User's Manual

RH850/V1R-M Radar Software

User's Manual : Radar API Part

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.0.01 Dec. XX, 2015

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.

(2012.4)

How to Use This Manual

1. Purpose and Target Readers

This manual is intended to give users of the software an understanding of the decoder functionality, performance, and usage of the software. It is targeted at people who wish to design application systems which use the software. It assumes readers hold general knowledge in the fields of programming languages, and microcontrollers.

Use this software after carefully reading the precautions. The precautions are stated in the main text of each section, at the end of each section, and in the usage precaution section.

The revision history summarizes major corrections and additions to the previous version. It does not cover all the changes. For details, refer to this manual.

2. Notation of Numbers and Symbols

3. Register Notation

4. List of Abbreviations and Acronyms

Abbreviation	Full Form
ANSI-C	American National Standards Institute - C
AUTOSAR	AUTomotive Open System ARchitecture
bps	bits per second
CPU	Central Processing Unit
DSP	Digital Signal Processor
I/O	Input/Output
LSB	Least Significant Bit
MSB	Most Significant Bit
OS	Operating System
Radar	Radio Detecting and Ranging
RAM	Random Access Memory
ROM	Read Only Memory
ANSI-C	American National Standards Institute - C

- Table of Contents -

- Figures -

- Tables -

1. Summary

1.1. Summary of this manual

This document is the user manual of Radar Application Interface (Radar API). It explains the specification of Radar API.

Please refer to the related documents shown in 1.3.

1.2. Software structure and the scope of this manual

Figure 1.1 shows software architecture of the Radar Software. The Radar Software is comprised of Radar API, each units and Radar Framework. Radar API is called from user application in CPU. Each units performs Radar signal processing in DSP. Radar Framework controls units.

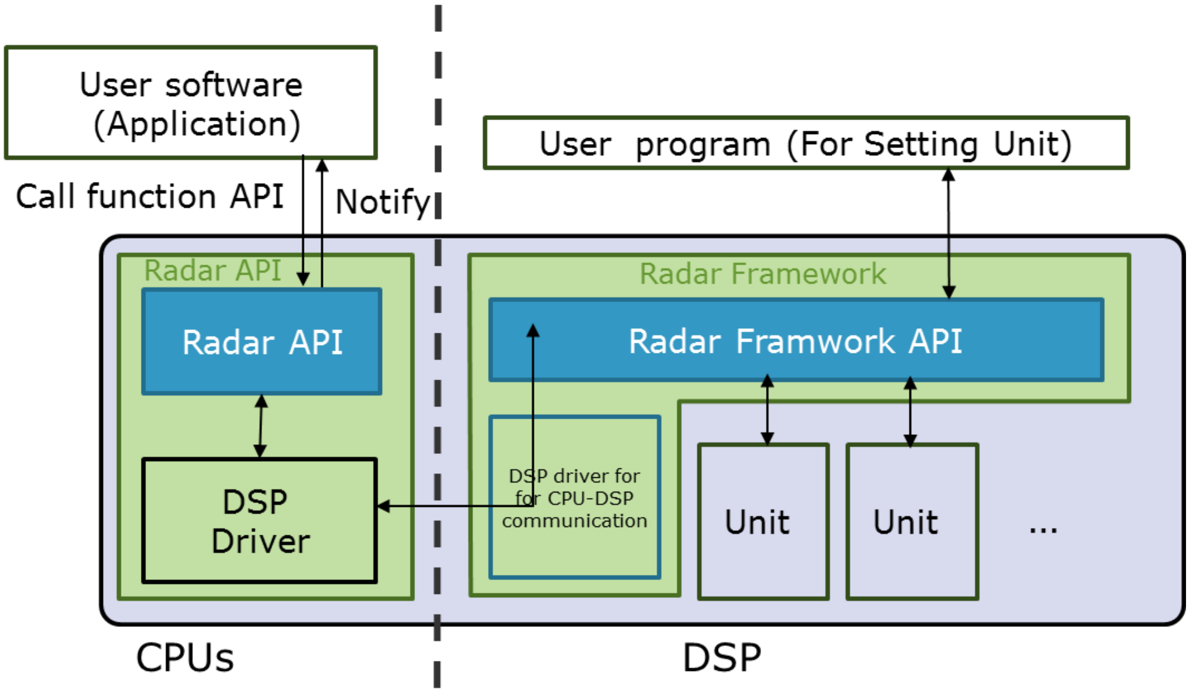


Figure 1.1 Software architecture

This document describes the specification of Radar API and Radar Framework.

Figure 1.2 shows the architecture when this software is applied to AUTOSAR.

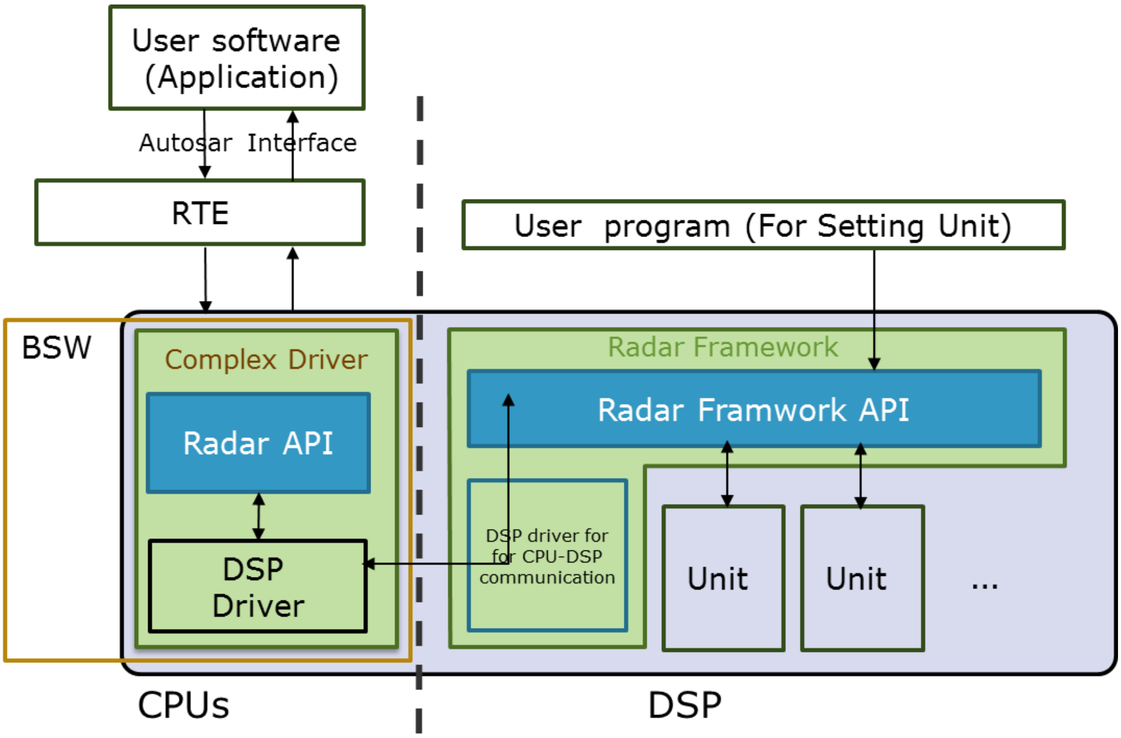


Figure 1.2 software architecture (AUTOSAR)

1.3. Related documents

Name	Revision
[1] RH850/V1R-M User's Manual:Hardware R7F701490EABG	Rev.0.10

1.4. Common data type definition

Table 1.1 shows the list of data type definition commonly used for Radar API and Radar Framework.

Table 1.1 The list of common data type definition

data type	size [byte(s)]		
RAI_S8	1	Signed 8bit integer	-128 to 127
RAI_S16	2	Signed 16bit integer	-32768 to 32767
RAI_S32	4	Signed 32bit integer	-2147483648 to 2147483647
RAI_U8	1	Unsigned 8bit integer	0 to 255
RAI_U16	2	Unsigned 16bit integer	0 to 65535
RAI_U32	4	Unsigned 32bit integer	0 to 4294967295
RAI_BOOL	4	Boolean(Signed 32bit integer)	(0 [RAI_FALSE] / not 0 [RAI_TRUE])

[note] The pointer of each data size is the same size (4 bytes).

1.5. Naming rules

Table 1.2 shows the naming rules for the symbols used for Radar API and Radar Framework. Never overlap the names if other applications are combined.

Table 1.2 Symbol naming rules

Type	Prefix
Function name	RAI_XXXX, rai_XXXX
Structure name	RAI_XXXX
Define/Enumeration name	RAI_XXXX

[note] XXXX consists of arbitrary alphanumeric characters.

2. Radar API

2.1. Function list

Table 2.1 shows the functions provided by Radar API.

Table 2.1 API function list

Function Name	Type	Outline
RAI_GetMemorySize	sync	Obtain the information of necessary memory size
RAI_Init[1]	sync	Initialize Radar API
RAI_DeInit[1]	sync	Deinitialize Radar API
RAI_GetHandle[1]	unsync	Obtain unit handle
RAI_FreeHandle[1]	unsync	Free unit handle
RAI_SetConfig[1]	unsync	Send config information to unit
RAI_SendCommand[1]	unsync	Send command to unit
RAI_GetStatus	sync	Obtain status information
RAI_GetVersion	sync	Obtain version information

Table 2.2 Interrupt handler registration function

Function Name	Outline
RAI_Isr	Interrupt handler used by Radar API

[1] Concurrent call from other task and call from interrupt handler are not supported.

2.2. Data type definition

This chapter shows the data type definition provided by Radar API.

2.2.1. Macro definition

Table 2.3 shows the error codes provided by Radar API.

Table 2.3 The list of error codes

Name	Description
RAI_E_OK	Normal termination
RAI_E_BUSY	Busy
RAI_E_TIMEOUT	Timeout
RAI_E_INVALID_HANDLE	Invalid handle
RAI_E_PARAMETER_ERROR	Parameter error
RAI_E_INIT_ERROR	Initialization error
RAI_E_SYSTEM_ERROR	System error

Table 2.4 shows other definitions provided by Radar API.

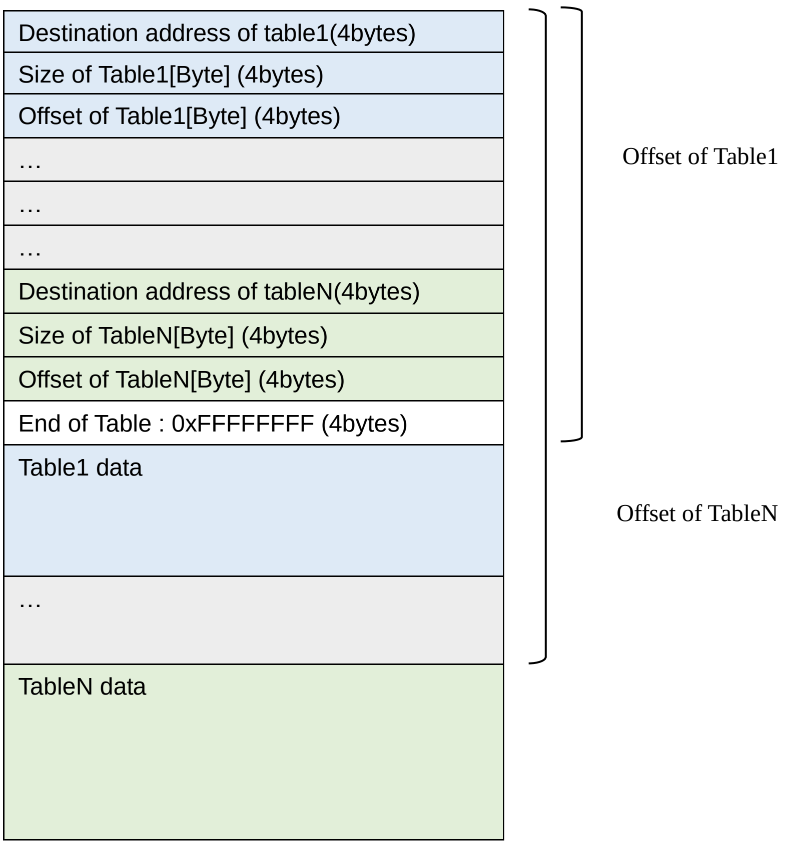
Table 2.4 list of definitions

Name	Initial value	**Description**
RAI_UNIT_MAX	8	Maximum number of units
RAI_CFG_MAX_SIZE	32	Maximum parameters of config information
RAI_CMD_MAX_SIZE	8	Maximum parameters of command
RAI_STATUS_COMMAND	0	Status ID to indicate the status of asynchronous API execution

2.2.2. RAI_HANDLE

Name:	**RAI_HANDLE**
Type:	void*

2.2.3. RAI_SYS_MEMINFO

Name:	**RAI_SYS_MEMINFO**		
Type:	Structure		
Element:	Type	Name	Description
	RAI_U32*	pSharedMem	The address of shared memory used by RadarAPI
	RAI_U32	nSharedMemSize	The size of shared memory address used by RadarAPI [byte]
	RAI_PTR	pDspFw	The start address of the binary object of DSP program.(see Note2)
Note:	Assign 4-byte aligned address to pSharedMem.		
Note 2	The format of the binary object (pDspFw) is shown below.  Destination address of table1(4bytes) Size of Table1[Byte] (4bytes) Offset of Table1[Byte] (4bytes) … … … Destination address of tableN(4bytes) Size of TableN[Byte] (4bytes) Offset of TableN[Byte] (4bytes) End of Table : 0xFFFFFFFF (4bytes) Table1 data … TableN data 		

2.3. Radar API function specification

Below shows the specification of API functions provided by Radar API.

2.3.1. RAI_GetMemorySize

Syntax	RAI_U32 RAI_GetMemorySize (RAI_U32 unitNum, RAI_U32 *sharedMemSize);			
Function	Obtain memory size used by Radar API.			
Arguments	Type	Name	I/O	Description
	RAI_U32	unitNum	I	The number of units to register with Radar Framework
	RAI_U32*	sharedMemSize	O	The size of shared memory [byte]
Return value	RAI_U32		error code	
	RAI_E_OK		OK	
	RAI_E_PARAMETER_ERROR		unitNum is 0 or exceeding maximum unit. number.sharedMemSize is NULL.	
Description	This function is to obtain the memory size used by Radar API. This function returns the necessary size to control units whose number is specified by unitNum. sharedMemSize is the memory size shared by CPU and DSP. Allocate the memory whose size is obtained by this API. Then specify it by RAI_Init function.			
Note 1	The maximum number of unitNum is RAI_UNIT_MAX.			
Note 2	T.B.D.			
Note 3	T.B.D.			

2.3.2. RAI_Init

Syntax	RAI_U32　RAI_Init (RAI_U32 unitNum, RAI_SYS_MEMINFO *memInfo);			
Function	Initialize Radar API.			
Arguments	Type	Name	I/O	Description
	RAI_U32	unitNum	I	The number of units to register with Radar Framework.
	RAI_SYS_MEMINFO*	memInfo	I	Memory information
Return value	RAI_U32	error code		
	RAI_E_OK	OK		
	RAI_E_INIT_ERROR	Already initialized		
	RAI_E_PARAMETER_ERROR	unitNum is 0 or exceeding maximum unit. memInfo is NULL.		
	RAI_E_TIMEOUT	Timeout without ACK from DSP		
Description	This function initializes Radar API. This function operates download of programs to DSP and start it. After confirmation of start of DSP, this function stops automatically.			
Note 1	Register with RAI_Isr for interrupt handler SINTR0 before calling this API.			
Note 2	The maximum number of unitNum is RAI_UNIT_MAX.			
Note 3	When this function is called before RAI_DeInit is called, it returns RAI_E_INIT_ERROR. If you want to register multiple units, call this function only at the first time and call RAI_GetHandle to register remaining units.			

2.3.3. RAI_DeInit

Syntax	RAI_U32　RAI_DeInit (void);	
Function	Deinitialize Radar API.	
Arguments	None	
Return value	RAI_U32	error codes
	RAI_E_OK	OK
	RAI_E_INIT_ERROR	when initialization process does not finished.
Description	This function deinitializes Radar API.	
Note 1	Release the memory specified by RAI_Init after this function returns RAI_E_OK.	
Note 2	T.B.D.	
Note 3	T.B.D.	

2.3.4. RAI_GetHandle

<table>
<tr><td>Syntax</td><td colspan="4">RAI_U32 RAI_GetHandle (
RAI_HANDLE *unitHandle,
RAI_U32 unitId,
RAI_CALLBACK callback,
RAI_U32 * sharedMem
);</td></tr>
<tr><td>Function</td><td colspan="4">Obtain the handle of specified unit</td></tr>
<tr><td rowspan="5">Arguments</td><td>Type</td><td>Name</td><td>I/O</td><td>Description</td></tr>
<tr><td>RAI_HANDLE *</td><td>unitHandle</td><td>O</td><td>unit handle</td></tr>
<tr><td>RAI_U32</td><td>unitId</td><td>I</td><td>unit ID(1-7)</td></tr>
<tr><td>RAI_CALLBACK</td><td>callback</td><td>I</td><td>callback function</td></tr>
<tr><td>RAI_U32*</td><td>sharedMem</td><td>I</td><td>The start address of shared memory used by the unit</td></tr>
<tr><td rowspan="7">Return value</td><td colspan="2">RAI_U32</td><td colspan="2">error codes</td></tr>
<tr><td colspan="2">RAI_E_OK</td><td colspan="2">OK</td></tr>
<tr><td colspan="2">RAI_E_INIT_ERROR</td><td colspan="2">when initialization process does not finished.</td></tr>
<tr><td colspan="2">RAI_E_PARAMETER_ERROR</td><td colspan="2">unitId is 0 or exceeding maximum unit.
sharedMem is NULL.</td></tr>
<tr><td colspan="2">RAI_E_INVALID_HANDLE</td><td colspan="2">when specified unit ID has been already used</td></tr>
<tr><td colspan="2">RAI_E_BUSY</td><td colspan="2">when another command is processed</td></tr>
<tr><td colspan="2">RAI_E_SYSTEM_ERROR</td><td colspan="2">system error (when shared memory administration register is 0)</td></tr>
<tr><td>Description</td><td colspan="4">This function obtains the handle of specified unit.
Set unitId to the unit ID defined by the target unit (it is the same as the ID registered with Radar Framework) The callback function specified by callback is called when the unit notifies an event.</td></tr>
<tr><td>Note 1</td><td colspan="4">This function is asynchronous. RAI_E_BUSY is returned if ACK for the previous asynchronous function is not received. Check that asynchronous function is acceptable by RAI_GetStatus before calling this function.</td></tr>
<tr><td>Note 2</td><td colspan="4">The callback function is called from interrupt handler RAI_Isr. If you want to implement such process as to wait for some input or to have high load operations, they should be processed by other thread handed necessary information.</td></tr>
<tr><td>Note 3</td><td colspan="4">Allocate shared memory for each unit and specify the start address to sharedMem. Necessary memory size is determined according to the specification of the unit.
The allocated area must be kept until the area is released by RAI_FreeHandle.</td></tr>
<tr><td>Note 4</td><td colspan="4">T.B.D.</td></tr>
</table>

2.3.5. RAI_FreeHandle

Syntax	RAI_U32　RAI_FreeHandle (　RAI_HANDLE　unitHandle,);			
Function	Release the handle of specified unit			
Arguments	Type	Name	I/O	Description
	RAI_HANDLE	unitHandle	I	unit handle
Return value	RAI_U32	error codes		
	RAI_E_OK	OK		
	RAI_E_INIT_ERROR	when initialization process does not finished.		
	RAI_E_INVALID_HANDLE	when the specified handle is not obtained.		
	RAI_E_BUSY	when another command is processed		
	RAI_E_SYSTEM_ERROR	system error (when shared memory administration register is 0)		
Description	This function releases the unit handle corresponded to unitHandle.			
Note 1	This function is asynchronous. RAI_E_BUSY is returned if ACK for the previous asynchronous function is not received. Check that asynchronous function is acceptable by RAI_GetStatus before calling this function.			
Note 2	T.B.D.			
Note 3	T.B.D.			

2.3.6. RAI_SetConfig

Syntax	RAI_U32　RAI_SetConfig (RAI_HANDLE unitHandle, RAI_U32 configId, RAI_U32 *param, RAI_U32 paramNum);			
Function	Send config information to specified unit			
Arguments	Type	Name	I/O	Description
	RAI_HANDLE	unitHandle	I	unit handle
	RAI_U32	configId	I	config ID
	RAI_U32 *	param	I	parameters
	RAI_U32	paramNum	I	the number of parameters
Return value	RAI_U32	error codes		
	RAI_E_OK	OK		
	RAI_E_INIT_ERROR	when initialization process does not finished.		
	RAI_E_PARAMETER_ERROR	when both paramNum is 1 or more and param is NULL. Or, paramNum exceeds maximum number.		
	RAI_E_INVALID_HANDLE	when the specified handle is not obtained.		
	RAI_E_BUSY	when another command is processed		
	RAI_E_SYSTEM_ERROR	system error (when shared memory administration register is 0)		
Description	This function sends config information to the unit specified by unitHandle. Set configId to the config ID defined by the unit corresponding to unitHandle. The information sent by this API is notified when Framework receives it. The timing doesn’t depend on the priority of the target unit.			
Note 1	This function is asynchronous. RAI_E_BUSY is returned if ACK for the previous asynchronous function is not received. Check that asynchronous function is acceptable by RAI_GetStatus before calling this function.			
Note 2	param must be defined as an array of RAI_U32.			
Note 3	paramNum is the number of parameters by the unit of RAI_U32. (1 parameter：4 bytes) The maximum number of paramNum is RAI_CFG_MAX_SIZE. If you want to use more parameters than RAI_CFG_MAX_SIZE, it is necessary to define and implement the way to pass parameters by the unit. One way to do it is to put the parameters in shared memory and pass the address by param.			
Note 4	This function allocates its own command area and copies the data specified by param to it. So param area can be released after this function returns response.			
Note 5	T.B.D.			

2.3.7. RAI_SendCommand

Syntax	RAI_U32 RAI_SendCommand (RAI_HANDLE unitHandle, RAI_U32 commandId, RAI_U32 *param, RAI_U32 paramNum);			
Function	Send command to specified unit			
Arguments	Type	Name	I/O	Description
	RAI_HANDLE	unitHandle	I	unit handle
	RAI_U32	commandId	I	command ID
	RAI_U32 *	param	I	parameters
	RAI_U32	paramNum	I	the number of parameters
Return value	RAI_U32	error codes		
	RAI_E_OK	OK		
	RAI_E_INIT_ERROR	when initialization process does not finished.		
	RAI_E_PARAMETER_ERROR	when both paramNum is 1 or more and param is NULL. Or, paramNum exceeds maximum number.		
	RAI_E_INVALID_HANDLE	when the specified handle is not obtained.		
	RAI_E_BUSY	when another command is processed		
	RAI_E_SYSTEM_ERROR	system error (when shared memory administration register is 0)		
Description	This function sends command to the unit specified by unitHandle. Set commandId to the command ID defined by the unit corresponding to unitHandle. The information sent by this API is notified according to the priority of the target unit.			
Note 1	This function is asynchronous. RAI_E_BUSY is returned if ACK for the previous asynchronous function is not received. Check that asynchronous function is acceptable by RAI_GetStatus before calling this function.			
Note 2	param must be defined as an array of RAI_U32.			
Note 3	paramNum is the number of parameters by the unit of RAI_U32. (1 parameter：4 bytes) The maximum number of paramNum is RAI_CMD_MAX_SIZE. If you want to use more parameters than RAI_CMD_MAX_SIZE, it is necessary to define and implement the way to pass parameters by the unit. One way to do it is to put the parameters in shared memory and pass the address by param.			
Note 4	This function allocates its own command area and copies the data specified by param to it. So param area can be released after this function returns response.			
Note 5	T.B.D.			

2.3.8. RAI_GetStatus

Syntax	RAI_U32 RAI_GetStatus (RAI_HANDLE unitHandle, RAI_U32 statusId, RAI_U32 **statusAddr);			
Function	Obtain the information of specified unit			
Arguments	Type	Name	I/O	Description
	RAI_HANDLE	unitHandle	I	unit handle
	RAI_U32	statusId	I	status ID
	RAI_U32 **	statusAddr	O	the address the status information is stored
Return value	RAI_U32			error codes
	RAI_E_OK			OK
	RAI_E_INIT_ERROR			when initialization process does not finished.
	RAI_E_PARAMETER_ERROR			when statusAddr is NULL.
	RAI_E_INVALID_HANDLE			when the specified handle is not obtained.
	RAI_E_SYSTEM_ERROR			system error (when shared memory administration register is 0)
Description	This function obtains the status of the unit specified by unitHandle. It is possible to set statusId to the status ID specified by unitHandle. The IDs which can be set is defined by the target unit. This function set statusAddr to the address the status information corresponding to status Id is stored. When you set statusId to RAI_STATUS_COMMAND, this function sets statusAddr to the address the value to indicate whether ACK for the previous asynchronous function has been received or not is stored. You can check which of the following the status is. RAI_TRUE: asynchronous API is executable RAI_FALSE: asynchronous API is not executable In this case, unitHandle is ignored. If the status obtained by using RAI_STATUS_COMMAND is RAI_FALSE, asynchronous API can’t be executed, so wait until the status changes to RAI_TRUE.			
Note 1	When this function is called with setting statusId to 0, the returned statusAddr is not changed until RAI_DeInit is called. So after statusAddr is set, whether asynchronous API can be executed or not can be easily checked by referring it.			
Note 2	statusAddr is allocated in shared memory and directly refers the information written by DSP. If it is necessary to limit access from DSP when CPU accesses the area, control function such as assignment of access flag or use of hardware semaphore should be implemented by user.			
Note 3	T.B.D.			

2.3.9. RAI_GetVersion

<table>
<tr><td><b>Syntax</b></td><td colspan="4">RAI_U32　RAI_GetVersion (
RAI_U32 *versionCodeAPI,
RAI_U32 *versionCodeFW
);</td></tr>
<tr><td><b>Function</b></td><td colspan="4">Obtain the version of Radar API</td></tr>
<tr><td rowspan="3"><b>Arguments</b></td><td>Type</td><td>Name</td><td>I/O</td><td>Description</td></tr>
<tr><td>RAI_U32*</td><td>versionCodeAPI</td><td>O</td><td>API version code</td></tr>
<tr><td>RAI_U32*</td><td>versionCodeFW</td><td>O</td><td>FW version code</td></tr>
<tr><td rowspan="2"><b>Return value</b></td><td colspan="2">RAI_U32</td><td colspan="2">error codes</td></tr>
<tr><td colspan="2">RAI_E_OK</td><td colspan="2">OK</td></tr>
<tr><td><b>Description</b></td><td colspan="4">This function obtains Radar Software version code.
The format of version code is as follows :
Customer ID (25bit～32bit)：0x00(standard version)、other(reserved)
Release ID (17bit～24bit)：0x00(official version)、0x01(sim version)、0xA0～0xAF(α version)、0xB0～0xBF(β version)、other(reserved)
Major ID(9bit～16bit)：0x00 ～ 0x99(major number)、other(reserved)
Minor ID (1bit～8bit)：0x00 ～ 0x99(minor number)、other(reserved)
Above 4 IDs are written in a 32 bit data.

The examples of version code are shown below.
・API version
Customer ID：standard version, Release ID：Sim version, Major ID：1, Minor ID：2
・FW version
Customer ID：standard version, Release ID：Sim version, Major ID：2, Minor ID：1
・The results
versionCodeAPI：0x00010102
versionCodeFW：0x00010201</td></tr>
<tr><td>Note 1</td><td colspan="4">versionCodeFW is available after calling RAI_Init.</td></tr>
<tr><td>Note 2</td><td colspan="4">When this function fails to obtain version information, the returned value is 0xFFFFFFFF.</td></tr>
<tr><td>Note 3</td><td colspan="4">T.B.D.</td></tr>
</table>

2.4. Radar API interrupt handler function specification

Below shows the specification of interrupt handler function provided by Radar API.

2.4.1. RAI_Isr

Syntax	void RAI_Isr (void);			
Function	Radar API interrupt handler function			
Arguments	Type	Name	I/O	Description
	None			
Return value	None			
Description	This function handles interruption from DSP. This function must be registered as SINTR0 interrupt handler before calling RAI_Init.			
Note 1	T.B.D.			
Note 2	T.B.D.			
Note 3	T.B.D.			

2.5. Radar API callback function specification

Below shows callback function specification provided by Radar API.

2.5.1. RAI_CALLBACK

Syntax	void (*RAI_CALLBACK)(RAI_HANDLE unitHandle, RAI_U32 eventId, RAI_U32 eventData);			
Function	callback called when an event from a unit happens			
Arguments	Type	Name	I/O	Description
	RAI_FW_HANDLE	unitHandle	I	unit handle
	RAI_U32	eventId	I	event ID
	RAI_U32	eventData	I	accompanying information with the event
Return value	None			
Description	This is callback function when an event from a unit happens. User should implement what to do and register it by RAI_GetHandle. Set eventId to the event ID defined by the target unit. And perform the process corresponding to the eventId with the callback function. The eventData stores the information corresponding to eventId defined by unit.			
Note 1	The callback function is called by interrupt handler RAI_Isr. If you want to process wait or high loaded operations, they should be processed by task with necessary information.			
Note 2	DSP can notify the next event after the process of this function is finished. If next event occurs while RAI_CALLBACK is running, the event is pended and notified after RAI_CALLBAKC is finished. If next event occurs while an event is pended and newer event comes from the same unit and the same event ID, newer one overwrites older one and only the newest is pended.			
Note 3	T.B.D.			

2.6. Memory

Below shows about memory used by Radar API.

2.6.1. List of memory

Below shows how much of each memory Radar API uses.
Please note the size shown is only as a guide. Refer to the memory map for the accurate size.
T.B.D

2.6.2. Section of memory

Below shows the allocation of the section of each memory.
T.B.D

3. Radar Framework

3.1. List of functions

Table 3.1 shows the functions provided by Radar Framework.
Radar Framework functions are called from user program to initialize Radar Framework, to register units with Radar Framework and to start Radar Framework. Refer to 3.3 for detail of each function.

Table 3.1 List of Framework functions

Function name	Outline
RAI_FW_GetMemorySize	Obtain necessary memory size for Radar Framework
RAI_FW_Init	Initialize Radar Framework
RAI_FW_RegisterUnit	Register units with Framework
RAI_FW_Execute	Start Radar Framework

Table 3.2 Function to register interrupt handler with Radar Framework

Function Name	Outline
RAI_FW_Isr	Interrupt handler used by Radar Framework

3.2. Type definition

3.2.1. Macro definition

Table 3.3 shows error codes provided by Radar Framework.

Table 3.3 list of error codes

Name	Description
RAI_FW_E_OK	OK
RAI_FW_E_PARAMETER_ERROR	Parameter error
RAI_FW_E_REGIST	Failed to register a unit
RAI_FW_E_INIT_ERROR	Initialization error
RAI_FW_E_SYSTEM_ERROR	System error

3.2.2. RAI_FW_HANDLE

Name:	**RAI_FW_HANDLE**
Type:	void*

3.2.3. RAI_FW_UNIT_INFO

Name:	**RAI_FW_UNIT_INFO**		
Type:	Structure		
Element:	Type	Name	Description
	RAI_U32	nUnitId	unit ID(1-7)
	RAI_U32	nPriority	The priority of a unit (the larger is the higher priority). When commands issued by multiple units, the scheduler of Radar Framework prioritizes the command by the unit with higher priority.
	RAI_FW_GET_HANDLE	pfGetHandle	Unit register function for obtaining handle. The function registered in this variable is executed by DSP when CPU calls RAI_GetHandle.
	RAI_FW_FREE_HANDLE	pfFreeHandle	Unit registration function for releasing handle. The function registered in this variable is executed by DSP when CPU calls RAI_FreeHandle.
	RAI_FW_SET_CONFIG	pfSetConfig	Unit registration function for configuration. The function registered in this variable is executed by DSP when CPU calls RAI_FreeHandle.
	RAI_FW_SEND_COMMAND	pfSendCommand	Unit registration function for command execution. The function registered in this variable is

			executed by DSP when CPU calls RAI_FreeHandle.
	RAI_U32	pUnitWork	The start address for work memory used by units.

3.3. Framework function specification

3.3.1. RAI_FW_GetMemorySize

Syntax	RAI_U32　RAI_FW_GetMemorySize (RAI_U32 * fwWorkSize);			
Function	Obtain memory size used by Radar Framework.			
Arguments	Type	Name	I/O	Description
	RAI_U32*	fwWorkSize	O	The size of DSP local memory used by Radar Framework [byte]
Return value	RAI_U32		error codes	
	RAI_FW_E_OK		OK	
Description	This function obtains the size of DSP local memory used by Radar Framework. The necessary memory size is calculated by the number of units specified by RAI_Init in RadarAPI. Secure to allocate the memory of the size obtained by this function before executing RAI_FW_Init.			
Note 1	This functions is available after boot up of DSP.			
Note 2	T.B.D			
Note 3	T.B.D			

3.3.2. RAI_FW_Init

Syntax	RAI_UI32　RAI_FW_Init (RAI_U32*　　　fwWork);			
Function	Initialize Framework			
Arguments	Type	Name	I/O	Description
	RAI_U32*	fwWork	I	The start address of DSP local memory (DSP-LRAM) used by Framework
Return value	RAI_U32		error codes	
	RAI_FW_E_OK		OK	
	RAI_FW_E_INIT_ERROR		already initalized	
	RAI_FW_E_SYSTEM_ERROR		system error (when shared memory administration register is 0)	
Description	This function initializes Framework. Run this function once after boot up of DSP and allocation of DSP local memory for Framework are done.			
Note 1	Register RAI_FW_Isr as interrupt handler of INTPE2DSP before call of this function.			
Note 2	The size of memory needed by Framework can be obtained by RAI_FW_GetMemorySize. Make sure to allocate equal to more than the size of memory obtained by RAI_FW_GetMemorySize.			
Note 3	If this function is called more than once, it returns RAI_FW_E_INIT_ERROR.			
Note 4	T.B.D			

3.3.3. RAI_FW_RegisterUnit

Syntax	RAI_U32　RAI_FW_RegisterUnit (RAI_U32*　　fwWork, RAI_FW_UNIT_INFO*　unitInfo);			
Function	Register unit with Framework			
Arguments	Type	Name	I/O	Description
	RAI_U32*	fwWork	I	The start address of DSP local memory (DSP-LRAM) used by Framework
	RAI_FW_UNIT_INFO*	unitInfo	I	The start address of unit information structure Refer to 3.2.3 about unit information structure
Return value	RAI_U32		error codes	
	RAI_FW_E_OK		OK	
	RAI_FW_E_INIT_ERROR		when initialization process does not finished.	
	RAI_FW_E_PARAMETER_ERROR		when the number of registered unit exceeds RAI_UNIT_MAX.	
	RAI_FW_E_SYSTEM_ERROR		system error (when shared memory administration register is 0)	
Description	This function registers a unit with Framework. Use this function to register a unit with Framework. When you want to register multiple units, run this function once for each unit. RAI_UNIT_MAX specifies the maximum number of units to be able to register. (※Note 4)			
Note 1	RAI_FW_Init is necessary to be called before calling this function. Specify the memory area initialized by RAI_FW_Init for DSP local memory for Framework.			
Note 2	The unit registration information should be specified by unit information structure according to the specification of each unit.			
Note 3	Allocate the work memory used by a unit separately and set unit information structure to the address before calling this function. The size need to be allocated is determined according to the specification of unit you use. The memory used by Framework and each unit should be managed by caller.			
Note 4	unit ID : 0 is reserved. The maximum number which can be registered is specified by RAI_UNIT_MAX.			
Note 5	T.B.D.			

3.3.4. RAI_FW_Execute

Syntax	RAI_U32 RAI_FW_Execute (RAI_U32* fwWork);			
Function	Execute Framework			
Arguments	Type	Name	I/O	Description
	RAI_U32*	fwWork	I	The start address of DSP local memory used by Framework
Return value	RAI_U32			error codes
	RAI_FW_E_OK			OK
	RAI_FW_E_INIT_ERROR			when initialization process does not finished.
	RAI_FW_E_SYSTEM_ERROR			system error (when shared memory administration register is 0)
Description	This function starts Framework, then waits for commands. Framework receives commands for each unit issued by CP, and execute unit registration functions. Framework also works as scheduler. If multiple units exists, it executes unit registration functions according to the priority of the units. See 3.2.3 for the priority. This function is terminated when RAI_DeInit is executed by RadarAPI. (T.B.D.)			
Note 1	It is necessary to execute RAI_FW_Init before calling this function. Specify the memory area initialized by RAI_FW_Init for DSP local memory for Framework.			
Note 2	It is necessary to register units by RAI_FW_RegisterUnit before calling this function.			
Note 3	T.B.D.			

3.4. Framework interrupt handler function

3.4.1. RAI_FW_Isr

Syntax	void RAI_FW_Isr (void);			
Function	Framework interrupt handler function			
Arguments	Type	Name	I/O	Description
	None			
Return value	None			
Description	This function handles interruption from CPU. Register this function as interrupt handler of INTPE2DSP before calling RAI_FW_Init.			
Note 1	T.B.D			
Note 2	T.B.D			
Note 3	T.B.D			

4. Radar Unit

This chapter shows the specification of unit registration function registered with Framework and the functions executable from unit registration function. These information will be useful when users develop unit.

4.1. List of functions

Table 4.1 and Table 4.2 shows unit registration functions and Framework unit functions.

Unit registration functions are the functions provided by a unit and they are registered with Framework when uses use a unit. These functions are callback functions executed by Framework. User should implement each of these function for development of unit. Refer 4.3 for the detail.

Framework unit functions are the functions executed from unit registration function when user develops unit. Refer 4.4 for the details.

Table 4.1 List of unit registration functions

Function name	Outline
RAI_FW_GET_HANDLE	Get handle of a unit
RAI_FW_FREE_HANDLE	Release handle of a unit
RAI_FW_SET_CONFIG	Configure a unit
RAI_FW_SEND_COMMAND	Process command to unit

Table 4.2 List of Framework unit functions

Function name	Outline
RAI_FW_SetNotify	Set of notification from Framework to CPU
RAI_FW_RegisterStatus	Register status information to Framework
RAI_FW_GetUnitWorkAddress	Obtain the address of work memory for a unit

4.2. Type definition

4.2.1. RAI_FW_RESULT

Name:	**RAI_FW_RESULT**	
Type:	Enumeration	
Range:	Name	Description
	RAI_FW_RESULT_COMPLETE	Process completed
	RAI_FW_RESULT_CONTINUE	Process continuing

4.3. Unit registration function specification

4.3.1. RAI_FW_GET_HANDLE

Syntax	RAI_FW_RESULT (*RAI_FW_GET_HANDLE)(RAI_FW_HANDLE unitHandle, RAI_U32* sharedMem);			
Function	Obtain unit handle			
Arguments	Type	Name	I/O	Description
	RAI_FW_HANDLE	unitHandle	I	unit handle of Framework
	RAI_U32*	sharedMem	I	The start address of memory shared by unit. The address specified by RAI_GetHandle in RadarAPI is set (see 2.3.4).
Return value	RAI_FW_RESULT		Results RAI_FW_RESULT_COMPLETE	
Description	This function is executed once from Framework when CPU calls RAI_GetHandle. Execute initialization and configuration of unit in this function. The sequence is shown in Figure 5.3. Use RAI_FW_GetUnitWorkAddress to obtain work memory for unit. The work memory can be used to hold information such as the start address of shared memory if necessary.			
Note 1	Error processing should be defined and implemented for each unit. Define event notification in unit if necessary and notify errors by RAI_FW_SetNotify. Framework will notify the events specified by RAI_FW_SetNotify after the end of this function. CPU should process corresponding to each event according to the specification of unit.			
Note 2	T.B.D			
Note 3	T.B.D			

4.3.2. RAI_FW_FREE_HANDLE

Syntax	RAI_FW_RESULT (*RAI_FW_FREE_HANDLE)(RAI_FW_HANDLE unitHandle,);			
Function	Release unit handle			
Arguments	Type	Name	I/O	Description
	RAI_FW_HANDLE	unitHandle	I	unit handle of Framework
Return value	RAI_FW_RESULT		Results RAI_FW_RESULT_COMPLETE	
Description	This function is executed once from Framework when CPU calls RAI_GetHandle. Execute termination of unit in this function. The sequence is shown in Figure 5.4.			
Note 1	Error processing should be defined and implemented for each unit. Define event notification in unit if necessary and notify errors by RAI_FW_SetNotify. Framework will notify the events specified by RAI_FW_SetNotify after the end of this function. CPU should process corresponding to each event according to the specification of unit.			
Note 2	T.B.D			
Note 3	T.B.D			

4.3.3. RAI_FW_SET_CONFIG

Syntax	RAI_FW_RESULT (*RAI_FW_SET_CONFIG)(
	    RAI_FW_HANDLE unitHandle,
	    RAI_U32 configId,
	    RAI_U32* param,
	    RAI_U32 paramNum
	);			

Function	Configure a unit			
Arguments	Type	Name	I/O	Description
	RAI_FW_HANDLE	unitHandle	I	unit handle of Framework
	RAI_U32	configId	I	config ID for unit. The command ID specified by RAI_GetHandle in RadarAPI is set (see 2.3.6).
	RAI_U32*	param	I	parameters
	RAI_U32	paramNum	I	the number of parameters
Return value	RAI_FW_RESULT		Results RAI_FW_RESULT_COMPLETE	
Description	This function is executed once from Framework when CPU calls RAI_GetHandle. Execute the process corresponding config ID defined by unit. The sequence is shown in Figure 5.5.			
Note 1	Error processing should be defined and implemented for each unit. Define event notification in unit if necessary and notify errors by RAI_FW_SetNotify. Framework will notify the events specified by RAI_FW_SetNotify after the end of this function. CPU should process corresponding to each event according to the specification of unit.			
Note 2	T.B.D			
Note 3	T.B.D			

4.3.4. RAI_FW_SEND_COMMAND

<table>
<tr><td><b>Syntax</b></td><td colspan="4">RAI_FW_RESULT (*RAI_FW_SEND_COMMAND)(
        RAI_FW_HANDLE   unitHandle,
        RAI_U32   commandId,
        RAI_U32*   param,
        RAI_U32   paramNum
);</td></tr>
<tr><td><b>Function</b></td><td colspan="4">Process command to unit</td></tr>
<tr><td rowspan="5"><b>Arguments</b></td><td>Type</td><td>Name</td><td>I/O</td><td>Description</td></tr>
<tr><td>RAI_FW_HANDLE</td><td>unitHandle</td><td>I</td><td>unit handle of Framework</td></tr>
<tr><td>RAI_U32</td><td>commandId</td><td>I</td><td>command ID to unit
The address specified by RAI_GetHandle in RadarAPI is set (see 2.3.7).</td></tr>
<tr><td>RAI_U32*</td><td>param</td><td>I</td><td>parameters</td></tr>
<tr><td>RAI_U32</td><td>paramNum</td><td>I</td><td>the number of parameters</td></tr>
<tr><td><b>Return value</b></td><td colspan="2">RAI_FW_RESULT</td><td colspan="2">Results
RAI_FW_RESULT_COMPLETE
RAI_FW_RESULT_CONTINUE</td></tr>
<tr><td rowspan="5"><b>Description</b></td><td colspan="4">This function is executed from Framework when CPU calls RAI_SendCommand.
Execute the process corresponding to command ID defined by unit in this function.
If you want to execute this function continuously in such case as dividing consecutive process, exit this function with setting the result to RAI_FW_RESULT_CONTINUE.
If the result is RAI_FW_RESULT_CONTINUE, Framework calls this function again.
Exit this function with setting the result to RAI_FW_RESULT_COMPLETE if all of consecutive process is done, or no need to run continuously.
The sequence is shown in Figure 5.6 and Figure 5.7.</td></tr>
<tr><td>Note 1</td><td colspan="3">Even while consecutive operation by RAI_FW_RESULT_CONTINUE by a unit, Framework executes commands of the unit if it has higher priority.
So when Framework needs to operate multiple units with different priority, the tasks by them can be operated according to priority by dividing process and return control to Framework.</td></tr>
<tr><td>Note 2</td><td colspan="3">The values of parameters do not change since the first execution while consecutive operation by RAI_FW_RESULT_CONTINUE. State control should be managed in this function.</td></tr>
<tr><td>Note 3</td><td colspan="3">Error processing should be defined and implemented for each unit.
Define event notification in unit if necessary and notify errors by RAI_FW_SetNotify. Framework will notify the events specified by RAI_FW_SetNotify after the end of this function. CPU should process corresponding to each event according to the specification of unit.</td></tr>
<tr><td>Note 4</td><td colspan="3">T.B.D</td></tr>
</table>

4.4. Framework unit function specification

4.4.1. RAI_FW_SetNotify

Syntax	RAI_U32 RAI_FW_SetNotify (RAI_FW_HANDLE* unitHandle, RAI_U32 eventId, RAI_U32 eventData,);			
Function	Set notification from Framework to CPU			
Arguments	Type	Name	I/O	Description
	RAI_FW_HANDLE*	unitHandle	I	unit handle of Framework
	RAI_U32	eventId	I	notification event ID(0-15)
	RAI_U32	eventData	I	notification event information
Return value	RAI_U32	error codes		
	RAI_FW_E_OK	OK		
	RAI_FW_E_PARAMETER_ERROR	parameter error		
Description	This function set notification of event from Framework to CPU. This function can be called only from unit registration function. Execute this function when a unit notifies event to CPU. Define notification event ID and notification event information by each unit.			
Note 1	Notification of event to CPU is done after execution of this function and control is returned to Framework.			
Note 2	If this function is called twice or more with the same notification event ID, only the last notification event is notified to CPU.			
Note 3	T.B.D			

4.4.2. RAI_FW_RegisterStatus

Syntax	RAI_U32 RAI_FW_RegisterStatus (RAI_FW_HANDLE* unitHandle, RAI_U32 statusId, RAI_U32* statusAddr,);			
Function	Register status information with Framework			
Arguments	Type	Name	I/O	Description
	RAI_FW_HANDLE*	unitHandle	I	unit handle of Framework
	RAI_U32	statusId	I	status ID
	RAI_U32*	statusAddr	I	The start address of status information
Return value	RAI_U32		error codes	
	RAI_FW_E_OK		OK	
	RAI_FW_E_PARAMETER_ERROR		parameter error	
Description	This function registers status ID and memory area of status information with Framework. This function can be called only from unit registration function. Execute this function in RAI_FW_GET_HANDLE if status ID is defined by RAI_GetStatus in RadarAPI. If you want to register multiple status ID, run this function once for each status ID. The maximum number of status which can be registered is 16.			
Note 1	Framework holds status IDs and the start addresses of status information as a table. And it returns the start address of corresponding status information from the table when RAI_GetStatus in RadarAPI is executed. Synchronization of status information between RAI_GetStatus and each unit should be managed by each unit (e.g. by notification of event on update of status).			
Note 2	T.B.D			

4.4.3. RAI_FW_GetUnitWorkAddress

Syntax	RAI_U32 RAI_FW_GetUnitWorkAddress (RAI_FW_HANDLE* unitHandle, RAI_U32** unitWorkAddr);			
Function	Obtain address of work area of unit			
Arguments	Type	Name	I/O	Description
	RAI_FW_HANDLE*	unitHandle	I	unit handle of Framework
	RAI_U32**	unitWorkAddr	O	the start address of work memory of unit
Return value	RAI_U32		error codes	
	RAI_FW_E_OK		OK	
	RAI_FW_E_PARAMETER_ERROR		parameter error	
Description	This function obtains the start address of work area of unit. This function can be called only from unit registration function. The start address of work area which can be obtained is the work area registered by RAI_FW_RegisterUnit. The work area should be defined and managed by each unit.			
Note 1	T.B.D			
Note 2				

5. Process flow

5.1. Initialization

Figure 5.1 shows initialization flow.

When the application calls RAI_Init, programs are downloaded to DSP, then DSP is reset, then user program in DSP starts. Initialization of Radar Framework and registration of units should be done in user program. Radar Framework notifies completion of initialization to Radar API when it is initialized by RAI_FW_Init and ready to receive commands. RAI_Init exits after the notification. Multiple units can be registered (max. RAI_UNIT_MAX). Call RAI_FW_RegisterUnit for the number of units to be registered. After all units are registered, call RAI_FW_Execute and start Radar Framework.

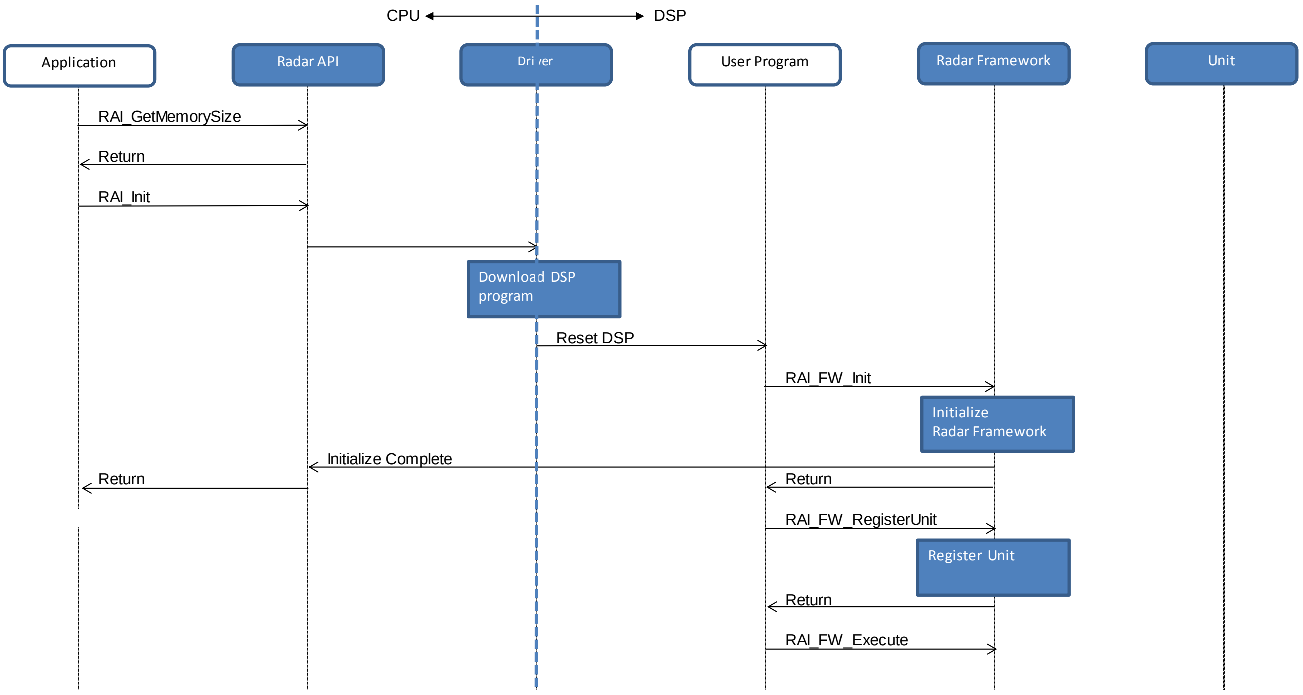


Figure 5.1 Initialization flow

After call of RAI_FW_Execute, the control does not return to user program until RAI_DeInit is called (T.B.D). So user program is omitted for the following flow charts.

5.2. Sending command

Figure 5.2 shows the flow of Radar API to send command from CPU to DSP.
In case when a Radar API function with sending command from CPU to DSP, the function exits without waiting for ACK from DSP (i.e. asynchronous function). Next asynchronous function can't be called until ACK is returned from DSP ((1) in following chart). To check whether ACK is returned from DSP or not, call RAI_GetStatus.

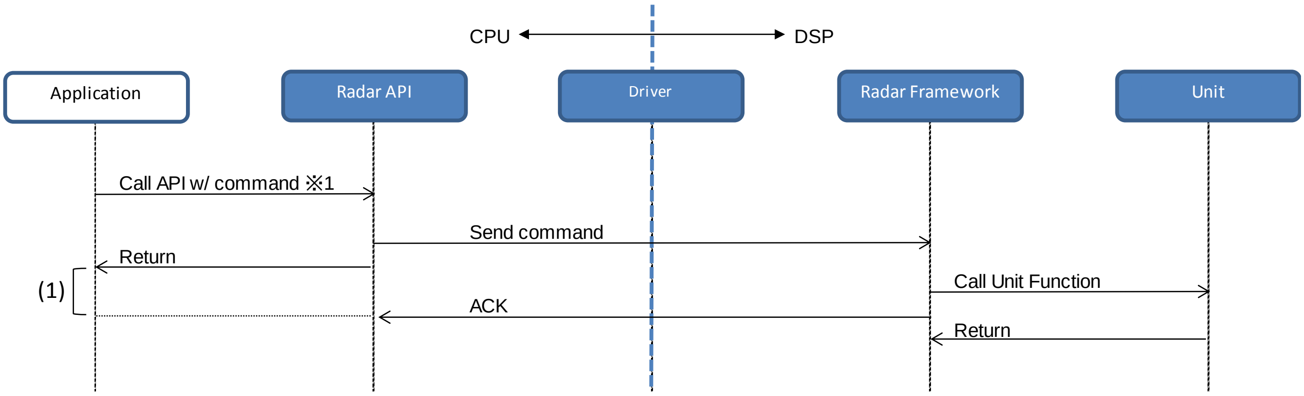


Figure 5.2 Flow to send command

※1 Table 5.1 shows the asynchronous Radar APIs with sending command from CPU to DSP

Table 5.1 list of asynchronous functions

Function Name	Description
RAI_GetHandle	Obtain unit handle
RAI_FreeHandle	Release unit handle
RAI_SetConfig	Send config information to unit
RAI_SendCommand	Send command to unit

5.2.1. RAI_GetHandle calling flow

Figure 5.3 shows the flow when RAI_GetHandle is called.
When RAI_GetHandle is called, Radar API send GetHandle command to Radar Framework. When When Radar Framework receives the command, it calls pfUnitGetHandle callback function specified by RAI_FW_RegisterUnit of corresponding unit.

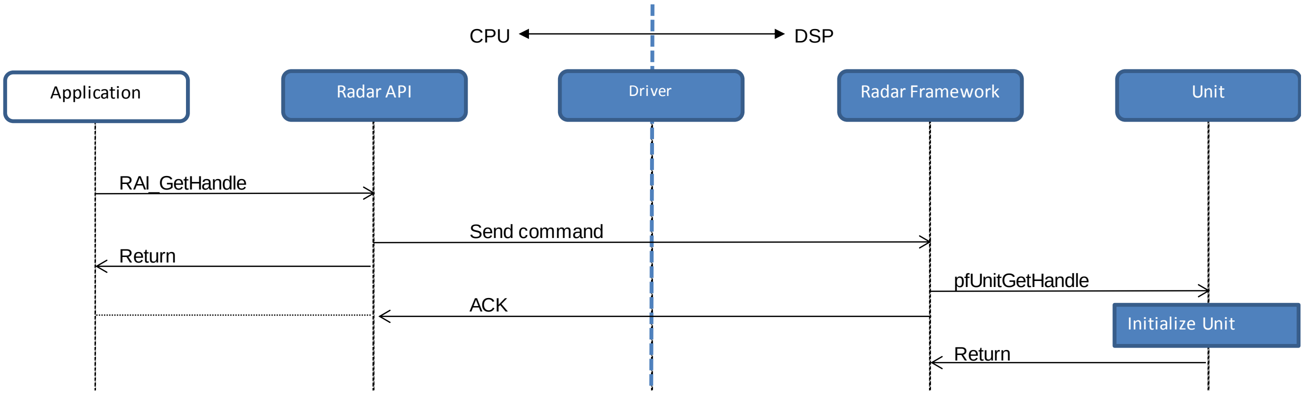


Figure 5.3 RAI_GetHandle flow

5.2.2. RAI_FreeHandle calling flow

Figure 5.4 shows the flow when RAI_FreeHandle is called.
When RAI_FreeHandle is called, Radar API send FreeHandle command to Radar Framework. When Radar Framework receives the command, it calls pfUnitFreeHandle callback function specified by RAI_FW_RegisterUnit of corresponding unit.

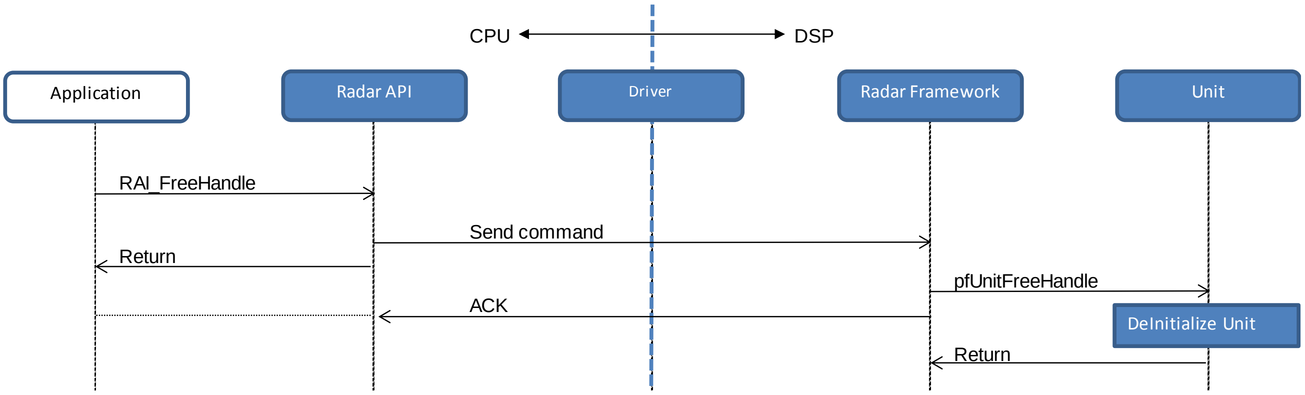


Figure 5.4 RAI_FreeHandle flow

5.2.3. RAI_SetConfig calling flow

Figure 5.5 shows the flow when RAI_SetConfig is called.
When RAI_SetConfig is called, Radar API send SetCofig command to Radar Framework. When Radar Framework receives the command, it calls pfUnitSetConfig callback function specified by RAI_FW_RegisterUnit of corresponding unit.

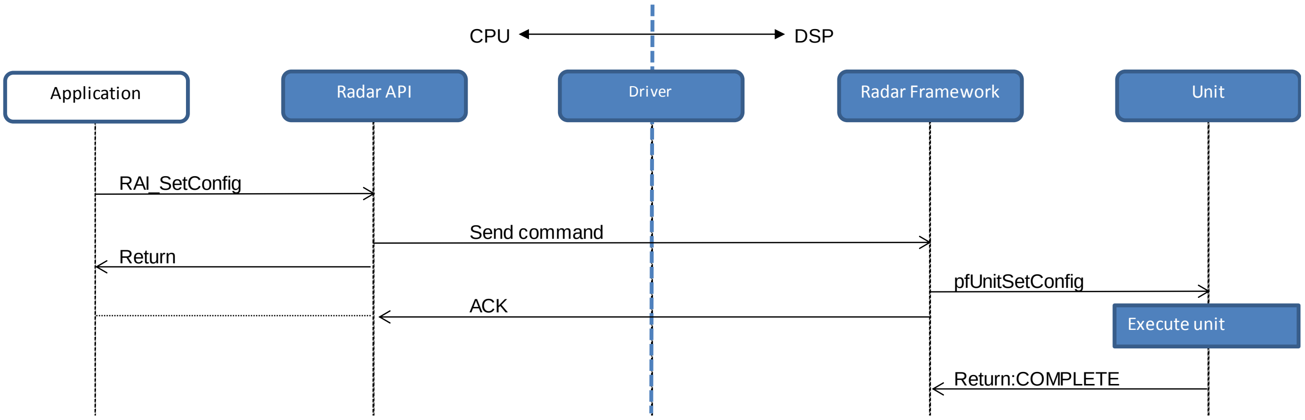


Figure 5.5 RAI_SetConfig flow

5.2.4. RAI_SendCommand calling flow

Figure 5.6 and Figure 5.7 show the flow when RAI_SendCommand is called.
When RAI_SendCommand is called, Radar API send SendCommand command to Radar Framework. When Radar Framework receives the command, it calls pfUnitSendCommand callback function specified by RAI_FW_RegisterUnit of corresponding unit.
If pfUnitSendCommand callback function returns RAI_FW_RESULT_COMPLETE, Framework finishes the process of pfUnitSendCommand callback function (Figure 5.6).

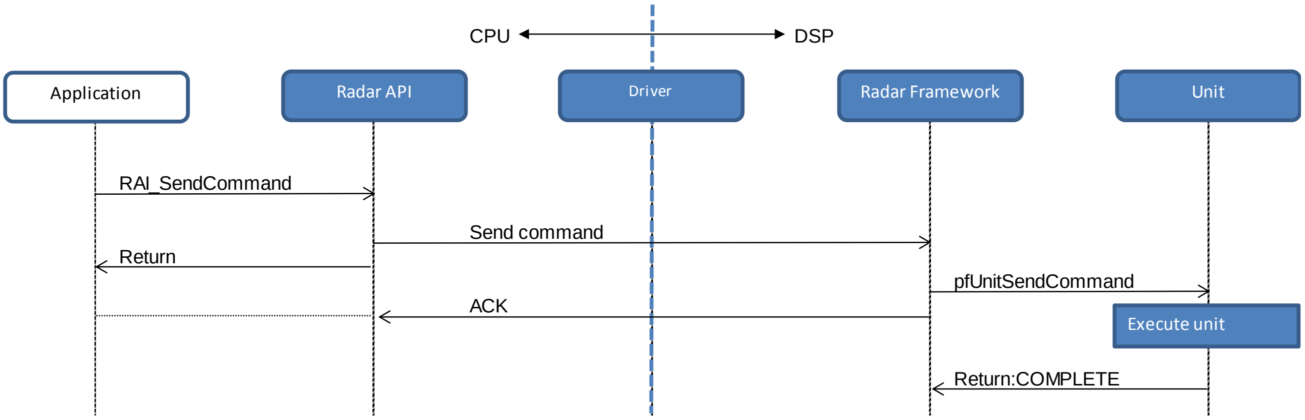


Figure 5.6 RAI_SendCommand flow(1)

If pfUnitSendCommand callback function returns RAI_FW_RESULT_CONTINUE, Framework schedules the task (Figure 5.7).
If another task by a unit with higher priority has already been scheduled, it is executed in advance. And if RAI_SendCommand is called for the same unit while CONTINUE, the tasks are called as scheduled order.

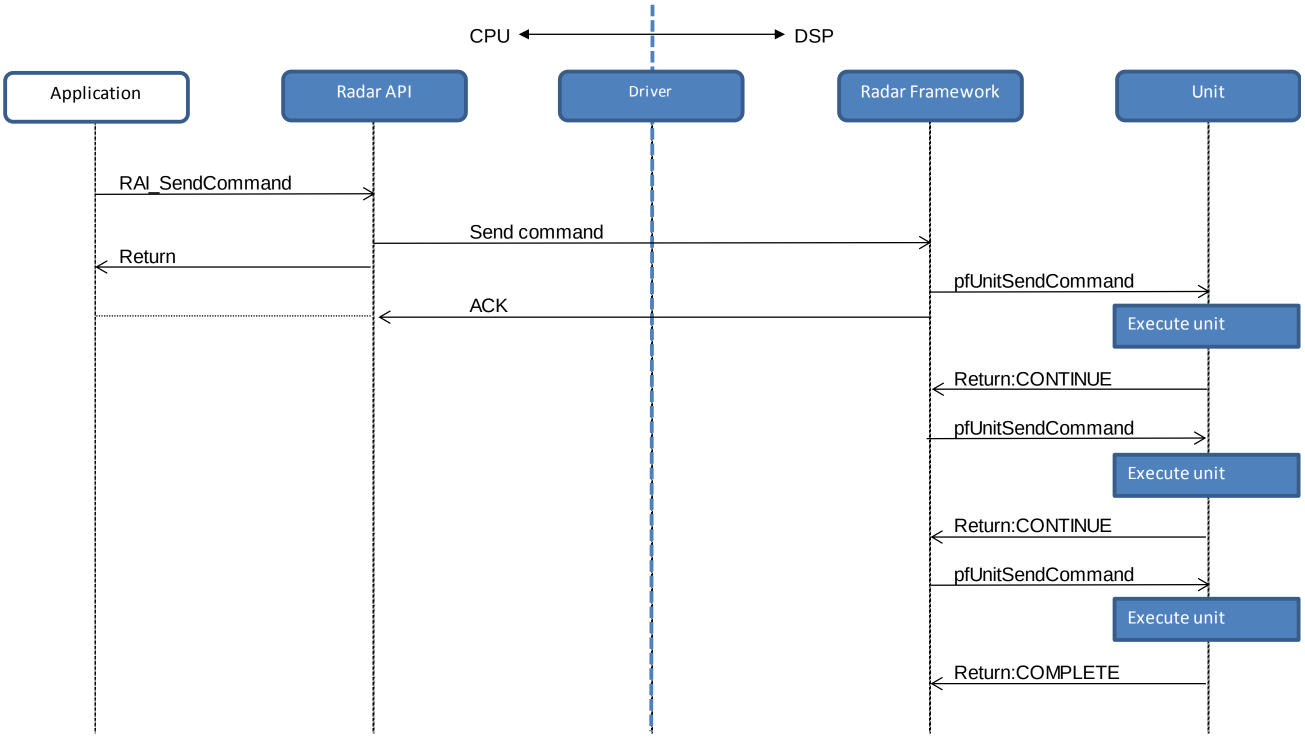


Figure 5.7 RAI_SendCommand flow (2)

6. Application consideration

Application consideration for developing user programs.

6.1. Function call

User programs which calls the functions in this specification should obey the calling rules of compiler.

6.1.1. The timing a function is executed

6.2. Other notes

6.2.1. Allocation of memory

Before calling the functions in this specification, allocate necessary memory area and each structure used for the parameters of each function.

6.2.2. Out of range memory access

The functions in this specification never access out of allocated memory or related I/O.

6.2.3. Combination with other applications

Take care not to duplicate symbol names when other applications are combined with radar programs.

6.2.4. Supervision of software

Supervise the system by watchdog timer and so on to avoid hang-up and implement timeout processing routine to upper program.

Revision history	RH850/V1R-M Radar Software User's Manual Radar API Part

Rev.	Date	Description	
		Page	Summary
0.01	2015.12.08	ー	Translated from Japanese version 0.01.

RH850/V1R-M Radar Software User's Manual
Radar API Part

Publication Date : Dec. XX, 2015 Rev. 0.01
Published by: Renesas Electronics Corporation

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
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: +44-1628-585-100, 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 1601-1611, 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, HALII Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.0

RH850/V1R-M Radar Software User's Manual

Renesas Electronics Corporation