

# Usage of SADDR Area and CALLT Instruction

CC-RL C Compiler for RL78 Family

Microcomputer Tool Product Marketing Department, Tool Business Division

Renesas System Design Co., Ltd.

Jun 30, 2015 Rev. 1.00

R20UT3476EJ0100

© 2015 Renesas System Design Co., Ltd. All rights reserved.

#### Introduction

- This document describes how to output a code that uses the SADDR area and CALLT instruction of the RL78 family when using the CC-RL C compiler for the RL78 family.
- Using the SADDR area and CALLT instruction reduces the code size.
- Each amount of code reduction shown in this document only applies to the corresponding example; the actual reduction will vary slightly between cases.
- The output assembly-language codes shown in this document are examples compiled with the medium model and the code size precedence option (-Osize) specified. Note that the output code will differ when a different type of optimization (default optimization or speed precedence optimization) is specified.
- This document uses the following tools and versions for description.
  - CC-RL C compiler for the RL78 family V.1.01.00
  - e<sup>2</sup> studio integrated development environment V.4.0.0.26
  - CS+ integrated development environment V.3.01.00





- How to Use the SADDR Area and CALLT Instruction
- Using the SADDR Area in a C Source File
- Using the CALLT Instruction in a C Source File
- Generating Variables/Functions Information File with Linker
- Using Variables/Functions Information File (e<sup>2</sup> studio)
- Using Variables/Functions Information File (CS+)



### How to Use the SADDR Area and CALLT Instruction

- Use the following methods to specify the SADDR area and the CALLT instruction.
  - Declaring in C source files
    - \_\_saddr declaration: SADDR area
    - #pragma saddr declaration: SADDR area
    - \_\_\_\_callt declaration: CALLT instruction
    - #pragma callt declaration: CALLT instruction
  - Using the variables/functions information file
    - Use the linker option -vfinfo to statically analyze the reference frequencies and generate a variables/functions information file in which variables and functions with #pragma saddr or #pragma callt declarations added are listed in the order of reference frequency.
    - Specify the generated file in the -preinclude option at compilation.



## Using the SADDR Area in a C Source File (1/2)

- Use \_\_saddr declaration for the frequently used external variables and static variables inside functions.
- For a one-bit field especially, \_\_\_\_saddr declaration can be expected to have a large effect.
- Declaring \_\_\_\_saddr allocates variables to the SADDR area, and the variables are accessed with direct manipulation instructions or small-size instructions.
- Example:
  - C source program

Before Change	After Change
typedef struct {	typedef struct {
unsigned char b0:1;	unsigned char b0:1;
unsigned char b1:1;	unsigned char b1:1;
unsigned char b2:1;	unsigned char b2:1;
unsigned char b3:1;	unsigned char b3:1;
unsigned char b4:1;	unsigned char b4:1;
unsigned char b5:1;	unsigned char b5:1;
unsigned char b6:1;	unsigned char b6:1;
unsigned char b7:1;	unsigned char b7:1;
} BITF;	} BITF;
BITF data0, data1;	saddr BITF data0, data1;
data0.b4 = data1.b1;	data0.b4 = data1.b1;

RENESAS

# Using the SADDR Area in a C Source File (2/2)

#### Example:

Output assembly-language program

Before Change			After Change		
movw	hl,#LOWW (_data1)	3			
mov1	CY,[hl].1	2	mov1	CY,_data1.1	3
movw	hl,#LOWW (_data0)	3			
mov1	[hl].4,CY	2	mov1	_data0.4,CY	3
	10 by	tes			6 bytes

#### Note:

 Alternatively, the variables/functions information file can be used to allocate variables to the SADDR area.





## Using the CALLT Instruction in a C Source File (1/2)

- Use \_\_\_\_callt declaration for frequently called functions.
- Declaring \_\_\_\_callt stores the addresses of the functions to be called in the callt table area [80H - BFH], and the functions are called with a smaller-size code than that for direct function calls.

- Example:
  - C source program

Before Change	After Change
void func_sub(void)	<pre>callt void func_sub(void)</pre>
{	{
;	;
}	}
void func()	void func()
{	{
func_sub();	func_sub();
;	;
func_sub();	func_sub();
}	}



# Using the CALLT Instruction in a C Source File (2/2)

- Example:
  - Output assembly-language program

Before Cha	ange			After Change	
				.SECTION .callt0,CALLT0 @_func_sub: .DB2 _func_sub	2
func	.SECTI	ON .textf,TEXTF		.SECTION .textf,TEXTF	
_func:	call	<pre>!!_func_sub</pre>	4	_func: callt [@_func_sub]	2
	call	!!_func_sub	4	callt [@_func_sub]	2
			8 bytes	6	bytes

#### Notes:

- A table of addresses for function calls is generated (.callt0).
- Due to generation of this table, code size reduction is not effective for a function called only once.
- The CALLT instruction requires more clock cycles for execution than the CALL instruction.
- Alternatively, the variables/functions information file can be used to specify declarations of the functions to be called through the CALLT instruction.





### Generating Variables/Functions Information File with Linker

#### Linker option -vfinfo

 This option selects variables and functions for which code reduction works most effectively based on their reference frequencies, adds declarations of saddr variables and callt functions through #pragma directives to the selected variables and functions, and outputs them to a header file (variables/functions information file).

#### Example:

```
/* RENESAS OPTIMIZING LINKER GENERATED FILE yyyy.mm.dd */
/*** variable information ***/
#pragma saddr data0 /* count: 10,size: 1,near,tp0.obj */
#pragma saddr data1 /* count: 5,size: 1,near,tp0.obj */
...
/* #pragma saddr datann */ /* count: 1,size: 1,near,tp1.obj */
...
/*** function information ***/
#pragma callt func_sub0 /* count: 4,far,tp0.obj */
#pragma callt func_sub1 /* count: 1,far,tp0.obj */
...
/* #pragma callt func0 */ /* count: 1,far,tp1.obj */
...
```



# Using Variables/Functions Information File (e<sup>2</sup> studio) (1/2)

Generating a variables/functions information file automatically

• Enable position optimization in the linker.

		□ <sup>1</sup> <sup>1</sup> <sup>1</sup> <sup>1</sup> <sup>1</sup> <sup>1</sup>	7
e <sup>2</sup> Properties for saddr_callt		Saddr_callt [HardwareDebug]	*
type filter text	Settings 🗇 🕆 🗸 🔻	Binaries	
<ul> <li>Presource</li> <li>Builders</li> <li>C/C++ Build</li> <li>Build Variables</li> <li>Change Toolchain Vers</li> <li>Dependency Scan</li> <li>Device</li> <li>Environment</li> <li>Logging</li> <li>Settings</li> <li>Tool Chain Editor</li> <li>C/C++ General</li> <li>Project References</li> <li>Run/Debug Settings</li> </ul>	<ul> <li>Tool Settings Build Steps Build Artifact Binary Parsers Error Parsers</li> <li>Common</li> <li>Optimization type:</li> <li>Optimizes branch instruction size</li> <li>Compiler</li> <li>Compiler</li> <li>Compiler</li> <li>Compiler</li> <li>Compiler</li> <li>Compiler</li> <li>Compiler</li> <li>Compiler</li> <li>Compiler</li> <li>Source</li> <li>Optimization</li> <li>Source</li> <li>Source&lt;</li></ul>	<ul> <li>Similares</li> <li>Includes</li> <li>Si cstart.asm</li> <li>Si cstart.asm</li> <li>Si hdwinit.asm</li> <li>In iodefine.h</li> <li>In saddr_callt_vfi.h</li> <li>Si stkinit.asm</li> <li>Si stkinit.asm</li> <li>Ic splo.c</li> </ul>	
4 >	Object         Optimization         Wer         Input         Advanced         Optimization         Optimization         Device         Output	<ul> <li>▷ ic tp1.c</li> <li>i HardwareDebug</li> <li>▷ ist</li> <li>▷ ist</li> <li>▷ isrc</li> <li>▷ is saddr_callt.abs - [rl78/le]</li> <li>▷ is saddr_callt.x - [rl78/le]</li> </ul>	H
?	OK Cancel	LinkerSubCommand.tmp	
• "F	Project name.h" is registered in the project tree	makefile.1st makefile 2nd saddr_callt_vfi.h saddr_callt.map sources.mk custom.bat	•

#### RENESAS

111

陷 Project Explorer 🛛

.€

Using Variables/Functions Information File (e<sup>2</sup> studio) (2/2)

Editing a variables/functions information file (after automatic generation)

11

- Disable position optimization that was enabled in the step shown in the previous page in the linker.
- Import the automatically generated "Project name.h" file to the src folder.
- Register the "Project name.h" file in [Include files at head of compiling units].

Properties for saddr_callt			
type filter text	Settings		<> ▼ <> ▼ ▼
<ul> <li>▷ Resource</li> <li>Builders</li> <li>▲ C/C++ Build</li> <li>Build Variables</li> <li>Change Toolchain Vers</li> <li>Dependency Scan</li> </ul>	Common     Bevice     CPU     Compiler     Compiler     Compiler     Compiler	Include file directories ["\$[TCINSTALL]\inc"	କି <b>କ୍ଷ</b> କି ବି ହା
Device Environment Logging Settings Tool Chain Editor ▷ C/C++ General Project References Run/Debug Settings	<ul> <li>Cangoage</li> <li>Colject</li> <li>Optimization</li> <li>User</li> <li>Assembler</li> <li>Cource</li> <li>Language</li> <li>Object</li> <li>Optimization</li> </ul>	Include files at head of compiling units saddr_callt_vfi.h	<b>ଇ ଛ</b> ହା ହା <sub>=</sub>
« [] )	User User Unker Advanced List Optimization Section Device Miscellaneous User	Macro definition	<u>원</u> 원 원 중1 원
?	1		OK Cancel

RENESAS



## Using Variables/Functions Information File (CS+) (1/2)

#### Generating a variables/functions information file automatically

Enable output of a variables/functions information file.

			Project Tree 4 X
~	CC-RL Property		2 🕜 🙎  🗃
	Specify execution start address Fill with padding data at the end of a section Address setting for specified area of vector table Address setting for unused vector area	No No Address setting for specified area of vector table[0]	RL78 G13 Tutorial Basic Operation CC (Project)* RSF100LE (Microcontroller) CC-RL (Build Tool) RL78 E1(Serial) (Debug Tool) File
4	Variables/functions information Output variables/functions information header file	Yes(-VFINFO)	
Sel If ")	Output folder for variables/functions information header file         Variables/functions information header file         Section         Verify         Message         Others         tput variables/functions information header file         ects whether to output a variables/functions information header file         ects whether to output a variables/functions information header file         ects whether to output a variables/functions information header         'es" is selected in this field, executes commands in the follow         'ommon Options        Compile Options          'Project name.h" is registered	ving order Ins Link Options / Hex Output Options / I/O	<pre>stkinit.asm iodefine.h □ - ド生成</pre>
			RL78_G13_Tutorial_Basic_Operation_CC_vfi.h

#### RENESAS

## Using Variables/Functions Information File (CS+) (2/2)

- Editing a variables/functions information file (after automatic generation)
  - Disable output of a variables/functions information file that was enabled in the step shown in the previous page.
  - Copy the "Project name.h" file to another folder (such as the source folder). (Although it can be used without copying, when output of a variables/functions information file is enabled, the tool overwrites and deletes the file.)
  - Register the "Project name.h" file in [Include files at head of compiling

	CC-RL Property Perform inter-module optimization	No
	Perform optimization considering type of data indicated by I	
	Outputs additional information for inter-module optimization	
4	Preprocess	
⊳	Additional include paths	Additional include paths[2]
Þ	System include paths	System include paths[0]
⊿	Include files at head of compiling units	Include files at head of compiling units[1]
	[0]	DefaultBuild\RL78_G13_Tutorial_Basic_Operation_CC_vfi.h
÷	Meere definition	Maara definition[0]
⊳	Macro undefinition	Macro undefinition[0]
⊳	Memory Model	
⊳	C Language	
⊳	Character Encoding	
	Output Code	

13

RENESAS



Renesas System Design Co., Ltd.

©2015 Renesas System Design Co., Ltd. All rights reserved.