# RENESAS

## RL78 Family Data Flash Library Type04 Package Ver.3.00 Release Note

Thank you for using the RL78 Family Data Flash Library Type04 Package Ver.3.00.

This document contains notes and points for caution on using the Data Flash Library Type04 Package Ver.3.00. Please read this document before use.

# Contents

Chapter 1	r 1 Target Product	
Chapter 2	User's Manual	2
Chapter 3	Revisions	2
Chapter 4	Supported Tools	3
Chapter 5	Installation	3
5.2 Unir	allation nstallation Configuration	3
Chapter 6	How to Build a Program	5
6.2 Buil 6.2.1 6.2.2 6.2.3 6.2.4 6.3 Buil 6.3.1 6.3.2 6.3.3	tware to be Used ding Using CS+(former CubeSuite+) Building a C Program Building an Assembly-Language Program Removing the automatically generated files (only when the CC-RL compiler is used) Building ding Using e <sup>2</sup> studio Creating a Project Building a C Program Building an Assembly-Language Program es at Build When the CA78K0R Compiler is Used	5 8 .10 .11 .11 .11 .13 .17 .22 .22
Chapter 7	How to Debug a Program	
7.1 Note	es on Debugging	
Chapter 8	Sample Program	
<ul> <li>8.1 Initial Settings of the Sample Program</li></ul>		



# Chapter 1 Target Product

In the Data Flash Library Type04 Package Ver.3.00, "RL78 Family Data Flash Library Type04 Ver.1.05 for LLVM Compiler" has been newly added.

Product Name	Ver.	Installer File Name	Ver.
Data Flash Library Type04 for CA78K0R	V1.05		
Compiler for the RL78 Family			
Data Flash Library Type04 for CC-RL	V1.05		
Compiler for the RL78 Family		RENESAS_RL78_FDL_T04_3V00.exe	V3.00
Data Flash Library Type04 for LLVM	V1.05		
Compiler for the RL78 Family			

# Chapter 2 User's Manual

The following user's manual is available for this version.

Title	Document Number
RL78 Family Data Flash Library Type04 User's Manual	R01US0049EJ0110

# Chapter 3 Revisions

The revised contents of this version are as follows:

No.	Package Ver.	Target	Contents
		Library V1.05 for CA78K0R Compiler	There is no change in the library from the package Ver.2.00.
		Library V1.05 for CC-RL Compiler	There is no change in the library from the package Ver.2.00.
1	V3.00	Library V1.05 for LLVM Compiler	Newly added.
		User's manual	Revised from Rev.1.06 to Rev.1.10. For details on the corrections to the user's manual in response to the revision, refer to the revision history of the user's manual.



# Chapter 4 Supported Tools

Use the following tool version when using the Data Flash Library Type04 Ver.1.05.

Target library	Tool Name	Version
Library for CA78K0R	Integrated development environment CubeSuite+	V1.00.00 or later
Compiler	Integrated development environment CS+	V3.00.00 or later
Library for CC-RL	Integrated development environment CS+	V3.01.00 or later
Compiler	Integrated development environment e <sup>2</sup> studio	Listed from Version: 2023-10 <sup>Note</sup>
Library for LLVM Compiler	Integrated development environment e <sup>2</sup> studio	Version: 2023-10 or later

Note: Available for e<sup>2</sup> studio with embedded CC-RL compiler V1.00 or later.

# Chapter 5 Installation

This chapter describes how to install and uninstall the Data Flash Library Type04 Package Ver.3.00.

#### 5.1 Installation

Install the Data Flash Library Type04 by using the following procedure:

- (1) Start Windows.
- (2) Decompress the file that contains the Data Flash Library Type04 Package and run of the installer.
- (3) Select "Asia/Oceania English" from the drop-down list.
- (4) Click on the "OK" button to proceed installation according to the instructions of the installer.



#### 5.2 Uninstallation

Uninstall the Data Flash Library Type04 by using the following procedure:

- (1) Start Windows.
- (2) Delete the folder that contains the Data Flash Library Type04 files and was placed at the location chosen by the user.



## 5.3 File Configuration

The file organization after this library is installed is shown below.



Notes: 1. x indicates the omitted numerals in version or revision numbers.

- 2. To use the sample program for CA78K0R, the program file (\*.c) and link directive file (\*.dr) should be embedded together.
- To use the sample program for CC-RL, the program file (\*.c) should be embedded. The link information for the sample program for CC-RL should be specified through the link setting window on the CS+ or the e<sup>2</sup> studio.
- 4. To use the sample program for LLVM, the program file (\*.c) and linker script file (\*.ld) should be embedded together.



# Chapter 6 How to Build a Program

This chapter describes how to build a program using the Data Flash Library Type04.

## 6.1 Software to be Used

The following integrated development environment is necessary for building programs using the Data Flash Library Type04.

- Integrated development environment CS+ V3.00.00 or later for CA78K0R compiler /Integrated development environment CubeSuite+ V1.00.00 or later for CA78K0R compiler
- Integrated development environment CS+ V3.01.00 or later for CC-RL compiler
   /Integrated development environment e<sup>2</sup> studio Listed from Version:2023-10 or later for CC-RL compiler <sup>Note</sup>
   Note: Available for e<sup>2</sup> studio with embedded CC-RL compiler V1.00 or later.
- Integrated development environment e<sup>2</sup> studio Version: 2023-10 or later for LLVM compiler

## 6.2 Building Using CS+(former CubeSuite+)

This section describes how to include the Data Flash Library Type04 in a user-created program and build the user program by using CS+. The target compilers for CS+ are CC-RL compiler and CA78K0R compiler.

#### 6.2.1 Building a C Program

(1) Creating a project and specifying the source file

Create a project by using CS+. In the Project Tree window displayed on the left, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 6-1). Next, click the Files of type drop-down list to display a list of the file types. Select C source file (\*.c), and then register the user-created program as the source file.



Figure 6-1. Registering the User Program File



#### (2) Specifying the include file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File.

The Add Existing File dialog box is displayed (as shown in Figure 6-2).

Next, click the Files of type drop-down list to display a list of the file types. Select Header file (\*.h;\*.inc), and then register the header files (pfdl.h, pfdl\_types.h) of the data flash library.

ė. 🔊 👘			
Add		Add File	
🕼 Add Existing File	<u> </u>	>	<
	« FDL → CA78K0R_110 → lib	v ♂ Search lib p	-
Organize 🔻 New	folder	III 🔹 🖬 🔇	
<ul> <li>This PC</li> <li>Desktop</li> <li>Documents</li> <li>Downloads</li> <li>Music</li> <li>Pictures</li> <li>Videos</li> <li>Windows (C:)</li> <li>Data (D:)</li> </ul>	Name 3 pfdl.h pfdl.inc pfdl_types.h		
	File <u>n</u> ame:	2 Header file (*.h; *.inc)	]
		4 <u>O</u> pen ▼ Cancel	

Figure 6-2. Registering the Include Files

(3) Specifying the library file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 6-3).

Next, click the Files of type drop-down list to display a list of the file types. Select Library file (\*.lib), and then register the data flash library file (pfdl.lib).



Figure 6-3. Registering the Library File



(4) Specifying the link directive file (only when the CA78K0R compiler is used)

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 6-4).

Next, click the Files of type drop-down list to display a list of the file types. Select Link directive file (\*.dr;\*.dir), and then register the link directive file that has the same name as the user-created program.

Add		•	🗋 Add	l File	
. 🕒 🔂 Rem	ove from Proiect	Shift+Del I	XI Add	New File	
🚳 Add Existing File					×
$\leftrightarrow$ $\rightarrow$ $\checkmark$ $\uparrow$	« Sample » C	√ Ū	Search C		Q
Organize 🔻 Ne	w folder			III -	•
<ul> <li>➡ This PC</li> <li>➡ Desktop</li> <li>➡ Documents</li> <li>➡ Downloads</li> <li>➡ Music</li> <li>➡ Pictures</li> <li>➡ Videos</li> <li>₩ Windows (C:)</li> <li>➡ Data (D:)</li> </ul>	Name 3 □ r_pfdl_sample	.c.dr			
	File <u>n</u> ame:	2 4	Link directi <u>O</u> pen	ve file (*.dr; *.dir)	

Figure 6-4. Registering the Link Directive File

#### 6.2.2 Building an Assembly-Language Program

(1) Creating a project and specifying the source file

Create a project by using CS+. In the Project Tree window displayed on the left, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 6-5). Next, click the Files of type drop-down list to display a list of the file types. Select Assemble file (\*.asm), and then register the user-created program as the source file.

Add	•	🛄 Add File	
E Remove from Project			
	Shirt-Dei	Add New File	
Add Existing File			×
$\leftarrow$ $\rightarrow$ $\checkmark$ $\uparrow$ $\blacksquare$ « user $\rightarrow$ asm	ٽ ~	Search asm	م
Organize 👻 New folder			
This PC Desktop Desktop Documents Downloads Music Pictures Videos Windows (C:) Data (D:) V	^		
File <u>n</u> ame:	2	·	~ ancel

Figure 6-5. Registering the User Program File

(2) Specifying the include file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File.

The Add Existing File dialog box is displayed (as shown in Figure 6-6).

Next, click the Files of type drop-down list to display a list of the file types. Select Header file (\*.h;\*.inc), and then register the header file (pfdl.inc) of the data flash library.



Add		•	Add File	
		NY N	Add Now File	
Add Existing File				×
· ← → • ↑ 📙	<pre>&lt;&lt; FDL &gt; CA78K0R_110 &gt; li</pre>	b v Ö Se	arch lib	م
Organize 🔻 Nev	v folder			•
<ul> <li>This PC</li> <li>Desktop</li> <li>Documents</li> <li>Downloads</li> <li>Music</li> <li>Pictures</li> <li>Videos</li> <li>Windows (C:)</li> <li>Data (D:)</li> </ul>	Name	^		
	File <u>n</u> ame:		eader file (*.h; *.inc)	~
		4	<u>O</u> pen <b>▼</b>	Cancel

Figure 6-6. Registering the Include File

(3) Specifying the library file

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 6-7).

Next, click the Files of type drop-down list to display a list of the file types. Select Library file (\*.lib), and then register the data flash library file (pfdl.lib).

1	Add
	The Remove from Project Shift+Del Add New File
	🕼 Add Existing File 🛛 🗙
	$\leftarrow \rightarrow \checkmark \uparrow$ $\bigcirc$ « FDL » CA78K0R_110 » lib $\checkmark$ $\circlearrowright$ Search lib $\checkmark$
	Organize 🔻 New folder 🛛 🔠 👻 🔟 📀
	<ul> <li>This PC</li> <li>Desktop</li> <li>Documents</li> <li>Downloads</li> <li>Music</li> <li>Pictures</li> <li>Videos</li> <li>Windows (C:)</li> <li>Data (D:)</li> </ul>
	File <u>n</u> ame: 2/ Library file(*.lib) ✓ 4 Open ▼ Cancel

Figure 6-7. Registering the Library File

(4) Specifying the link directive file (only when the CA78K0R compiler is used)

In the CS+ Project Tree window, right-click the File node, click Add, and then click Add File. The Add Existing File dialog box is displayed (as shown in Figure 6-8).

Next, click the Files of type drop-down list to display a list of the file types. Select Link directive file (\*.dr;\*.dir), and then register the link directive file that has the same name as the user-created program.

Image: Sample → C       Image: Sample → C       Image: Search C         Organize ▼       New folder       Image: Search C         Image: This PC       Name       Image: Search C	
← → ▼ ↑     _ « Sample > C     ▼ ひ     Search C       Organize ▼     New folder     Image: I	_
Organize  New folder	×
This PC	Q
This PC Name	?
<ul> <li>□ Desktöp</li> <li>□ Documents</li> <li>□ Downloads</li> <li>□ Music</li> <li>□ Pictures</li> <li>□ Videos</li> <li>□ Windows (C:)</li> <li>□ Data (D:)</li> </ul>	
File <u>n</u> ame: 2 Link directive file (*.dr; *.dir) 4 <u>Open</u> ▼ Cancel	~

Figure 6-8. Registering the Link Directive File

# 6.2.3 Removing the automatically generated files (only when the CC-RL compiler is used)

CS+ for the CC-RL compiler automatically generates some files under the File node in the Project Tree window. Among these, the processing of the "main.c" and "hdwinit.asm" files is included in the data flash library. Therefore, remove these two files from the target of the build process.

. .

To use assembly language, only "main.c" is removed because the sample program is not used.

			After removal
Brite RSF100LE-FSL (Project	)		
R5F100LE (Microc	١	Assemble	Bring R5F100LE-FSL (Project)*
🔨 CC-RL (Build Tool		Open	R5F100LE (Microcontroller)
RL78 Simulator (D	3	Open with Internal Editor	CC-RL (Build Tool)
🖃 👔 File		Open with Selected Application	
em hdwinit.asm	2	Open Folder with Explorer	📄 🗍 File
stkinit.asm	E	Windows Explorer Menu	- 📶 Build tool generated files
C main.c		Add	cstart.asm
h i ta la Cara h	-		stkinit.asm
iodenne.n	6	Remove from Project Shift+Del	iodefine.h
	P <sub>D</sub>	Copy Ctrl+C	
	Ē	Paste Ctrl+V	
	ajje	Rename F2	
	ajje	Change Extension	
		Property	





#### 6.2.4 Building

From the CS+ Build menu, click Build Project to build the project.

#### 6.3 Building Using e<sup>2</sup> studio

This section describes how to include the Data Flash Library Type04 in a user-created program and build the user program by using  $e^2$  studio. The target compilers for  $e^2$  studio are CC-RL and LLVM.

#### 6.3.1 Creating a Project

The  $e^2$  studio starts and from the [File] menu, select [New] – [C/C++ Project], the "Templates for New C/C++ Project" window will open.

e² e	e² e2_studio - e² studio				
File	Edit Source Refactor Navigate Search	Project Renesas Views Run Window Help			
	New	Alt+Shift+N > 🕅 Makefile Project with Existing Code			
	Open File	C/C++ Project			
	Open Projects from File System	Project			
	Close	Ctrl+W Convert to a C/C++ Project (Adds C/C++ Nature)			

Figure 6-10. Create a New Project

- When using the CC-RL compiler, select [Renesas CC-RL C/C++ Executable Project] displayed after selection in [Renesas RL78], and press "next" button.

New C/C++ Project	S New C/C++ Project — — >				
Templates for New C/O	C++ Project				
Make	[Deprecated] GCC for Renesas RL78 C/C++ Exect A C/C++ Executable Project for Renesas RL78 using the G RL78 Toolchain.				
Renesas Debug Renesas RL78	[Deprecated] GCC for Renesas RL78 C/C++ Libran A C/C++ Library Project for Renesas RL78 using the GCC I Toolchain.				
F	LLVM for Renesas RL78 C/C++ Executable Project A C/C++ Executable Project for Renesas RL78 using LLVM Toolchain.		esas RL78		
F	LLVM for Renesas RL78 C/C++ Library Project	Renesas	RL78 Too	lchain.	
e	Renesas CC-RL C/C++ Executable Project	°C-RL too.	lchain.		
F	Renesas CC-RL C/C++ Library Project Renesas RL78 using the CC-RL C/C++ Library Project for Renesas RL78 using the CC-RL	L toolcha	in.		
?	< <u>B</u> ack <u>N</u> ext > <u>Finish</u>		Cance	9	

Figure 6-11. Select the CC-RL Compiler for the Tool Chain

Input "project name" on "New Renesas CC-RL Executable Project" window, and press "Next" button.



- When using the LLVM compiler, Select [LLVM for Renesas RL78 C/C++ Executable Project] displayed after selection in [Renesas RL78], and press "Next" button.

New C/C++ Project		_		×
Templates for New C/C++ Project				
CMake RL78	GCC for Renesas RL78 C/C++ Executable Project A C/C++ Executable Project for Renesas RL78 using the GCC for Renesas RL78 Toolch	ain.		
	GCC for Renesas RL78 C/C++ Library Project A C/C++ Library Project for Renesas RL78 using the GCC for Renesas RL78 Toolchain.			
	LLVM for Renesas RL78 C/C++ Executable Project A C/C++ Executable Project for Renesas RL78 using LLVM for Renesas RL78 Toolchair	7.		
	LLVM for Renesas RL78 C/C++ Library Project A C/C++ Library Project for Renesas RL78 using LLVM for Renesas RL78 Toolchain.			
	Renesas CC-RL C/C++ Executable Project A C/C++ Executable Project for Renesas RL78 using the CCRL toolchain.			
	Renesas CC-RL C/C++ Library Project A C/C++ Library Project for Renesas RL78 using the CCRL toolchain.			
?	< <u>B</u> ack <u>Next &gt;</u> Einish		Cancel	

Figure 6-12. Select the LLVM Compiler for the Tool Chain

Input "Project name" on "New LLVM for Renesas RL78 Executable Project" window, and press "Next" button.

Select the [Target Device] of [Device Settings] and select "RL78 – G13" - "R5F100LE". (When the target device is RL78/G13 [Part Number: R5F100LE].)

It is a premise that E2 Lite is selected as a debugging tool and on-chip debugging is executed. Put a check mark to "Create Hardware Debug Configuration" by [Configurations]. And select "E2 Lite(RL78)". Press "Finish" button.

Select toolchain, d	evice & debug settings	
Toolchain Settings Language: Toolchain: Toolchain Version:	● C ○ C++ Renesas CC-RL ~	
Device Settings Target Board: Cu	Manage Toolchains	Configurations
Target Device: R5	Download additional boards	Create Hardware Debug Configuration E2 Lite (RL78) Create Debug Configuration RL78 Simulator
Endian: Lit Project Type: De		Create Release Configuration

Figure 6-13. Device Selection



## 6.3.2 Building a C Program

(1) Specifying the source and include files

Specifying the data flash library file in the created project.

- CC-RL: Register the data flash library files "pfdl.h", "pfdl\_types.h", "pfdl.lib" and "r\_pfdl\_sample\_c.c" in the "src" folder output by e<sup>2</sup> studio. (Figure 6-14)



Figure 6-14. Specifying the Source and Include Files (CC-RL)

- LLVM: Register the data flash library files "pfdl.h", "pfdl\_types.h", "libpfdl.a", "r\_pfdl\_sample\_c.c" and "r\_pfdl\_sample\_c.ld" in the "src" folder output by e<sup>2</sup> studio. (Figure 6-15)



Figure 6-15. Specifying the Source and Include Files (LLVM)



Exclusion of the file automatically added by the function of e<sup>2</sup> studio.

There are files added automatically in the created project. The same file as these exists also in the

"sample" folder of FDL Type04. Therefore, using the function of IDE, Select those files from tree, and excludes from a project.

Clicks the right mouse button for the file of tree. And On the [Settings] screen displayed by the

- "Properties", put a check mark to [Exclude resource from build] and exclude a target file (target folder).
- (Exclusion of a folder is also possible)
- CC-RL: Target files are "hdwinit.asm" in a [project name]/generate folder, and [project name] .c ("FDLType04\_PJ01.c") in a [project name]/src folder.
- LLVM: Target files are "linker\_script.ld" in a [project name]/generate folder, and [project name] .c
- ("FDLType04\_PJ01.c") in a [project name]/src folder.



Figure 6-16. File Exclusion Example



#### (2) Specifying the library file

- CC-RL: Click the right mouse button for the project in a tree, and select "Properties". In the "Add file" window that appears by clicking the "+" button to the right of "Relocatable files, object files, and library files" on the "C/C++ Build" [Settings] – "Linker" [Input] screen, change the [Format] to "library", and register the path to the data flash library file "pfdl.lib". (Figure 6-17)

Properties for FDLType04_PJ01					
type filter text	Settings		<	⇔ -	
<ul> <li>&gt; Resource</li> <li>Builders</li> <li>C/C++ Build     <li>Build Variables     <li>Environment</li> <li>Logging</li> </li></li></ul>	S S Common     S Compiler     S S Assembler     S S Linker     S Input     Buput     B Advanced	<ul> <li>Use standard/mathematical libraries (-library)</li> <li>Use C99 edition libraries (-library)</li> <li>Check memory smashing on releasing memory (-library)</li> <li>Use runtime libraries (-library)</li> <li>Relocatable files, object files and library files (-input/-library/-binary)</li> </ul>	<b>6 8</b> §	<mark>화</mark> 상태	
Settings Stack Analysis Tool Chain Editor	🖉 List 🖄 Optimization	"\${workspace_loc:/\${ProjName}/src/lib/pfdl.lib}"			

Figure 6-17 (a). Specifying the Library File (CC-RL)

📴 Add file	X
Format:	library
File name:	\${workspace_loc:/\${ProjName}/src/lib/pfdl.lib}
	Workspace File system
Module name:	
Section name:	
Boundary alignment:	1 ~
Section attribute:	None v
Symbol name:	
	OK Cancel

Figure 6-17 (b). Specifying the Library File (CC-RL)

 LLVM: Click the right mouse button for the project in a tree, and select "Properties". Register the file path of the data flash library file "libpfdl.a" in the "Additional input files" field on the screen displayed in "C/C++ Build" [Settings] – "Linker" [Source].

Properties for FDLType	4_PJ01 — 🗆 X
type filter text	Settings $\diamondsuit \checkmark \checkmark \ $
<ul> <li>&gt; Resource Builders</li> <li>&gt; C/C++ Build Build Variables Environment</li> <li>Logaina Settings Tool Chain Editor</li> <li>&gt; C/C++ General Project Natures Project References Refactoring History Renesas QE Run/Debug Settings</li> </ul>	Configuration:       HardwareDebug [Active]       Manage Configurations.         Tool Settings       Toolchain       Device       Build Steps       Build Artifact       Binary Parsers       "1         CPU       Optimization       Entry point:       -WI,-e_PowerON_Reset       Inker script       Source       Source       Source       Source       Source       Source       Achives       Miscellaneous       Additional input files       Source
	Solution of the second se

Figure 6-18. Specifying the Library File (LLVM)

(3) Specifying the linker script file (only when the LLVM compiler is used)

Click the right mouse button for the project in a tree, and select "Properties". Register the file path of the linker script file ".ld" in the "Linker script" field on the screen displayed in "C/C++ Build" [Settings] – "Linker" [Source].

Here, select the file path of "r\_pfdl\_sample\_c.ld" prepared for the FDL Type04.

Properties for FDLType04	_PJ01		— 🗆 X
type filter text	Settings		
<ul> <li>Resource</li> <li>Builders</li> <li>C/C++ Build</li> <li>Build Variables</li> </ul>	Configuration: Hardware	Debug [Active]	V Manage Configurations.
Environment Logging Settings Tool Chain Editor > C/C++ General Project Natures Project References Refactoring History Renesas QE Run/Debug Settings	<ul> <li>Tool Settings Toolcl</li> <li>CPU</li> <li>Optimization</li> <li>Debug</li> <li>Warnings</li> <li>Library Generator</li> <li>Compiler</li> <li>Socompiler</li> <li>Socompiler</li> <li>Source</li> <li>Archives</li> </ul>	hain Sevice PowerON_Reset Entry point: -WI,-e_PowerON_Reset Linker script  \$\mathbf{script} \$\mathbf{script}\text{ProjName}/src/r_pfdI_sample_1 \$\mathbf{script} \$\mathbf{scrip}	- 

Figure 6-19. Specifying the Linker Script File

Note: Refer to each reference manual of LLVM about the descriptive content of linker script file, and the details of the description method.



#### (4) Building

Right-click on the [Project] in the e<sup>2</sup> studio project tree and select "Build Project" to build the project.

#### 6.3.3 Building an Assembly-Language Program

(1) Specifying the source and include files

Specifying the data flash library file in the created project.

- CC-RL: Register user program file ("xxxxx.asm"), the data flash library files "pfdl.inc" and "pfdl.lib" in the "src" folder output by e<sup>2</sup> studio. (Figure 6-20)





- LLVM: Register user program file ("xxxxx.S"), the data flash library files "pfdl\_asm.h", "pfdl.lib" and "r pfdl sample c.ld" in the "src" folder output by e<sup>2</sup> studio. (Figure 6-21)



Figure 6-21. Specifying the Source and Include Files (LLVM)



Exclusion of the file automatically added by the function of IDE.

- There are files added automatically in the created project. The same file as these exists also in the
- "sample" folder of FDL Type04. Therefore, using the function of IDE, Select those files from tree, and excludes from a project.
- Clicks the right mouse button for the file of tree. And On the [Settings] screen displayed by the
- "Properties", put a check mark to [Exclude resource from build] and exclude a target file (target folder). (Exclusion of a folder is also possible)
- CC-RL: Target file is [project name] .c ("FDLType04\_PJ01.c") in a [project name]/src folder.
- LLVM: Target files are linker\_script.ld in a [project name]/generate folder, and [project name] .c ("FDLType04\_PJ01.c") in a [project name]/src folder.

#### (2) Specifying the library file

- CC-RL: Click the right mouse button for the project in a tree, and select "Properties". In the "Add file" window that appears by clicking the "+" button to the right of "Relocatable files, object files, and library files" on the "C/C++ Build" [Settings] – "Linker" [Input] screen, change the [Format] to "library", and register the path to the data flash library file "pfdl.lib". (Figure 6-22)

type filter text       Settings       Image: Common state of the state of	Properties for FDLType04_PJ01					
Builders       > So Common       ✓ Use standard/mathematical libraries (-library)         V C/C++ Build       > So Compiler       Use C99 edition libraries (-library)         Build Variables       > So Assembler       □ Check memory smashing on releasing memory (-library)         Environment       > So Linker       ✓ Use runtime libraries (-library)         Logging       > Mathematical libraries (-library)         Settings       > Advanced         Stack Analysis       > List	type filter text	Settings			<b>⇔</b> -	
	Builders C/C++ Build Build Variables Environment Logging Settings	<ul> <li>S Compiler</li> <li>S Assembler</li> <li>S Linker</li> <li>Input</li> <li>Advanced</li> <li>List</li> </ul>	<ul> <li>☐ Use C99 edition libraries (-library)</li> <li>☐ Check memory smashing on releasing memory (-library)</li> <li>☑ Use runtime libraries (-library)</li> <li>Relocatable files, object files and library files (-input/-library/-binary)</li> </ul>			

Figure 6-22 (a). Specifying the Library File (CC-RL)

📴 Add file		X
Format:	library	~
File name:	{workspace_loc:/{{ProjName}/src/lib/pfdl.lib}	
		Workspace File system
Module name:		
Section name:		
Boundary alignment:	1	~
Section attribute:	None	~
Symbol name:		
		OK Cancel

Figure 6-22 (b). Specifying the Library File (CC-RL)

 LLVM: Click the right mouse button for the project in a tree, and select "Properties". Register the file path of the data flash library file "libpfdl.a" in the "Additional input files" field on the screen displayed in "C/C++ Build" [Settings] – "Linker" [Source]. (Figure 6-23)

	01				$\times$
type filter text So	ettings		<	⇔ - ⇒	<b>₩</b> 8
Build Variables Environment	Configuration: HardwareD Tool Settings Toolcf CPU CPU CPU CPU CPU CPU CPU CPU	Debug [Active] Device PowerON_Reset Entry point: -WI,-e_PowerON_Reset Linker script  \${workspace_loc:/\${ProjName}/src/r_pfdl_sample_ Additional input files  \${workspace_loc:/\${ProjName}/src/lib/libpfdl.a}	Binary Par	e Configura sers 71	

Figure 6-23. Specifying the Library File (LLVM)

(3) Specifying the linker script file (only when the LLVM compiler is used)

Click the right mouse button for the project in a tree, and select "Properties". Register the file path of the linker script file ".ld" in the "Linker script" field on the screen displayed in "C/C++ Build" [Settings] – "Linker" [Source]. Here, select the file path of "r\_pfdl\_sample\_c.ld" prepared for the FDL Type04. (Figure 6-24)

Properties for FDLType0	4_PJ01 — 🗆 X
type filter text	Settings $\Leftrightarrow \checkmark \Leftrightarrow \checkmark \S$
<ul> <li>Resource Builders</li> <li>C/C++ Build Build Variables Environment</li> </ul>	Configuration: HardwareDebug [Active]
Settings Tool Chain Editor > C/C++ General Project Natures Project References Refactoring History Renesas QE Run/Debug Settings	<ul> <li>Tool Settings Toolchain Device Puild Steps Puild Artifact Binary Parsers "</li> <li>Entry point: -WI,-e_PowerON_Reset</li> <li>CPU</li> <li>Cpu Entry point: -WI,-e_PowerON_Reset</li> <li>Linker script</li> <li>Compiler</li> <li>Source</li> <li>Archives</li> </ul>

Figure 6-24. Specifying the Linker Script File

Note: Refer to each reference manual of LLVM about the descriptive content of linker script file, and the details of the description method.

the description method.



## (4) Building

Right-click on the [Project] in the e<sup>2</sup> studio project tree and select "Build Project" to build the project.

#### 6.4 Notes at Build

#### 6.4.1 When the CA78K0R Compiler is Used

(1) When the on-chip debugging function is in use

After the on-chip debugging function is enabled in the CS+, building a program generates the following type of error.

RA78K0R error E3212: Default segment can't allocate to memory - ignored Segment '??OCDROM' at xxxxxH-200H

This error occurs when the segment for the monitor area (OCDROM) used by the on-chip debugging function cannot be allocated. Therefore, to avoid this error, add the following code to the link directive file (\*.dr) embedded in the project and prepare a separate area for allocating the segment.

MEMORY OCD\_ROM : ( 0xxxxxH, 00200H )

Remarks: 1. xxxxx indicates the start address at which the error has occurred.

2. The area name "OCD\_ROM" is an example of the notation.



## 6.4.2 When the CC-RL Compiler is Used

(1) When the on-chip debugging function is in use

After the on-chip debugging function is enabled in the CS+, building a program may generate the following type of error.

E0562321:Section ".monitor2" overlaps section "xxxxx"

This error occurs when the section for the monitor area (OCDROM) used by the on-chip debugging function cannot be allocated. Therefore, to avoid this error, right click the CC-RL (Build Tool) node (1) in the CS+ Project Tree window, select Property to open the CC-RL Property panel (2), and select the Link Options tab (3). In the Section category (4), modify the setting for Section start address (5) so that no other areas overlap the area where the section for the on-chip debugger monitor is allocated (monitor2: the initial address range is 0xFE00 to 0xFFFF in R5F100LE). (See Figure 6-25)

For details of the section settings, refer to the CC-RL Compiler User's Manual.

Remark 1. xxxxx: Indicates the section name.

Project Tree 7 ×	Property			<b>→</b> X
2 @ 2 2 2	CC-RL Property		2	+ – ۹
<u>CC EN FDL SampleTest (Project)</u>				<u>^</u>
	> Device			
🛺 🎤 Pin Configurator (Design Tool)	> Output Code			
Code Generator (Derign Tool)	> List			_
	Variables/functions information     Section			
Program Analyzer (Analyze Tool)	Layout sections automatically Yes(-AUTO_SECTION_LAYOUT)			
	5 Section start address .const,.RLIB,.SLIB,.textf,.constf,.c			_COD/02
Build tool generated files	Section that outputs external defined symbols to the file Section that outputs external defined sym	bols to the	file[0]	
estart.asm	ROM to RAM mapped section     ROM to RAM mapped section[2]     Verify			
estatusini stkinit.asm	> Verny > Message			
iodefine.h	> Others			
r_pfdl_sample_c.c				~
	Section			
pfdLh	2			
plath	<u> </u>			
placypes.n	Common Options / Compile Options / Assemble Options / Link Options / Hex Output Op	tions 🖌 I/	0 Header	File G / ₹
	Error List			<b>д X</b>
	😵 1 Errors 🔒 0 Warnings 🚯 0 Messages 🛛 🏹 🗸			
	Number Message	File	Line	Project
	E0562321 E0562321:Section ".monitor2" overlaps section ".text"			CC_EN_FDL
	🚵 Output 🐖 Error List			

Figure 6-25. Modifying the Section Allocation



# Chapter 7 How to Debug a Program

For details on how to perform debugging by using IECUBE or the on-chip debugging emulator E1, E2, E2 emulator Lite or E20, see the following document:

Title		
CubeSuite+ Integrated Development Environment User's Manual: RL78 Debug[CS+ for CA,CX] <sup>Note</sup>		
CS+ Integrated Development Environment User's Manual: RL78 Debug Tool[CS+ for CC] Note		
e <sup>2</sup> studio Integrated Development Environment User's Manual: Getting Started Guide		

Note: You can download this document from the "CS+ Integrated Development Environment" or "e<sup>2</sup> studio Integrated Development Environment" page of the Renesas Electronics website.

## 7.1 Notes on Debugging

- (1) When a command of the Data Flash Library Type04 Ver.1.05 is executed using the on-chip debugging emulator E1, E2, E2 emulator Lite or E20 in a CS+ whose version is earlier than Ver.1.01, do not execute a break until completion of the sequencer has been confirmed. Otherwise, the sequencer does not operate correctly.
- (2) The data flash library cannot be debugged by a simulator. To perform debugging, either use the on-chip debugging function of the RL78 microcontroller or prepare the IECUBE.



# Chapter 8 Sample Program

The attached sample program (r\_pfdl\_sample\_c.c) is provided to enable the usage method of the Data Flash Library Type04 to be easily confirmed on the QB-R5F100LE-TB boards with R5F100LEA (RL78/G13) as the target microcontrollers. The sample program is just a reference example and the user program does not have to be created to match the sample program. The sample program should be used as a simple program to confirm operation.

- The link directive file (r\_pfdl\_sample\_c.dr) for the sample program for the CA78K0R compiler has a purpose to specify that a stack or data buffer used by the sample program is not allocated to an area where allocation is prohibited <sup>Note1</sup>. When using the sample program, this file should also be embedded with the sample program.<sup>Note2, 3</sup>
- The sample program for the CC-RL compiler, should be allocated appropriately in the section category on the "Link Options" tabbed page in the CS+ window, or on the "Linker" [Section] page in the e<sup>2</sup> studio, so that a stack or data buffer used by the sample program is not allocated to an area where allocation is prohibited <sup>Note1</sup>.
- The linker script file (r\_pfdl\_sample\_c.ld) for the sample program for the LLVM compiler has a purpose to specify that a stack or data buffer used by the sample program is not allocated to an area where allocation is prohibited <sup>Note1</sup>. When using the sample program, this file should also be embedded with the sample program.<sup>Note3</sup>

Notes: 1. For details, refer to chapter "2.2 Software Environment" in the user's manual.

- 2. In the supplied link directive file, the RAM area size is set to 2 Kbytes. Even when the target microcontroller has 2 Kbytes or larger RAM, the sample program (r\_pfdl\_sample\_c.c) can be used for building without modifying the defined area setting.
- 3. The data in usage may be placed at an unintended area depending on how the environment in use or the program is changed. After an execution module is generated, the map file and allocation state of programs or data must be confirmed. For the definition method and allocation conditions of each code or data, refer to the user's manual of the compiler used.

#### 8.1 Initial Settings of the Sample Program

The sample program operates with the following initial settings. When these settings need to be changed, modify the sample program.

- CPU operating frequency: High-speed on-chip oscillator 32 MHz
- Voltage mode: High-speed mode



## 8.2 Settings of Option Byte and On-Chip Debugging

(1) When using CA78K0R or CC-RL compiler with the CS+

When performing on-chip debug, set "Set enable/disable on-chip debug by link option" to "Yes" and specify "84" for "Option byte values for OCD". For the CC-RL compiler, set "Set debug monitor area" to "Yes".

The sample program normally operates by setting the high-speed on-chip oscillator at 32 MHz.

After setting "Set user option byte" to "Yes" on the "Link Options" tabbed page, specify "xxxxE8" for "User option byte value" and set the high-speed on-chip oscillator at 32 MHz.

CA78K0R Property		<b>₽ −</b> +	
> Debug Information		-	
> Input File			
> Output File			
> Library			
Device			
Set enable/disable on-chip debug by link option	Yes(-go)	-	
Option byte values for OCD	HEX 84	=	
Debug monitor area start address	FE00		
Debug monitor area size[byte]	512		
Set user option byte	Yes(-gb)		
User option byte value			
Specify mirror area Set flash start address	MAA=0(-mi0) No		
Boot area load module file name	INO		
Control allocation to self RAM area	No		
Message	NO		
> Stack			
Set enable/disable on-chip debug by link option Specify this option, to set a value of the on-chip debug function and to secure area of the debug monitor. This option corresponds to the -go option.			

Figure 8-1 (a). Setting of Option Byte when Using the CS+ (CA78K0R Compiler)

	Property	- ×
∕∿	CC-RL Property	<b>a p</b> -+
⊳	Debug Information	
⊳	Optimization	
⊳	Input File	
⊳	Output File	
⊳	Library	
⊿	Device	
-	Set enable/disable on-chip debug by link option	Yes(-OCDBG)
	Option byte values for OCD	HEX 84
	Set debug monitor area	Yes(Specify address range)(-DEBUG_MONITOR= <address range="">)</address>
	Range of debug monitor area	FE00-FFFF
	Set user option byte	Yes(-USER_OPT_BYTE)
	User option byte value	HEX EFFFE8
	Control allocation to self RAM area	No
⊳	Output Code	
⊳	List	
⊳	Variables/functions information	
⊳	Section	
	Verify	
	Message	
⊳	Others	
De	vice	
/_c	ommon Options / Compile Options / Assemb	leOptions Link Options Hex Output Optio / I/O Header File G / 🔻

Figure 8-1 (b). Setting of Option Byte when Using the CS+ (CC-RL Compiler)

#### (2) When using the CC-RL compiler with the e<sup>2</sup> studio

Select "C/C++ Build" [Settings] - "Linker" [Device]. And set device items on the displayed screen.

When performing on-chip debug, put a check mark to "Set enable/disable on-chip debug by link option" and specify "84" for "On-chip debug control value". Put a check mark to "Secure memory area of OCD monitor".

The sample program normally operates by setting the high-speed on-chip oscillator at 32 MHz. Put a check mark to "Set user option byte" on the "Tool Settings" tabbed page, specify "xxxxE8" for "User option byte value" and set the high-speed on-chip oscillator at 32 MHz.

type filter text	Settings		
<ul> <li>&gt; Resource Builders</li> <li>~ C/C++ Build Build Variables Environment Logging</li> <li>Settings</li> </ul>	Configuration: HardwareDu	ebug [ Active ] Device 🎤 Build Steps 🙅 Build Artifact 🖬 Binary Parsers 😣 E	V Manag
Stack Analysis Tool Chain Editor > C/C++ General Project Natures Project References	> 🐯 Common > 🐯 Compiler > 🐯 Assembler ~ 🐯 Linker > 🎉 Input	Security ID value (-security_id) Serial Programming Security ID value (-flash_security_id) Reserve working memory for RRM/DMM function (-rrm) Start address area (-rrm= <value>)</value>	0
Renesas QE Run/Debug Settings	<ul> <li>List</li> <li>Optimization</li> <li>Section</li> <li>Device</li> <li>Output</li> <li>Miscellaneous</li> </ul>	Secure memory area of OCD monitor (-debug_monitor) Memory area (-debug_monitor= <start address="">-<end address="">)     Set user option byte (-user_opt_byte) User option byte value (-user_opt_byte=<value>)     Set enable/disable on-chip debug by link option (-ocdbg)</value></end></start>	OFE00-OFFFF FFFFE8
	> 🛞 Converter	On-chip debug control value (-ocdbg= <value>) Set security option byte (-security opt byte)</value>	84

Figure 8-2. Setting of Option Byte when Using the e<sup>2</sup> studio (CC-RL)

(3) When using LLVM compiler with the e<sup>2</sup> studio

The device item settings are set in the "vects.c" file. In the "vects.c" file provided in the sample program, the option byte value and user option byte value are set in "Option Bytes" as follows.

The sample program normally operates by setting the high-speed on-chip oscillator at 32 MHz. Therefore, set the user option byte value "xxxxe8" and the on-chip debug option byte value in the "Option\_Bytes" of the "vects.c" file as follows:

#### [The example for RL78/G13]

```
"0xff, 0xff, 0xe8, 0x84" (WDT Enable, LVD reset mode, HS mode /32MHz, Enable on-chip debug operation)
```



Note: Be sure to confirm the contents of "User option byte" of the chapter of "Option Bytes" and "On-chip debug option byte" by the user's manual of a target device. And describe the set value used with user application.



## 8.3 Compilation Switch for the C-Language Sample Program

The sample program has a compilation switch as shown below. This compilation switch is used to turn on the LED to confirm operation on the QB-R5F100LE-TB board. To use this, modify "#if 0" to "#if 1" so that the #define declaration for the target CPU board becomes valid.

/**************************************	
/* Symbol for program switch of sample program */	
/**************************************	
/* Can be set when QB-R5F100LE-TB is used alone */	
#if 0	Can be modified to #if 1
#defineQB_R5F100LE_TB	when QB-R5F100LE-TB is used
/* Other boards */	
#else	
#defineNON_TARGET	
#endif	



## 8.4 Defining the Internal RAM Area

## 8.4.1 When the CA78K0R Compiler is Used

When the CA78K0R compiler is used, the entire internal RAM area is automatically defined as an area with the name "RAM" in the initial state. Unless otherwise stated in the link directive file, the stack and data buffers are to be allocated to this area <sup>Note</sup>. However, in this case, the stack and data buffers would be allocated by default to an area (FFE20H to FFEFFH in self-RAM) for which use by the data flash library is prohibited, so the program may not run correctly.

In the attached link directive file for the sample program, as a solution, re-define the area with the name "RAM" so that it does not include the above area, ensuring that stack and so on are not allocated to the area for which usage is prohibited.

MEMORY RAM :(0FEF88H, 000E98H)

The above statement redefines the area with the name "RAM" to be the E98H bytes area starting from the address FEF88H (FEF88H to FFE1FH)<sup>Note</sup>. This prevents attempted use of the area which the data flash library is prohibited to use by excluding the prohibited portion from the area with the name "RAM".

However, if this is the only change setting that is explicitly made, the area from FFE20H to FFEFFH is also unusable for any other purpose. Accordingly, separately add the following definition. No particular restrictions apply to the name of this area.

MEMORY SADDR\_RAM:(0FFE20H, 0000E0H)

If there is a self-RAM area, automatic allocation of variables to this area can be restricted by defining its range as an area with the name "SELFRAM".

MEMORY SELFRAM :(0FEF00H, 000088H)

An example of the settings for an RL78/G13 (the product with 4 Kbytes of RAM and 64 Kbytes of ROM) is given below.



Note: The CA78K0R linker allocates data with a non-specified destination for allocation (segment types DSEG and BSEG) to the internal RAM area according to the re-allocation attribute of the data. Accordingly, specific data may not be allocated to the area with the name "RAM" in some situations.

For details on the methods of defining and allocating the individual categories of data, refer to the user's manual for CS+.

Reference to the map file (\*.map) generated at the time of building is required to confirm the state of allocation.



#### 8.4.2 When the CC-RL Compiler is Used

(1) Adding the include path

In CS+ and e<sup>2</sup> studio, no include path is specified in the initial state: The include path for the header files used by the data flash library need to be added. The data flash library uses header files "pfdl.h", "pfdl\_types.h", and "iodefine.h" (this file is automatically generated by CS+ and e<sup>2</sup> studio).

- In CS+, add the include path where each file resides in [Compile Options] [Preprocessing] [Additional Include Path].
- In e<sup>2</sup> studio, in the "Properties" window, add the include path where each file exists in the "Include file directories(-I)" field on the screen displayed by "C/C++ Build" [Settings] "Compiler" [Source].

#### (2) Defining sections

The sections used for the ROM and RAM areas need to be defined.

In this example, the PFDL\_COD section that are necessary for operation of the sample program is added.

- Sections can be defined in the Section category on the Link Options tabbed page in the CS+ window. When the Layout sections automatically property is set to No, select the Section start address property to open the Section Settings dialog box and add the sections necessary for the data flash library to the ROM area (Figure 8-3).

Section Settings			×			
Address	Section		<u>A</u> dd			
0x02000	.const		Mar differ			
	.text		<u>M</u> odify		Add Section	×
	.RLIB		New <u>O</u> verlay			
	.SLIB		Remove	N	Section name:	
	.textf				PFDL_COD	$\sim$
	.constf		<u>U</u> p <u>D</u> own			
	.data				OK Cancel <u>H</u> elp	
	.sdata					:
0xFEF00	.dataR					
	.bss				Enter the section name (PFDL_COD) and	
0xFFE20	.sdataR		Import		click the OK button.	
	.sbss		Export			
	ОК	Cancel	<u>H</u> elp			
Section Settings			×		$\mathbf{N}$	
Address	Section		<u>A</u> dd	M /		
0x02000	.const		Modify	$\checkmark$		
	.text					
	.RLIB		New Overlay			
	.SLIB		Remove			
	.textf					
	.constf		<u>U</u> p <u>D</u> own			
	.data			After	adding all necessary sections, click the OK	
	.sdata				n to close the Section Settings dialog box.	
	PFDL_COD					
0xFEF00	.dataR					
	.bss					
0xFFE20	.sdataR		Import			
	.sbss		Export			
		1				
	ОК	Cancel	<u>H</u> elp			
				•		

Figure 8-3. Example of Section Settings for the Data Flash Library when Using CS+ (ROM Area)



- Setting of the section items on e<sup>2</sup> studio inputs in the "Properties" window. Select "C/C++ Build" [Setting] - "Linker" [Section]. And set section items on the displayed screen. Remove a check mark to [Layout sections automatically(-auto\_section\_layout)]. Press the " … " button of the right-hand side which sections are displaying, and a "Section Viewer" screen is displayed and add the sections necessary for the data flash library to the ROM area (Figure 8-4).

Address	Section Name	
0x00002000	.const	
	.text	
	.data	Add Section
	.sdata	New Overlay
	.RLIB	Remove Section
	.SLIB	Move Up
	.textf	
	.constf	Move Down
	PFDL_COD	Import
0x000FEF00	.dataR	Export
	.bss	
0x000FFE20	.sdataR	
	.sbss	
Override Lir	ker Script	
		Browse
	Re-Apply	

Figure 8-4. Example of Section Settings for the Data Flash Library when Using e<sup>2</sup> studio (ROM Area)

#### (3) Allocating the Self-RAM Area

In the initial state of the section settings in CS+ for the CC-RL compiler, the user RAM area is allocated at the beginning of the internal RAM area (from address FEF00H for R5F100LEA, which is the target microcontroller of the sample program). However, in R5F100LEA, the data flash library uses the address range from 0xFEF00 to 0xFEF87 as the self-RAM area. Therefore, the user RAM area must be allocated outside this area. In this example, the user data start address 0xFEF00 is changed to 0xFEF88.



Figure 8-5. Example of Changing the User RAM Area Allocation when Use CS+ (RAM Area)

		×
Section Viewer		
Address	Section Name	
0x00002000	.const	
	.text	
	.data	Add Section
	.sdata	New Overlay
	.RLIB	Remove Section
	.SLIB	Move Up
	.textf	•
	.constf	Move Down
	PFDL_COD	Import
0x000FEF88	.dataR	Export
	.bss	
0x000FFE20	.sdataR	
	.sbss	
Override Lin	nker Script	
	inter benjet	
		Browse
	Re-Apply	
		OK Cancel

Figure 8-6. Example of Changing the User RAM Area Allocation when Use e<sup>2</sup> studio (RAM Area)

Note: The sections including the user-specified sections are automatically re-allocated when the Layout sections automatically property is temporarily set to No, the user RAM allocation is changed, and then the property is again set to Yes. In this case, sections may be allocated to areas that are not specified by the user; that is, data may be placed in unintended areas. Be sure to refer to the map file to check if the software resources (especially RAM data) used by the data flash library are placed in relocatable areas.

#### 8.4.3 When the LLVM Compiler is Used

#### (1) Adding the include path

In e<sup>2</sup> studio, no include path is specified in the initial state: The include path for the header files used by the data flash library need to be added. The data flash library uses header files "pfdl.h", "pfdl\_types.h", and "iodefine.h" and "iodefine\_ext.h" (this file is automatically generated by e<sup>2</sup> studio).

In e<sup>2</sup> studio, in the "Properties" window, add the include path where each file exists in the "Include file directories(-I)" field on the screen displayed by "C/C++ Build" [Settings] – "Compiler" [Includes].



#### (2) Allocating the Self-RAM area

The LLVM compiler describes the link settings to be performed in the build in a linker script file (\*.ld).

In the linker script file (linker\_script.ld) output from e<sup>2</sup> studio, the built-in RAM area is defined as "RAM" section. In addition, the software resources used by the data flash library are defined as an area called "SELFRAM" section. (Only for devices that require "Self-RAM" area)

In the linker script file "r\_pfdl\_sample\_c.ld" included with the sample program, the "RAM" section and "SELFRAM" section are defined so that they do not overlap.

Note: The "r\_pfdl\_sample\_c.ld" provided in the sample program is prepared on the assumption that R5F100LE will be used. When using other devices, please check the Self-RAM list and modify it according to the device. Refer to each reference manual of LLVM about the descriptive content of linker script file (\*.ld), and the details of the description method.



All trademarks and registered trademarks are the property of their respective owners.



#### Notice

- 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.
- Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims 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, drawings, charts, programs, algorithms, and 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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
- 5. You shall not alter, modify, copy, or reverse engineer 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, copying or reverse engineering.
- 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; 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.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; 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 any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.

- 7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
- 8. When using 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 dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
- 9. 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. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a 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. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using 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.
- 11. 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 comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
- 12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
- This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
   Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.
- (Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
- (Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

## **Corporate Headquarters**

TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

#### Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

## **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: <a href="http://www.renesas.com/contact/">www.renesas.com/contact/</a>.