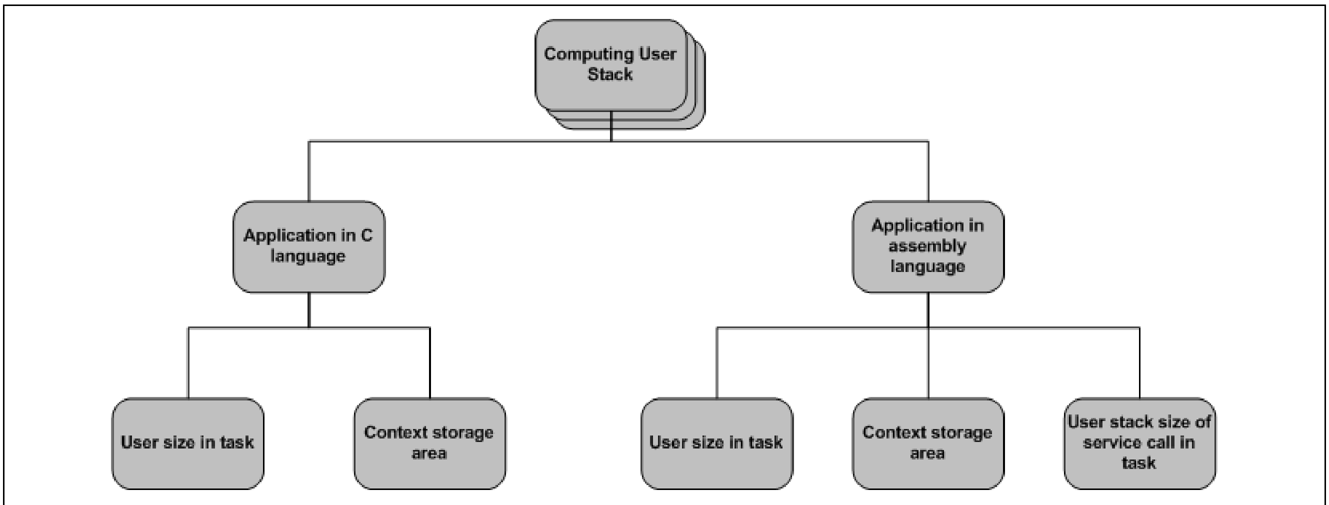


Figure 4 Components in User Stack Computation

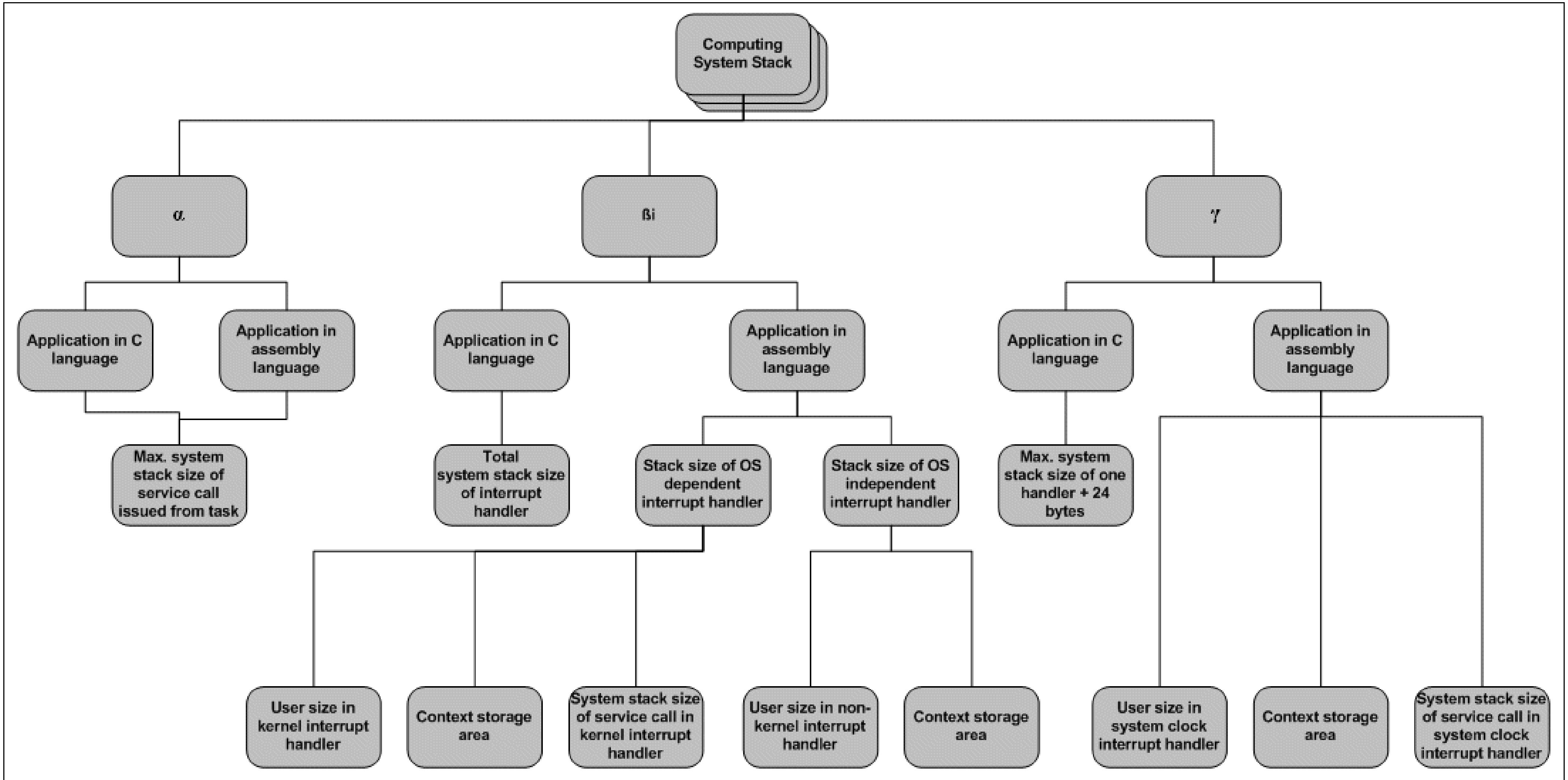


Figure 5 Components in System Stack Computation

3. Settings for STK Viewer Utility

STK Viewer utility is required for the computation of stack size for program written in C language. The utility analyzes the inspector information added to the absolute module file ("mot") and calculates the stack size needed for each task and the relationship of function calls.

To generate the inspector information in the absolute module file, specific compiler options will need to be enabled in the HEW environment. Figure 6 illustrates the steps to set the compiler options.

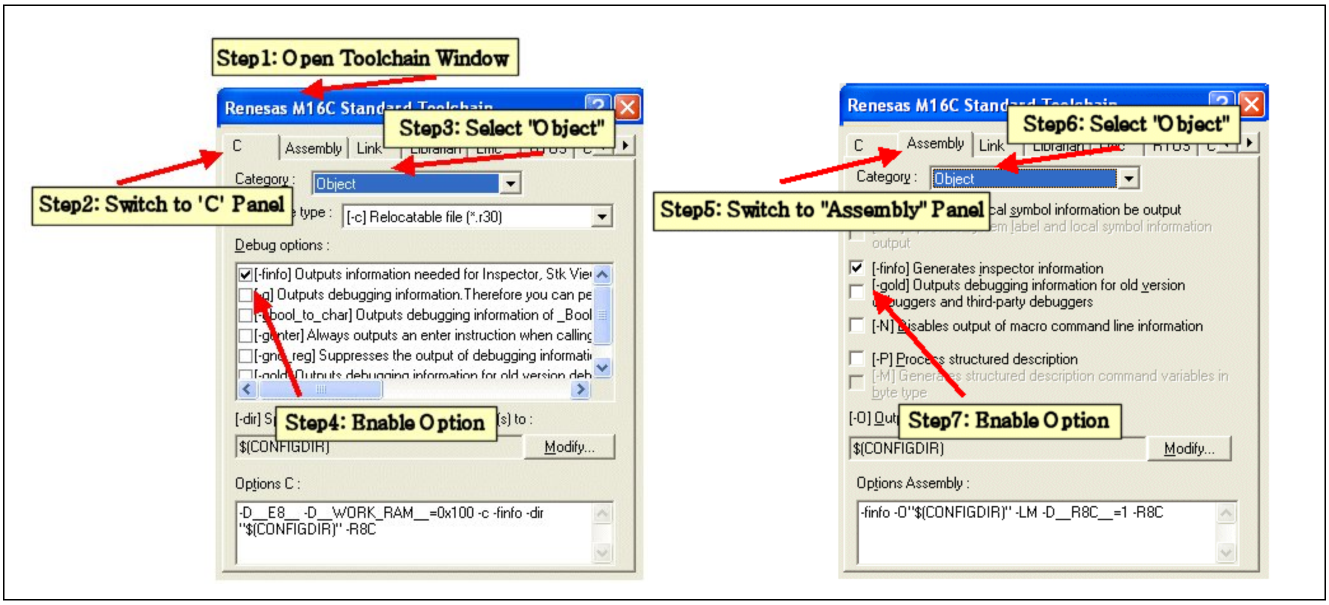


Figure 6 Setting Compiler Options

4. Computing User Stack Size

The computation methods for user stack size differ between an application written in C language and assembly language.

4.1 Application written in C Language

The formula for the computation is as shown in Figure 7.

User Stack Size = (Note 1)User Size + (Note 2)Context Storage Area

Note 1: Sum of stack size shown in STK Viewer
Note 2: Fixed size of 20 bytes

Figure 7 Formula for User Stack Size Computation for Application in C Language

In addition, 5 bytes must be added to the above sum if any of the following service calls is used in the task: "get_tid" and "pol_flg".

4.1.1 User Size

To calculate the user size for individual task, use the STK Viewer utility to tabulate the total stack size of all the functions in the specific task. If an unknown, recursive or indirect function exist within the task, user will need to define the stack size for the function and add it manually to the task stack size (for more information, refer to STK Viewer manual). Figure 8 illustrates the method of retrieving the task stack size using STK Viewer.

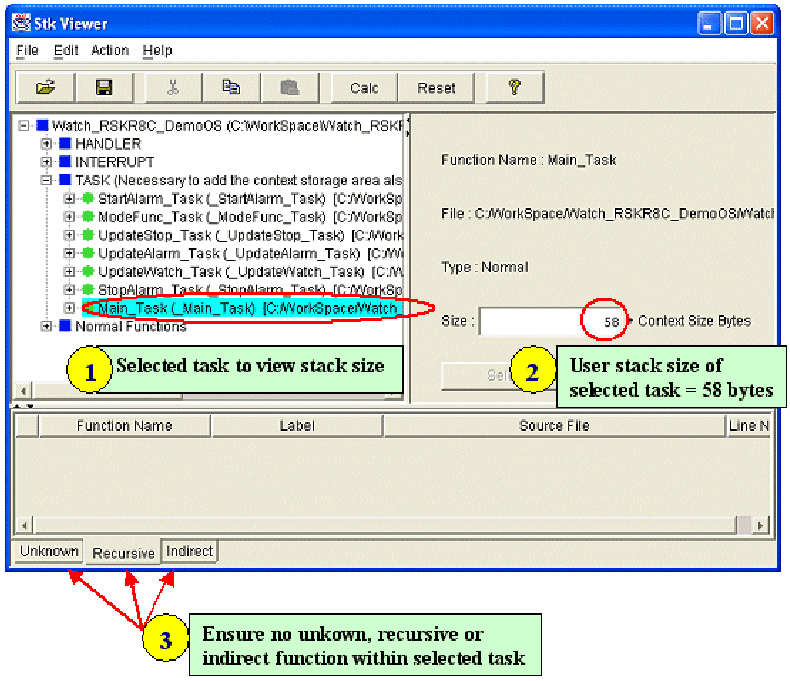


Figure 8 Reading Task Stack Size of "Main_Task" using STK Viewer

4.1.2 Context Storage Area

Prior to executing a service call, context of the task will be saved in the user stack area. The registers within the context to be saved are shown in Figure 9. When an application is written in C language, all registers (R0, R1, R2, R3, A0 A1, SB & FB) are to be saved. Thus the context storage area is fixed at 20 bytes with inclusion of program counter and flag registers.

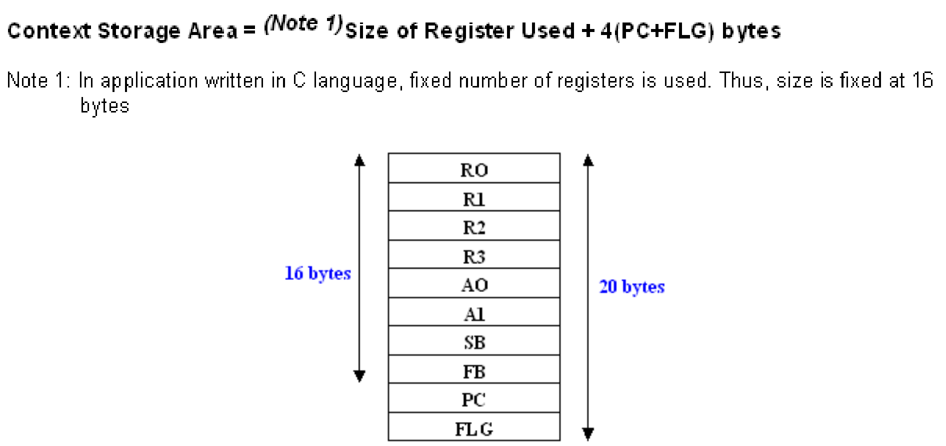


Figure 9 Context Storage Area Calculation

4.1.3 Example

With reference to the task "Main_Task" in Figure 8,

User stack size
= User size + context storage area
= 58 bytes + 20 bytes
= 78 bytes

Therefore, task "Main_Task" should be allocated a minimum stack size of 78 bytes (refer to Figure 10).

```
//Task Definition
task[1]{
    name            = ID_Task1_Main;
    entry_address    = Main_Task();
    stack_size  = 78;
    priority    = 6;
    initial_start    = ON;
    exinf    = 0x0;
};
```

Figure 10 Stack Size Allocation of Task "Main_Task"

4.2 Application written in Assembly Language

When an application is written in assembly language, the formula for the computation is shown in Figure 11.

User Stack Size = (Note 1)**User Size +**
(Note 2)**Context Storage Area +**
(Note 3)**User Stack of Service Call**

Note 1: Stack size used in subroutine call and variable declaration within the task
Note 2: Dependent on registers selected in configurator file
Note 3: Maximum stack size required by one of service calls within the task

Figure 11 Formula of Computing User Stack Size for Application Written in Assembly Language

4.2.1 User Size

Incorporating function/subroutine calls in a Task section is an almost indispensable need. The total stack size required by the subroutine calls is defined as the "User Size".

4.2.2 Context Storage Area

When an application is written in assembly language, user is able to define the registers to be used in a task (refer to Figure 12). If the context is not defined, all the registers (R0, R1, R2, R3, A0 A1, SB & FB) are used. Thus, context storage area is a summation of the stack size required by the registers used, the program counter and flag registers. In Figure 12, total context storage area required for ID_TASK1 is 12 bytes (R0=R1=R2=A0=2 bytes) including program counter and flag registers.

```
task[1]{
    name            = ID_TASK1;
    entry_address   = Task1();
    stack_size      = 50;
    priority        = 1;
    context         = R0,R1,R2,A0;
    initial_start   = ON;
    exinf           = 0x0;
};
```

Define only R0, R1, R2 and A0 registers are used in ID_TASK1

Figure 12 Task Definition in Configurator File

4.2.3 User Stack of Service Call

The "User Stack of Service Call" refers to the maximum user stack size that required by one of the service calls that can be issued from task or handlers (refer to Appendix III).

With reference to Appendix III, if both or either of the service calls "sns_ctx" and "sns_loc" is/are issued from the task, "User Stack of Service Call" will be 10 bytes.

4.2.4 Example

Figure 13 illustrates an example of calculating the required user stack size for a task with the following characteristics:

- Consist of a function call
- A 2 bytes variable declared inside the task
- Service calls "sta_tsk", "sns_ctx" and "sns_loc" are issued from the task

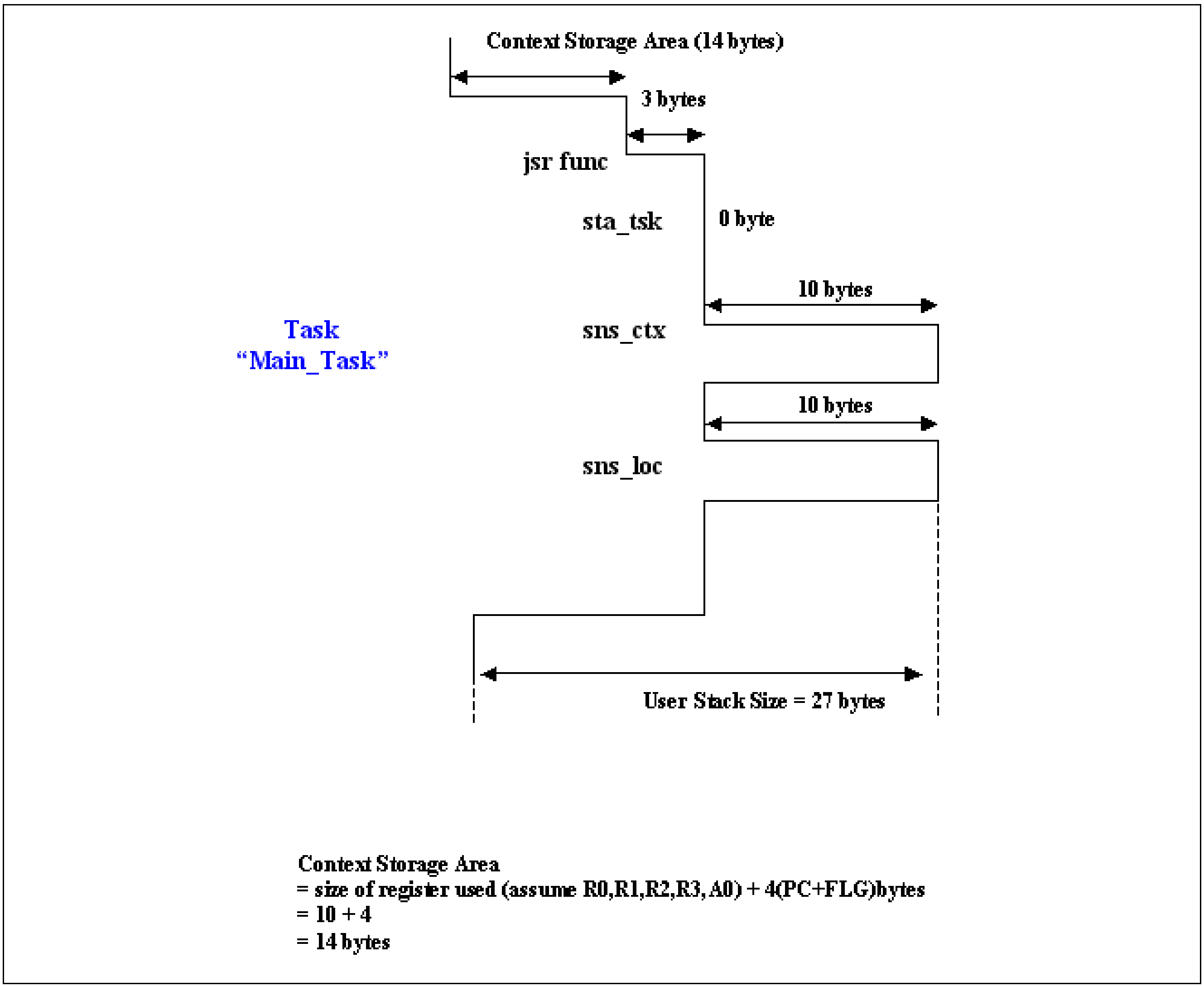


Figure 13 An Example of Calculating User Stack Size for an Application written in Assembly Language

5. Computing System Stack Size

The formula for the computation is as shown in Figure 14.

Necessary size of the system stack = $\alpha \Sigma \beta i (\gamma)$

***α**: Maximum system stack size among service calls to be used*
***βi**: Stack size to be used by interrupt handler*
***γ**: Stack size to be used by system clock interrupt handler*

Figure 14 Formula for System Stack Size Computation

The system stack size computation for nested interrupts is illustrated in Figure 15.

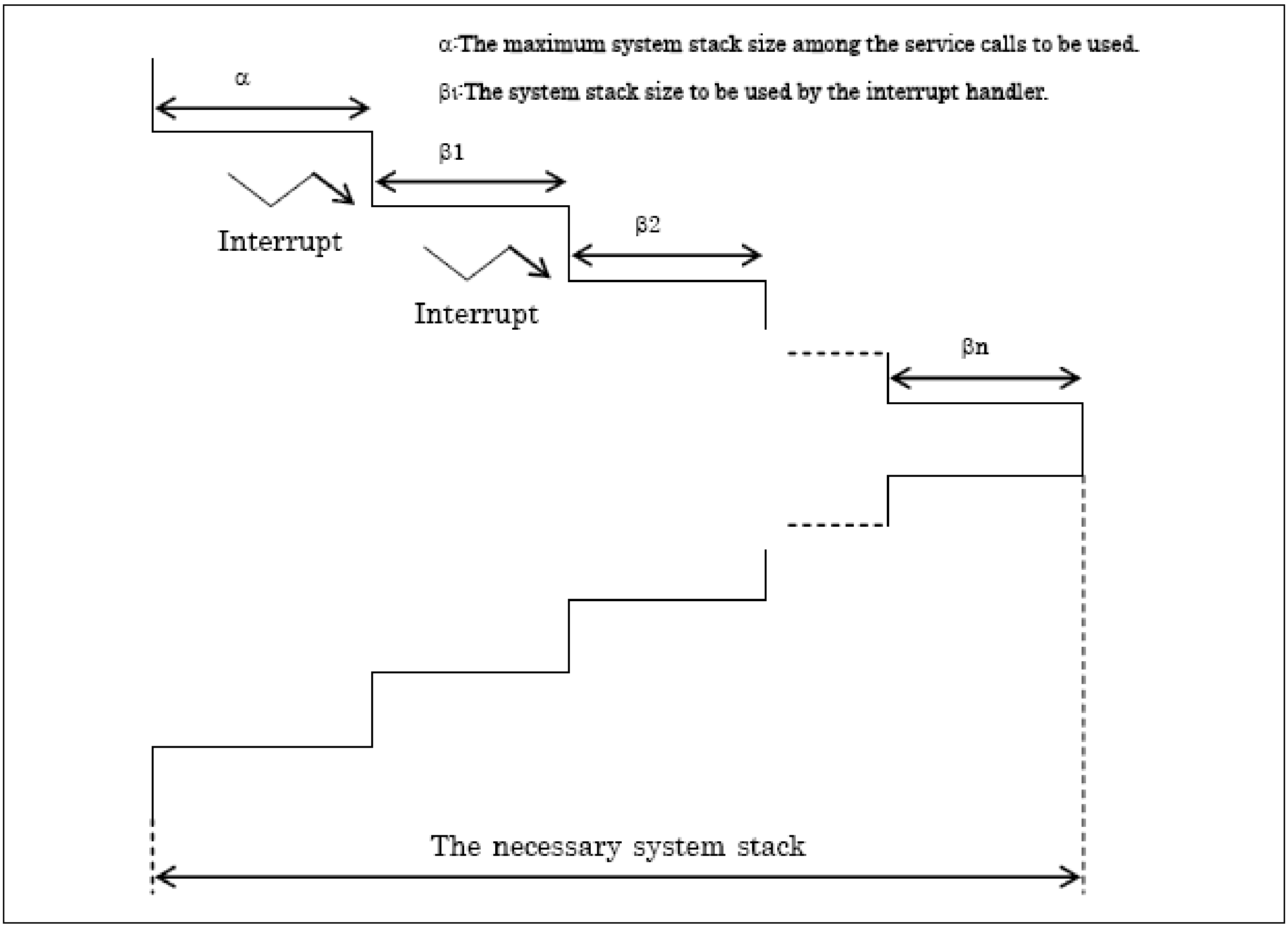


Figure 15 System Stack Computation for Nested Interrupts

5.1 Computing α

The method for computing **α** is the same for program written in C or assembly language. **α** denotes the maximum system stack size required by one of the service calls issued in the Task. Figure 16 illustrates an example of the computation.

1 Assume only 2 tasks created in application.

```
void Task1(VP_INT stacd)
{
    ercd = sta_tsk(ID_Task2);
    ercd = slp_tsk();
}
void Task2(VP_INT stacd)
{
    ercd = sig_sem(ID_Sem1);
    ercd = wup_tsk(ID_TASK1);
    ercd = sta_alm(ID_Alm1,0);
    ercd = sta_tsk(ID_Task1);
}
```

2 With reference to the table below, system stack required for all the service calls issued within the application can be identified.

Service Call	Stack Size		Service Call	Stack Size	
	User Stack	System Stack		User Stack	System Stack
sta_tsk	0	**2**	wai_flg	(5)	2
ext_tsk	0	0	pol_flg	10(5)	0
ter_tsk	0	4	snd_dtq	0	4
chg_pri	0	4	psnd_dtq	0	2
slp_tsk	0	**2**	rcv_dtq	(5)	2
wup_tsk	0	**2**	prcv_dtq	(5)	2
can_wup	10	0	dis_dsp	4	0
rel_wai	0	4	sta_cyc	10	0
sus_tsk	0	2	stp_cyc	10	0
rsm_tsk	0	2	sta_alm	10	**0**
dly_tsk	0	4	stp_alm	10	0
sig_sem	0	**2**	get_tid	10(5)	0
wai_sem	0	2	loc_cpu	4	0
pol_sem	10	0	unl_cpu	0	0
set_flg	0	6	ref_ver	12	0
clr_flg	10	0	ena_dsp	0	0

3 "sig_sem", "wup_tsk", "slp_tsk" and "sta_tsk" required the largest system stack (2bytes), thus **α** for application is 2 bytes

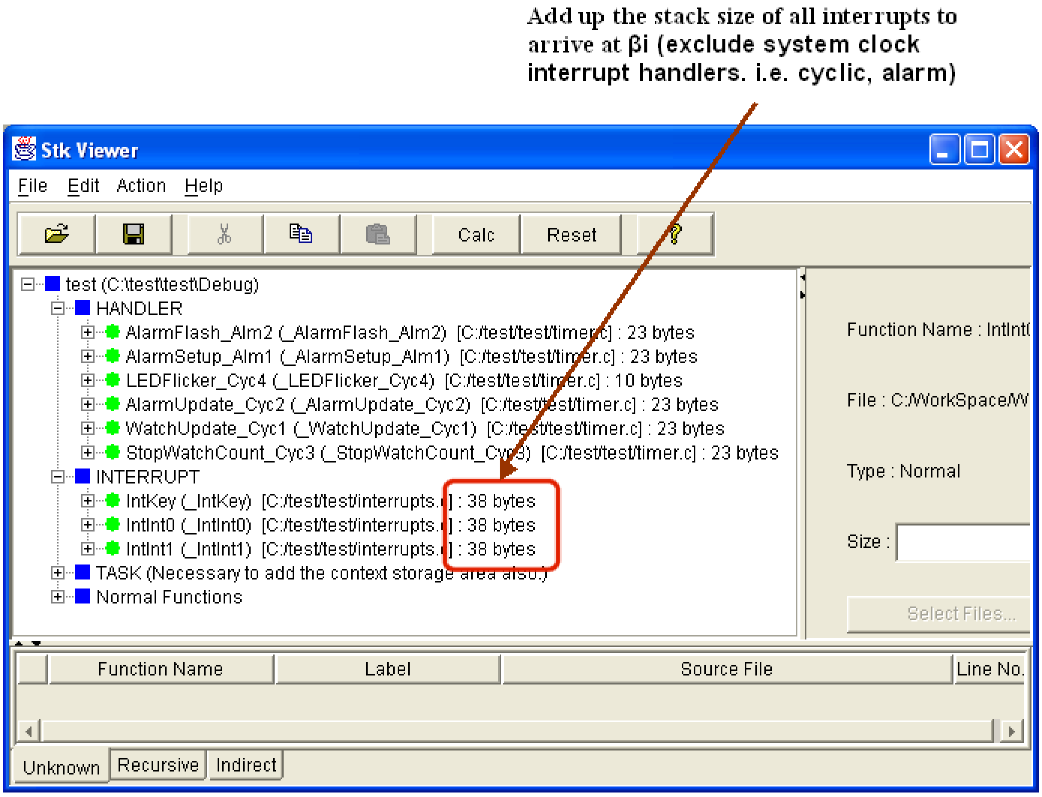
Figure 16 An Example of Computing α

5.2 Computing βi

The stack size used by an interrupt handler that is invoked during a service call is calculated differently for applications written in C or assembly language.

5.2.1 Application written in C Language

To calculate **βi** for an application written in C language, use the STK Viewer utility to tabulate the total stack size of all the interrupts (Kernel) and handlers (Non-kernel) within the application. Figure 17 illustrates an example of the computation.



$$\beta i = 38+38+38 = 114 \text{ bytes}$$

Figure 17 An Example of Computing βi for Application Written in C Language

5.2.2 Application written in Assembly Language

To calculate **βi** for an application written in assembly language, it is required to identify interrupts that are Kernel and Non-kernel. The formulae for computing **βi** are illustrated in Figure 18.

1. Bi = Stack Size of Kernel Interrupt Handler +
   Stack Size of Non-Kernel Interrupt Handler

2. Stack size of Kernel Interrupt Handler
   = Context Storage Area +
   User Size +
   System Stack Size of Service Call

3. Stack size of Non-Kernel Interrupt Handler
   = Context Storage Area +
   User Size

Figure 18 Formula of computing βi for Application Written in Assembly Language

Context storage area can be computed as defined in section 4.2.2. It is a summation of the size of registers defined in configurator file, program counter and flag registers. “User Size” refers to the stack size required by subroutine calls and variable declarations in the interrupt handler (refer to section 4.2.1). “System Stack Size of Service Call” refers to the maximum system stack size of the service calls issued within the kernel (OS-dependent) interrupt handler. The stack size of individual service calls issued from handlers can be found in the table in APPENDIX II.

Figure 19 illustrates an example of calculating the stack size for a kernel interrupt handler.

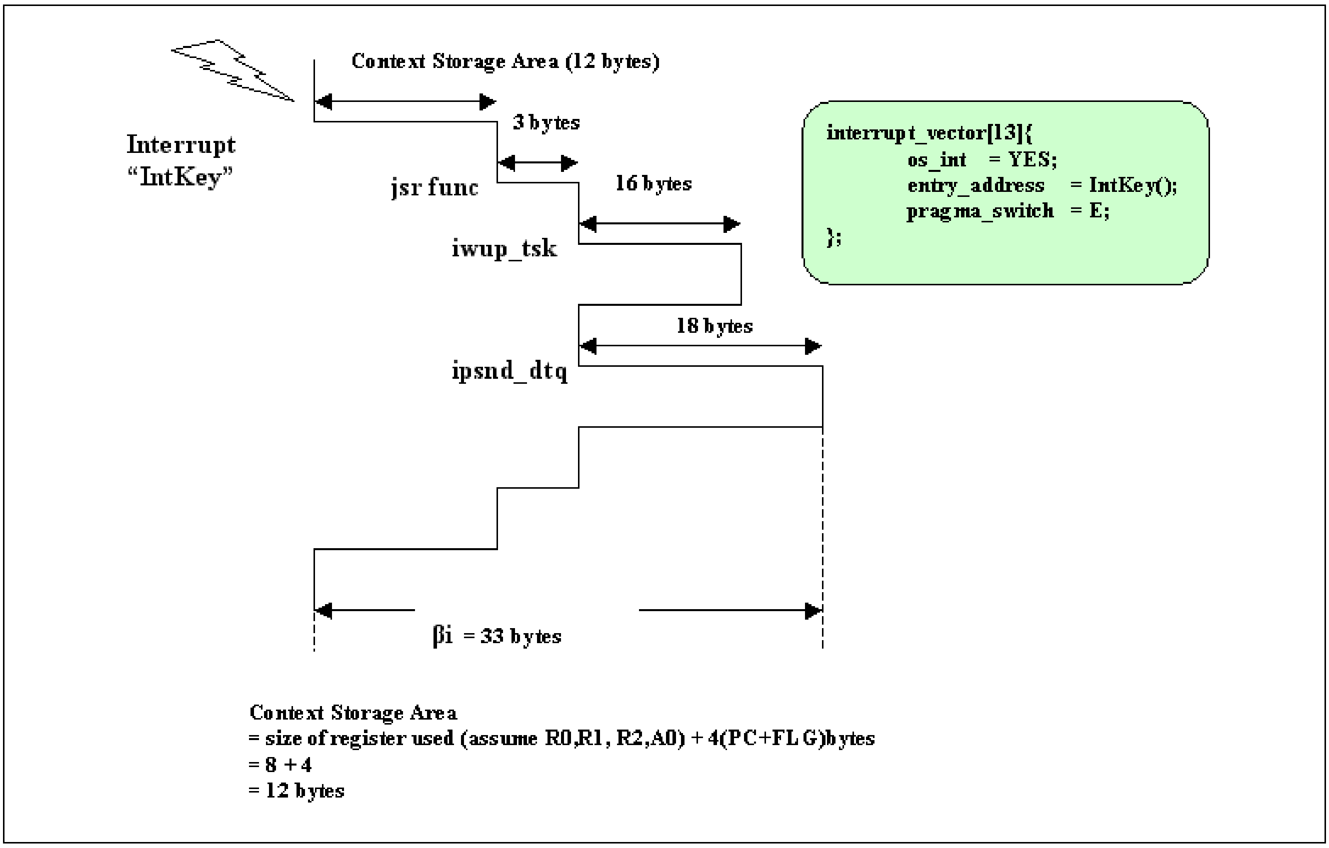


Figure 19 An Example of Calculating βi for an kernel Interrupt Handler

5.3 Computing γ

γ denotes the system stack size used by system clock interrupt handler. A system timer is not used when cyclic handler, alarm handler and none of the service calls under Time Management Function (i.e. isig_tim) is used. In the case where a system timer is not used, system stack size is equivalent to zero. The computation methods can be summarized in Figure 20.

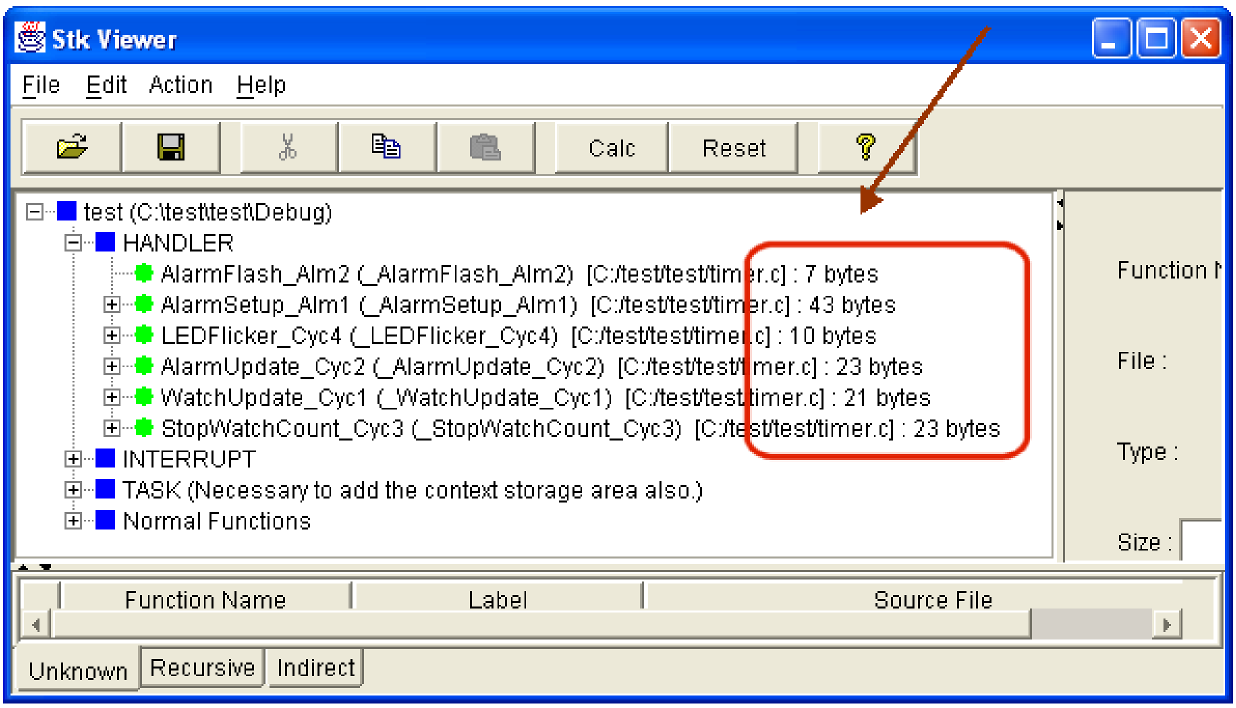
Scenario	γ (Bytes)
Time Management Function service call, cyclic handler and alarm handler not used	0
Time Management Function service calls used only	14
Cyclic handler and/or alarm handler used	24 + maximum size of cyclic handler or alarm handler (whichever is larger)

Figure 20 Summary of Computation Method for γ

5.3.1 Application written in C Language

In the computation of system stack for system clock interrupt handlers; use the STK Viewer utility to display the system stack size required by the individual handlers. To compute γ, identify the maximum stack required among the handlers and add 24 bytes to the stack size (refer to Figure 21).

Choose the maximum system stack required among the cyclic and alarm handlers. In this case, 43 bytes is the maximum system stack used



$$\gamma = 24 + \text{maximum system stack size of handler}$$
$$= 24 + 43$$
$$= 67$$

Figure 21 An Example of Computing γ for Application Written in C Language

5.3.2 Application written in Assembly Language

The formula for the computation of γ for an application written in assembly language is shown in Figure 22.

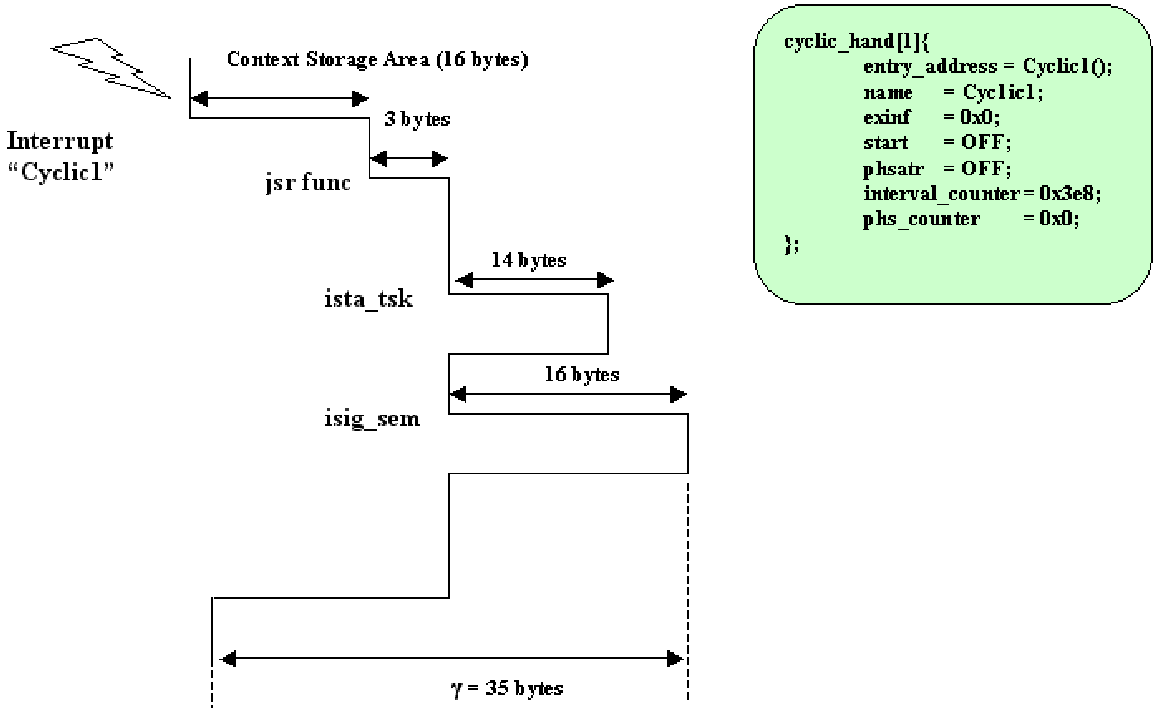
γ = (Note 1) User Size +
(Note 2) Context Storage Area +
(Note 3) System Stack Size of Service Call

Note 1: Stack size used in subroutine call and variable declaration within the handler
Note 2: Dependent on registers selected in configurator file
Note 3: Maximum stack size required by one of service calls within the handler

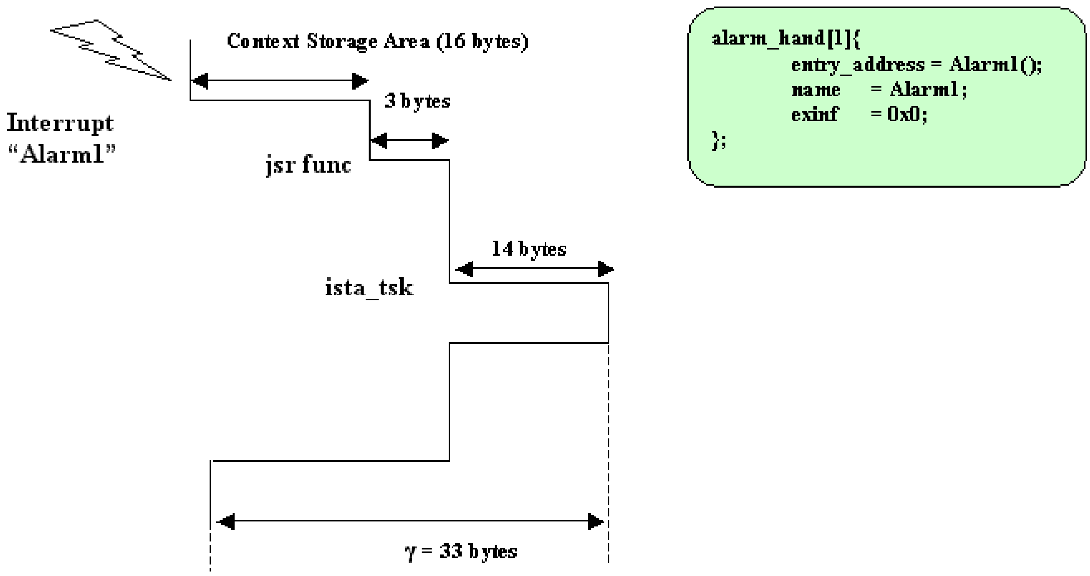
Figure 22 Formula of computing γ for Application Written in Assembly Language

"User Size" refers to the stack size required by subroutine calls and variable declarations in the system clock interrupt handler. Context storage area refers to the summation of the size of registers defined in configurator file, program counter and flag registers. "System Stack Size of Service Call" refers to the maximum system stack size of the service calls issued within the system clock interrupt handler. The system stack size of individual service calls issued from handlers can be found in the table in APPENDIX II.

Figure 23 illustrates an example of calculating the stack size for the system stack size used by the system clock interrupt handler.



Context Storage Area
= size of register used (assume R0,R1,R2,R3,A0,A1) + 4(PC+FLG)bytes
= 12 + 4
= 16 bytes



Context Storage Area
= size of register used (assume R0,R1,R2,R3,A0,A1) + 4(PC+FLG)bytes
= 12 + 4
= 16 bytes

Assuming only one alarm handler (Alarm1) and one cyclic handler (Cyclic1) are being used in the application. As shown above, γ of Cyclic1 is greater than that of Alarm1, so the overall system stack size γ is 35bytes

Figure 23 An Example of Computing γ for Application Written in Assembly Language

6. Reference Documents

User's Manual

- MR8C/4 V1.00 User's Manual
- STK Viewer User's Manual
- R8C Family Software Manual

The latest version can be downloaded from the Renesas Technology website

7. Appendix I: Listing of Stack Sizes used by Service Calls Issued from Task (Bytes)

Service call	Stack size		Service call	Stack size	
	User stack	System stack		User stack	System stack
sta_tsk	0	2	sta_cyc	10	0
ext_tsk	0	0	stp_cyc	10	0
ter_tsk	0	4	sta_alm	10	0
chg_pri	0	4	stp_alm	10	0
slp_tsk	0	2	get_tid	10(5)	0
wup_tsk	0	2	loc_cpu	4	0
can_wup	10	0	unl_cpu	0	0
rel_wai	0	4	ref_ver	12	0
sus_tsk	0	2	dis_dsp	4	0
rsm_tsk	0	2	ena_dsp	0	0
dly_tsk	0	4	snd_dtq	0	4
sig_sem	0	2	psnd_dtq	0	2
wai_sem	0	2	rcv_dtq	(5)	2
pol_sem	10	0	prcv_dtq	(5)	2
set_flg	0	6			
clr_flg	10	0			
wai_flg	(5)	2			
pol_flg	10(5)	0			

(): Stack sizes used by service call in C programs.

8. Appendix II: Listing of System Stack Sizes used by Service Calls Issued from Handlers (Bytes)

Service call	Stack size	Service call	Stack size
iwup_tsk	14	iset_flg	22
irel_wai	14	ipsnd_dtq	6
isig_sem	4	ret_int	10
ista_tsk	14		

(): Stack sizes used by service call in C programs.

9. Appendix III: Listing of Stack Sizes used by Service Calls that can be issued either from Tasks or Handlers (Bytes)

Service call	Stack size	Service call	Stack size
sns_ctx	10	sns_loc	10
sns_dsp	10		

Website and Support

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

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

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jan.01.10	—	First edition issued