RL78

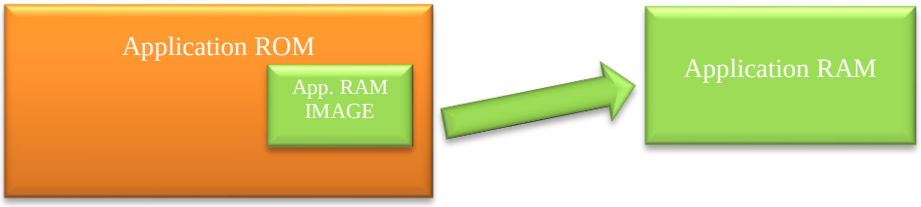
Application execution from RAM

R01AN2080ED0000
Rev.1.00
April 01, 2014

Introduction

Today, a lot of applications require the code execution from RAM like for example due to safety reasons or e.g. in case of bootloader for flash self-programming. Usually such an application have to be divided into two parts

- The main part that will be executed from flash. This part represents the main application like e.g. a boot-loader.
- Image within the main application which will be copied during the runtime to the RAM.



Considering the above described software concept there are several issues to be considered for development:

- How to create an application which will be stored in flash, but executed from RAM during the runtime?
- How to debug an application which is built within a different project?

This document will help you to set-up the above described two projects based on the IAR environment. The sample application related to this "Application Note" can be downloaded here:

http://www.renesas.eu/update?oc=QB-R5F10BMG-TB

For details please refer to "Chapter 4 Sample application".

Contents

1. Common resources 3

2. Create an application for execution from RAM 4

2.1 Memory map 4

2.1.1 Scatter loading 5

2.1.2 Common linker file for shared information 5

2.2 CSTARTUP 6

2.3 Replacement of reset vector 8

2.4 General project options 9

2.5 Instruction fetch and pre-fetch in RAM 9

2.6 Linker configuration 10

2.6.1 Program entry 10

2.6.2 Output files 11

3. Create the main application (located in ROM) 12

3.1 Linker configuration 12

3.1.1 Memory mapping 12

3.1.2 Add binary image 12

3.1.3 Symbol definition for image access 13

3.2 Debugger configuration 13

3.3 Access the image from C code 14

3.3.1 Copy image segments to RAM 14

3.3.2 Execute the RAM application function 14

3.4 Debugging 14

3.4.1 Software breakpoints usage within RAM 14

4. Sample application 15

4.1 Run the sample application 15

5. Website and Support 16

Revision History 1

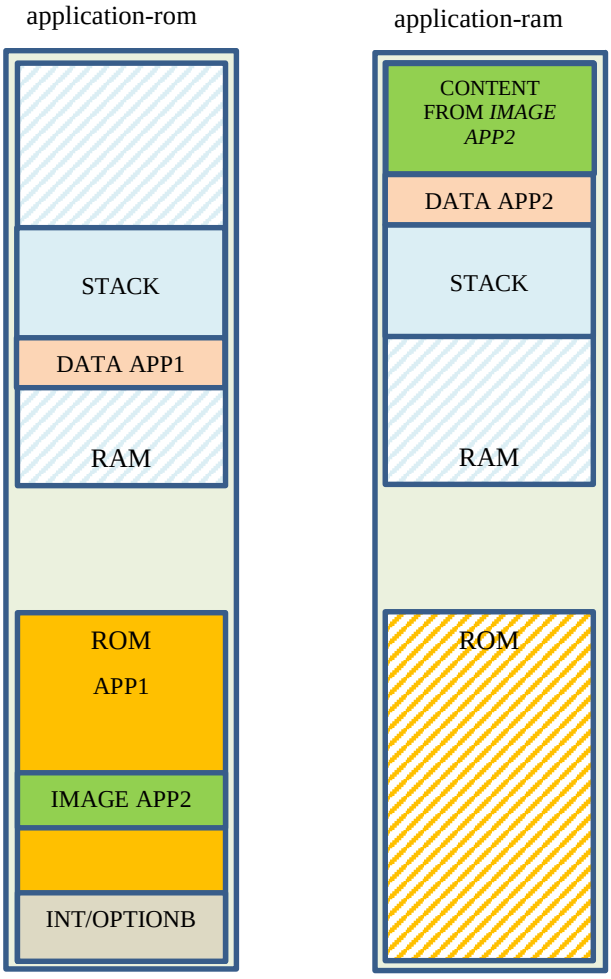
General Precautions in the Handling of MPU/MCU Products 2

1. Common resources

The separation of two applications requires a concept on how to handle common resources like e.g. RAM.

- RAM
  - Stack and heap definition
    - one stack and heap for both applications
    - separate stack for each application
  - SADDR and data RAM should be partitioned according to the application requirements
- OPTION byte
  - The OPTION BYTE will be define within one of the applications. All configurations within this option byte like e.g. watchdog shall be valid for both applications, because it cannot be changed during the run-time.
- Interrupts
  - By using the self-programming library there is a possibility to use a common interrupt service routine for all interrupts within RAM.

Following figure illustrates the memory mapping of both applications. Here the stack is used as common resource for both applications.

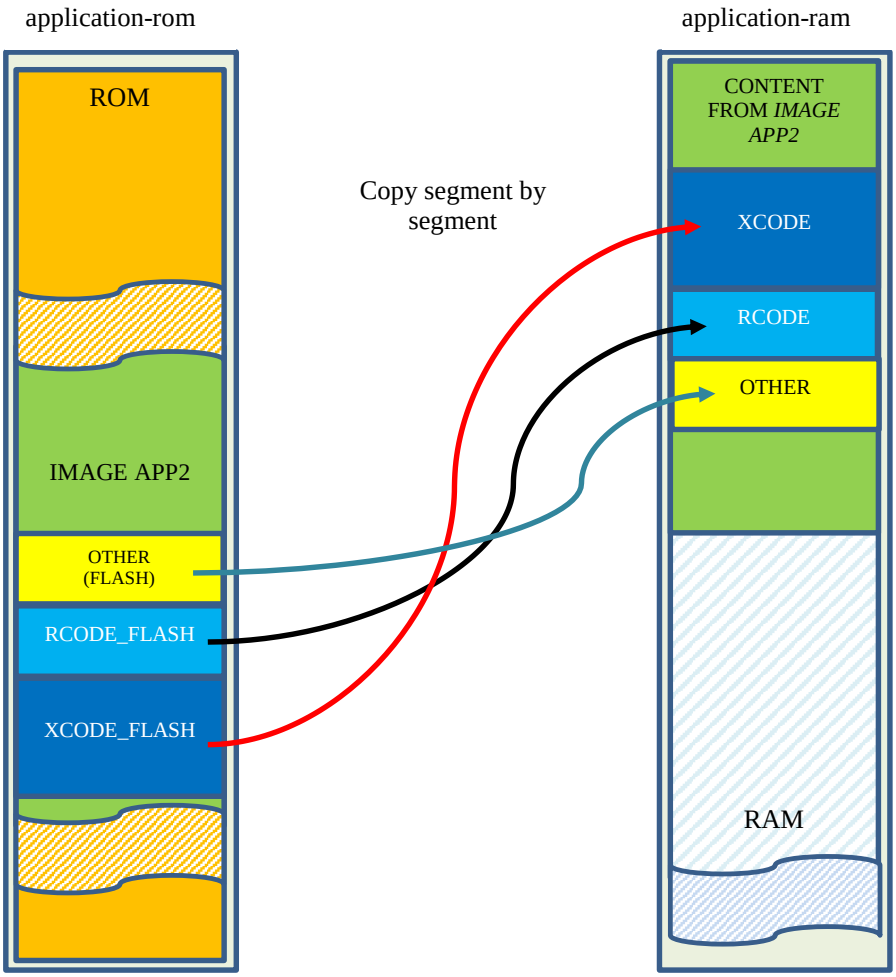


2. Create an application for execution from RAM

This chapter describes how to set-up an application which will be loaded via an image from the main application flash and executed from RAM. Please note that the below described configuration is based on one common Stack for both applications.

2.1 Memory map

Due to the fact that the application within RAM is a standalone application all the standard segments used by the Compiler/Assembler like RCODE, XCODE, FAR_I, FAR_CONST etc. have to be defined accordingly. The difference compared to the usual application is that this application will be executed from RAM and not from flash. That means the segments are initially located within the image (flash) and will be later copied segment by segment to the RAM and executed from there. Please note that it is not possible to copy the whole image directly in case the application has more than one segment.



The reason for this is that the segment location within the image might be different to the location within RAM and therefore all segments have to be copied segment by segment. Due to the fact that the application will be executed from RAM the linker has to be informed that the segment will be placed within flash, but executed from RAM. This can be done by the linker feature “scatter loading” which will be introduced within the next chapter.

2.1.1 Scatter loading

By using the linker “scatter loading” feature it is possible to place a segment within the flash, but execute it later from RAM. For this we need to define two segments:

```
-Z(CODE)XCODE=0xFF300-0xFF400
-Z(CODE)XCODE_FLASH=0x100-0x200
```

The XCODE segment is a standard segment used by the Compiler for far code. It will be mapped to RAM. All the code will be compiled according to the defined address range. The XCODE_FLASH segment is just a segment which will be placed into flash and contains the image. After the definition of this two segments we can define the scatter loading feature which defines that the content of the segment XCODE_FLASH will be later executed from the segment XCODE in RAM. This will be done by the following instruction.

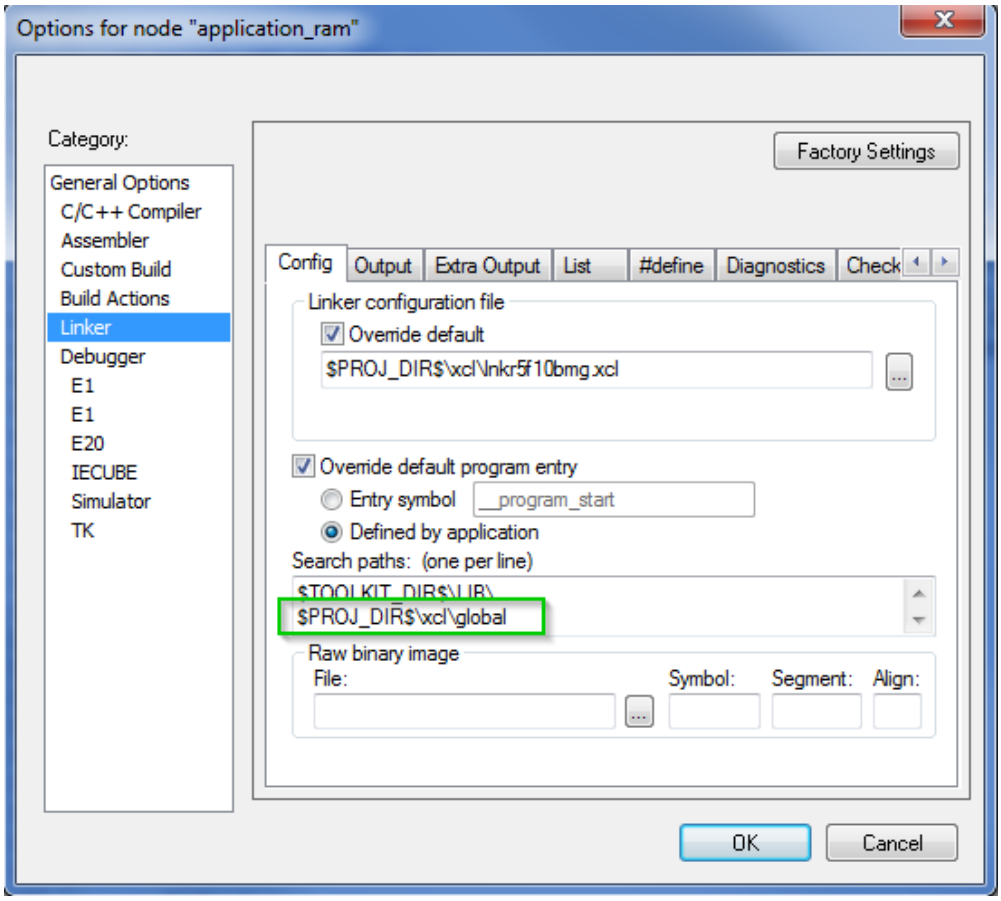
```
-QXCODE=XCODE_FLASH
```

2.1.2 Common linker file for shared information

As described in Chapter ”2.1 Memory map” the main application has to copy the code segment by segment to RAM. However, due to the fact that only the binary image is included the segment information is missing. The easiest way to solve this issue is to use a common XCL file which can be used for both application. You can include the common XCL file to your project specific XCL file by using the linker option ‘–f’ like shown below.

```
-f common.xcl
```

Please note that here the absolute path is not used. To allow the linker to find this file within the correct path please add the search path to the linker within the IDE.



This `common.xcl` file can be even used for structuring the RAM resources for example to avoid segment overlapping. Please refer to the sample application which configures all the segments based on the definition within the `common.xcl` file.

2.2 CSTARTUP

The RAM application is a complete standalone application and therefore needs to perform the CSTARTUP. However, due to the fact that the CSTARTUP is already performed by the main application, the common resources like “Stack” etc. don’t need to be re-initialized. That means we need a modified cstartup. The standard cstartup file can be copied from the IAR installation.

<IAR_INSTALL_PATH>\rl78\src\lib\cstartup.s87

Following adaptations have to be done.

- Remove stack segment placement, because the stack is already placed within the main application

```
;------------------------------------------------------------------------------
;       The stack segment.
;       The stack size is defined in the linker command file
;------------------------------------------------------------------------------

        MODULE  ?CSTARTUP

;;;;;;;;        RSEG    CSTACK:DATA:ROOT(1)
```

- Remove reset vector placement

```
;------------------------------------------------------------------------------
;       The interrupt vector segment.
;       Interrupt functions with defined vectors will reserve
;       space in this area as well as conformingly written assembly
;       language interrupt handlers
;------------------------------------------------------------------------------

;;;;;;;;        COMMON  INTVEC:CODE:ROOT(1)

;;;;;;;;        DC16    __program_start                 ; Reset vector
```

- Remove stack initialization

```
?C_STARTUP:
`@cstart`:
__program_start:
        DI

;;;;;;;;        MOV     A, #(_NEAR_CONST_LOCATION & 1)  ; Near/Far, set mirror a
;;;;;;;;        MOV1    CY, A.0
;;;;;;;;        MOV1    PMC.0, CY
;;;;;;;;
;;;;;;;;        MOVW    SP, #sfe(CSTACK)
;;;;;;;;
;;;;;;;;
        ; Init stack segment for as the generated code may sometimes
        ; access the 4th byte of a return address before it is initialized
;;;;;;;;        MOVW    HL, #sfb(CSTACK)
;;;;;;;;        MOVW    BC, #LWRD(sizeof(CSTACK))
;;;;;;;;        CMP0    C
;;;;;;;;        SKZ
;;;;;;;;        INC     B
;;;;;;;;        MOV     A, #0xCD        ; 0xCD to fool C-SPY's stack limit check
;;;;;;;;loop_s:
;;;;;;;;        MOV     [HL], A
;;;;;;;;        INCW    HL
;;;;;;;;        DEC     C
;;;;;;;;        BNZ     loop_s
;;;;;;;;        DEC     B
;;;;;;;;        BNZ     loop_s

        MOV     CS, #0
```

- Remove copy process FAR_ID → FAR_I from the cstartup.
  Background of this issue is that we would like to avoid double copy process (one during cstartup and the other by user). Usually the FAR_ID segment is located within flash and contains the initialization data for pre-initialized `__far` variables. This initialization values will be copied during the cstartup process into the FAR_I segment which is located in RAM. However, due to the fact that this application is located completely within RAM (after the user copy process) the segment copy process by cstartup can be skipped.

```
        MODULE  ?__INIT_FAR_I

        RSEG    FAR_I:DATA(0)
        RSEG    FAR_ID:DATA(0)
        RSEG    RCODE:CODE:NOROOT(0)

        PUBLIC  __INIT_FAR_I

__INIT_FAR_I:
;;;;;;;;        ; First make sure FAR_I and FAR_ID have the same size
;;;;;;;;        LIMIT   sizeof(FAR_I)-sizeof(FAR_ID),0,0,"FAR_I and FAR_ID not same size"
;;;;;;;;
;;;;;;;;        ; Sanity check
;;;;;;;;        LIMIT   (sfb(FAR_I)-sfb(FAR_ID))==0,0,0,"FAR_I and FAR_ID have same start address"
;;;;;;;;
;;;;;;;;        ; FAR_I and FAR_ID must start at the same offset in a 64k page, unless sizeof
;;;;;;;;        ; FAR_I is less than 64k, then it's enugh if both segments reside within a 64k
;;;;;;;;        ; boundary
;;;;;;;;        LIMIT   (((sfb(FAR_I)^sfb(FAR_ID)) & 0xFFFF) == 0) || ( (sizeof(FAR_I)< 0x10000) && (((sfb(FAR_I)^sfe(FAR_I)) & 0xF0000) ==
;;;;;;;;
;;;;;;;;
;;;;;;;;
;;;;;;;;        ;         LIMIT   (sfb(FAR_I)^sfb(FAR_ID)) & 0xFFFF,0,0,"FAR_I and FAR_ID must start at the same offset into a 64k page"
;;;;;;;;        MOV     ES, #BYTE3(sfb(FAR_ID))
;;;;;;;;        MOVW    HL, #LWRD(sfb(FAR_ID))
;;;;;;;;        MOV     CS, #BYTE3(sizeof(FAR_ID))    ; CS is used as counter
;;;;;;;;        MOVW    AX, #LWRD(sizeof(FAR_ID))
;;;;;;;;        MOVW    BC, AX
;;;;;;;;        CMP0    C
;;;;;;;;        SKZ
;;;;;;;;        INC     B
;;;;;;;;        CMP0    B
;;;;;;;;        SKZ
;;;;;;;;        INC     CS                            ; counter
;;;;;;;;        MOV     A, #BYTE3(sfb(FAR_I))
;;;;;;;;        MOVW    DE, #LWRD(sfb(FAR_I))
;;;;;;;;        MOV     X, A
;;;;;;;;loop:
;;;;;;;;        MOV     A, ES:[HL]
;;;;;;;;        XCH     A, X
;;;;;;;;        XCH     A, ES
;;;;;;;;        XCH     A, X
;;;;;;;;        MOV     ES:[DE], A
;;;;;;;;        XCH     A, X
;;;;;;;;        XCH     A, ES
;;;;;;;;        XCH     A, X
;;;;;;;;        INCW    HL
;;;;;;;;        MOV     A, H
;;;;;;;;        OR      A, L
;;;;;;;;        SKNZ
;;;;;;;;        INC     ES
;;;;;;;;        INCW    DE
;;;;;;;;        MOV     A, D
;;;;;;;;        OR      A, E
;;;;;;;;        SKNZ
;;;;;;;;        INC     X
;;;;;;;;        DEC     C
;;;;;;;;        BNZ     loop
;;;;;;;;        DEC     B
;;;;;;;;        BNZ     loop
;;;;;;;;        DEC     CS                            ; counter
;;;;;;;;        BNZ     loop

        ENDMOD
```

- Define the CALL to the application RAM function and whether the standard exit function should be executed at the end or not. In our application we will not perform the exit function, because the RAM application should return to the main application located in ROM.

```
;------------------------------------------------------------------------------
;       Enter main
;
;       Call the actual "main" function
;------------------------------------------------------------------------------

        MODULE  ?__MAIN_CALL

        RSEG    RCODE:CODE:NOROOT(0)

        PUBLIC  __MAIN_CALL
        PUBLIC  `@cend`             ; Renesas debugger specific

        EXTERN  ram_main
        EXTERN  exit

__MAIN_CALL:
#if defined(__FAR_MODEL__)
        CALL    F:ram_main
        RET
        ;;;;;;;;CALL    F:exit
#else
        ;;;;;;;;CALL    ram_main
        ;;;;;;;;CALL    exit
#endif

`@cend`:

;       STOP                            ; Should not return

        ENDMOD
```

2.3 Replacement of reset vector

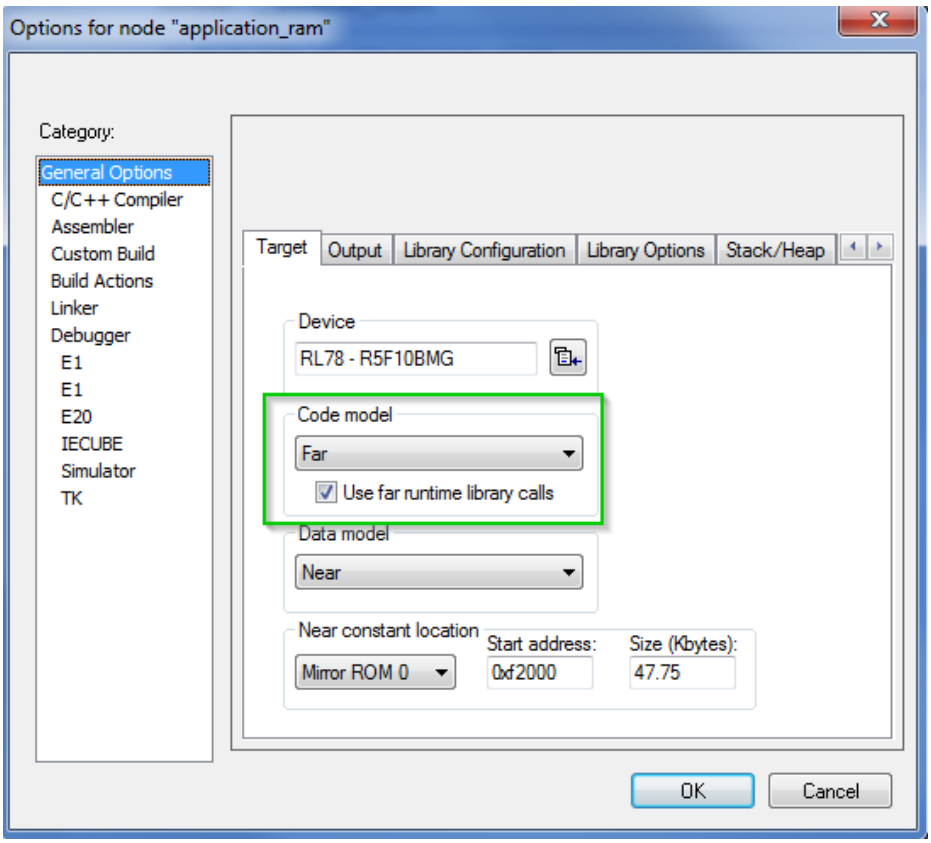
As described before the application within RAM doesn't have a reset vector. However, it is important to know from where the application starts, because the main application has to call the entry point. Usually the cstartup is the entry point for the application, but the cstartup is located within the RCODE segment which can be internally restructured, so that the address of the cstartup might be changed. For that reason it is necessary to define an address specific starting point which can be called by the main application independent of the RCODE location. The easiest way to do this is to place one branch instruction to the cstartup on a defined address.

```
;------------------------------------------------------------------------------
;       The actual startup code
;
;       Entry:  __program_start
;------------------------------------------------------------------------------
        NAME    MY_BR_INSTRUCT

        RSEG    BR_INST:CODE:ROOT
        EXTERN  __program_start
;------------------------------------------------------------------------------
;       Perform jump to the program_start
;------------------------------------------------------------------------------
        BR    F:__program_start


        END
```

2.4 General project options

Within the “General Options” menu the following settings shall be configured

- Code model = Far
  Application will be executed from RAM and therefore the code model shall be configured for “Far” execution.
- “Use far runtime library calls” feature shall be activated
  Due to the fact that the application might call the runtime library functions, the runtime library functions shall be called via the “Far” addressing method



2.5 Instruction fetch and pre-fetch in RAM

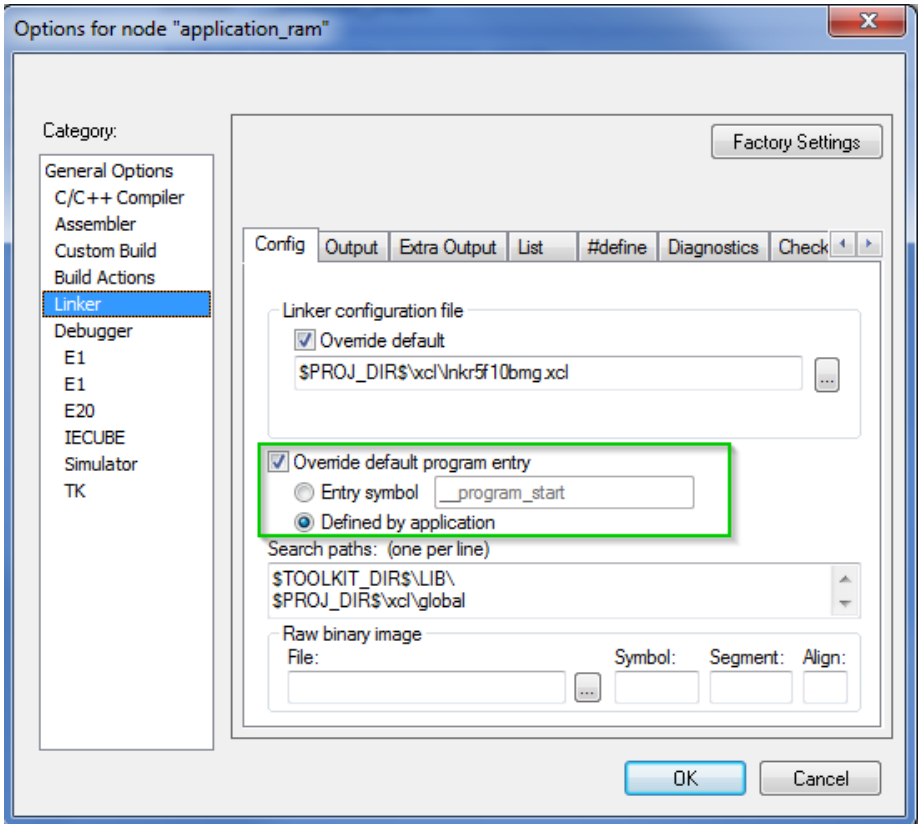
On the RL78 device family it is mandatory to initialize the RAM before reading the data. The same applies to the area used for code execution from RAM. For that reason, it is important that the whole RAM area consumed by the application is initialized.

Please note that the pipeline of the RL78 devices performs an instruction pre-fetch of 10 bytes. Therefore the pre-fetch area after the last instruction has also to be initialized. The easiest way to do this is to extend the segment to be copied to RAM by 10bytes. The initialization will then be done by the copy process automatically.

2.6 Linker configuration

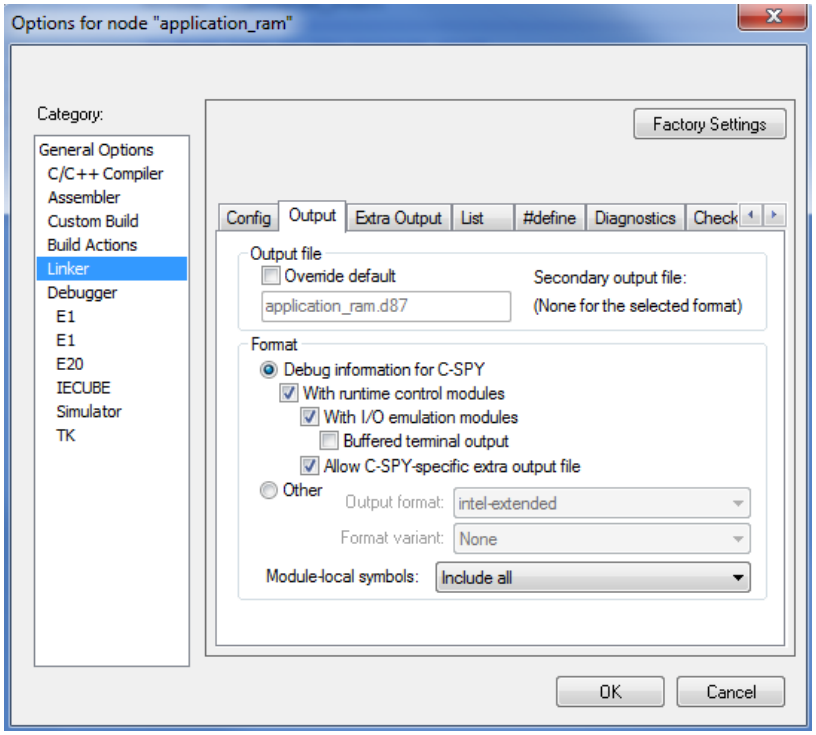
2.6.1 Program entry

Within the linker configuration the program entry shall be set to “defined application”, because the reset vector will not be generated.

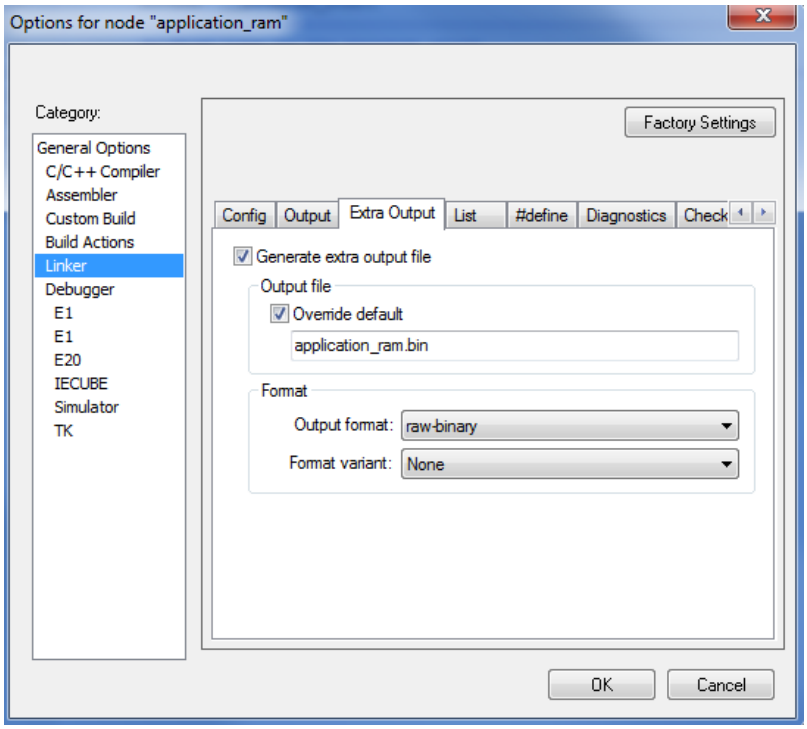


2.6.2 Output files

For debugging purpose please configure the C-SPY output file and allow the extra output file for the binary image.



Within the extra output dialog the binary image can be defined.



3. Create the main application (located in ROM)

Within the previous chapter we introduced the procedure on how to creation an application which will be execute from RAM. This chapter will describe how to integrate this application into the main application and run it from RAM.

3.1 Linker configuration

3.1.1 Memory mapping

To avoid any overlapping of segments defined within the RAM application the symbols defined in `common.xcl` file should be used. Please refer to chapter "2.1.2 Common linker file for shared information" for details.

3.1.2 Add binary image

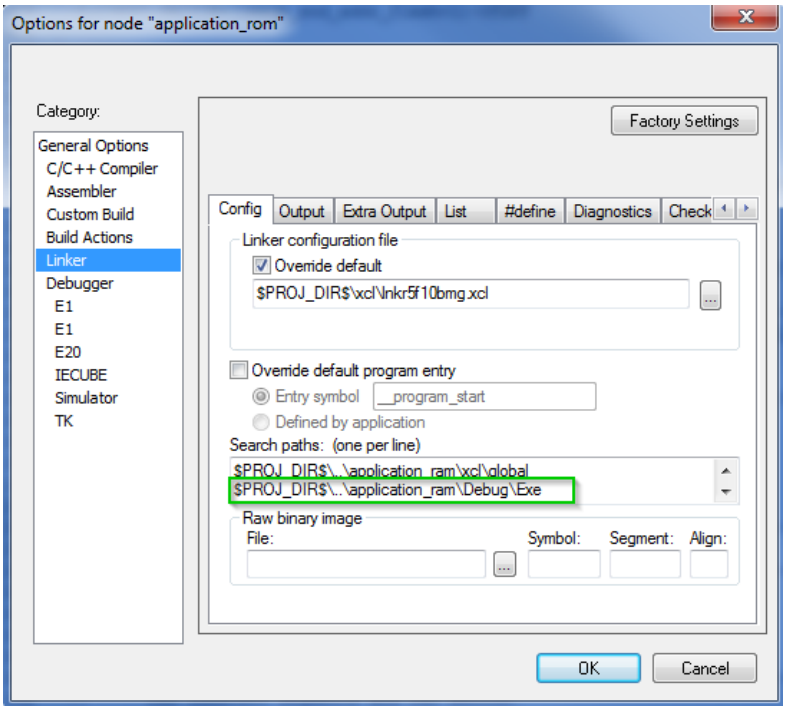
The binary image generated within the RAM application project can be loaded via the linker option `--image_input`. The parameter for this option are defined as follows.

```
--image_input=<BIN IMAGE>,<APP SYM>,<APP SEGMENT>,<ALIGNMENT>
```

```
Sample:
--image_input=application_ram.bin,app_ram_img_src_symbol,RAM_APP_IMGAGE,1
```

`application_ram.bin:`	Binary image
`app_ram_img_src_symbol:`	Symbol definition for the binary image. This symbol can be accessed from the source code.
`RAM_APP_IMGAGE:`	Segment for image placement
`1:`	1 byte alignment

Please note that the usage of a binary image without a defined path is only possible, if the path is added to the search path of the linker. You can define it here:



In our application we will not access the symbol "`app_ram_img_src_symbol`" defined during the image load. Therefore the symbol is not referenced and will be removed by the linker due to optimization reason. To avoid the removal of the image the linker option "-g" can be used.

```
-gapp_ram_img_src_symbol
```

3.1.3 Symbol definition for image access

The image is now integrated into the project and so that we can define the access method to the functions, constant etc. within the image. There are several ways to do this, like e.g.

- Usage of C function pointers to access functions etc…..
- Definition of symbols within the linker file and access directly within the C code

Due to the fact that our RAM application is a standalone application we don't need to access each function separately. We only have to call the address where the branch instruction is located (see 2.3 Replacement of reset vector).

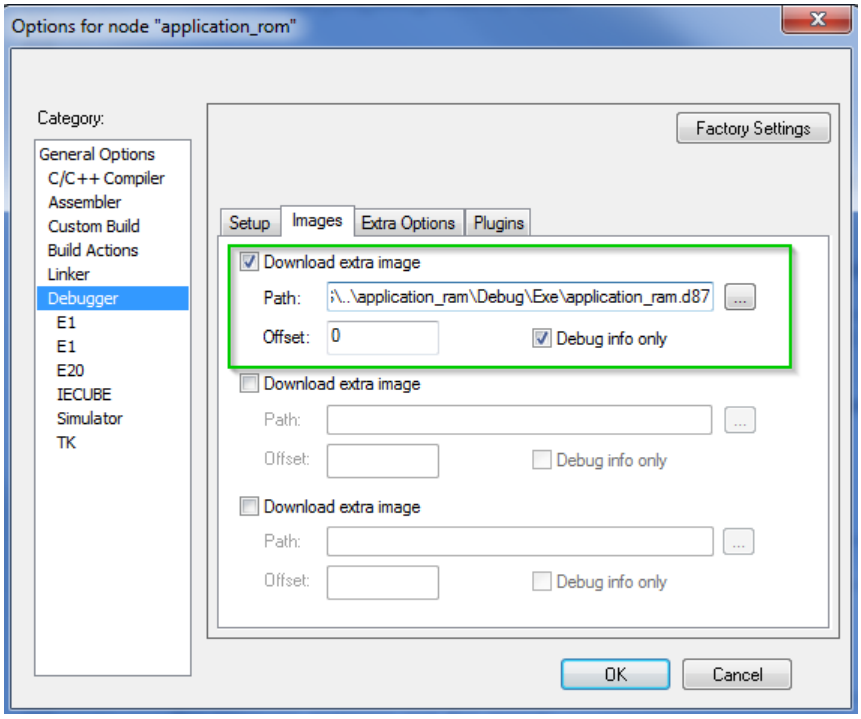
The branch instruction is mapped to the image start address and therefore we can use the symbol `image_start_addr_RAM` defined in `common.xcl` as shown below.

```
-Dmy_app_ram=image_start_addr_RAM
```

The symbol `my_app_ram` will be later used as a function entry within the C code.

3.2 Debugger configuration

Since the binary image doesn't contains any symbol definition for debugging we have to load an additional file for symbol information. Load the symbol information from the C-SPY debug file within the debugger configuration here.



Please check the check box "Debug info only", because the image itself is already loaded via the linker.

3.3 Access the image from C code

3.3.1 Copy image segments to RAM

First of all the image have to be copied segment by segment to the RAM. The segment address information within our sample is available from the common.xcl file and can be accessed within the C code like shown below.

Import the addresses from the common.xcl file by using the extern keyword.

```
extern __far void __far * XCODE_start_addr;
extern __far void * RAM_XCODE_start_addr;
extern __far void * XCODE_size;
```

Implement a function for copy the segments from flash to RAM.

```
void copySegmentToRam(UCHAR __far *src, UCHAR *dst, ULONG size)
{
  while( size > 0){
      *dst++ = *src++;
      size--;
  }
}
```

Note: Precondition for the above code is that the whole source segment is located within a 64KB flash page. If the segment overlaps two 64KB pages, the pointer increment "src++" is not enough, because it can be incremented within one 64KB page only. On IAR version V1.40.1 and later you can use the __huge pointer to avoid this problem.

Copy the segment by calling the copy function.

```
copySegmentToRam((UCHAR __far *)&XCODE_start_addr,
                 (UCHAR*)&RAM_XCODE_start_addr, (ULONG)&XCODE_size);
```

3.3.2 Execute the RAM application function

Within our sample the application within RAM is a standalone application and therefore we just need to call it. If the application is finished it will return to the application located in ROM.

First of all we have to define the function prototype by using the symbol defined within the ROM application xcl file.

```
extern __far_func void my_app_ram(void);
```

After the copy process described within the chapter before we can call the function and execute code from RAM.

```
my_app_ram();
```

3.4 Debugging

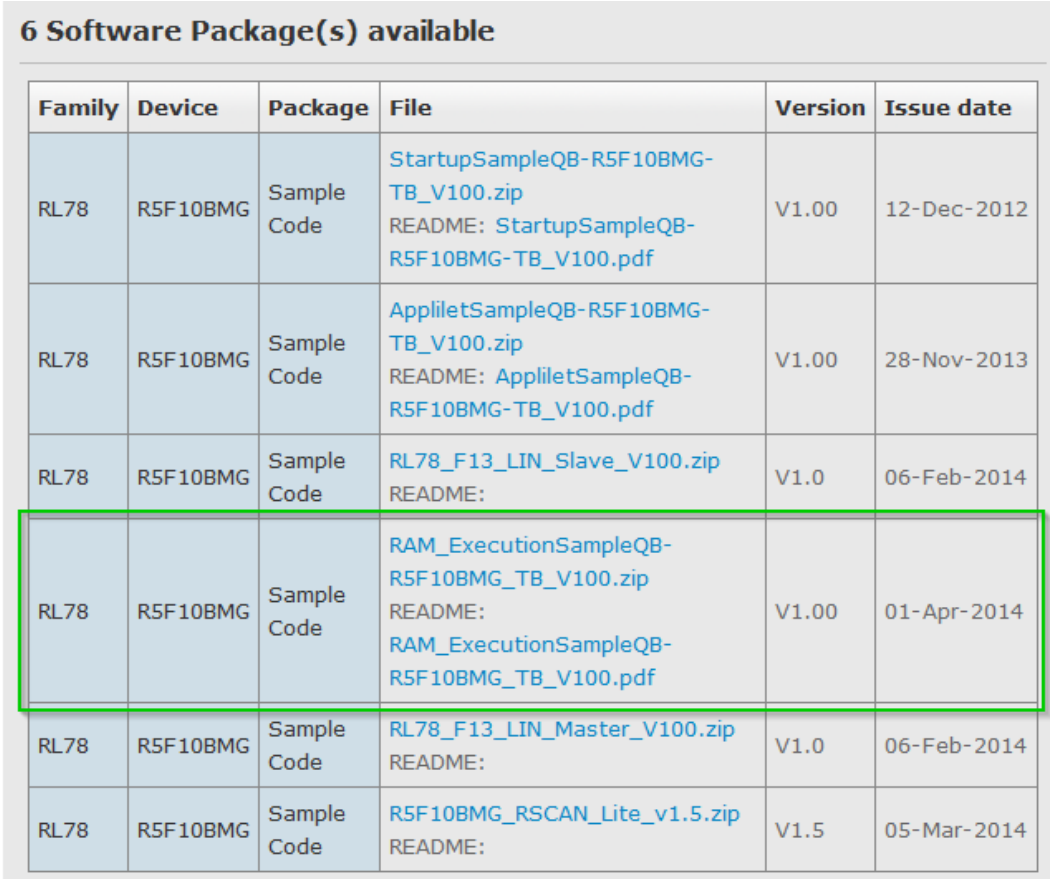
3.4.1 Software breakpoints usage within RAM

The software breakpoints within RAM can only be used after the copy process of the segments is performed. Otherwise the breakpoint instruction within RAM would be deleted by the segment copy process.

4. Sample application

In order to provide you an easy start into the above described software concept, Renesas provides a sample application which covers all the described aspects. The sample application can be downloaded from the following site:

http://www.renesas.eu/update?oc=QB-R5F10BMG-TB



6 Software Package(s) available

Family	Device	Package	File	Version	Issue date
RL78	R5F10BMG	Sample Code	StartupSampleQB-R5F10BMG-TB_V100.zip README: StartupSampleQB-R5F10BMG-TB_V100.pdf	V1.00	12-Dec-2012
RL78	R5F10BMG	Sample Code	AppliletSampleQB-R5F10BMG-TB_V100.zip README: AppliletSampleQB-R5F10BMG-TB_V100.pdf	V1.00	28-Nov-2013
RL78	R5F10BMG	Sample Code	RL78_F13_LIN_Slave_V100.zip README:	V1.0	06-Feb-2014
RL78	R5F10BMG	Sample Code	RAM_ExecutionSampleQB-R5F10BMG_TB_V100.zip README: RAM_ExecutionSampleQB-R5F10BMG_TB_V100.pdf	V1.00	01-Apr-2014
RL78	R5F10BMG	Sample Code	RL78_F13_LIN_Master_V100.zip README:	V1.0	06-Feb-2014
RL78	R5F10BMG	Sample Code	R5F10BMG_RSCAN_Lite_v1.5.zip README:	V1.5	05-Mar-2014

This sample application is designed for the target board QB-R5F10BMG-TB where the RL78/F13 (R5F100BMG) device is mounted.

4.1 Run the sample application

Please follow the steps to run the sample application on your target board.

1. Unpack the sample application and open the `workspace.eww` file via RL78 IAR environment V1.30.5 or higher
2. Build application “application_ram”.
3. Build application “application_rom”.
4. Connect the target board with E1 and start the debug session.
5. Press button “Go” to start the debug session. During the execution of the application “application_ram” in RAM the LED1 on the target board should blink.

Please note that the “application_ram” cannot be debugged as a standalone application. Start debug session for application “application_rom” only.

5. Website and Support

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

Inquiries
http://www.renesas.com/contact/

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
V1.0	01/04/2014		Initial version

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation 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.
2. 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.
3. 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.
4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.
   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). 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 for which it is not intended. 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 which the product is not intended by Renesas Electronics.
6. 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.
7. 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 systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please 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.
9. 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. You should not use Renesas Electronics products or 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. When exporting the Renesas Electronics 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.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.
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.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Ku, Seoul, 135-920, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0