RZ/T1 Group

Initial Settings of the Microcomputers Incorporating the R-IN Engine

R01AN2989EJ0131
Rev.1.31
Jul. 13, 2018

Outline

This application note describes a sample program for making initial settings of the RZ/T1 microcomputers incorporating the R-IN Engine.

The features of this program are as follows:

- The Cortex-R4 core makes initial settings after it is released from the reset state and releases the Cortex-M3 from reset. The R4 core then generates an inter-CPU interrupt, and makes LED 0 flash with a predetermined period.
- The Cortex-M3 core makes initial settings after it is released from the reset state and waits for the inter-CPU interrupt request from the Cortex-R4. On reception of the interrupt, the M3 core lights up LED 1.
- Pressing the switch SW2 makes the Cortex-R4 write “LED data” including the state of LED 0 (on or off) to the shared memory area. For its own part, the M3 core constantly reads the LED data in the shared memory and reflects the state read from there in LED 1.

Target Devices

RZ/T1 group

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation and testing of the modified program.

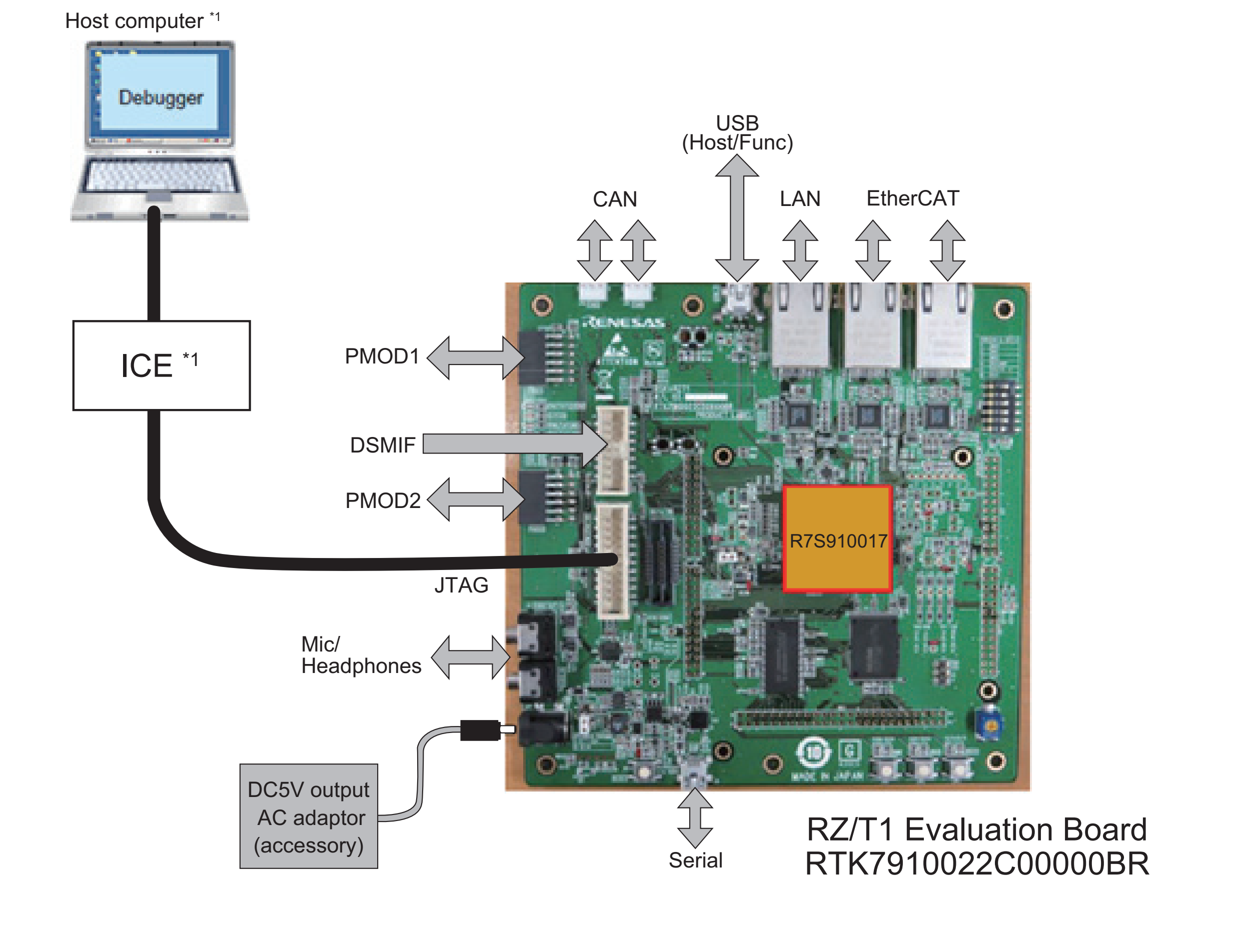
Table of Contents

1. Specifications

Table 1.1 lists the peripheral modules to be used and their applications and Figure 1.1 shows the operating environment.

Table 1.1 Peripheral Modules and Applications

Peripheral Module	Application
Clock generator (CPG)	Supplying the CPU clock and low-speed on-chip oscillator clock
Interrupt controller (ICUA)	Handling interrupts through an external interrupt input pin (IRQ5) and inter-CPU interrupts
Extended internal RAM	The shared memory area (extended internal instruction RAM, “RAM I”) and memory area for the program of the Cortex-M3 (extended internal data RAM, “RAM D”)
Error control module (ECM)	Initializing the ERROROUT# pin
General-purpose input/output ports	Pins which control switching the LEDs on and off



Note 1. Indicates the device that the user needs to prepare.

Figure 1.1 Operating Environment

2. Operating Environment

The sample program described in this application note is for the environment below.

Table 2.1 Operating Environment

Item	Description
Microcomputer	RZ/T1 group
Operating frequency	CPUCLK (Cortex-R4): 450 MHz ICLK (Cortex-M3): 150 MHz
Operating voltage	3.3 V
Integrated development environment	Embedded Workbench® for Arm, version 8.20.2 from IAR Systems DS-5™ 5.26.2 from Arm e2 studio 6.1.0 from Renesas
Operating mode	SPI boot mode 16-bit-bus boot mode
Board	RZ/T1 Evaluation Board (RTK7910022C00000BR)
Devices (functions to be used on the board)	• NOR flash memory (connected to CS0 and CS1 spaces) Manufacturer: Macronix International Co., Ltd. Model: MX29GL512FLT2I-10Q • SDRAM (connected to CS2 and CS3 spaces) Manufacturer: Integrated Silicon Solution Inc. Model: IS42S16320D-7TL • Serial flash memory Manufacturer: Macronix International Co., Ltd. Model: MX25L51245G

3. Related Application Notes

The application note related to this document is listed below for reference.

- RZ/T1 Group Application Note: Initial Settings

4. Peripheral Modules

The basics of the clock generator (CPG), the interrupt controller (ICUA), the error control module (ECM), extended internal RAM, and general-purpose input/output ports are described in the “RZ/T1 Group User's Manual: Hardware.”

5. Hardware

5.1 Hardware Structure Example

Figure 5.1 shows an example of hardware structure.

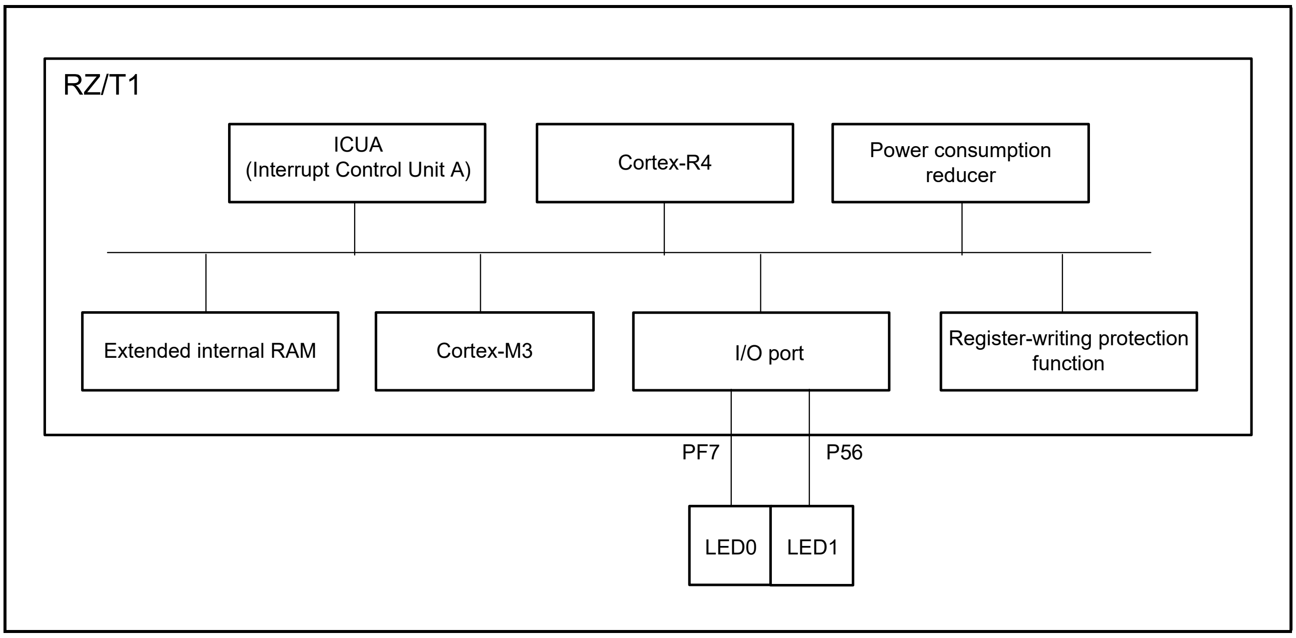

RZ/T1
ICUA (Interrupt Control Unit A)
Cortex-R4
Power consumption reducer
Extended internal RAM
Cortex-M3
I/O port
Register-writing protection function
PF7
P56
LED0
LED1

Figure 5.1 Example of Hardware Structure

5.2 Pins Used

Table 5.1 lists the pins to be used and their functions.

Table 5.1 Pins and Functions

Pin Name	Input/Output	Description

MD0	Input	Select the operating mode
		MD0 = L and MD1 = L and MD2 = L (SPI boot mode)
		MD0 = L and MD1 = H and MD2 = L (16-bit-bus boot mode)

MD1	Input	
MD2	Input	
IRQ5	Input	SW2 (IRQ pin interrupt)
PF7	Output	Switches LED 0 on and off.
P56	Output	Switches LED 1 on and off.

6. Software

This section explains the case of EWARM (from IAR systems) unless otherwise stated.

6.1 Operation Overview

The sample program described in this application note makes initial settings of the Cortex-R4 and Cortex-M3 cores.

1. The program for the Cortex-R4 core makes initial settings after power for the core is switched on. This core copies the program code for the Cortex-M3 from the external flash memory into the extended internal RAM, and releases the M3 core from the reset state. Note that the R4 core does not copy the code for the M3 core if the program is to be executed directly from the RAM. After making settings for the LED-controlling port pins and interrupts, the R4 core generates an inter-CPU interrupt to notify the M3 core of completion of the initial settings.
2. The program for the Cortex-M3 core makes initial settings after it is released from the reset state (software reset 2) and waits for an inter-CPU interrupt from the Cortex-R4 core.
3. The Cortex-R4 core makes LED 0 flash on and off with a predetermined period. The state of LED 0 (switched on or off) is written to the shared memory area (extended internal RAM) on detection of the switch SW2 being pressed. On its side, the Cortex-M3 core constantly reads LED data from the shared memory and turns LED 1 on or off according to the value read (the initial value is “on”). For example, if SW2 is pressed while LED 0 is off, the state of LED 0 is reflected in LED 1, resulting in LED 1 being switched off.

Figure 6.1 shows the processing for initial settings of the microcomputers incorporating the R-IN engine in outline.

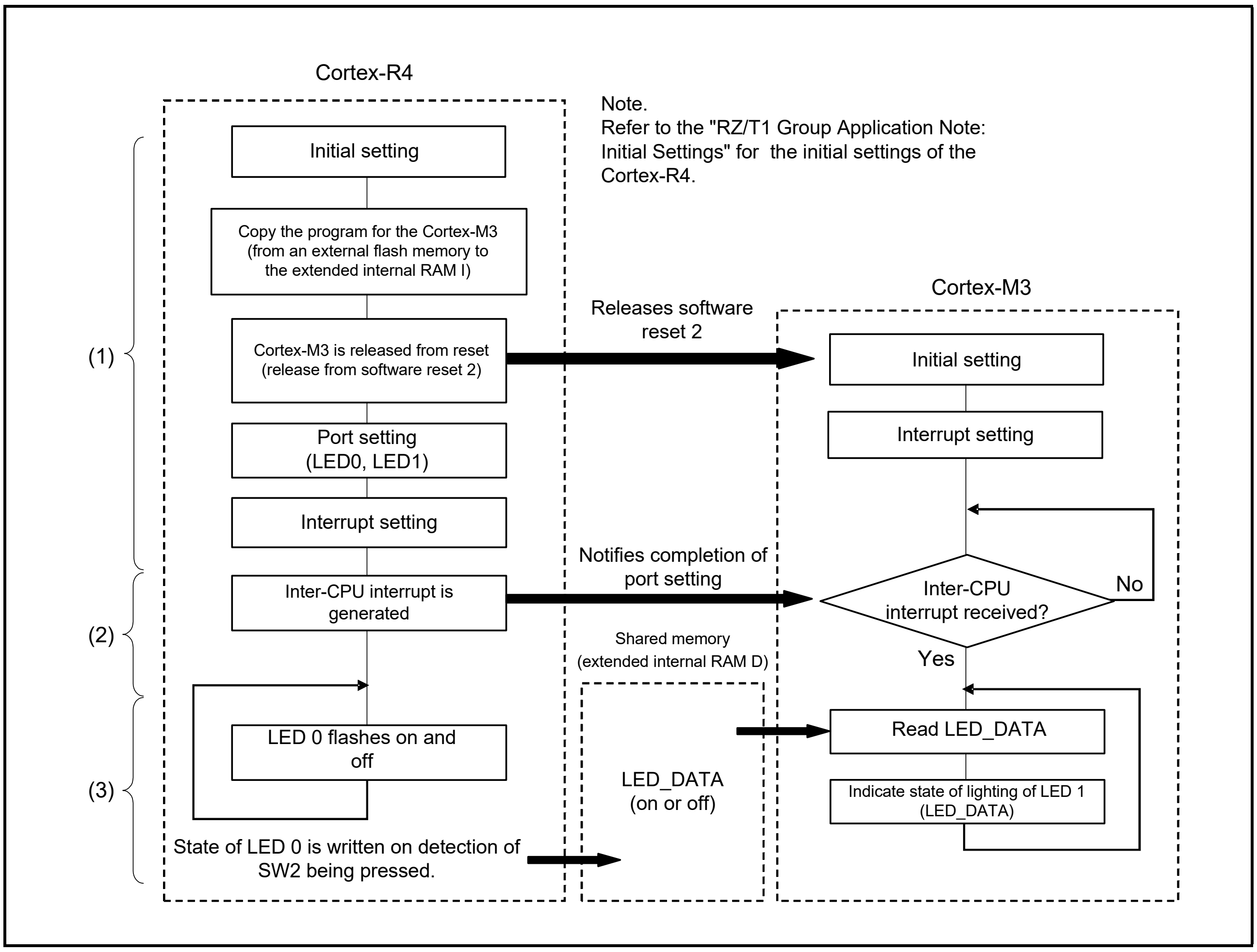

Cortex-R4
Note.
Refer to the "RZ/T1 Group Application Note: Initial Settings" for the initial settings of the Cortex-R4.
Initial setting
Copy the program for the Cortex-M3 (from an external flash memory to the extended internal RAM I)
Cortex-M3 is released from reset (release from software reset 2)
Port setting (LED0, LED1)
Interrupt setting
Inter-CPU interrupt is generated
LED 0 flashes on and off
State of LED 0 is written on detection of SW2 being pressed.
(1)
(2)
(3)
Releases software reset 2
Notifies completion of port setting
Shared memory (extended internal RAM D)
LED_DATA (on or off)
Cortex-M3
Initial setting
Interrupt setting
Inter-CPU interrupt received?
No
Yes
Read LED_DATA
Indicate state of lighting of LED 1 (LED_DATA)

Figure 6.1 Overview of Processing for Initial Settings of the Microcomputers incorporating the R-IN Engine

6.1.1 Preparation

Make appropriate setting for SW4 on the RZ/T1 Evaluation Board (RTK7910022C00000BR) depending on the project to be used. Table 6.1 shows the settings of SW4. Detailed settings of SW4 are described in the “RZ/T1 Evaluation Board RTK7910022C00000BR User's Manual” (see Section 8, Related Documents).

Table 6.1 Settings of SW4

Sample Program	SW4-1	SW4-2	SW4-3	SW4-4	SW4-5	SW4-6
16-bit-bus boot mode version	ON	OFF	ON	ON	ON	OFF
SPI boot mode version	ON	ON	ON	ON	ON	OFF
RAM exectuion version	Either of the above combinations.					

6.2 Memory Map

The address space of the RZ/T1 group and the memory map of the RZ/T1 Evaluation Board are described in the “RZ/T1 Group Application Note: Initial Settings” and the “RZ/T1 Group User's Manual: Hardware.”

6.2.1 Section Assignment for the Sample Program

Table 6.2 lists the sections to be used in this sample program. Figure 6.2 shows the assignment of sections (for the Cortex-R4 in the 16-bit-bus boot version) as an example.

The details of the sections for the Cortex-R4 are described in the “RZ/T1 Group Application Note: Initial Settings”.

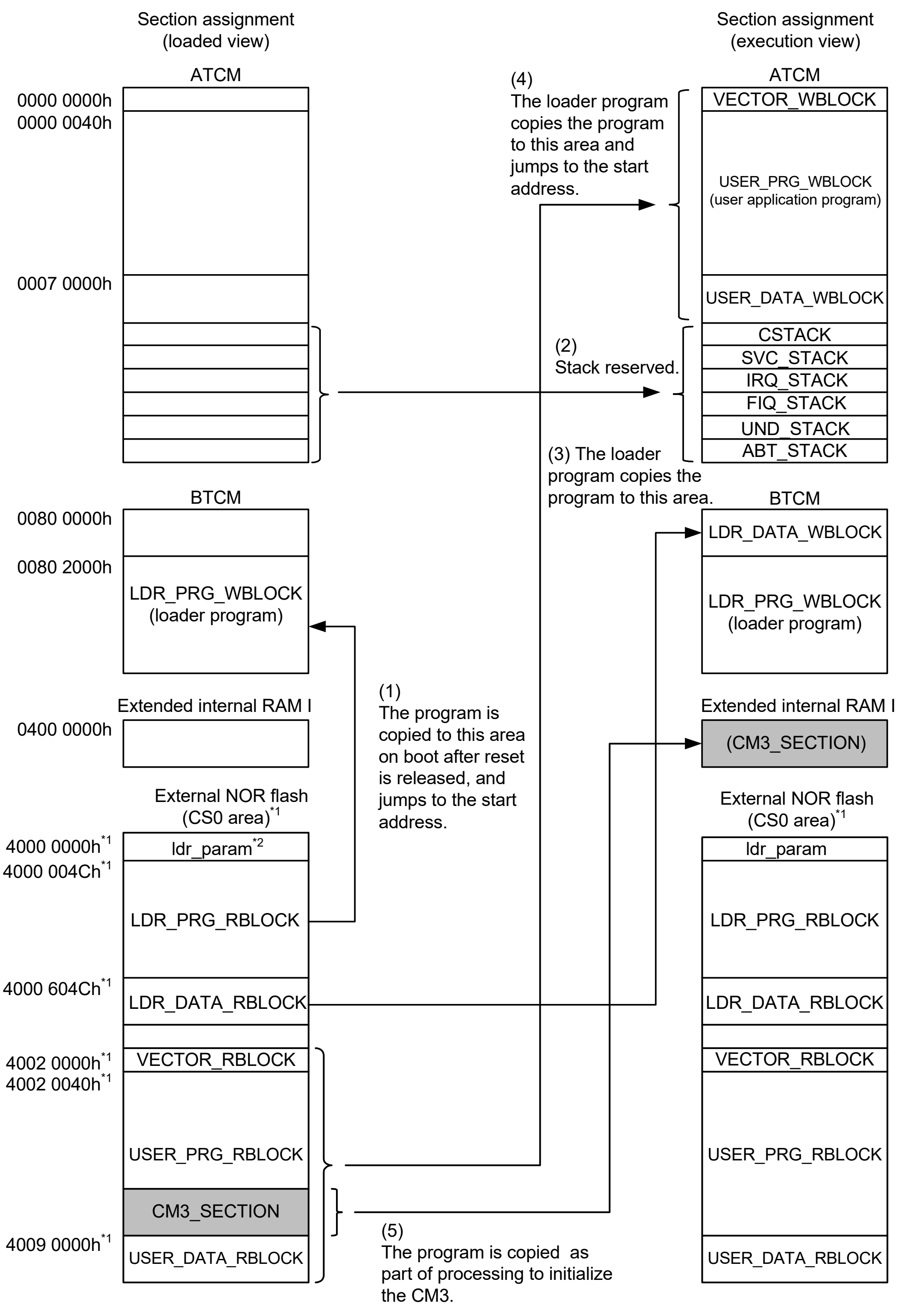
Table 6.2 Sections to be Used (Cortex-R4)

Area Name	Description	Type	Loading Area	Execution Area
VECTOR_WBLOCK	Reset and exception processing vector table	Code	—	ATCM
USER_PRG_WBLOCK	User application program area (for execution)	Code	—	ATCM
USER_DATA_WBLOCK	User application program variable area (for execution)	Data	—	ATCM
CSTACK	Stack area	Data	—	ATCM
SVC_STACK	Supervisor (SVC) mode stack area	Data	—	ATCM
IRQ_STACK	IRQ mode stack area	Data	—	ATCM
FIQ_STACK	FIQ mode stack area	Data	—	ATCM
UND_STACK	Undefined (UND) mode stack area	Data	—	ATCM
ABT_STACK	Abort (ABT) mode stack area	Data	—	ATCM
LDR_DATA_WBLOCK	Loader program variable area (for execution)	Data	—	BTCM
LDR_PRG_WBLOCK	Loader program area (for execution)	Code	—	BTCM
ldr_param	Loader parameters*1	Data	FLASH*2	—
LDR_PRG_RBLOCK	Loader program area (for storing)*1	Code	FLASH*2	—
LDR_DATA_RBLOCK	Loader program variable area (for storing)*1	Data	FLASH*2	—
VECTOR_RBLOCK	Reset and exception processing vector table area (for storing)*1	Code	FLASH*2	—
USER_PRG_RBLOCK	User application program area (for storing)*1 User application program area for the Cortex-M3 (for storing)*3	Code	FLASH*2	—
USER_DATA_RBLOCK	User application program variable area (for storing)*1	Data	FLASH*2	—

Note 1. The RAM execution version does not have this area.
Note 2. This is assigned to the NOR flash memory in the case of the 16-bit-bus boot mode version and to the serial flash memory in the case of the SPI boot mode version.
Note 3. The RAM execution version does not have this area. Instead of this area, according to the section assignment of the Cortex-M3, the development tool directly downloads the program to the area in extended internal RAM I from which it will be executed.

Table 6.3 Sections to be Used (Cortex-M3)

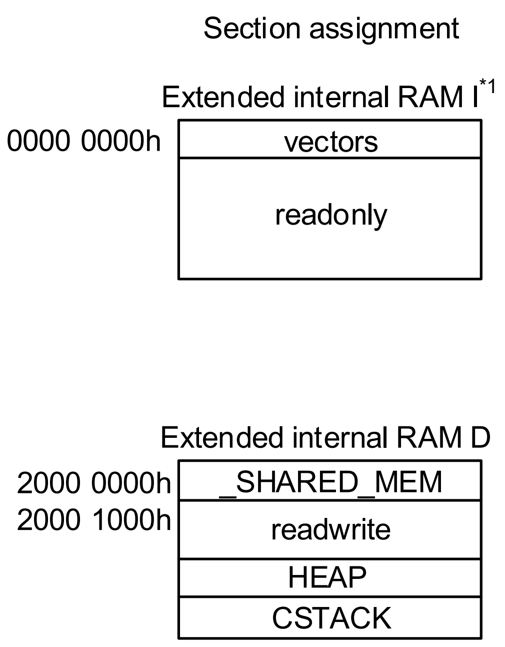
Area Name	Description	Type	Loading Area	Execution Area
vectors	Vector area	Code	Extended internal RAM I	Extended internal RAM I
readonly	User application program area	Code	Extended internal RAM I	Extended internal RAM I
_SHARED_MEM	Shared memory area	Data	Extended internal RAM D	Extended internal RAM D
readwrite	User application program variable area	Data	Extended internal RAM D	Extended internal RAM D
HEAP	Heap area	Data	Extended internal RAM D	Extended internal RAM D
CSTACK	Stack area	Data	Extended internal RAM D	Extended internal RAM D


Section assignment (loaded view)
Section assignment (execution view)
ATCM
0000 0000h
0000 0040h
0007 0000h
VECTOR_WBLOCK
USER_PRG_WBLOCK (user application program)
USER_DATA_WBLOCK
CSTACK
SVC_STACK
IRQ_STACK
FIQ_STACK
UND_STACK
ABT_STACK
(4) The loader program copies the program to this area and jumps to the start address.
(2) Stack reserved.
(3) The loader program copies the program to this area.
BTCM
0080 0000h
0080 2000h
LDR_DATA_WBLOCK
LDR_PRG_WBLOCK (loader program)
(1) The program is copied to this area on boot after reset is released, and jumps to the start address.
Extended internal RAM I
0400 0000h
(CM3_SECTION)
External NOR flash (CS0 area)*1
4000 0000h*1
4000 004Ch*1
4000 604Ch*1
4002 0000h*1
4002 0040h*1
4009 0000h*1
ldr_param*2
ldr_param
LDR_PRG_RBLOCK
LDR_DATA_RBLOCK
VECTOR_RBLOCK
USER_PRG_RBLOCK
CM3_SECTION
USER_DATA_RBLOCK
(5) The program is copied as part of processing to initialize the CM3.

Note 1. This corresponds to the serial flash (SPIBSC area) in case of SPI boot version. Read addresses as if the start address is 1000 0000h.

Note 2. The loader parameters (ldr_param section) are used in boot processing. Refer to the “RZ/T1 Group User's Manual: Hardware for details.

Figure 6.2 Assignment of Sections (for the Cortex-R4 in the 16-Bit-Bus Boot Version)



Note: The sections in extended internal RAM I are copied by the Cortex-R4 as the CM3_SECTION.

Figure 6.3 Assignment of Sections (Cortex-M3)

6.2.2 MPU Settings

MPU settings are described in the “RZ/T1 Group Application Note: Initial Settings.”

6.2.3 Exception Processing Vector Table

Exception processing vector table is found in the “RZ/T1 Group Application Note: Initial Settings.”

6.3 Interrupts

Table 6.4 lists the interrupts used in the sample program.

Table 6.4 Interrupts Used in Sample Program

Interrupt (Source ID)	Priority	Process in Outline
Inter-CPU interrupt (IRQ1)	15	The Cortex-R4 notifies the Cortex-M3 of the completion of initial settings (applicable to the Cortex-R4 and Cortex-M3).
IRQ pin interrupt 5 (IRQ9)	15	State of lighting of LED 0 (switched on or off) is written to the shared memory on detection of SW2 being pressed (applicable to the Cortex-R4).

6.4 Fixed-Width Integer Types

Table 6.5 lists the fixed-width integers for sample program.

Table 6.5 Fixed-Width Integers for Sample Program

Symbol	Description
int8_t	8-bit signed integer (defined in the standard library)
int16_t	16-bit signed integer (defined in the standard library)
int32_t	32-bit signed integer (defined in the standard library)
int64_t	64-bit signed integer (defined in the standard library)
uint8_t	8-bit unsigned integer (defined in the standard library)
uint16_t	16-bit unsigned integer (defined in the standard library)
uint32_t	32-bit unsigned integer (defined in the standard library)
uint64_t	64-bit unsigned integer (defined in the standard library)

6.5 Constants/Error Codes

Table 6.6 lists the constants used in the sample program. Detail on the program of the Cortex-R4 core is found in the "RZ/T1 Group Application Note: Initial Settings."

Table 6.6 Constants Used in the Sample Program

Constant Name	Value	Description
LED_OUTPUT_HIGH	(1)	Output value indicating the lighting state of the LED
SHM_SUCCESS	(0)	"Successful" control flag for the shared memory driver
SHM_ERR	(-1)	"Unsuccessful" control flag for the shared memory driver
SHM_SEMFNO_TOTAL	(8)	Total number of the semaphore flags for the shared memory driver
SHM_ SEMFNO_0	(0)	Semaphore flag 0 for the shared memory driver
SHM_ SEMFNO_1	(1)	Semaphore flag 1 for the shared memory driver
SHM_ SEMFNO_2	(2)	Semaphore flag 2 for the shared memory driver
SHM_ SEMFNO_3	(3)	Semaphore flag 3 for the shared memory driver
SHM_ SEMFNO_4	(4)	Semaphore flag 4 for the shared memory driver
SHM_ SEMFNO_5	(5)	Semaphore flag 5 for the shared memory driver
SHM_ SEMFNO_6	(6)	Semaphore flag 6 for the shared memory driver
SHM_ SEMFNO_7	(7)	Semaphore flag 7 for the shared memory driver

6.6 Structures/Unions/Enumerated Types

Figure 6.4 is the structures, unions, and enumerated types used in the sample program. Detail on the program of the Cortex-R4 core is described in the "RZ/T1 Group Application Note: Initial Settings."

```
/* Shared memory struct */
/* Size MAX 4KB */
struct st_shm
{
   uint32_t LED_DATA;
};



/* struct of [SYSTEM.SYTSEMFn] register */
typedef struct
{
        union
        {
                    unsigned long LONG;
                    struct
                     {
                               unsigned long SEMF:1;
                               unsigned long :31;
                     } BIT;
        } SYTSEMF;
} st_sytsemf;
```

Figure 6.4 Structures/Unions/Enumerated Types Used in Sample Program

6.7 Global Variables

Table 6.7 lists the global variables. Detail on the program of the Cortex-R4 core is described in the "RZ/T1 Group Application Note: Initial Settings."

Table 6.7 Global Variables

Type	Variable Name	Description	Function
uint8_t	g_ready_flag	Initialization completion flag of the Cortex-R4	init_main.c (Cortex-M3)
uint32_t	g_led_data	Variable to indicate the state of the LED	main.c (Cortex-R4) init_main.c (Cortex-M3)

6.8 Functions

Table 6.8 lists the functions. Detail on the program of the Cortex-R4 core is described in the "RZ/T1 Group Application Note: Initial Settings."

Table 6.8 Functions

Function Name	Page
R_SHM_Init	17
R_SHM_memcpy	17
R_SHM_Load_uint32	17
R_SHM_Load_int32	18
R_SHM_Load_uint16	18
R_SHM_Load_int16	18
R_SHM_Load_uint8	18
R_SHM_Load_int8	19
main (Cortex-R4)	19
init_cm3 (Cortex-R4)	19
R_IRQ9_isr (Cortex-R4)	20
main (Cortex-M3)	19
IRQ_INTERCPU_IRQHandler(Cortex-M3)	20

Note: The functions are common among cores unless otherwise specified.

6.9 Specification of Functions

Specifications of the functions used in the sample program are as follows.

6.9.1 R_SHM_Init

R_SHM_Init		
Synopsis	Initializing the shared-memory driver	
Declaration	int32_t R_SHM_Init (uint32_t semfno)	
Description	This function makes initial settings of the specified semaphore register to be used for the shared memory driver.	
Arguments	uint32_t semfno	Specifies the number of the semaphore register to be initialized as a value from 0 to 7.
Return value	SHM_SUCCESS: succeeded. SHM_ERR: failed.	
Remarks	None	

6.9.2 R_SHM_memcpy

R_SHM_memcpy		
Synopsis	Copies data to and from the shared memory area	
Declaration	int32_t R_SHM_memcpy(void *dst, void *src, size_t size)	
Description	This function copies multiple bytes of data to and from the shared memory area.	
Arguments	void *dst	Pointer to the address of the destination memory area
	void *src	Pointer to the address of the source memory area
	size_t size	Size of memory to be transferred.
Return value	SHM_SUCCESS: succeeded SHM_ERR: failed	
Remarks	None	

6.9.3 R_SHM_Load_uint32

R_SHM_Load_uint32		
Synopsis	Function for loading four bytes of unsigned int type data to and from the shared memory	
Declaration	int32_t R_SHM_Load_uint32(uint32_t *dst, uint32_t *src)	
Description	This function loads four bytes of unsigned int type data to and from the shared memory.	
Arguments	uint32_t *dst	Pointer to the address of the destination memory area
	uint32_t *src	Pointer to the address of the source memory area
Return value	None	
Remarks	None	

6.9.4 R_SHM_Load_int32

R_SHM_Load_int32		
Synopsis	Function for loading four bytes of signed int type data to and from the shared memory	
Declaration	int32_t R_SHM_Load_int32(int32_t *dst, int32_t *src)	
Description	This function loads four bytes of singed int type data to and from the shared memory.	
Arguments	int32_t *dst	Pointer to the address of the destination memory area
	int32_t *src	Pointer to the address of the source memory area
Return value	None	
Remarks	None	

6.9.5 R_SHM_Load_uint16

R_SHM_Load_uint16		
Synopsis	Function for loading two bytes of unsigned int type data to and from the shared memory area	
Declaration	int32_t R_SHM_Load_uint16(uint16_t *dst, uint16_t *src)	
Description	This function loads two bytes of unsigned int type data to and from the shared memory area.	
Arguments	uint16_t *dst	Pointer to the address of the destination memory area
	uint16_t *src	Pointer to the address of the source memory area
Return value	None	
Remarks	None	

6.9.6 R_SHM_Load_int16

R_SHM_Load_int16		
Synopsis	Function for loading two bytes of signed int type data to and from the shared memory area	
Declaration	int32_t R_SHM_Load_int16(int16_t *dst, int16_t *src)	
Description	This function loads two bytes of signed int type data to and from the shared memory area.	
Arguments	int16_t *dst	Pointer to the address of the destination memory area
	int16_t *src	Pointer to the address of the source memory area
Return value	None	
Remarks	None	

6.9.7 R_SHM_Load_uint8

R_SHM_Load_uint8		
Synopsis	Function for loading one byte of unsigned int type data to and from the shared memory area	
Declaration	int32_t R_SHM_Load_uint8(uint8_t *dst, uint8_t *src)	
Description	This function loads one byte of unsigned int type data to and from the shared memory area.	
Arguments	uint8_t *dst	Pointer to the address of the destination memory area
	uint8_t *src	Pointer to the address of the source memory area
Return value	None	
Remarks	None	

6.9.8 R_SHM_Load_int8

R_SHM_Load_int8		
Synopsis	Function for loading one byte of signed int type data to and from the shared memory area	
Declaration	int32_t R_SHM_Load_int8(int8_t *dst, int8_t *src)	
Description	This function loads one byte of signed int type data to and from the shared memory area.	
Arguments	int8_t *dst	Pointer to the address of the destination memory area
	int8_t *src	Pointer to the address of the source memory area
Return value	None	
Remarks	None	

6.9.9 main (Cortex-R4)

main	
Synopsis	Main processing
Declaration	int main (void)
Description	This is the user application program for the Cortex-R4 core.
Arguments	None
Return value	None
Remarks	None

6.9.10 init_cm3 (Cortex-R4)

init_cm3	
Synopsis	Initialization processing of the Cortex-M3 core
Declaration	void init_cm3(void)
Description	This function copies the program code for the Cortex-M3 from an external flash memory to the extended internal RAM and releases the M3 from the reset state.
Arguments	None
Return value	None
Remarks	When the program is run directly from RAM, it does not have to be copied, so only releasing the Cortex-M3 from the reset state proceeds in this case.

6.9.11 main (Cortex-M3)

main	
Synopsis	Main processing
Declaration	int main(void)
Description	This is the user application program for the Cortex-M3 core.
Arguments	None
Return value	None
Remarks	None

6.9.12 R_IRQ9_isr (Cortex-R4)

R_IRQ_9_isr	
Synopsis	IRQ9 interrupt (IRQ pin interrupt 5) handling
Declaration	void R_IRQ9_isr(void)
Description	This function writes the state of lighting of LED 0 to the shared memory.
Arguments	None
Return value	None
Remarks	None

6.9.13 IRQ_INTERCPU_IRQHandler

IRQ_INTERCPU_IRQHandler	
Synopsis	Inter-CPU interrupt handling
Declaration	void IRQ_INTERCPU_IRQHandler(void)
Description	This function handles interrupt processing in response to interrupt requests from the Cortex-R4 core and sets the g_ready_flag.
Arguments	None
Return value	None
Remarks	None

6.10 Flowchart

6.10.1 Initialization Processing of Shared Memory Driver

Figure 6.5 shows the initialization processing of the shared memory driver.

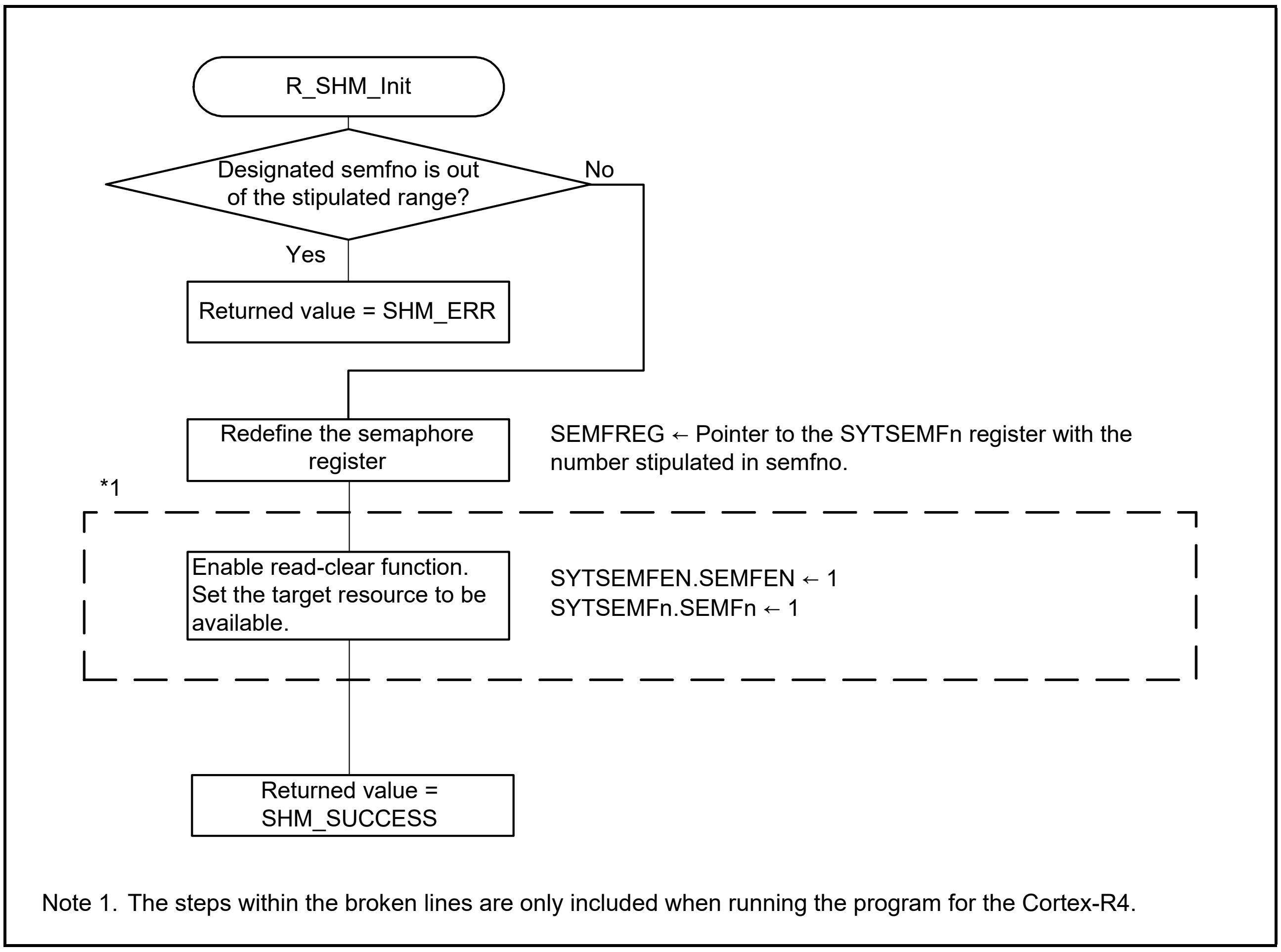

R_SHM_Init
Designated semfno is out of the stipulated range?
No
Yes
Returned value = SHM_ERR
Redefine the semaphore register
SEMFREG ← Pointer to the SYTSEMFn register with the number stipulated in semfno.
*1
Enable read-clear function. Set the target resource to be available.
SYTSEMFEN.SEMFEN ← 1
SYTSEMFn.SEMFn ← 1
Returned value = SHM_SUCCESS
Note 1. The steps within the broken lines are only included when running the program for the Cortex-R4.

Figure 6.5 Initialization Processing of Shared Memory Driver

6.10.2 Processing to Copy Ranges of Memory to and from the Shared Memory Area

Figure 6.6 shows the processing to copy ranges of memory to and from the shared memory area.

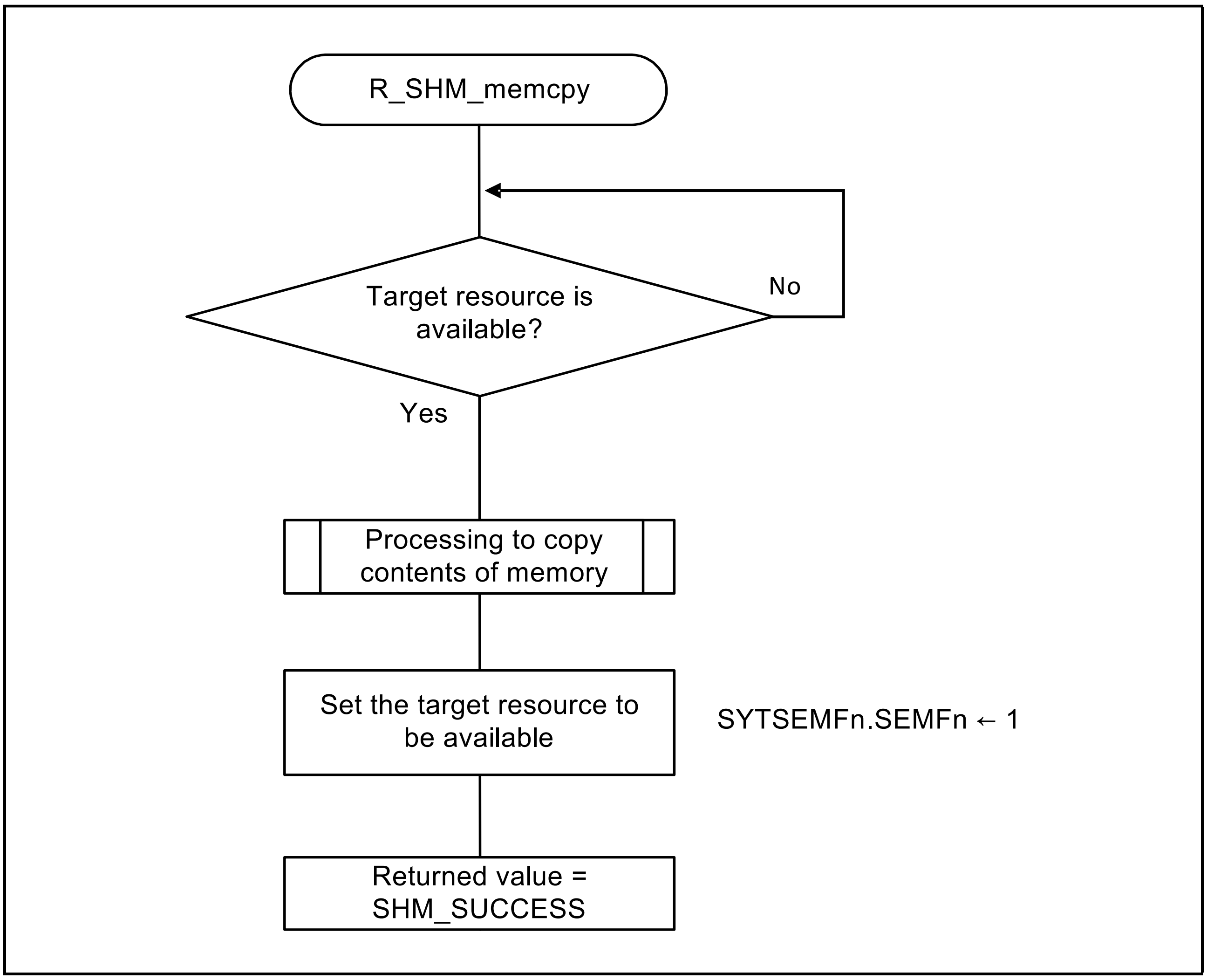

R_SHM_memcpy
Target resource is available?
No
Yes
Processing to copy contents of memory
Set the target resource to be available
SYTSEMFn.SEMFn ← 1
Returned value = SHM_SUCCESS

Figure 6.6 Processing to Copy Ranges of Memory to and from the Shared Memory Area

6.10.3 Processing to Load a Value (4-Byte Unsigned Int Type) to and from the Shared Memory Area

Figure 6.7 shows the processing to load four bytes of unsigned int type data to and from the shared memory area.

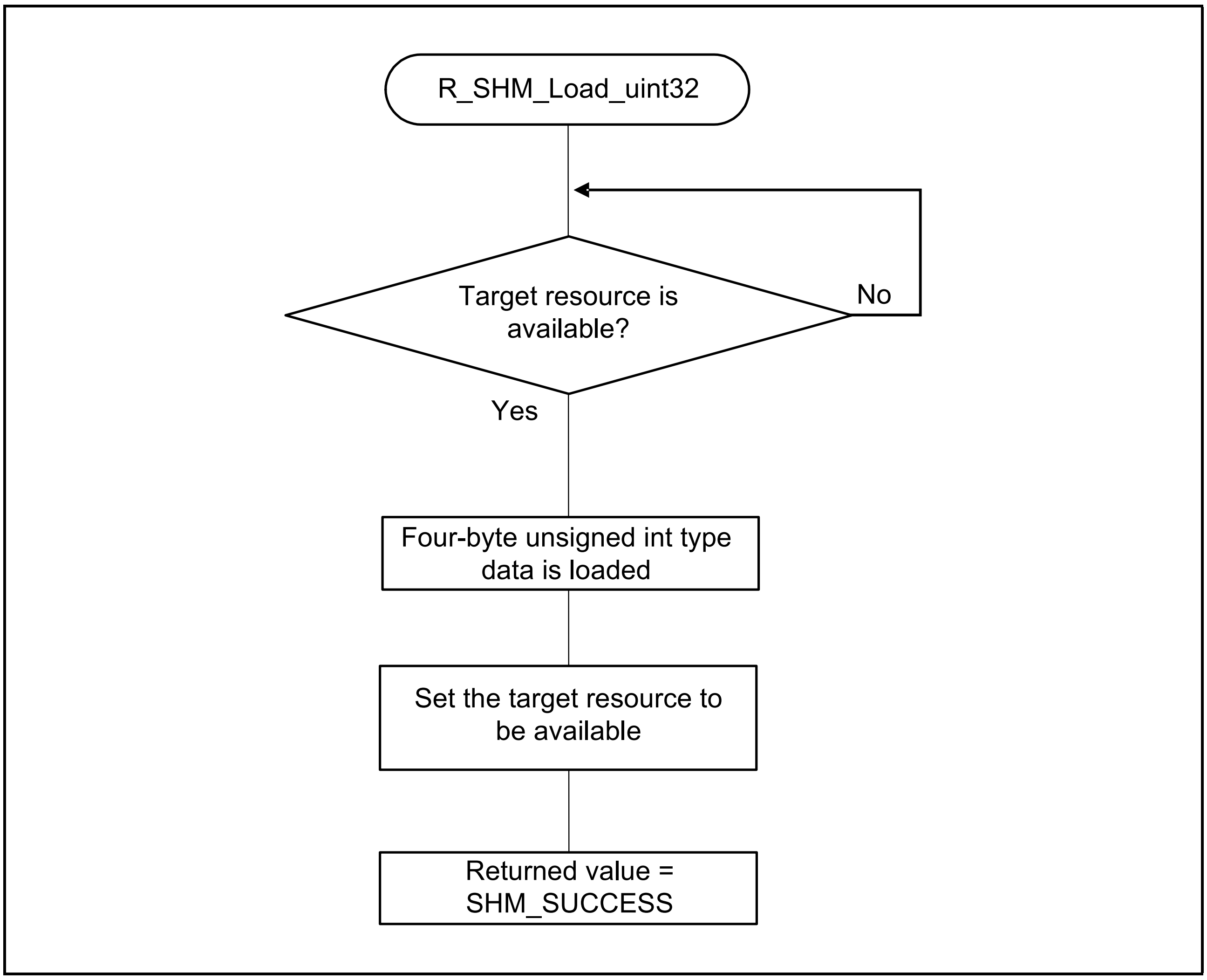

R_SHM_Load_uint32
Target resource is available?
No
Yes
Four-byte unsigned int type data is loaded
Set the target resource to be available
Returned value = SHM_SUCCESS

Figure 6.7 Processing to Load a Value (4-Byte Unsigned Int Type) to and from the Shared Memory Area

6.10.4 Processing to Load a Value (4-Byte Signed Int Type) to and from the Shared Memory Area

For the flowchart of this processing, see Figure 6.7. Note that the value loaded in this processing is of the four-byte signed int type.

6.10.5 Processing to Load a Value (2-Byte Unsigned Int Type) to and from the Shared Memory Area

For the flowchart of this processing, see Figure 6.7. Note that the value loaded in this processing is of the two-byte unsigned int type.

6.10.6 Processing to Load a Value (2-Byte Signed Int Type) to and from the Shared Memory Area

For the flowchart of this processing, see Figure 6.7. Note that the value loaded in this processing is of the two-byte signed int type.

6.10.7 Processing to Load a Value (1-Byte Unsigned Int Type) to and from the Shared Memory Area

For the flowchart of this processing, see Figure 6.7. Note that the value loaded in this processing is of the one-byte unsigned int type.

6.10.8 Processing to Load a Value (1-Byte Signed Int Type) to and from the Shared Memory Area

For the flowchart of this processing, see Figure 6.7. Note that the value loaded in this processing is of the one-byte signed int type.

6.10.9 Main Processing (Cortex-R4)

Figure 6.8 shows the flowchart of main processing of the Cortex-R4.

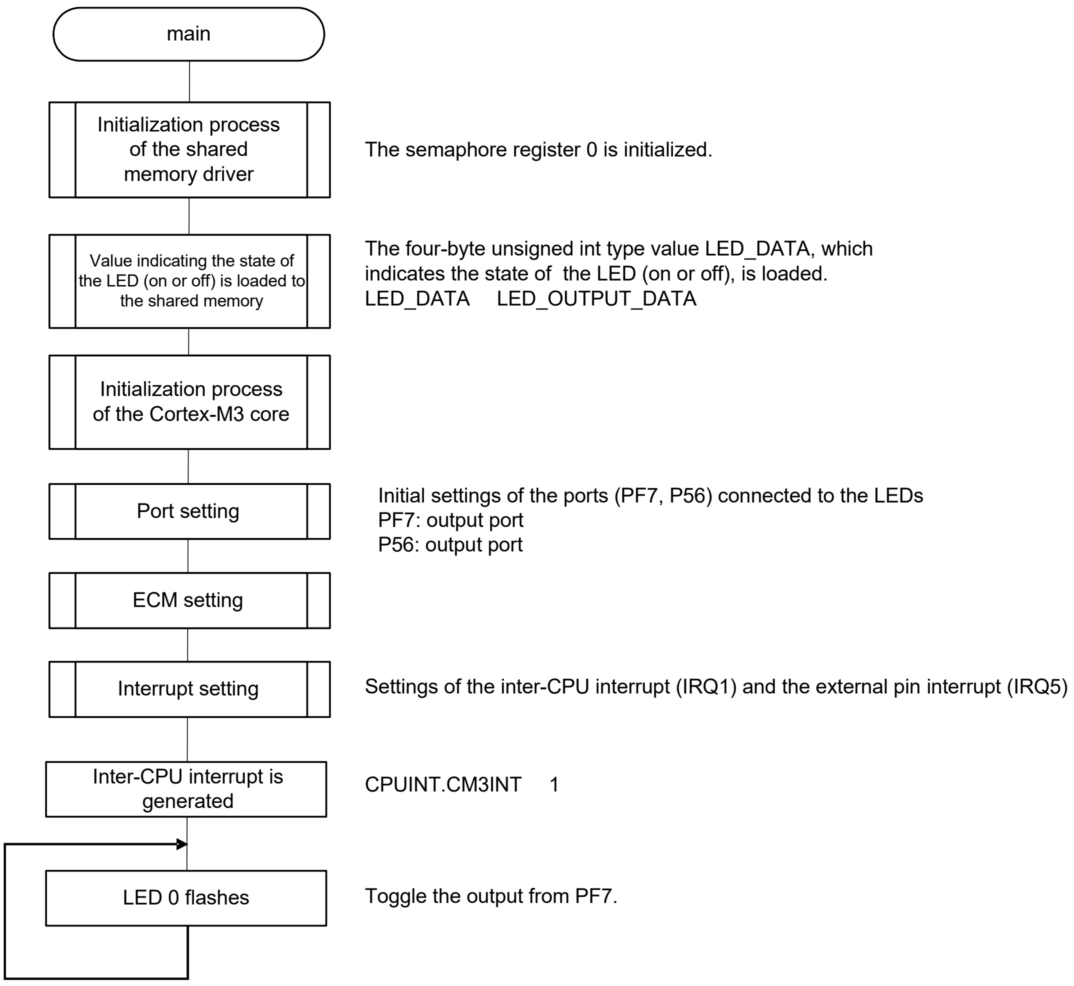
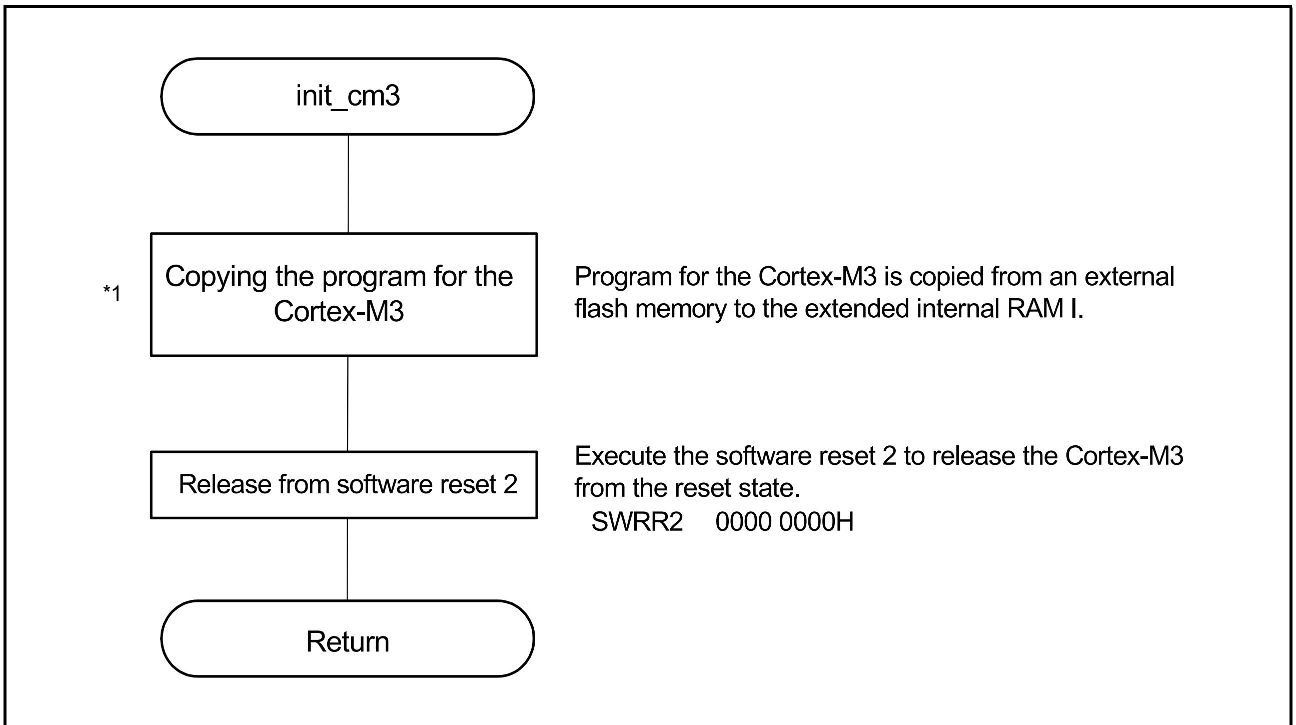

main
Initialization process of the shared memory driver
The semaphore register 0 is initialized.
Value indicating the state of the LED (on or off) is loaded to the shared memory
The four-byte unsigned int type value LED_DATA, which indicates the state of the LED (on or off), is loaded.
LED_DATA LED_OUTPUT_DATA
Initialization process of the Cortex-M3 core
Port setting
Initial settings of the ports (PF7, P56) connected to the LEDs
PF7: output port
P56: output port
ECM setting
Interrupt setting
Settings of the inter-CPU interrupt (IRQ1) and the external pin interrupt (IRQ5)
Inter-CPU interrupt is generated
CPUINT.CM3INT 1
LED 0 flashes
Toggle the output from PF7.

Figure 6.8 Main Processing (Cortex-R4)

6.10.10 Initialization Processing of the Cortex-M3 Core

Figure 6.9 shows the flowchart of initialization processing of the Cortex-M3 core.


init_cm3
*1
Copying the program for the Cortex-M3
Program for the Cortex-M3 is copied from an external flash memory to the extended internal RAM I.
Release from software reset 2
Execute the software reset 2 to release the Cortex-M3 from the reset state.
SWRR2 0000 0000H
Return

Note 1. This step is not included when the program is to be executed from the RAM.

Figure 6.9 Initialization Processing of Cortex-M3

6.10.11 Main Processing (Cortex-M3)

Figure 6.10 shows the flowchart of main processing of the Cortex-M3.

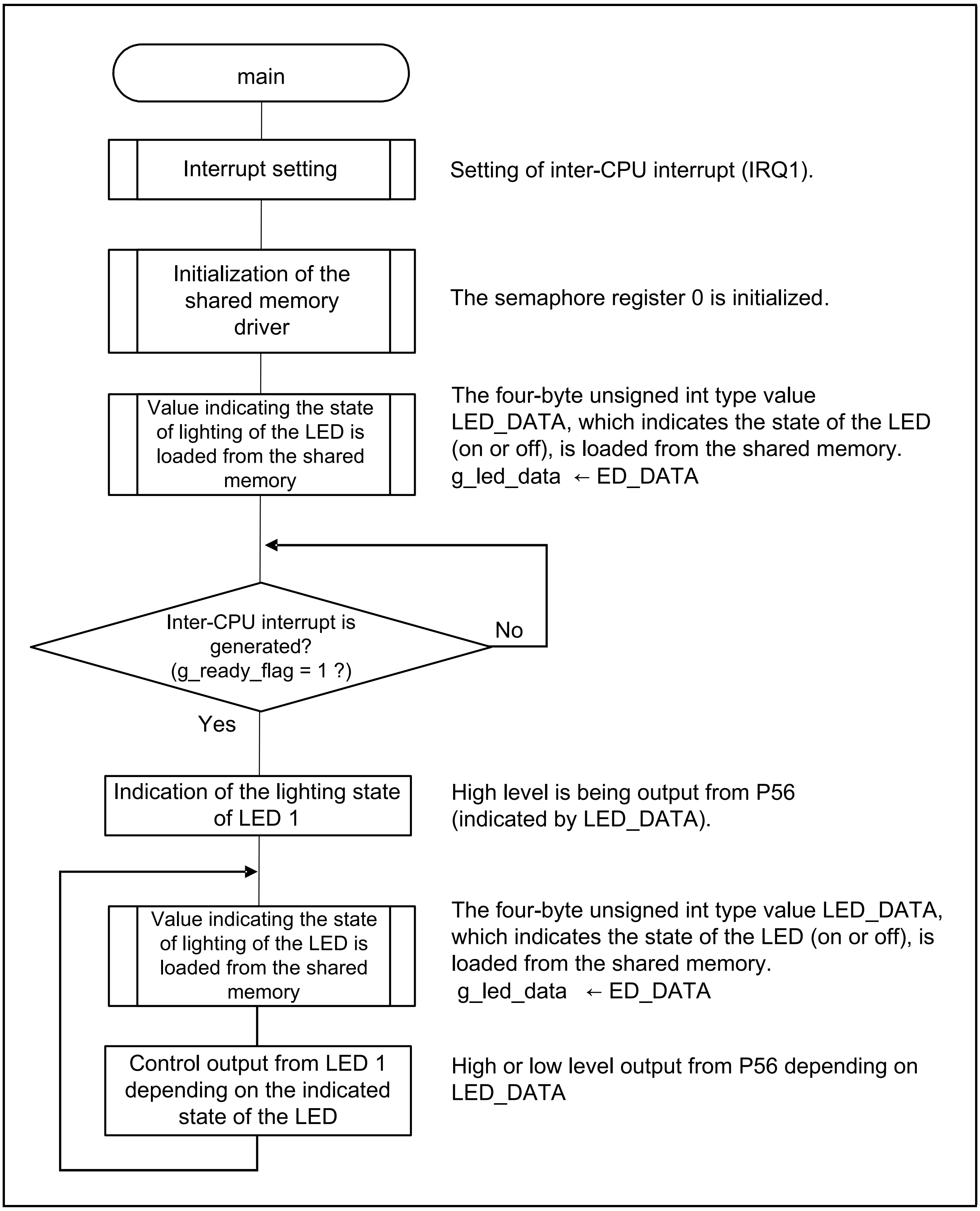

main
Interrupt setting
Setting of inter-CPU interrupt (IRQ1).
Initialization of the shared memory driver
The semaphore register 0 is initialized.
Value indicating the state of lighting of the LED is loaded from the shared memory
The four-byte unsigned int type value LED_DATA, which indicates the state of the LED (on or off), is loaded from the shared memory.
g_led_data ← ED_DATA
Inter-CPU interrupt is generated? (g_ready_flag = 1 ?)
No
Yes
Indication of the lighting state of LED 1
High level is being output from P56 (indicated by LED_DATA).
Value indicating the state of lighting of the LED is loaded from the shared memory
The four-byte unsigned int type value LED_DATA, which indicates the state of the LED (on or off), is loaded from the shared memory.
g_led_data ← ED_DATA
Control output from LED 1 depending on the indicated state of the LED
High or low level output from P56 depending on LED_DATA

Figure 6.10 Main Processing (Cortex-M3)

6.10.12 R_IRQ9 Interrupt (IRQ Pin Interrupt 5) Processing

Figure 6.11 shows the flowchart of R_IRQ9 interrupt (IRQ Pin interrupt 5) processing.

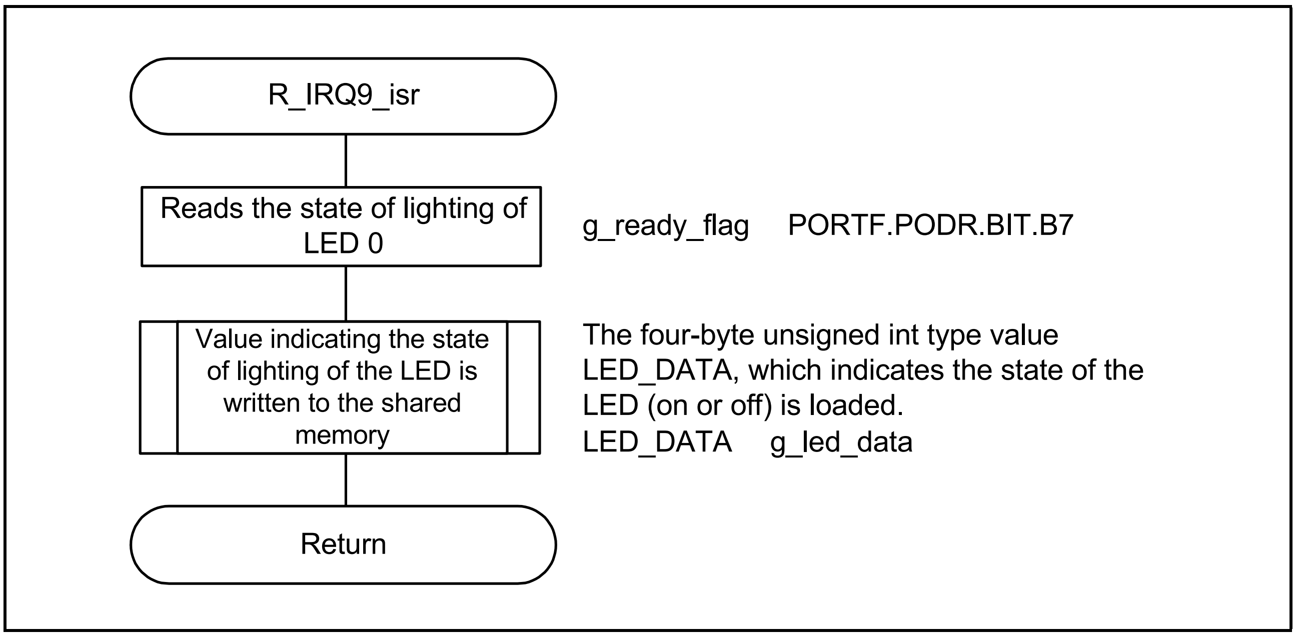

R_IRQ9_isr
Reads the state of lighting of LED 0
g_ready_flag PORTF.PODR.BIT.B7
Value indicating the state of lighting of the LED is written to the shared memory
The four-byte unsigned int type value LED_DATA, which indicates the state of the LED (on or off) is loaded.
LED_DATA g_led_data
Return

Figure 6.11 R_IRQ9 Interrupt (IRQ Pin Interrupt 5) Processing

6.10.13 Inter-CPU Interrupt Processing

Figure 6.12 shows the flowchart of inter-CPU interrupt processing.

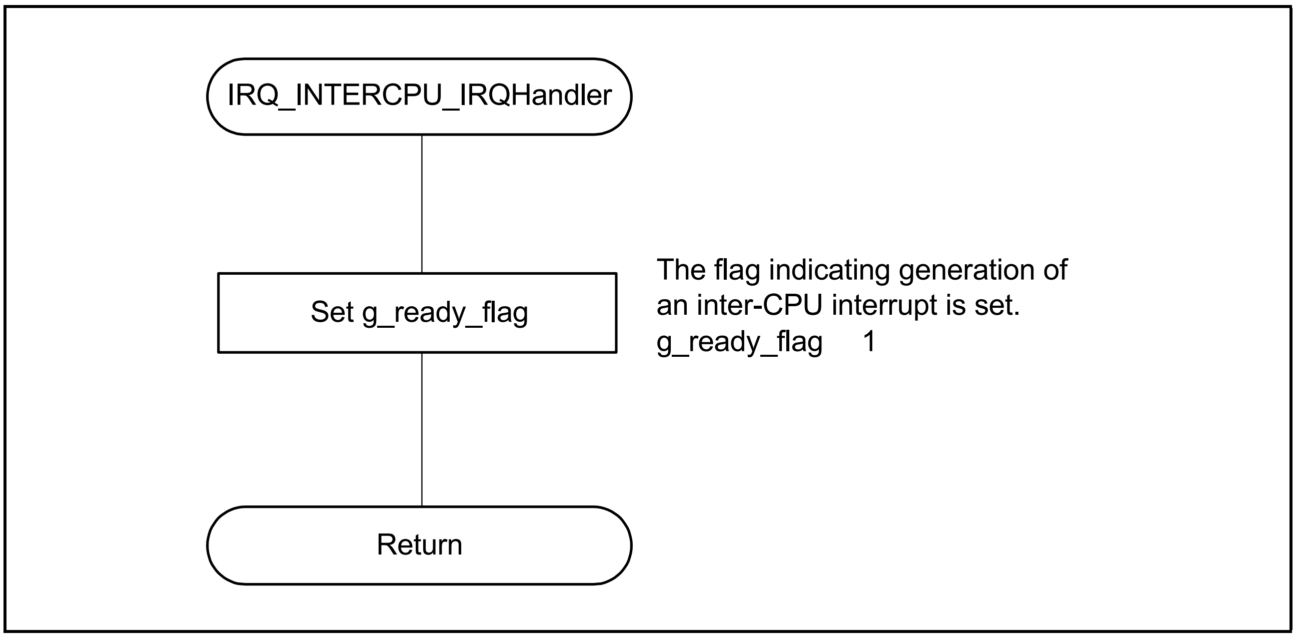

IRQ_INTERCPU_IRQHandler
Set g_ready_flag
The flag indicating generation of an inter-CPU interrupt is set.
g_ready_flag 1
Return

Figure 6.12 Inter-CPU Interrupt Processing

7. Sample Program

Download the sample program from the Renesas Electronics website.

8. Related Documents

- User's Manual: Hardware
  RZ/T1 Group User's Manual: Hardware
  Download the latest version from the Renesas Electronics website.

  RZ/T1 Evaluation Board RTK7910022C00000BR User's Manual
  Download the latest version from the Renesas Electronics website.

- Technical Update/Technical News
  Download the latest version from the Renesas Electronics website.

- User's Manual: Development Environment
  The latest version for the IAR integrated development environment (IAR Embedded Workbench® for Arm) is available from the IAR Systems website.
  The latest version for the Arm integrated development environment (Development Studio 5^{TM}) is available from the Arm website.

Appendix-1. Supplementary Notes on Development Environments

This section shows the settings required to perform debugging for the sample program for the Cortex-R4 core and the R-IN engine (Cortex-M3 core) in each available development environment and the debugging procedures.

(EWARM from IAR systems)

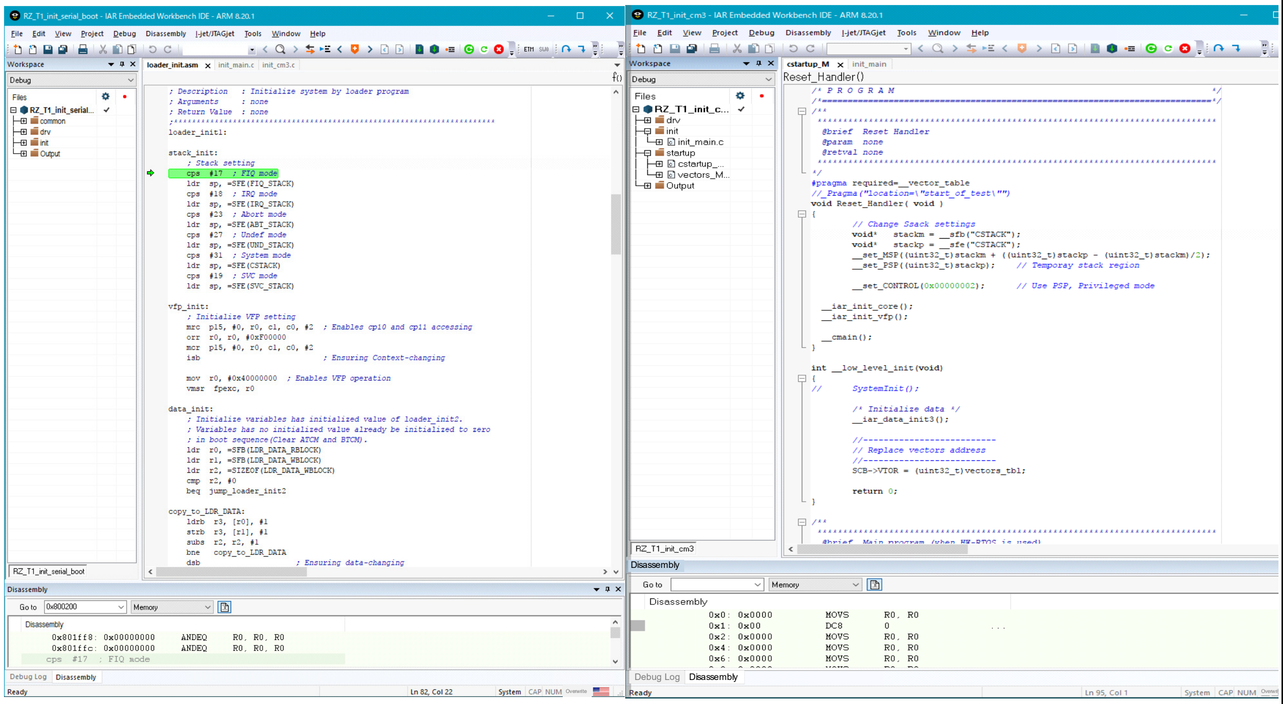
<Steps up to debugging of the sample program (when booting is through the SPI)>

(1) Start up the EWARM environment. Go to [File] > [Open] > [Workspace] and then specify RZ_T1_init_cm3.eww. The project for the Cortex-M3 will start.

(2) Select the [Rebuild All] item from the [Project] menu to rebuild the project. The binary file RZ_T1_init_cm3.bin is generated.

(3) Start up another EWARM environment. Go to [File] > [Open] > [Workspace] and then specify RZ_T1_init_serial.eww. The project for the Cortex-R4 will start. Rebuild the project in the same manner as in step (2).

(4) While the RZ/T1 evaluation board and I-jet are connected, click on the [Download and debug] button in the [Project] toolbar in the project for the Cortex-R4. Similarly, click on the [Attach to Running Target] button in the [Project] toolbar in the project for the Cortex-M3.


RZ_T1_init_serial_boot - IAR Embedded Workbench IDE - ARM 8.20.1
File Edit View Project Debug Disassembly I-jet/JTAGjet Tools Window Help
Workspace
Debug
Files
RZ_T1_init_serial...
common
drv
init
Output
RZ_T1_init_serial_boot
loader_init.asm
init_main.c
init_cm3.c

```
; Description   : Initialize system by loader program
; Arguments     : none
; Return Value  : none
;*********************************************************************
loader_init1:

stack_init:
    ; Stack setting
    cps  #17  ; FIQ mode
    ldr  sp, =SFE(FIQ_STACK)
    cps  #18  ; IRQ mode
    ldr  sp, =SFE(IRQ_STACK)
    cps  #23  ; Abort mode
    ldr  sp, =SFE(ABT_STACK)
    cps  #27  ; Undef mode
    ldr  sp, =SFE(UND_STACK)
    cps  #31  ; System mode
    ldr  sp, =SFE(CSTACK)
    cps  #19  ; SVC mode
    ldr  sp, =SFE(SVC_STACK)

vfp_init:
    ; Initialize VFP setting
    mrc  p15, #0, r0, c1, c0, #2  ; Enables cp10 and cp11 accessing
    orr  r0, r0, #0xF00000
    mcr  p15, #0, r0, c1, c0, #2
    isb                           ; Ensuring Context-changing

    mov  r0, #0x40000000  ; Enables VFP operation
    vmsr  fpexc, r0

data_init:
    ; Initialize variables has initialized value of loader_init2.
    ; Variables has no initialized value already be initialized to zero
    ; in boot sequence(Clear ATCM and BTCM).
    ldr  r0, =SFB(LDR_DATA_RBLOCK)
    ldr  r1, =SFB(LDR_DATA_WBLOCK)
    ldr  r2, =SIZEOF(LDR_DATA_WBLOCK)
    cmp  r2, #0
    beq  jump_loader_init2

copy_to_LDR_DATA:
    ldrb  r3, [r0], #1
    strb  r3, [r1], #1
    subs  r2, r2, #1
    bne   copy_to_LDR_DATA
    dsb                        ; Ensuring data-changing
```

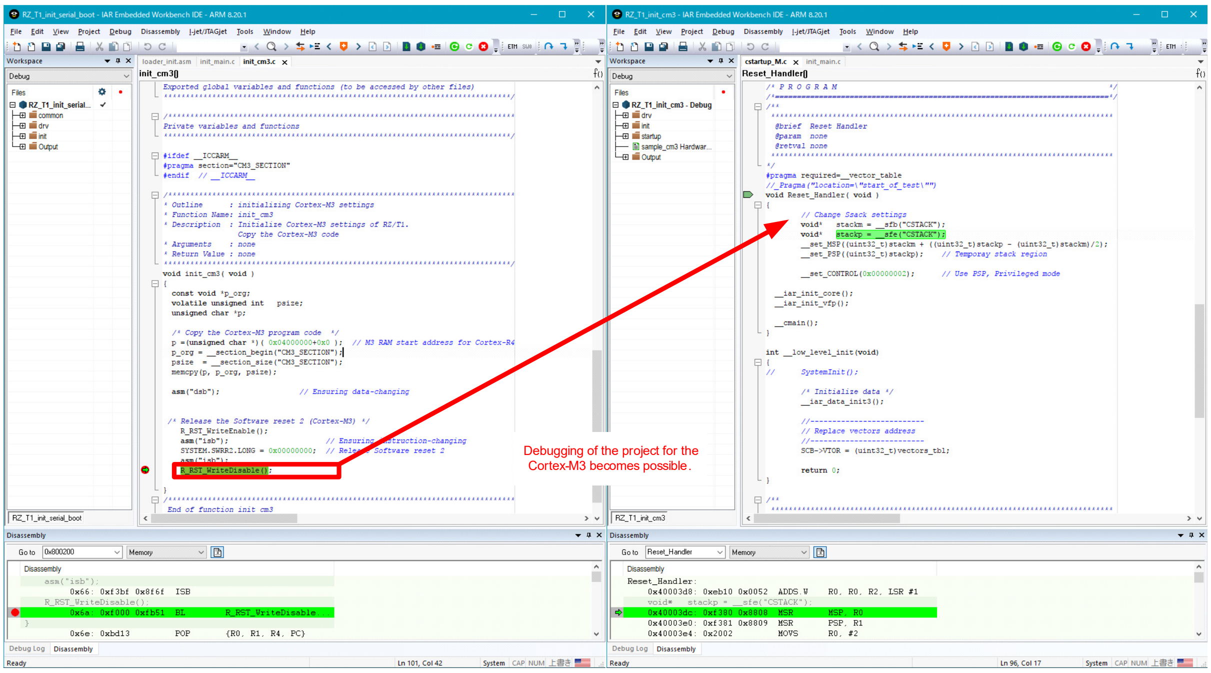
```
Disassembly
    0x801ff8: 0x00000000    ANDEQ    R0, R0, R0
    0x801ffc: 0x00000000    ANDEQ    R0, R0, R0
  cps  #17  ; FIQ mode
```

```
/* P R O G R A M                                                          */
/*========================================================================*/
/**
 **************************************************************************
  @brief  Reset Handler
  @param  none
  @retval none
 **************************************************************************
*/
#pragma required=__vector_table
//_Pragma("location=\"start_of_test\"")
void Reset_Handler( void )
{
        // Change Ssack settings
        void*   stackm = __sfb("CSTACK");
        void*   stackp = __sfe("CSTACK");
        __set_MSP((uint32_t)stackm + ((uint32_t)stackp - (uint32_t)stackm)/2);
        __set_PSP((uint32_t)stackp);    // Temporay stack region

        __set_CONTROL(0x00000002);      // Use PSP, Privileged mode

  __iar_init_core();
  __iar_init_vfp();

  __cmain();
}

int __low_level_init(void)
{
//      SystemInit();

        /* Initialize data */
        __iar_data_init3();

        //-------------------------
        // Replace vectors address
        //-------------------------
        SCB->VTOR = (uint32_t)vectors_tbl;

        return 0;
}

/**
 **************************************************************************
  @brief  Main program (when HW-RTOS is used)
```

```
Disassembly
        0x0: 0x0000       MOVS     R0, R0
        0x1: 0x00         DC8      0              ...
        0x2: 0x0000       MOVS     R0, R0
        0x4: 0x0000       MOVS     R0, R0
        0x6: 0x0000       MOVS     R0, R0
```

Note: Debugging of the project for the Cortex-M3 is not possible at this point because the core is in the reset state. Debugging of the project for the Cortex-R4 can proceed.

(5) Release the Cortex-M3 core from the reset state by the project for the Cortex-R4 by executing the process in its function init_cm3. The Cortex-M3 core is released from the reset state and debugging of the program for the Cortex-M3 core is then possible.


RZ_T1_init_serial_boot - IAR Embedded Workbench IDE - ARM 8.20.1
RZ_T1_init_cm3 - IAR Embedded Workbench IDE - ARM 8.20.1
Debugging of the project for the Cortex-M3 becomes possible.

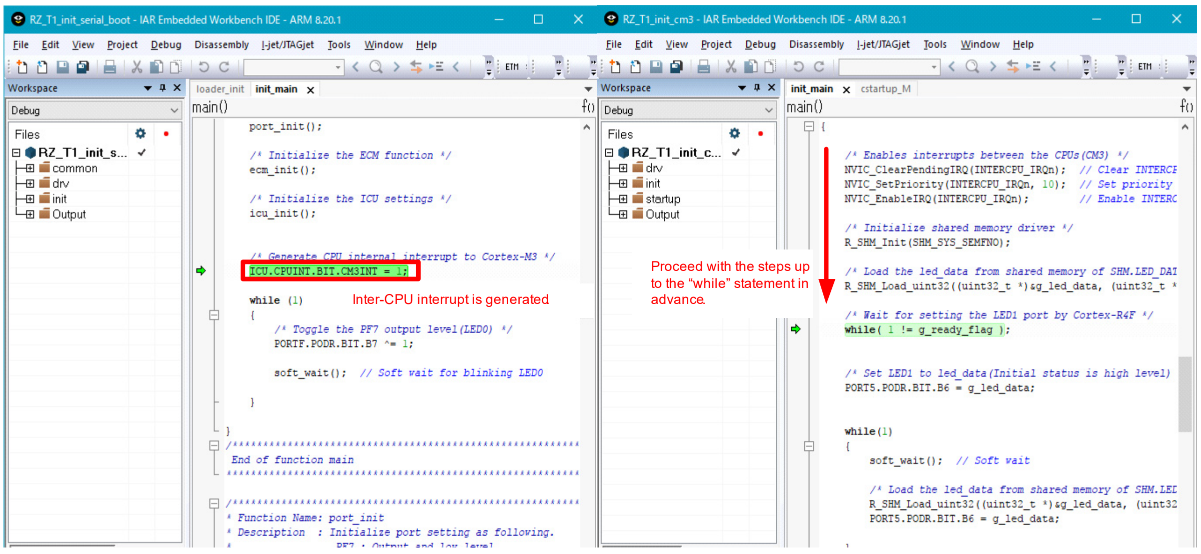
```
init_cm3()
 Exported global variables and functions (to be accessed by other files)
 ***********************************************************************/

/***********************************************************************
Private variables and functions
***********************************************************************/

#ifdef __ICCARM__
#pragma section="CM3_SECTION"
#endif  // __ICCARM__

/***********************************************************************
* Outline      : initializing Cortex-M3 settings
* Function Name: init_cm3
* Description  : Initialize Cortex-M3 settings of RZ/T1.
                 Copy the Cortex-M3 code
* Arguments    : none
* Return Value : none
***********************************************************************/
void init_cm3( void )
{
  const void *p_org;
  volatile unsigned int   psize;
  unsigned char *p;

  /* Copy the Cortex-M3 program code  */
  p =(unsigned char *)( 0x04000000+0x0 );  // M3 RAM start address for Cortex-R4
  p_org = __section_begin("CM3_SECTION");
  psize  = __section_size("CM3_SECTION");
  memcpy(p, p_org, psize);

  asm("dsb");                  // Ensuring data-changing


 /* Release the Software reset 2 (Cortex-M3) */
    R_RST_WriteEnable();
    asm("isb");                     // Ensuring instruction-changing
    SYSTEM.SWRR2.LONG = 0x00000000;  // Release Software reset 2
    asm("isb");
    R_RST_WriteDisable();

}
/***********************************************************************
 End of function init_cm3
```

```
Disassembly
    asm("isb");
        0x66: 0xf3bf 0x8f6f   ISB
    R_RST_WriteDisable();
        0x6a: 0xf000 0xfb51   BL        R_RST_WriteDisable...
    }
        0x6e: 0xbd13          POP       {R0, R1, R4, PC}
```

```
Reset_Handler()
/* P R O G R A M                                                       */
/*=====================================================================*/
/**
 ***********************************************************************
  @brief  Reset Handler
  @param  none
  @retval none
 ***********************************************************************
*/
#pragma required=__vector_table
//_Pragma("location=\"start_of_test\"")
void Reset_Handler( void )
{
        // Change Ssack settings
        void*   stackm = __sfb("CSTACK");
        void*   stackp = __sfe("CSTACK");
        __set_MSP((uint32_t)stackm + ((uint32_t)stackp - (uint32_t)stackm)/2);
        __set_PSP((uint32_t)stackp);    // Temporay stack region

        __set_CONTROL(0x00000002);      // Use PSP, Privileged mode

  __iar_init_core();
  __iar_init_vfp();

  __cmain();
}

int __low_level_init(void)
{
//      SystemInit();

        /* Initialize data */
        __iar_data_init3();

        //-------------------------
        // Replace vectors address
        //-------------------------
        SCB->VTOR = (uint32_t)vectors_tbl;

        return 0;
}

/**
 ***********************************************************************
```

```
Disassembly
Reset_Handler:
    0x400003d8: 0xeb10 0x0052  ADDS.W    R0, R0, R2, LSR #1
    void*   stackp = __sfe("CSTACK");
    0x400003dc: 0xf380 0x8808  MSR       MSP, R0
    0x400003e0: 0xf381 0x8809  MSR       PSP, R1
    0x400003e4: 0x2002         MOVS      R0, #2
```

(6) This sample program uses inter-CPU interrupts from the Cortex-R4 core to the Cortex-M3 core. If you want to debug in the order of the actual operation, execute the project for the Cortex-M3 in advance through the interrupt settings and loading of data from the shared memory, and then execute the project for the Cortex-R4 to generate the inter-CPU interrupt.


RZ_T1_init_serial_boot - IAR Embedded Workbench IDE - ARM 8.20.1
RZ_T1_init_cm3 - IAR Embedded Workbench IDE - ARM 8.20.1
Inter-CPU interrupt is generated
Proceed with the steps up to the "while" statement in advance.

```
main()
    port_init();

    /* Initialize the ECM function */
    ecm_init();

    /* Initialize the ICU settings */
    icu_init();


    /* Generate CPU internal interrupt to Cortex-M3 */
    ICU.CPUINT.BIT.CM3INT = 1;

    while (1)
    {
        /* Toggle the PF7 output level(LED0) */
        PORTF.PODR.BIT.B7 ^= 1;

        soft_wait();  // Soft wait for blinking LED0

    }

}
/**********************************************************************
 End of function main
 *********************************************************************

/**********************************************************************
* Function Name: port_init
* Description  : Initialize port setting as following.
```

```
main()
{

    /* Enables interrupts between the CPUs(CM3) */
    NVIC_ClearPendingIRQ(INTERCPU_IRQn);  // Clear INTERCF
    NVIC_SetPriority(INTERCPU_IRQn, 10);  // Set priority
    NVIC_EnableIRQ(INTERCPU_IRQn);        // Enable INTERC

    /* Initialize shared memory driver */
    R_SHM_Init(SHM_SYS_SEMFNO);

    /* Load the led_data from shared memory of SHM.LED_DAT
    R_SHM_Load_uint32((uint32_t *)&g_led_data, (uint32_t *

    /* Wait for setting the LED1 port by Cortex-R4F */
    while( 1 != g_ready_flag );


    /* Set LED1 to led_data(Initial status is high level)
    PORT5.PODR.BIT.B6 = g_led_data;


    while(1)
    {
        soft_wait();  // Soft wait

        /* Load the led_data from shared memory of SHM.LED
        R_SHM_Load_uint32((uint32_t *)&g_led_data, (uint32
        PORT5.PODR.BIT.B6 = g_led_data;
```

(7) Debugging of the cores can then proceed without following given procedures.

<Project settings for the sample program>

Settings of EWARM related to the sample program for making initial settings of the microcomputers incorporating the R-IN engine are shown below. The details of the project for the Cortex-R4 core are given in the "RZ/T1 Group Application Note: Initial Settings."

Table. Settings in the Project for the Sample Program (Cortex-R4)

		Build Action	
Build action setting	Pre-build command line	cmd /c "copy /Y "$PROJ_DIR$¥..¥..¥Cortex-M3¥Device¥Renesas¥RIN_Engine¥Source¥Project¥Init¥IAR¥Debug¥Exe¥RZ_T1_init_cm3.bin" "$PROJ_DIR$¥cm3.bin¥RZ_T1_init_cm3.bin""	
		Linker	
Input*1	Keep symbols	CM3_SECTION	
	Raw binary image	File: $PROJ_DIR$¥cm3.bin¥RZ_T1_init_cm3.bin Symbol: CM3_SECTION Section: CM3_SECTION Alignment: 4	
		Debugger	
Extra options	Use command line options	Switch checkmarks on	
		Synchronous	Asynchronous
		--drv_default_breakpoint=1 --macro_param_etm_rute=2 --jet_emu_param=UseCTI=0*2	--drv_default_breakpoint=1 --macro_param_etm_rute=2 --jet_sigprobe_opt=shared
Multicore	Symmetric multicore	The number of cores: 1	
	Asymmetric multicore Enable multicore master mode	Synchronous	Asynchronous
		Switch checkmarks on Port: 53461 Slave work space: $PROJ_DIR$\..\..\Cortex-M3\Device\Rensas\RIN_Engine\Project\Init\IAR\RZ_T1_init_cm3.eww Slave project: RZ_T1_init_cm3 Slave configuration: Debug Attach the slave to running target: Switch checkmarks on	Don't switch checkmarks on

Note 1. The binary image for the Cortex-M3 is fetched into the section CM3_SECTION.
Note 2. This is required for EWARM V8.10.1 or later.

Table. Settings in the Project for the Sample Program (Cortex-M3)

		Output Converter	
Output	Generation of additional output file	Switch checkmarks on: Output format: binary	
	Output file Override default	Switch checkmarks on: $PROJ_DIR$¥RZ_T1_init_cm3.bin	
		Debugger	
Extra options	Use command line options	Synchronous	Asynchronous
		--drv_default_breakpoint=1	--drv_default_breakpoint=1 --jet_sigprobe_opt=shared

Details on the option settings are described in the C-SPY debugging guide C-SPY Macro from the IAR Systems and in related documents.

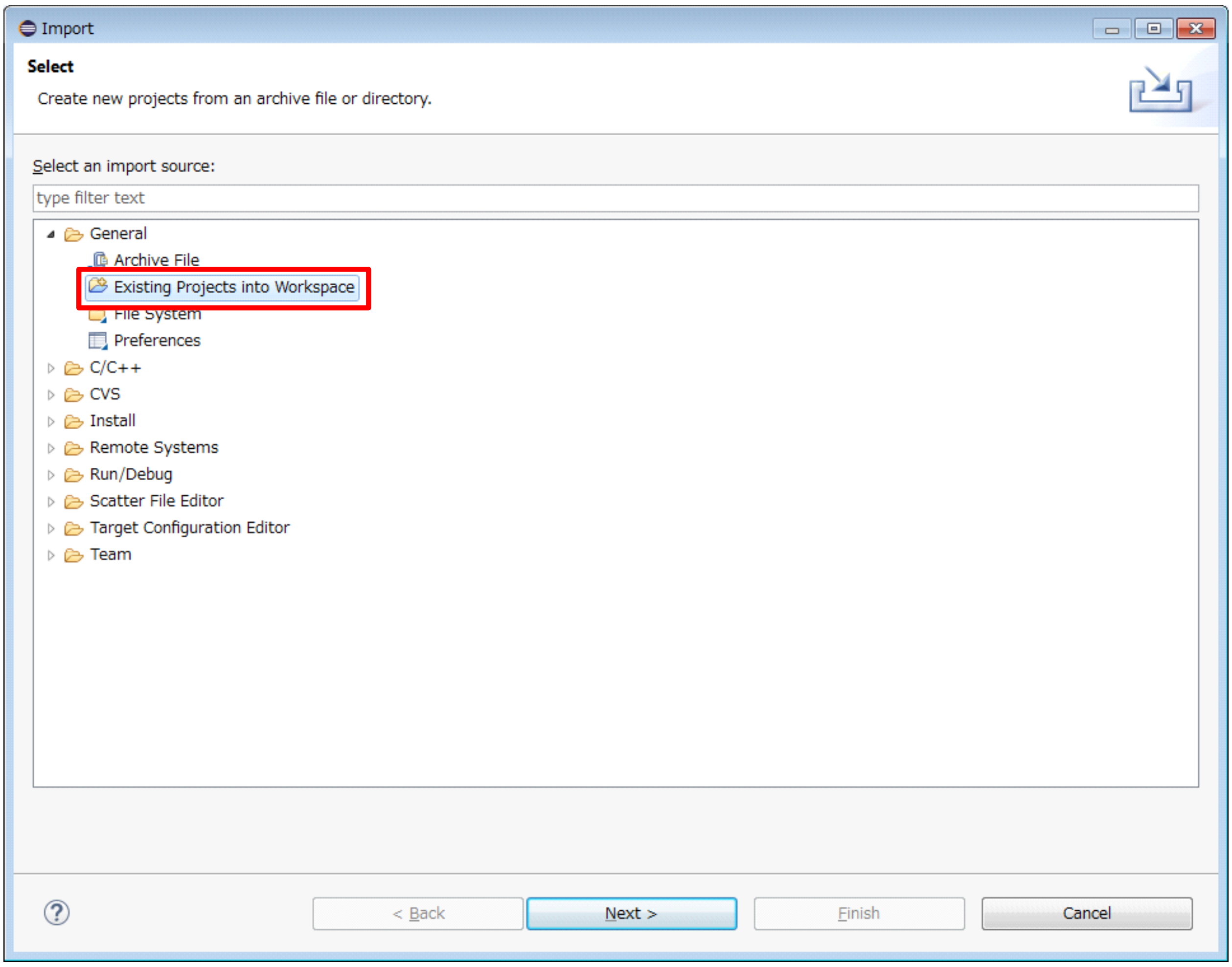
(DS-5 from Arm)

<Steps up to debugging of the sample program (when booting is through the SPI)>

(1) Create a folder to use as a workspace on your PC, where the sample program will be stored.

(2) Start up the DS-5 environment. Specify the folder in which you stored the sample programs in step 1 as the workspace.

(3) Go to [File] > [Import]. On the [Import] window, select [Existing Projects into Workspace] in the [General] folder and click on the [Next] button.


Import

Select

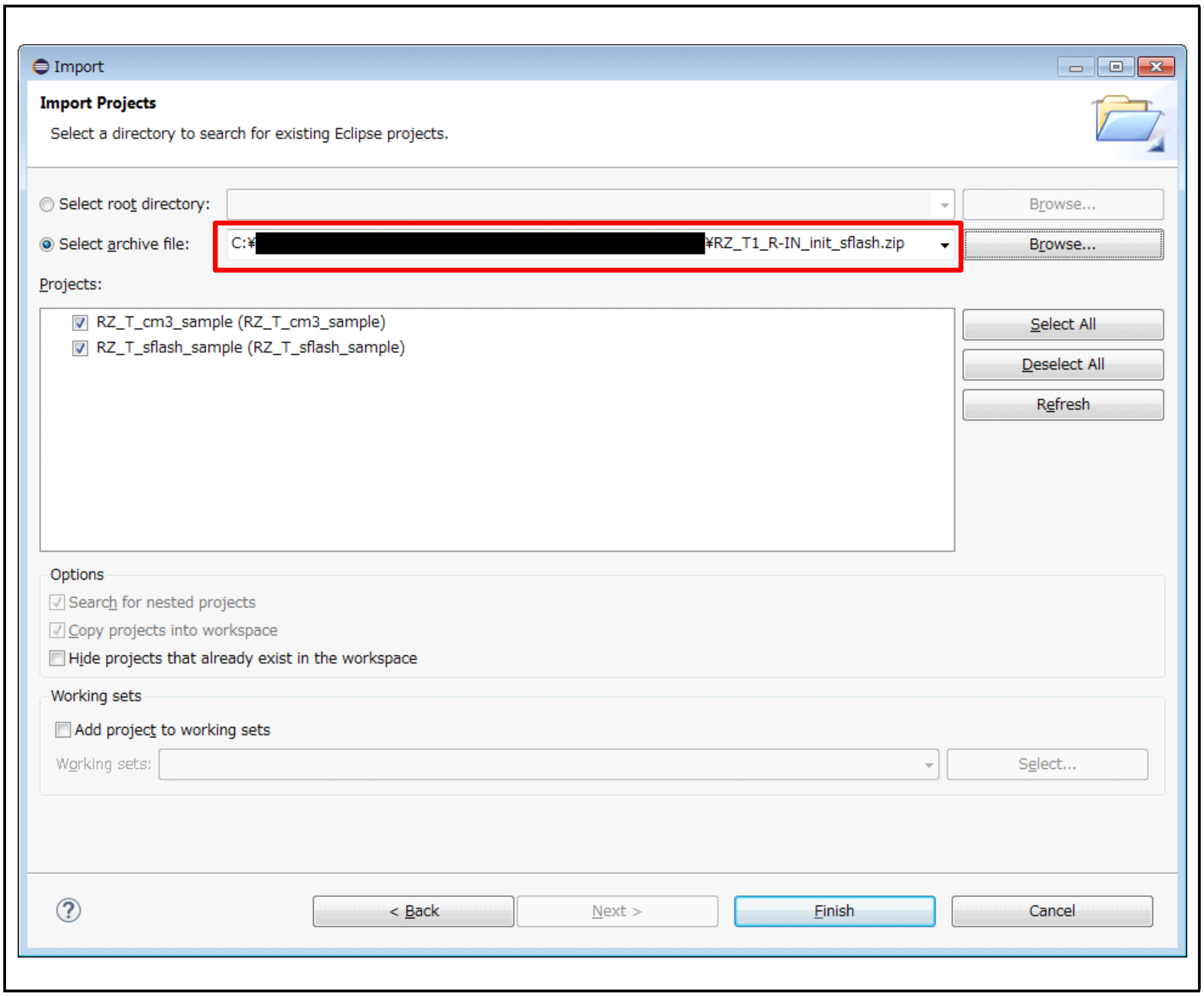
Create new projects from an archive file or directory.

Select an import source:

type filter text

- General
  - Archive File
  - Existing Projects into Workspace
  - File System
  - Preferences
- C/C++
- CVS
- Install
- Remote Systems
- Run/Debug
- Scatter File Editor
- Target Configuration Editor
- Team

(4) Select the [Select archive file:] radio button and click on the [Browse..] button. Select the compressed program file "RZ_T1_R-IN_init_sflash.zip" on the list in the window and click on the [Finish] button.



Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory: [ ]

[✓ Select archive file:] C:¥[redacted]¥RZ_T1_R-IN_init_sflash.zip

Projects:

- ☑ RZ_T_cm3_sample (RZ_T_cm3_sample)
- ☑ RZ_T_sflash_sample (RZ_T_sflash_sample)

Options

- ☑ Search for nested projects
- ☑ Copy projects into workspace
- ☐ Hide projects that already exist in the workspace

Working sets

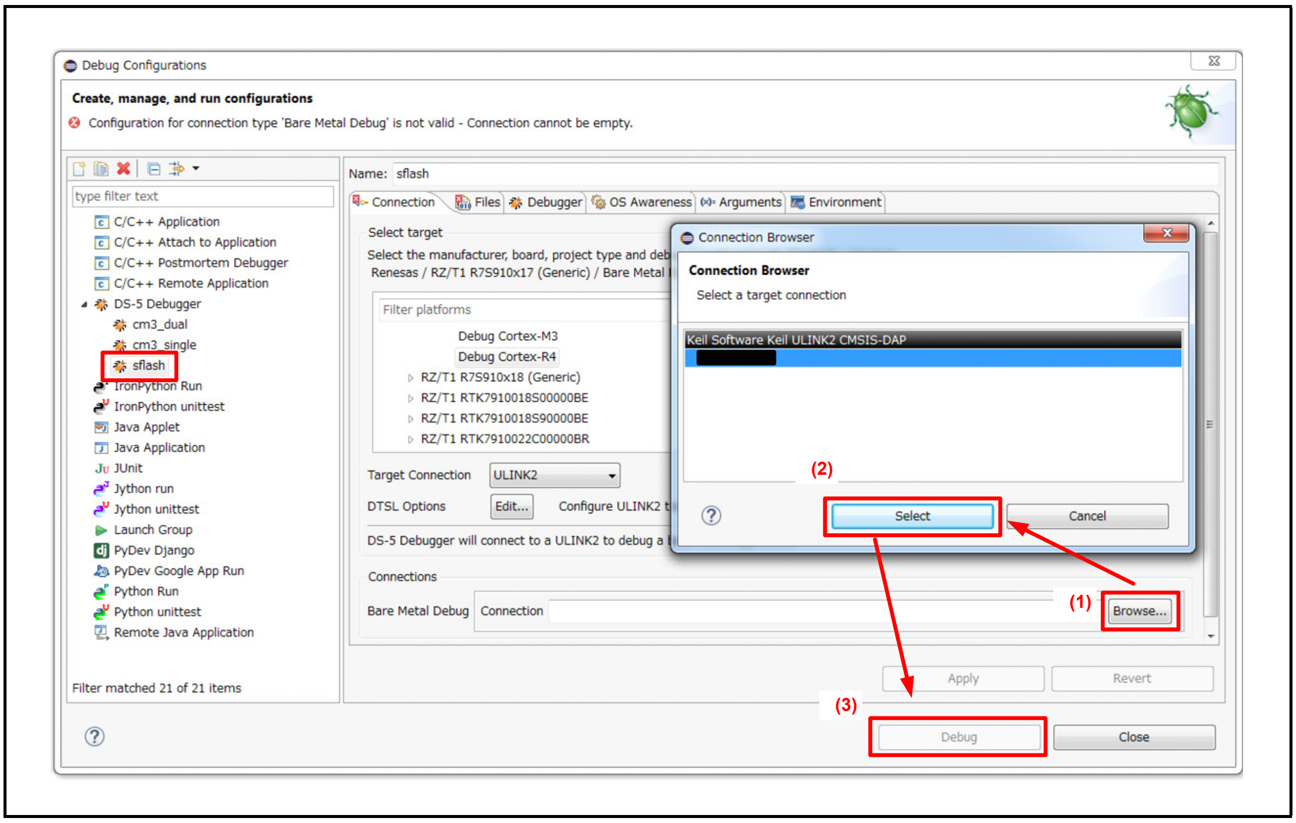
- ☐ Add project to working sets

Working sets:

(5) Select [Build All] from the [Project] menu. On completion of building, select [Run] from the [Project] menu and then select [Debug Configurations].

(6) While the RZ/T1 evaluation board and ULINK2 are connected, go to [DS-5 Debugger] and select [sflash] to show the window for configurations. Then, click on [Browse] and open the connection browser window as shown below. Select the target connection from the list in the window. Click on [Debug] in the debug configurations window and start debugging of the Cortex-R4.

Note: If you are using the sample program to be run from the RAM, skip the following step (7) to step (9), so only release the debug connection ram (Cortex-R4) from reset.



Create, manage, and run configurations

Configuration for connection type 'Bare Metal Debug' is not valid - Connection cannot be empty.

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- DS-5 Debugger
  - cm3_dual
  - cm3_single
  - sflash
- IronPython Run
- IronPython unittest
- Java Applet
- Java Application
- JUnit
- Jython run
- Jython unittest
- Launch Group
- PyDev Django
- PyDev Google App Run
- Python Run
- Python unittest
- Remote Java Application

Filter matched 21 of 21 items

Name: sflash

Connection | Files | Debugger | OS Awareness | Arguments | Environment

Select target

Select the manufacturer, board, project type and deb

Renesas / RZ/T1 R7S910x17 (Generic) / Bare Metal

Filter platforms

- Debug Cortex-M3
- Debug Cortex-R4
- RZ/T1 R7S910x18 (Generic)
- RZ/T1 RTK7910018S00000BE
- RZ/T1 RTK7910018S90000BE
- RZ/T1 RTK7910022C00000BR

Target Connection: ULINK2

DTSL Options: Edit... Configure ULINK2 t

DS-5 Debugger will connect to a ULINK2 to debug a

Connections

Bare Metal Debug: Connection

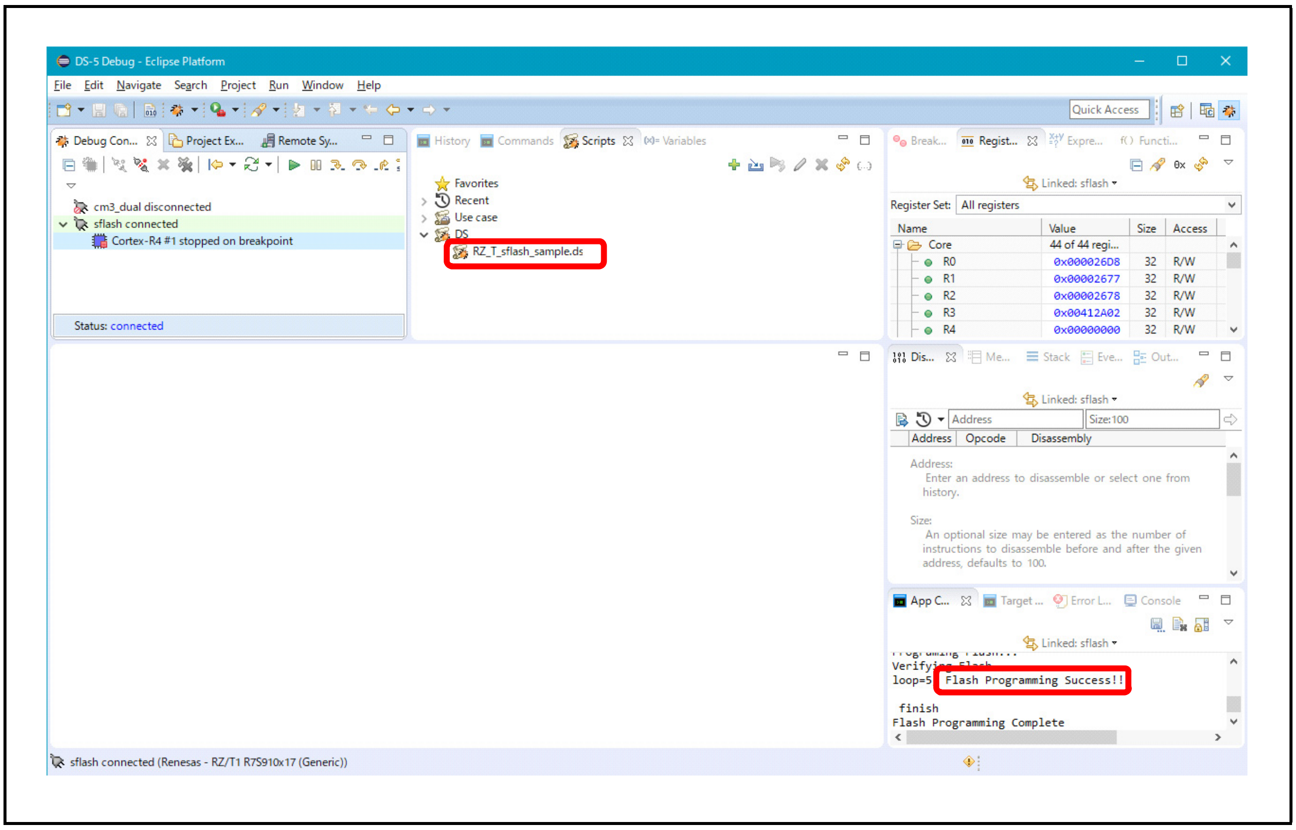
Connection Browser

Select a target connection

Keil Software Keil ULINK2 CMSIS-DAP

[redacted]

(7) When the PC is connected to the target, RZ_T_sflash_sample.ds, the script file for writing to serial flash memory, is automatically executed from the ¥RZ_T_sflash_sample¥script_sflash folder.



cm3_dual disconnected

sflash connected

Cortex-R4 #1 stopped on breakpoint

Status: connected

Register Set: All registers

Name	Value	Size	Access
Core	44 of 44 regi...		
R0	0x000026D8	32	R/W
R1	0x00002677	32	R/W
R2	0x00002678	32	R/W
R3	0x00412A02	32	R/W
R4	0x00000000	32	R/W

Address | Size:100

Address	Opcode	Disassembly

Address:
Enter an address to disassemble or select one from history.

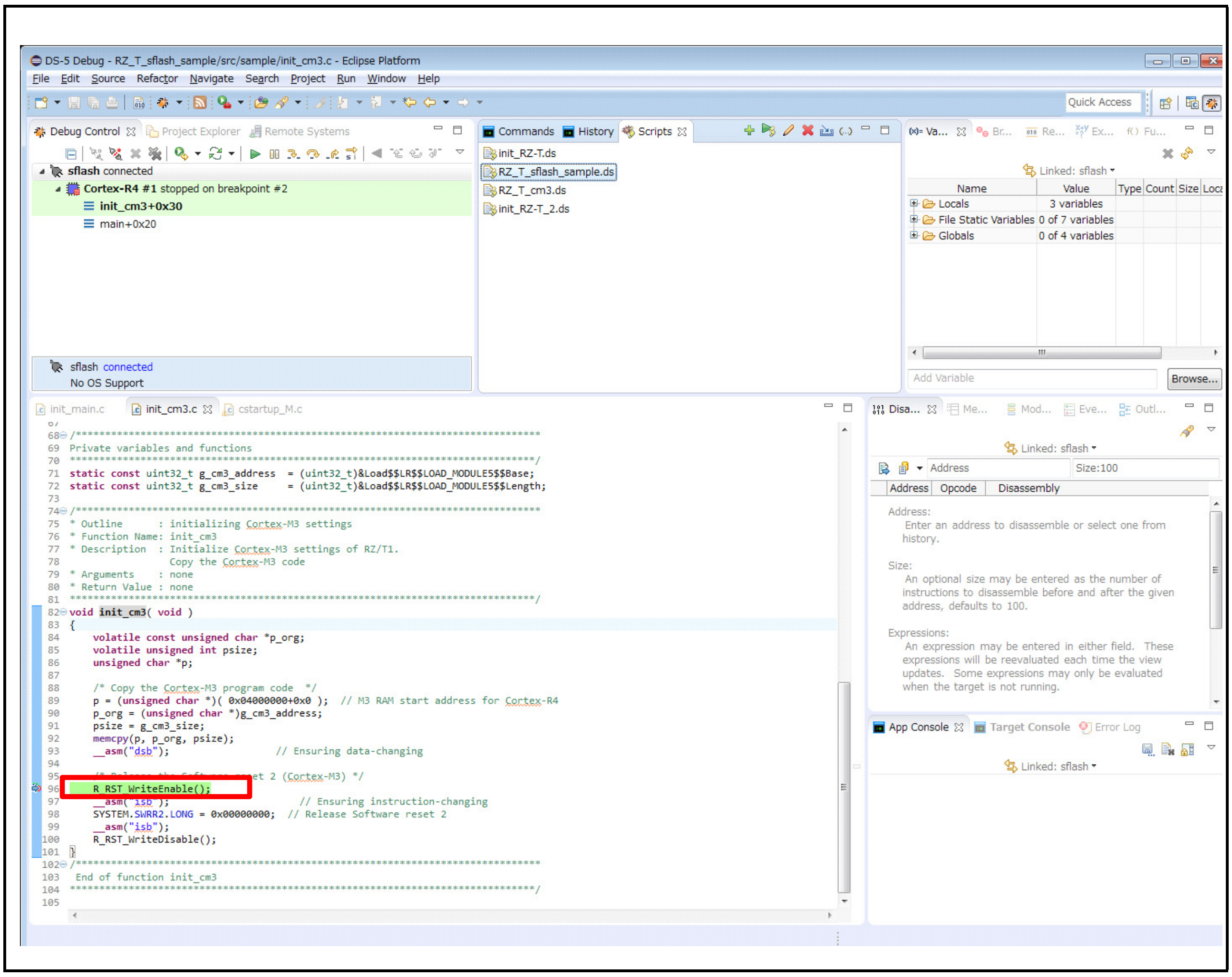
Size:
An optional size may be entered as the number of instructions to disassemble before and after the given address, defaults to 100.

```
Programming Flash...
Verifying Flash
loop=5 Flash Programming Success!!

 finish
Flash Programming Complete
```

sflash connected (Renesas - RZ/T1 R7S910x17 (Generic))

(8) On completion of writing to the flash memory by the script, the message "Flash Programming Complete" appears in the application console window.

(9) Apply a system reset. Set the hardware breakpoint immediately before the init_cm3 function, which is used to release the Cortex-M3 core from the reset state, and proceed with the steps up to the target process. To set a hardware breakpoint, right-click on the number for the row of the target process and select [DS-5 Breakpoint] from the menu, then select [Toggle Hardware Breakpoint].



sflash connected
Cortex-R4 #1 stopped on breakpoint #2
init_cm3+0x30
main+0x20

sflash connected
No OS Support

init_RZ-T.ds
RZ_T_sflash_sample.ds
RZ_T_cm3.ds
init_RZ-T_2.ds

Name	Value	Type	Count	Size	Loca
Locals	3 variables				
File Static Variables	0 of 7 variables				
Globals	0 of 4 variables				

Address Size:100

Address Opcode Disassembly

Address:
Enter an address to disassemble or select one from history.

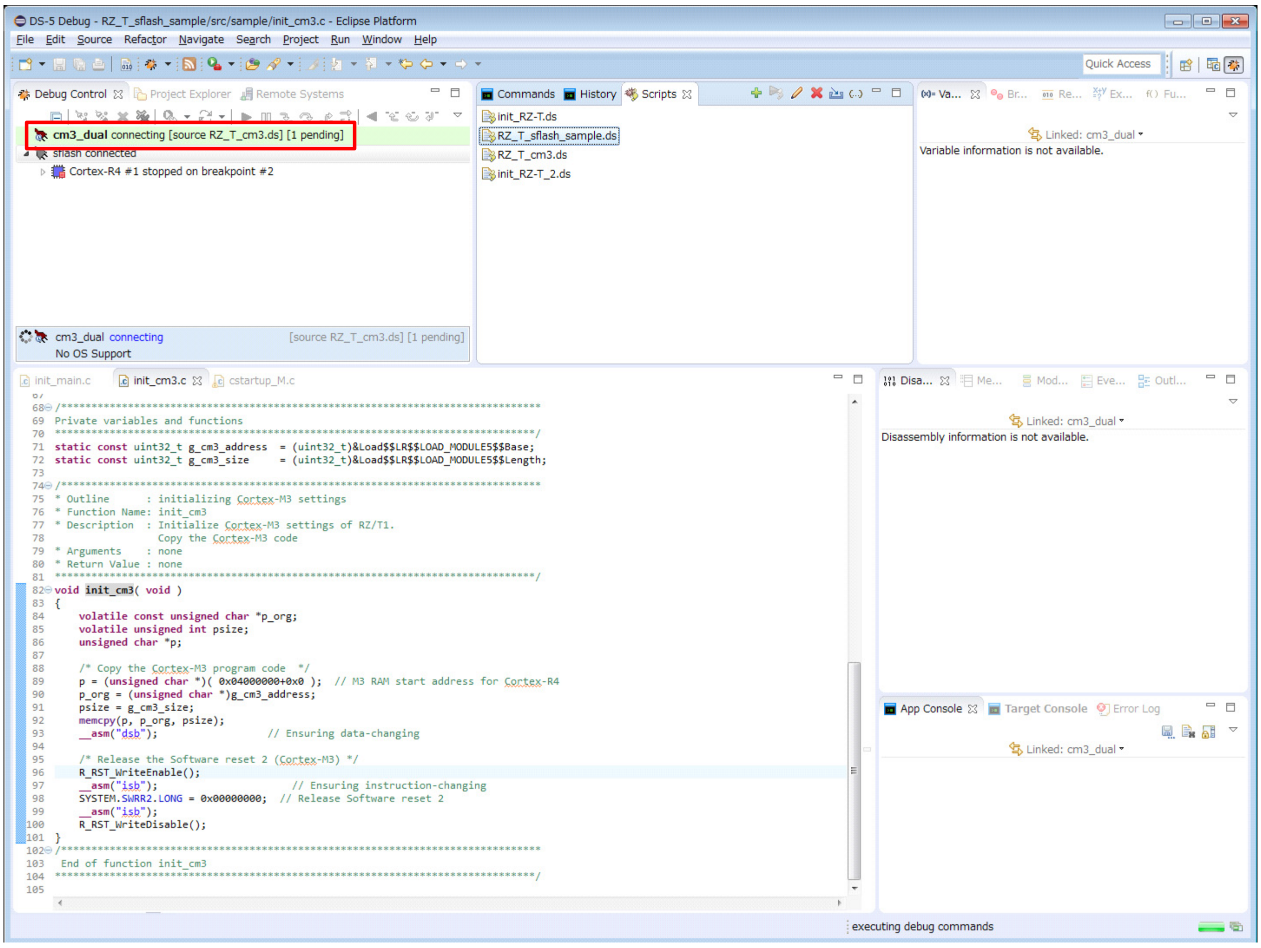
Size:
An optional size may be entered as the number of instructions to disassemble before and after the given address, defaults to 100.

Expressions:
An expression may be entered in either field. These expressions will be reevaluated each time the view updates. Some expressions may only be evaluated when the target is not running.

init_main.c init_cm3.c cstartup_M.c

```
/******************************************************************************
Private variables and functions
******************************************************************************/
static const uint32_t g_cm3_address  = (uint32_t)&Load$$LR$$LOAD_MODULE5$$Base;
static const uint32_t g_cm3_size     = (uint32_t)&Load$$LR$$LOAD_MODULE5$$Length;

/******************************************************************************
* Outline      : initializing Cortex-M3 settings
* Function Name: init_cm3
* Description  : Initialize Cortex-M3 settings of RZ/T1.
                 Copy the Cortex-M3 code
* Arguments    : none
* Return Value : none
******************************************************************************/
void init_cm3( void )
{
    volatile const unsigned char *p_org;
    volatile unsigned int psize;
    unsigned char *p;

    /* Copy the Cortex-M3 program code  */
    p = (unsigned char *)( 0x04000000+0x0 );  // M3 RAM start address for Cortex-R4
    p_org = (unsigned char *)g_cm3_address;
    psize = g_cm3_size;
    memcpy(p, p_org, psize);
    __asm("dsb");                     // Ensuring data-changing

    /* Release the Software reset 2 (Cortex-M3) */
    R_RST_WriteEnable();
    __asm("isb");                         // Ensuring instruction-changing
    SYSTEM.SWRR2.LONG = 0x00000000;  // Release Software reset 2
    __asm("isb");
    R_RST_WriteDisable();
}
/******************************************************************************
 End of function init_cm3
******************************************************************************/
```

(10) Go to [Run] > [Debug Configurations…] and open the widow of debug configurations. Select [DS-5 Debugger] and then [cm3_dual] and select the target for connection with ULINK2 in the same manner as in step (6), and make the debugging connection. At this point, reset vector catch is enabled on the Cortex-M3 by the script file "RZ_T_cm3.ds".


DS-5 Debug - RZ_T_sflash_sample/src/sample/init_cm3.c - Eclipse Platform

cm3_dual connecting [source RZ_T_cm3.ds] [1 pending]
sflash connected
Cortex-R4 #1 stopped on breakpoint #2

init_RZ-T.ds
RZ_T_sflash_sample.ds
RZ_T_cm3.ds
init_RZ-T_2.ds

Linked: cm3_dual
Variable information is not available.

cm3_dual connecting [source RZ_T_cm3.ds] [1 pending]
No OS Support

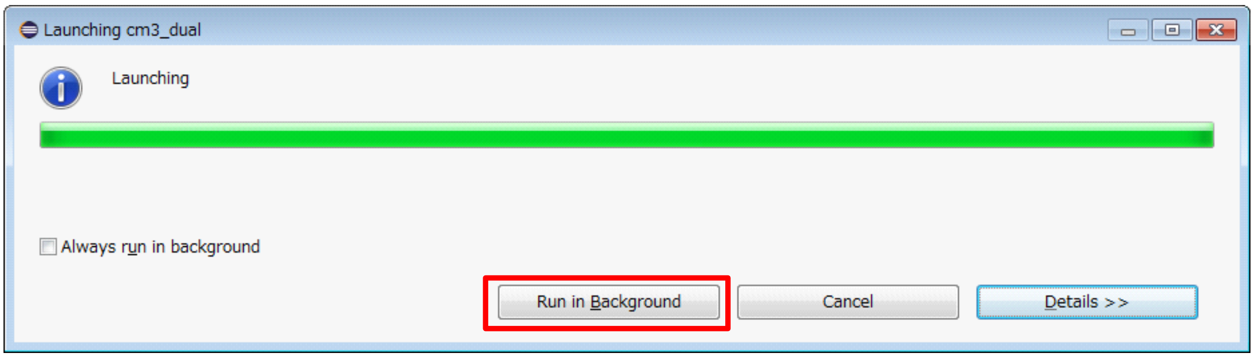
```
/*******************************************************************************
Private variables and functions
*******************************************************************************/
static const uint32_t g_cm3_address  = (uint32_t)&Load$$LR$$LOAD_MODULE5$$Base;
static const uint32_t g_cm3_size     = (uint32_t)&Load$$LR$$LOAD_MODULE5$$Length;

/*******************************************************************************
* Outline      : initializing Cortex-M3 settings
* Function Name: init_cm3
* Description  : Initialize Cortex-M3 settings of RZ/T1.
                 Copy the Cortex-M3 code
* Arguments    : none
* Return Value : none
*******************************************************************************/
void init_cm3( void )
{
    volatile const unsigned char *p_org;
    volatile unsigned int psize;
    unsigned char *p;

    /* Copy the Cortex-M3 program code  */
    p = (unsigned char *)( 0x04000000+0x0 );  // M3 RAM start address for Cortex-R4
    p_org = (unsigned char *)g_cm3_address;
    psize = g_cm3_size;
    memcpy(p, p_org, psize);
    __asm("dsb");                  // Ensuring data-changing

    /* Release the Software reset 2 (Cortex-M3) */
    R_RST_WriteEnable();
    __asm("isb");                      // Ensuring instruction-changing
    SYSTEM.SWRR2.LONG = 0x00000000;  // Release Software reset 2
    __asm("isb");
    R_RST_WriteDisable();
}
/*******************************************************************************
 End of function init_cm3
*******************************************************************************/
```

Linked: cm3_dual
Disassembly information is not available.

Linked: cm3_dual

executing debug commands

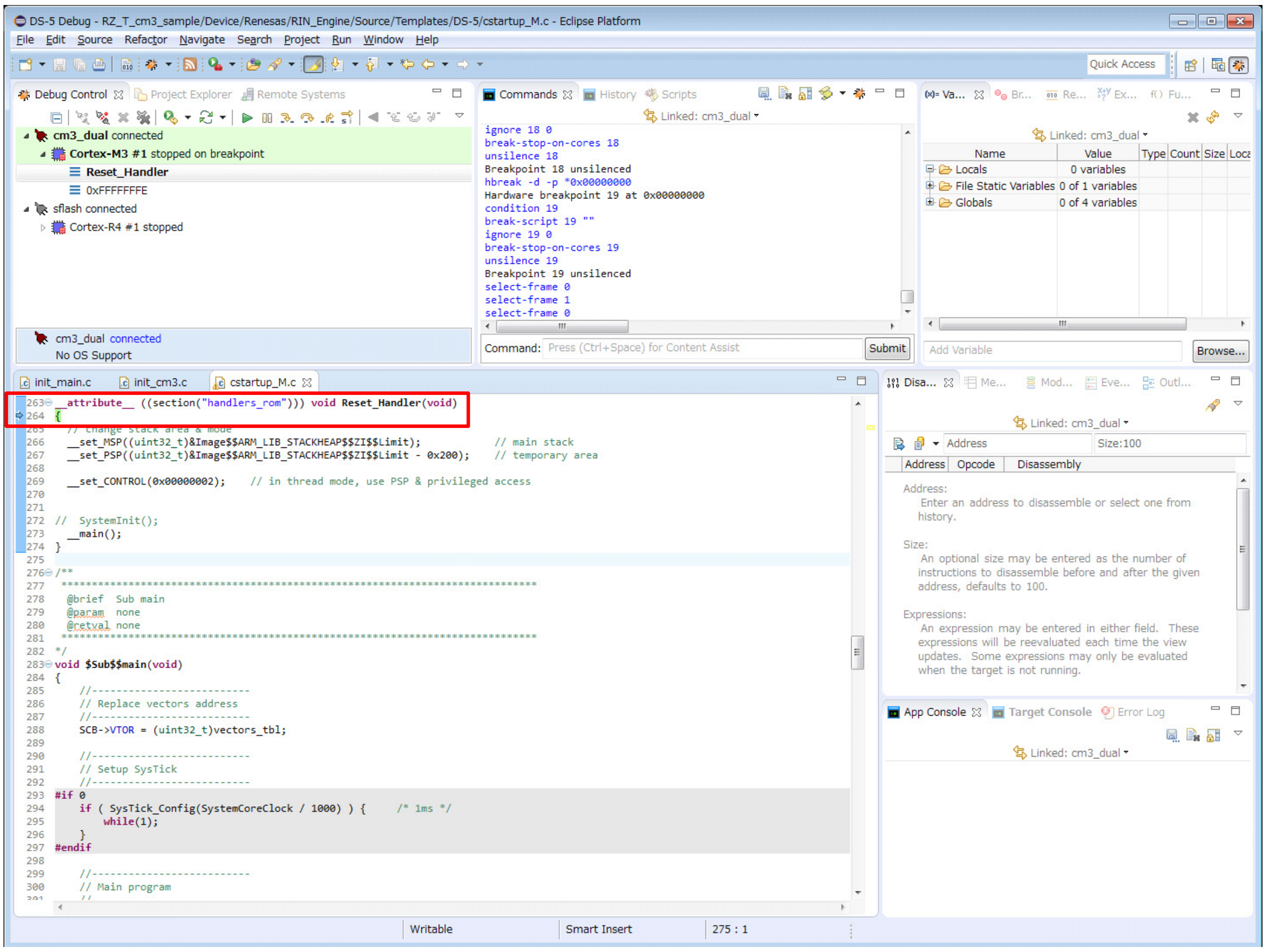
Note: Debugging of the project for the Cortex-M3 is not possible at this point because the core is in the reset state.


Launching cm3_dual

Launching

Note: The window shown above may pop up while connecting to the Cortex-M3. In this case, select [Run in Background].

(11) Connect "sflash" (the Cortex-R4) as the target. When the Cortex-M3 is released from the reset state, debugging of "cm3_dual" connection (the Cortex-M3) can then start. The program stops at reset handler, which is set as the breakpoint in the previous step.


DS-5 Debug - RZ_T_cm3_sample/Device/Renesas/RIN_Engine/Source/Templates/DS-5/cstartup_M.c - Eclipse Platform

Debug Control:

```
cm3_dual connected
  Cortex-M3 #1 stopped on breakpoint
    Reset_Handler
    0xFFFFFFFE
sflash connected
  Cortex-R4 #1 stopped

cm3_dual connected
No OS Support
```

Commands (Linked: cm3_dual):

```
ignore 18 0
break-stop-on-cores 18
unsilence 18
Breakpoint 18 unsilenced
hbreak -d -p *0x00000000
Hardware breakpoint 19 at 0x00000000
condition 19
break-script 19 ""
ignore 19 0
break-stop-on-cores 19
unsilence 19
Breakpoint 19 unsilenced
select-frame 0
select-frame 1
select-frame 0
```

Command: Press (Ctrl+Space) for Content Assist

Variables (Linked: cm3_dual):

Name	Value	Type	Count	Size	Loca
Locals	0 variables				
File Static Variables	0 of 1 variables				
Globals	0 of 4 variables				

cstartup_M.c:

```
__attribute__ ((section("handlers_rom"))) void Reset_Handler(void)
{
  // change stack area & mode
  __set_MSP((uint32_t)&Image$$ARM_LIB_STACKHEAP$$ZI$$Limit);          // main stack
  __set_PSP((uint32_t)&Image$$ARM_LIB_STACKHEAP$$ZI$$Limit - 0x200);    // temporary area

  __set_CONTROL(0x00000002);    // in thread mode, use PSP & privileged access


//  SystemInit();
  __main();
}

/**
 ******************************************************************************
  @brief  Sub main
  @param  none
  @retval none
 ******************************************************************************
*/
void $Sub$$main(void)
{
    //-------------------------
    // Replace vectors address
    //-------------------------
    SCB->VTOR = (uint32_t)vectors_tbl;

    //-------------------------
    // Setup SysTick
    //-------------------------
#if 0
    if ( SysTick_Config(SystemCoreClock / 1000) ) {     /* 1ms */
        while(1);
    }
#endif

    //-------------------------
    // Main program
    //
```

Writable   Smart Insert   275 : 1

Disassembly (Linked: cm3_dual): Address, Size:100

Address:
Enter an address to disassemble or select one from history.

Size:
An optional size may be entered as the number of instructions to disassemble before and after the given address, defaults to 100.

Expressions:
An expression may be entered in either field. These expressions will be reevaluated each time the view updates. Some expressions may only be evaluated when the target is not running.

(12) Debugging of further cores can be done by following the same procedure as described from step (6) of the procedure for EWARM from IAR Systems.

Note: Debugging of the Cortex-M3 alone is possible by using the debug connection "cm3_single". Note that the target for connection with ULINK2 needs to be selected in the debugging configuration.

<Project settings for the sample program>

Setting of DS-5 related to the sample program for making the initial settings of the LSI chips incorporating the R-IN engine is shown below.

Table. Settings in the Project for the Sample Program (Cortex-R4)

Project -> Properties -> C/C++ Build -> Settings		
Build Steps*1	Pre-build steps Command:	fromelf --bin --output =../cm3.bin ../../RZ_T_cm3_sample/Debug/RZ_T_cm3_sample.axf
	Post-build steps Command:	after_build.bat ${ProjName} This command is not specified when the program is run directly from RAM.

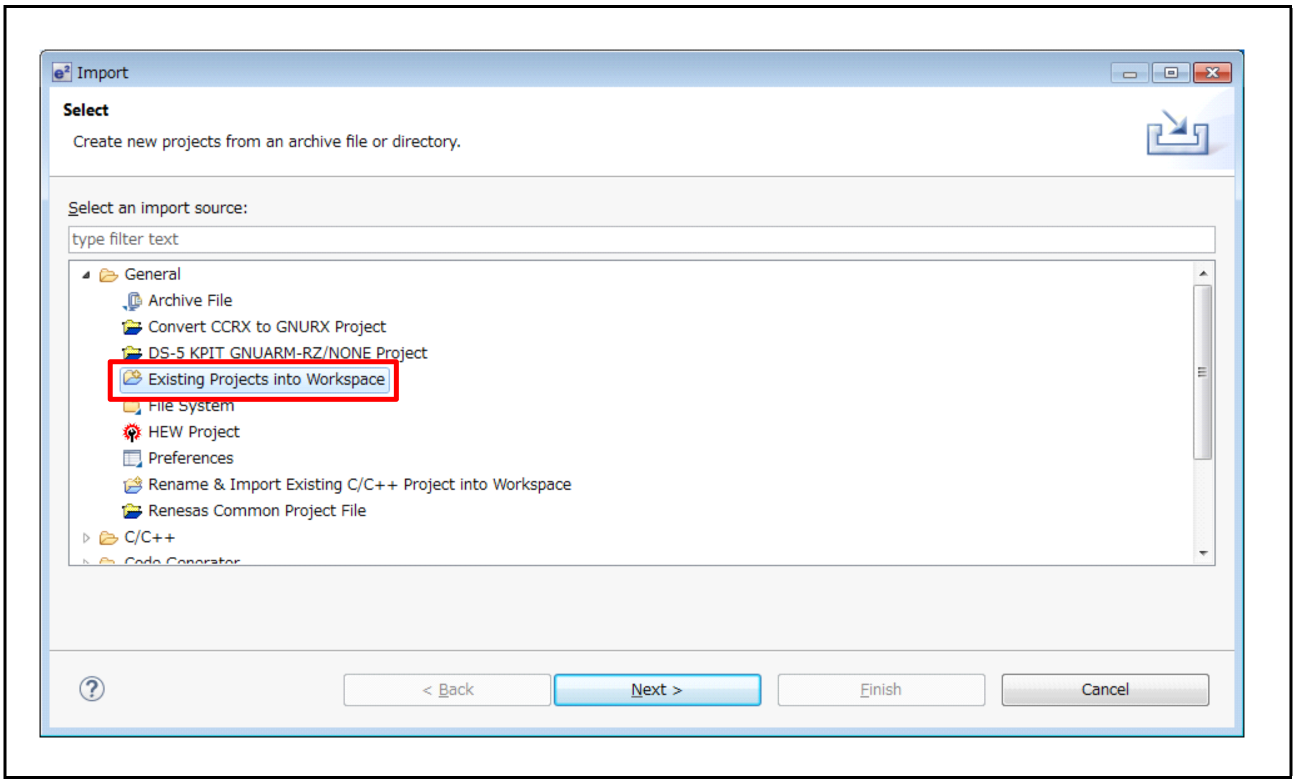
Note 1. The binary image for the Cortex-M3 is fetched into the section CM3_SECTION.

Details on the option setting are described in the related documents from Arm.

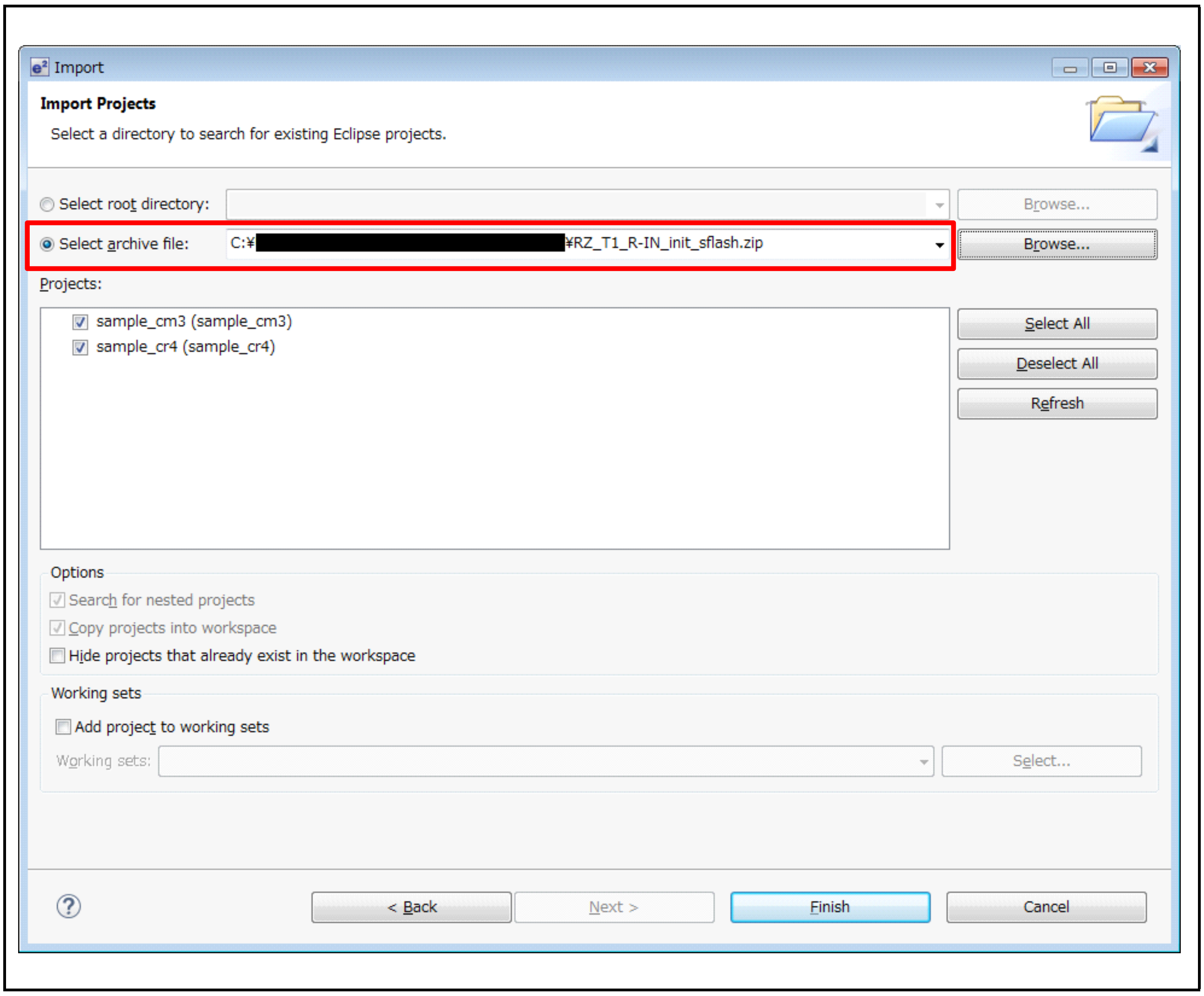
(e2 studio from Renesas)

<Steps up to debugging of the sample program (when booting is through the SPI)>

(1) Create a folder to use as a workspace on your PC, where the sample program will be stored.
(2) Start up the e2 studio environment. Specify the folder in which you stored the sample programs in step 1 as the workspace.
(3) Go to [File] > [Import]. On the [Import] window, select [Existing Projects into Workspace] in the [General] folder and click the [Next] button.


Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Convert CCRX to GNURX Project
DS-5 KPIT GNUARM-RZ/NONE Project
Existing Projects into Workspace
File System
HEW Project
Preferences
Rename & Import Existing C/C++ Project into Workspace
Renesas Common Project File
C/C++
Code Generator
< Back
Next >
Finish
Cancel

(4) Select the [Select archive file:] radio button and click on the [Browse..] button. Select the compressed program file "RZ_T1_R-IN_init_sflash.zip" and click on the [Finish] button.


Import

Import Projects

Select a directory to search for existing Eclipse projects.

Select root directory:

Select archive file: C:¥[redacted]¥RZ_T1_R-IN_init_sflash.zip

Projects:

- [✓] sample_cm3 (sample_cm3)
- [✓] sample_cr4 (sample_cr4)

Options

- [✓] Search for nested projects
- [✓] Copy projects into workspace
- [ ] Hide projects that already exist in the workspace

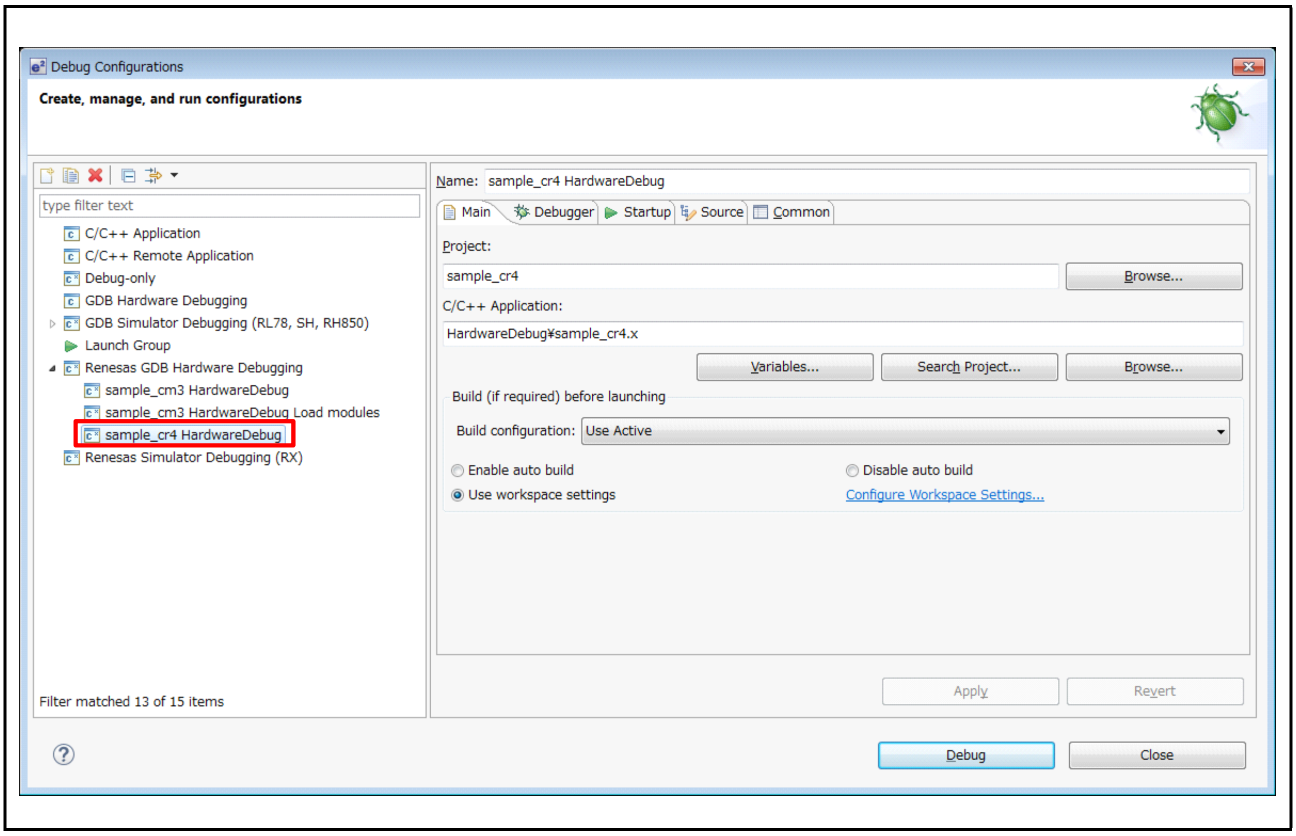
Working sets

- [ ] Add project to working sets

Working sets:

(5) Select [Build All] from the [Project] menu. On completion of building, select [Run] from the [Project] menu and then select [Debug Configurations].

(6) While the RZ/T1 evaluation board and J-Link are connected, go to [Renesas GDB Hardware Debugging] and select [sample_cr4 HardwareDebug] to show the window for configurations. Click on [Debug] in the debug configurations window and start debugging of the Cortex-R4.



Filter matched 13 of 15 items

Name: sample_cr4 HardwareDebug

Project: sample_cr4

C/C++ Application: HardwareDebug¥sample_cr4.x

Build (if required) before launching

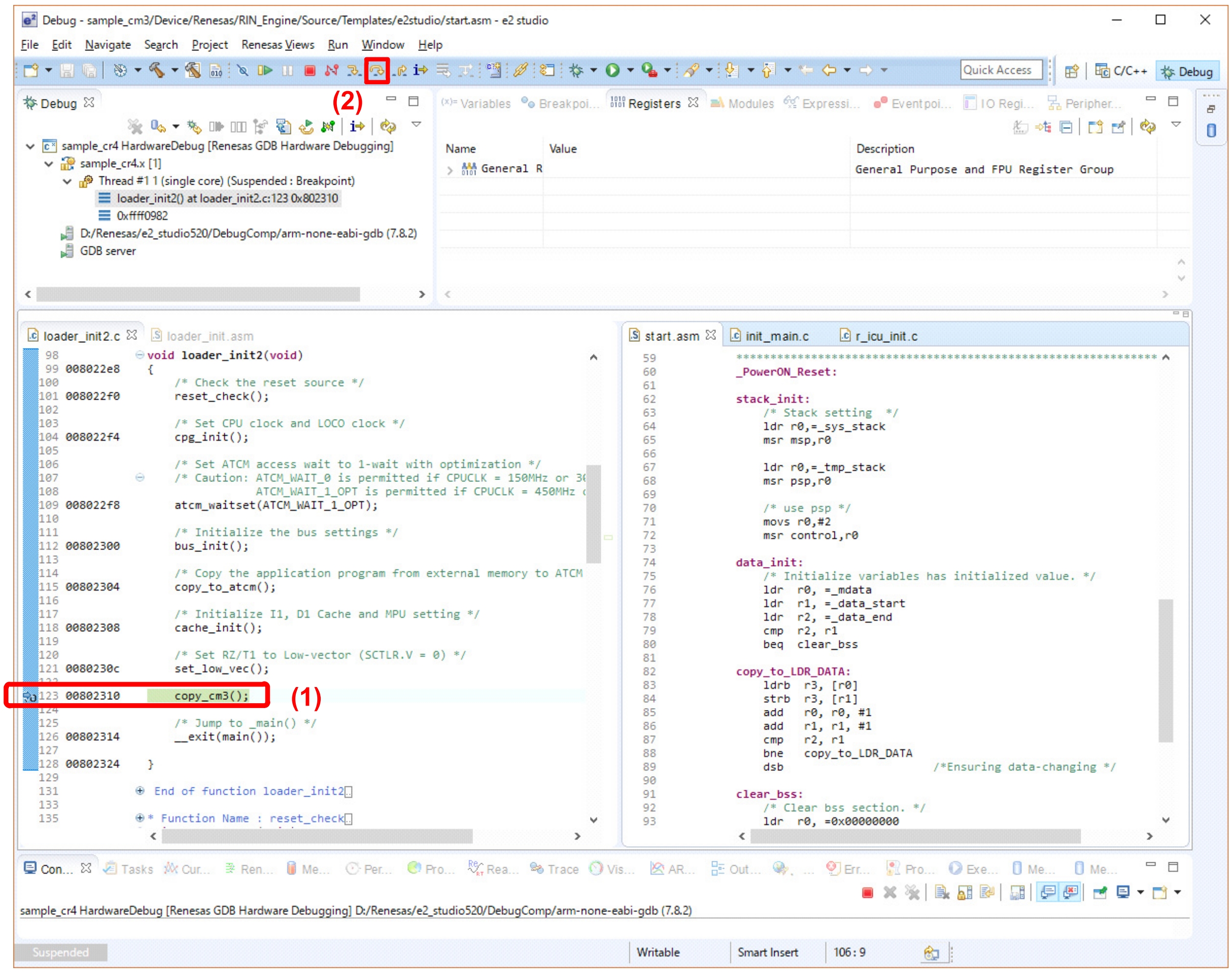
Build configuration: Use Active

Enable auto build

Disable auto build

Use workspace settings [✓]

(7) After connecting to the target, resume the program by clicking on the [Resume] button, and proceed with the steps up to the process of writing the program for the Cortex-M3.



```
sample_cr4 HardwareDebug [Renesas GDB Hardware Debugging]
  sample_cr4.x [1]
    Thread #1 1 (single core) (Suspended : Breakpoint)
      loader_init2() at loader_init2.c:123 0x802310
      0xffff0982
  D:/Renesas/e2_studio520/DebugComp/arm-none-eabi-gdb (7.8.2)
  GDB server
```

Name	Value	Description
General R		General Purpose and FPU Register Group

loader_init2.c / loader_init.asm

```
         void loader_init2(void)
008022e8 {
             /* Check the reset source */
008022f0     reset_check();

             /* Set CPU clock and LOCO clock */
008022f4     cpg_init();

             /* Set ATCM access wait to 1-wait with optimization */
             /* Caution: ATCM_WAIT_0 is permitted if CPUCLK = 150MHz or 30
                         ATCM_WAIT_1_OPT is permitted if CPUCLK = 450MHz o
008022f8     atcm_waitset(ATCM_WAIT_1_OPT);

             /* Initialize the bus settings */
00802300     bus_init();

             /* Copy the application program from external memory to ATCM
00802304     copy_to_atcm();

             /* Initialize I1, D1 Cache and MPU setting */
00802308     cache_init();

             /* Set RZ/T1 to Low-vector (SCTLR.V = 0) */
0080230c     set_low_vec();

00802310     copy_cm3();

             /* Jump to _main() */
00802314     __exit(main());

00802324 }

          End of function loader_init2

          * Function Name : reset_check
```

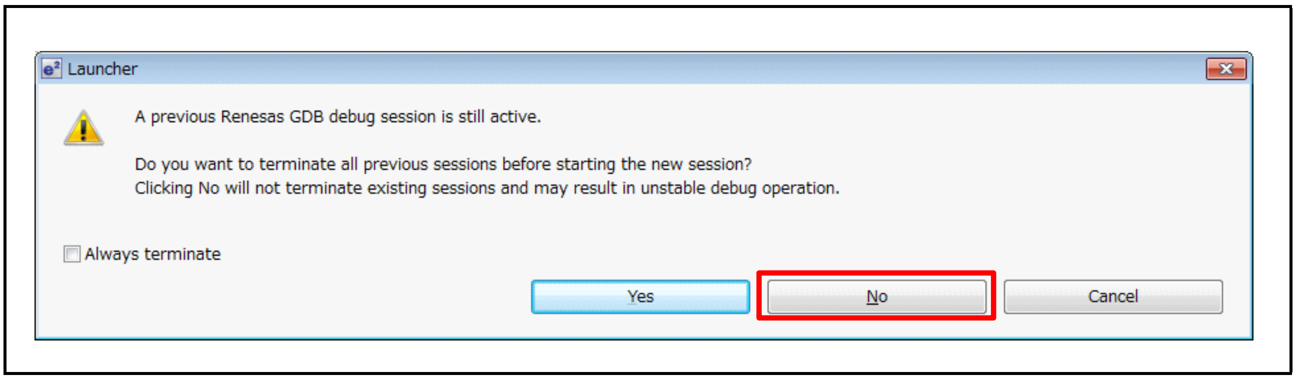
start.asm / init_main.c / r_icu_init.c

```
     ****************************************************************
     _PowerON_Reset:

     stack_init:
         /* Stack setting  */
         ldr r0,=_sys_stack
         msr msp,r0

         ldr r0,=_tmp_stack
         msr psp,r0

         /* use psp */
         movs r0,#2
         msr control,r0

     data_init:
         /* Initialize variables has initialized value. */
         ldr  r0, =_mdata
         ldr  r1, =_data_start
         ldr  r2, =_data_end
         cmp  r2, r1
         beq  clear_bss

     copy_to_LDR_DATA:
         ldrb  r3, [r0]
         strb  r3, [r1]
         add   r0, r0, #1
         add   r1, r1, #1
         cmp   r2, r1
         bne   copy_to_LDR_DATA
         dsb                        /*Ensuring data-changing */

     clear_bss:
         /* Clear bss section. */
         ldr  r0, =0x00000000
```

sample_cr4 HardwareDebug [Renesas GDB Hardware Debugging] D:/Renesas/e2_studio520/DebugComp/arm-none-eabi-gdb (7.8.2)

(8) From [Run] on the menu bar, click on [Debug Configurations…] to open the debug configurations window, go to [Renesas GDB Hardware Debugging] and select [sample_cm3 HardwareDebug]. Click on [Debug] in the debug configurations window.
Since the program for the Cortex-M3 core has not yet been written, debugging the project for the Cortex-M3 is not possible at this point.

Note: On clicking the [Debug] button, a message window appears as below asking whether to disconnect the previous debugging session or not. Here, select [No] because this is for both the Cortex-R4 core and the R-IN engine (Cortex-M3 core).



A previous Renesas GDB debug session is still active.

Do you want to terminate all previous sessions before starting the new session?
Clicking No will not terminate existing sessions and may result in unstable debug operation.

(9) On the [Debug] tabbed page, select “Thread #xxx” from the “Sample_cr4.x” folder. Then, from [Run] on the menu bar, click on [Step Over F6] to execute writing of the program for the Cortex-M3.

(10) Resume the program by clicking the [Resume] button while “Thread #xxx” from the “Sample_cm3.x” folder is selected. Debugging on the Cortex-M3 can then start.

Note: The debugger releases the Cortex-M3 from the reset state on the debugging process, which is followed by releasing of the reset by the Cortex-R4, and there is no problem with these processes. When the debugger is not used, only the latter process proceeds.

(11) Debugging of further cores can be done by following the same procedure as described from step (6) of the procedure for EWARM from IAR Systems.

Note: Debugging on the Cortex-R4 is possible by selecting the “Thread #xxx” in the “Sample_cr4.x” folder and on the Cortex-M3 by selecting the “Thread #xxx” in the “Sample_cm3.x” folder.

<Project settings for the sample program>

Setting of e2 studio related to the sample program for making the initial settings of the microcomputers incorporating the R-IN engine is shown below.

Table. Settings in the Project for the Sample Program (Cortex-R4)

Project -> Properties -> Project References		
Project references for 'sample_cr4'*1	sample_cm3	Switch checkmark on

Note 1. Building the project for the Cortex-R4 automatically builds the project for the Cortex-M3..

Table. Settings in the Project for the Sample Program (Cortex-M3)

Project -> Properties -> C/C++ Build -> Settings		
Build Steps*1	Post-build steps Command:	arm-none-eabi-objcopy -I elf32-littlearm -O binary sample _cm3.x cm3.bin & arm-none-eabi-objcopy -I binary -O elf32-littlearm -B arm --rename-section .data=.cm3,alloc,data,readonly,load,contents cm3.bin cm3.o & copy /Y cm3.o..¥..¥sample_cr4¥cm3.bin¥cm3.o

Note 1. The data is copied from the Cortex-M3 and fetched into the section ".cm3" in the Cortex-R4 as the binary image.

Table. Debug Configuration for the Sample Program

Run -> Debug Configurations		
Debugger -> Connection Settings -> Connection	Reset on connection	"NO"
	Reset before run	[sample_cr4 HardwareDebug]: "YES" [sample_cm3 HardwareDebug], [sample_cm3 HardwareDebug Load modules]: "NO"
Startup	Runtime options	[sample_cr4 HardwareDebug, sample_cm3 HardwareDebug] Switch checkmark on the "Set break point at:" and set "_PowerON_Reset". *1
	Run commands	[sample_cm3 HardwareDebug] Set "set $pc=&_PowerON_Reset".*1

Note 1. "_PowerON_Reset" is the entry point for each project. Replace it if the entry point is different.

Website and Support

Renesas Electronics website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision History	Application Note: Initial Settings of the Microcomputers Incorporating the R-IN Engine

Rev.	Date	Description	
		Page	Summary
1.00	Nov. 30, 2015	—	First Edition issued
1.10	Mar. 01, 2016	2. Operating Environment	
		5	Table 2.1 Operating Environment, Integrated development environment, partially amended
		6. Software	
		9	Description added
		15	6.2.4 Required Memory Size, title of Table 6.7 amended, Table 6.8 and Table 6.9 added
		Appendix 1. Supplementary Notes on Development Environments	
		36	“DS-5 from ARM” and “e2studio from Renesas”, fully amended
1.20	Aug. 09, 2017	All	Cortex-R4F → Cortex-R4, modified
		2. Operating Environment	
		5	Table 2.1 Operating Environment: The description of the integrated development environment, modified
		6. Software	
		—	6.2.4 Required Memory Size, deleted
		Appendix-1. Supplementary Notes on Development Environments	
		31	(EWARM from IAR systems) <Steps up to debugging of the sample program (when booting is through the SPI)> (4) modified: [Debug without Downloading] → [Attach to Running Target]
		33	(EWARM from IAR systems) <Project settings for the sample program> Table. Settings in the Project for the Sample Program (Cortex-R4): Modified. Notes 2 and 3, deleted.
		44	(e2 studio from Renesas) <Steps up to debugging of the sample program (when booting is through the SPI)> (7): The description modified. The figure modified.
		45	(e2 studio from Renesas) <Steps up to debugging of the sample program (when booting is through the SPI)> (8): The description added
		45	(e2 studio from Renesas) <Steps up to debugging of the sample program (when booting is through the SPI)> (9): The description modified. The figure deleted.
		46	(e2 studio from Renesas) <Project settings for the sample program> Table. Debug Configuration for the Sample Program: Note 1, added
1.30	Apr. 27, 2018	All	"ARM" changed to "Arm"
		2. Operating Environment	
		5	Table 2.1 Operating Environment: The description in the integrated development environments, modified; The version number of the integrated development environment from IAR Systems, modified; The description of “Devices (functions to be used on the board)”, modified
		6. Software	
		9	6.1 Operation Overview: The error in step 3, corrected
		10	Table 6.1 Settings of SW4: The names of the sample programs, modified
		10	6.2.1 Section Assignment for the Sample Program: The error, corrected
		10	Table 6.2 Sections to be Used (Cortex-R4): The description in Note 1 and Note 3, modified
		11	Table 6.3 Sections to be Used (Cortex-M3): The HEAP line moved to below the readwrite line; The MAIN_STACK area, deleted
		13	Figure 6.3 Assignment of Sections (Cortex-M3): The HEAP line moved to below the readwrite line; The MAIN_STACK area, deleted
		16	Table 6.7 Global Variables: The function of type uint32_t, modified
		7. Sample Program	
		29	The description, modified
		8. Related Documents	
		30	The description, modified

Revision History	Application Note: Initial Settings of the Microcomputers Incorporating the R-IN Engine

Rev.	Date	Description	
		Page	**Summary**
1.30	Apr. 27, 2018	Appendix-1. Supplementary Notes on Development Environments	
		31, 32	(EWARM from IAR systems): Images in steps (4) to (6) for debugging the sample program, modified
		33	Table. Settings in the Project for the Sample Program (Cortex-R4): Debugger: The description in Debugger/Extra options/Synchronous, modified. Note 2, added.
		36	(DS-5 from ARM): The description and image in step (7) for debugging the sample program, modified
		45	Table. Settings in the Project for the Sample Program (Cortex-M3): Modified
		45	Table. Debug Configuration for the Sample Program: The description in Startup/Run commands, modified
1.31	Jul. 13, 2018	—	Revision with updating of the sample program

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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 Microprocessing unit or Microcontroller unit products 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.

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.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

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

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

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, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

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

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

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0