- Table of Contents -

[1. Overview 3](#_Toc6909597)

[1.1. Overview 3](#_Toc6909598)

[1.2. The architecture of the Software and scope of this document 3](#_Toc6909599)

[1.3. Specification overview 4](#_Toc6909600)

[1.4. Memory specification 4](#_Toc6909601)

[1.5. Interrupt specification 4](#_Toc6909602)

[1.6. Related documents 5](#_Toc6909603)

[1.7. Type definitions 5](#_Toc6909604)

[2. Software specification 6](#_Toc6909605)

[2.1. State transition 6](#_Toc6909606)

[2.1.1. Initialization Stage 7](#_Toc6909607)

[2.1.2. Runtime Initialization Stage 8](#_Toc6909608)

[2.1.3. Execution Stage 9](#_Toc6909609)

[2.2. Memory structure 10](#_Toc6909610)

[2.3. Message structure 11](#_Toc6909611)

[2.4. Opcodes 12](#_Toc6909612)

[2.5. Structures 13](#_Toc6909613)

[2.6. Messages 14](#_Toc6909614)

[2.6.1. XF\_UNREGISTER 14](#_Toc6909615)

[2.6.2. XF\_REGISTER 14](#_Toc6909616)

[2.6.3. XF\_ROUTE 14](#_Toc6909617)

[2.6.4. XF\_UNROUTE 15](#_Toc6909618)

[2.6.5. XF\_ALLOC 15](#_Toc6909619)

[2.6.6. XF\_FREE 15](#_Toc6909620)

[2.6.7. XF\_SET\_PARAM 15](#_Toc6909621)

[2.6.8. XF\_GET\_PARAM 16](#_Toc6909622)

[2.6.9. XF\_EMPTY\_THIS\_BUFFER 16](#_Toc6909623)

[2.6.10. XF\_FILL\_THIS\_BUFFER 17](#_Toc6909624)

[2.6.11. XF\_FLUSH 17](#_Toc6909625)

[2.6.12. XF\_PAUSE 18](#_Toc6909626)

[2.6.13. XF\_RESUME 18](#_Toc6909627)

[2.6.14. XF\_MMAP\_THIS\_BUFFER 18](#_Toc6909628)

[3. Notes 19](#_Toc6909629)

[3.1. Function Call 19](#_Toc6909630)

[3.2. Other notes 19](#_Toc6909631)

[3.2.1. Allocation of memory 19](#_Toc6909632)

[3.2.2. Out of range memory access 19](#_Toc6909633)

[3.2.3. Combination with other applications 19](#_Toc6909634)

[3.2.4. Monitoring on Performance 19](#_Toc6909635)

- List of Figures -

[Figure 1‑1 The software architecture 3](#_Toc6909636)

[Figure 2‑1 ADSP Plugin state transition 6](#_Toc6909637)

[Figure 2‑2 Initialization Stage flow chart 7](#_Toc6909638)

[Figure 2‑3 Runtime Initialization Stage flow chart 8](#_Toc6909639)

[Figure 2‑4 Execution Stage　flow chart 9](#_Toc6909640)

[Figure 2‑5 Shared buffer structure 10](#_Toc6909641)

[Figure 2‑6 The structure of message buffer 11](#_Toc6909642)

- List of Tables -

[Table 1‑1 Specification overview 4](#_Toc6909643)

[Table 1‑2 Memory map 4](#_Toc6909644)

[Table 1‑3 Interrupt specification 4](#_Toc6909645)

[Table 1‑4 Related documents 5](#_Toc6909646)

[Table 1‑5 Type definitions 5](#_Toc6909647)

[Table 2‑1 Shared buffer information 10](#_Toc6909648)

[Table 2‑2 The list of macros to configure shared buffer 10](#_Toc6909649)

[Table 2‑3 Explanation of message structure 11](#_Toc6909650)

[Table 2‑4 Opcodes 12](#_Toc6909651)

[Table 2‑5 Structures 13](#_Toc6909652)

RCG3AHFWN0101ZDPE

Rev. 1.00

Jul. 10, 2017

ADSP Framework User's Manual

# Overview

## Overview

In this chapter, overview of is explained. All expression in this manual is based on default parameters. If the software is customized from original source code, make the appropriate change according to the customization.

## The architecture of the Software and scope of this document

The architecture of is shown in Figure 1‑1. is a software which operates on ADSP and control according to messages from .

ARM

User Application

ADSP

Codec

DAC/ADC

ARM

Audio HW

SCU/ SSI/

ADMA

ADSP Interface

NC/EC

ADSP Driver

ADSP Framework

ADSP Reference Renderer Plugin

ADSP Reference Equalizer Plugin

User Space

Kernel Space

This document’s target is in side of red square.

Figure 1‑1 The software architecture

## Specification overview

Table 1‑1 shows the specification overview of .

Table 1‑1 Specification overview

|  |  |
| --- | --- |
| Item | Outline |
| DSP | HiFi2 by Cadence Design Systems, Inc. |
| IDE | Xtensa Xplorer 7.0.4 (RG-2016.4) |
| Configuration | hifi2\_rcar\_rg20164c |
| Memory map | adspfwk\_r001c |
| Interrupt | call0 ABI |
| Compiler | Xtensa C and C++ Compiler (Version 12.0.4) |
| Endian | Little endian |
| OS | XOS by Cadence Design Systems, Inc. |

[note1] This software adopts call0 ABI. Windowed ABI is not supported.

[note2] This software adopts XOS. XTOS is not supported. XTHAL can also be used.

## Memory specification

Table 1‑2 Memory specification of . See 2.2 for the structure of shared buffer area.

Table 1‑2 Memory map

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Allocation | Item | Cache type | start address | size |
| Internal memory | I-TCM0 | － | 0xECE8\_0000 | 0x0001\_0000 |
| I-TCM1 | － | 0xECE9\_0000 | 0x0001\_0000 |
| D-TCM0 | － | 0xECE7\_8000 | 0x0000\_8000 |
| D-TCM1 | － | 0xECE6\_0000 | 0x0001\_0000 |
| External memory | Debug area | Read Cache / Writeback Cache | 0x5700\_0000 | 0x0010\_0000 |
| Code area | Read Cache / Writeback Cache | 0x5710\_0000 | 0x0030\_0000 |
| Shared buffer area | Read Cache / Writeback Cache | 0x5740\_0000 | 0x00C0\_0000 |

## Interrupt specification

Table 1‑3 Interrupt specification shows the interrupts used by .

Table 1‑3 Interrupt specification

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | type | source | direction | usage |
| 15 | edge | IRQIN\_SET[0] | CPU to ADSP | send command |
| － | Level | IRQOUT\_SET | ADSP to CPU | send response |
| 0 | Level | Output FIFO 0 | FIFO→ADSP | Output FIFO 0 Interrupt |
| 1 | Level | Input FIFO 0 | FIFO→ADSP | Input FIFO 0 Interrupt |
| 2 | Level | Output FIFO 1 | FIFO→ADSP | Output FIFO 1 Interrupt |
| 3 | Level | Input FIFO 1 | FIFO→ADSP | Input FIFO 1 Interrupt |
| 4 | Level | Output FIFO 2 | FIFO→ADSP | Output FIFO2 Interrupt |
| 5 | Level | Input FIFO 2 | FIFO→ADSP | Input FIFO 2 Interrupt |
| 8 | Level | Output FIFO 3 | FIFO→ADSP | Output FIFO 3 Interrupt |
| 9 | Level | Input FIFO 3 | FIFO→ADSP | Input FIFO3 Interrupt |

## Related documents

Table 1‑4 shows related documents and references.

Table 1‑4 Related documents

|  |  |  |
| --- | --- | --- |
| No. | Name | Published by |
| [1] | R-Car Series, 3rd Generation User’s Manual: Hardware | Renesas Electronics Corporation |
| [2] | Xtensa® LX4 Microprocessor Data Book | Tensilica, Inc. |
| [3] | Xtensa® XOS Reference Manual | Cadence Design Systems, Inc. |
| [4] | Xtensa® System Software Reference Manual | Cadence Design Systems, Inc. |

## Type definitions

Table 1‑5 shows the type definitions used by .

Table 1‑5 Type definitions

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Size [byte] | Description | | | |
| s8 | 1 | signed 8bit integer | -128 | to | 127 |
| s16 | 2 | signed 16bit integer | -32768 | to | 32767 |
| s32 | 4 | signed 32bit integer | -2147483648 | to | 2147483647 |
| u8 | 1 | unsigned 8bit integer | 0 | to | 255 |
| u16 | 2 | unsigned 16bit integer | 0 | to | 65535 |
| u32 | 4 | unsigned 32bit integer | 0 | to | 4294967295 |

[note] The size of pointer depends on the architecture this software is used.

# Software specification

is an event-driven framework. Commands are stored in shared buffer, and their execution is triggered by interruption to . When notifies information, the information is stored in shared buffer and then is performed by storing information to shared buffer and issues interrupt.

## State transition

Figure 2‑1 shows state transition of each ADSP Plugin. See 2.1.1, 2.1.2 or 2.1.3 for the flows in each stage.

Initialization Stage

Runtime Initialization Stage

Execution Stage

Figure 2‑1 state transition

### Initialization Stage

FD\_FWK\_CMN\_001

API Handle Allocation

api = NULL

cmd = XA\_API\_CMD\_GET\_API\_SIZE

param = <output value ptr>

Pre-Initialization

cmd = XA\_API\_CMD\_INIT

idx = XA\_CMD\_TYPE\_INIT\_API\_PRE\_CONFIG\_PARAMS

param = NULL

Configuration

cmd = XA\_API\_CMD\_SET\_CONFIG\_PARAM

idx = <param-id>

param = <param-value-ptr>

Post-Initialization

cmd = XA\_API\_CMD\_INIT

idx = XA\_CMD\_TYPE\_INIT\_API\_POST\_CONFIG\_PARAMS

param = NULL

Memory Allocation

cmd = XA\_API\_CMD\_SET\_MEM\_PTR

idx = <index-of-memory-buffer>

param = <memory-buffer-ptr>

[Covers: RD\_010]

Figure 2‑2 Initialization Stage flow chart

Initialization Stage performs allocation of resources for ADSP Plugin and configures static parameters. This stage consists of the following steps.

1. API Handle Allocation  
   It is the start of ADSP Plugin lifecycle. It requires the size of ADSP Plugin API structure.
2. Pre-Initialization  
   It is the entry point for ADSP Plugin. It initializes the API structure allocated by ADSP Framework and configures the default value.
3. Configuration  
   It configures the static parameters of ADSP Plugin.
4. Post-Initialization  
   It means the configure of the static parameters is finished. It fixes persistent area, scratch area and input buffer area. The size of each buffer can’t be increased after this step.
5. Memory Allocation  
   ADSP Framework allocates ADSP Plugin buffer and registers it to ADSP Plugin. It is possible to exchange data with ADSP Plugin after this step.

### Runtime Initialization Stage

FD\_FWK\_CMN\_002

Input/Output Buffer Submission

cmd = XA\_API\_CMD\_SET\_MEM\_PTR

idx = <io-buffer-idx>

param = <io-buffer-ptr>

Runtime Initialization

cmd = XA\_API\_CMD\_INIT

idx = XA\_CMD\_TYPE\_INIT\_PROCESS

param = NULL

Runtime Initialization Status Querying

cmd = XA\_API\_CMD\_INIT

idx = XA\_CMD\_TYPE\_INIT\_DONE

param = <status-ptr>

Query Effective Parameters

cmd = XA\_API\_CMD\_GET\_CONFIG\_PARAM

idx = <param-id>

param = <param-value-ptr>

Runtime Initialized?

No

Yes

[Covers: RD\_010]

Figure 2‑3 Runtime Initialization Stage flow chart

Runtime Initialization Stage performs configuration of parameter which depends on input data and so on. It checks input data but never make output data.

### Execution Stage

FD\_FWK\_CMN\_003

Input/Output Buffer Submission

cmd = XA\_API\_CMD\_SET\_MEM\_PTR

idx = <io-buffer-idx>

param = <io-buffer-ptr>

Execution

cmd = XA\_API\_CMD\_EXECUTE

idx = XA\_CMD\_TYPE\_DO\_EXECUTE

param = NULL

Execution Completion Status Querying

cmd = XA\_API\_CMD\_EXECUTE

idx = XA\_CMD\_TYPE\_DONE\_QUERY

param = <status-ptr>

Reset Execution Runtime

cmd = XA\_API\_CMD\_EXECUTE

idx = XA\_CMD\_TYPE\_DO\_RUNTIME\_INIT

param = NULL

Execution Completed?

No

Yes

[Covers: RD\_010]

Figure 2‑4 Execution Stage　flow chart

Execution Stage performs ADSP Plugin processing. The way to process input and output data in ADSP Plugin should be determined at the beginning of this stage.

ADSP Framework usually processes input and output data successively except when a physical cause inputs or outputs data e.g. Renderer plugin and Capturer plugin.

When ADSP Framework receives notification that input data is ended (see 2.6.9), it provides all input data, output all output data, and then finishes this stage. After this stage is finished, it is necessary to shift the state to Runtime Initialization Stage to run ADSP Plugin process.

It is possible to shift the state to Runtime Initialization Stage (see 2.6.10) at any time in this stage.

## Memory structure

Figure 2‑5 shows the memory structure of shared buffer in . Table 2‑1 shows the default configuration of the shared buffer. It is possible to extend the size of control area and data area in total up to 12 MB. Table 2‑2 shows the macro which defines buffer.

Command Control

Response Control

Data Area

0x5740\_0000

0x5740\_3000

XF\_PROXY\_DATA\_ADDRESS

XF\_CFG\_REMOTE\_IPC\_POOL\_SIZE

XF\_PROXY\_DATA\_SIZE

XF\_PROXY\_MESSAGE\_QUEUE\_LENGTH

Padding

Figure 2‑5 Shared buffer structure

Table 2‑1 Shared buffer information

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| type | buffer size | number of buffers | total size | remarks |
| command control area | 16 | 256 | 4,096 | shared buffer information |
| 8 | 1 | 8 | index information |
| response control area | 16 | 256 | 4,096 | shared buffer information |
| 8 | 1 | 8 | index information |
| data area | － | － | 262,144 | shared buffer information |

[note] There may be padding area between each area though not shown in Figure 2‑5.

Table 2‑2 The list of macros to configure shared buffer

|  |  |  |
| --- | --- | --- |
| macro | initial value | remarks |
| XF\_PROXY\_DATA\_ADDRESS | 0x57400000 | The start address of shared buffer |
| XF\_PROXY\_DATA\_SIZE | 0xC00000 | The size of shared buffer |
| XF\_PROXY\_MESSAGE\_QUEUE\_LENGTH | 256 | The number of communication control area |
| XF\_CFG\_REMOTE\_IPC\_POOL\_SIZE | 262,144 | The effective size of data area |

## Message structure

id

opcode

length

address

0bit

31

buffer

(See Section 2.5)

0bit

31

Command / Response  
Control

Data Area

next

Figure 2‑6 The structure of message buffer

Table 2‑3 Explanation of message structure

|  |  |
| --- | --- |
| Member | Outline |
| next | This is the pointer to point to next item in the list. |
| id | This is the ID for internal administration. Users need not take care of it. |
| opcode | This is message opcode. See 2.4 for details.  It is configured according to “opcode” parameter when xf\_command in ADSP Interface is executed. |
| length | It is configured according to “length” parameter when xf\_command in ADSP Interface is executed. |
| address | It is configured according to “buffer” parameter when xf\_command in ADSP Interface is executed. |
| buffer | This is the buffer used for communication. The structure changes according to opcode. |

## Opcodes

Table 2‑4 shows the opcodes provided by this software. See 2.6 for details.

Table 2‑4 Opcodes

|  |  |
| --- | --- |
| Opcodes | Outline |
| XF\_UNREGISTER | This opcode unregisters client. |
| XF\_REGISTER | This opcode registers client. |
| XF\_ROUTE | This opcode connects the ports between 2 ADSP plugin. |
| XF\_UNROUTE | This opcode disconnects the connection between the ports of 2 ADSP plugin. |
| XF\_ALLOC | This opcode allocates shared memory buffer. |
| XF\_FREE | This opcode deallocates the shared memory buffer. |
| XF\_SET\_PARAM | This opcode sets parameters. |
| XF\_GET\_PARAM | This opcode gets parameters. |
| XF\_EMPTY\_THIS\_BUFFER | This opcode sends input buffer. It also notifies the end of file by sending buffer with 0 byte data. |
| XF\_FILL\_THIS\_BUFFER | This opcode sends output buffer. It also shifts the state of to Runtime Initialization Stage by sending buffer with 0 byte data. |
| XF\_FLUSH | This opcode clears buffers for specific port. |
| XF\_PAUSE | This opcode pause component operation. |
| XF\_RESUME | This opcode resumes component operation. |
| XF\_MMAP\_THIS\_BUFFER | This opcode maps buffer in shared memory/local memory to data buffer in ADSP plugin |

## Structures

Table 2‑5 shows the structures to define the structure of buffer. See 2.6 for details.

Table 2‑5 Structures

|  |  |  |
| --- | --- | --- |
| Name | Outline | Remarks |
| xf\_set\_param\_msg\_t | Structure for XF\_SET\_PARAM |  |
| xf\_get\_param\_msg\_t | Structure for XF\_GET\_PARAM |  |
| xf\_start\_msg\_t | Structure for Runtime Initialization Stage | It is used for response transmit when Runtime Initialization Stage is finished. |
| xf\_user\_msg\_t | Structure for user message | Used for response transmit. |

## Messages

Below show the expression of message specification in this chapter.

Template for this chapter:

|  |  |
| --- | --- |
| Command code for opcode parameter of xf\_command in ADSP Interface. | |
| Synopsis | Explanation of command code |
| Buffer structure | Buffer structure for buffer parameter of xf\_command in ADSP Interface. |
| Parameters | Explanation of each parameter in buffer structure |

FD\_FWK\_CMN\_004

|  |  |
| --- | --- |
| XF\_UNREGISTER | |
| Synopsis | This opcode sends XF\_UNREGISTER message to ADSP Plugin.  It will terminate component and used to generate error-response message for originator. |
| Buffer structure | None |
| Parameters | None |

[Covers: RD\_010]

FD\_FWK\_CMN\_005

|  |  |  |
| --- | --- | --- |
| XF\_REGISTER | | |
| Synopsis | This opcode sends XF\_REGISTER message to ADSP Plugin.  It will allocate new client id and create new component along with its port specification. | |
| Buffer structure | void \*buffer | |
| Parameters | buffer | ID string of component. |

[Covers: RD\_010]

FD\_FWK\_CMN\_006

|  |  |  |
| --- | --- | --- |
| XF\_ROUTE | | |
| Synopsis | This opcode sends XF\_ROUTE message to ADSP Plugin.  It routes output port and create message to communicate between 2 ADSP plugins. | |
| Buffer structure | typedef struct xf\_route\_port\_msg  {  u32 src;  u32 dst;  u32 alloc\_number;  u32 alloc\_size;  u32 alloc\_align;  } xf\_route\_port\_msg\_t; | |
| Parameters | src | Source port specification. |
| dst | Destination port specification. |
| alloc\_number | Number of buffers to allocate. |
| alloc\_size | Length of buffers to allocate. |
| alloc\_align | Alignment restriction for a buffer. |

[Covers: RD\_010]

FD\_FWK\_CMN\_007

|  |  |  |
| --- | --- | --- |
| XF\_UNROUTE | | |
| Synopsis | This opcode sends XF\_UNROUTE message to ADSP Plugin.  It un-routes output port and destroy all memory buffers allocated between 2 ADSP plugin. | |
| Buffer structure | typedef struct xf\_unroute\_port\_msg  {  u32 src;  u32 dst;  } xf\_unroute\_port\_msg\_t; | |
| Parameters | src | Source port specification. |
| dst | Destination port specification. |

[Covers: RD\_010]

|  |  |
| --- | --- |
| XF\_ALLOC | |
| Synopsis | This opcode sends XF\_ALLOC message to .  It allocates shared memory buffer. Both CPU and ADSP can read and write in this area. |
| Buffer structure | None |
| Parameters | None |

|  |  |  |
| --- | --- | --- |
| XF\_FREE | | |
| Synopsis | This opcode sends XF\_FREE message to .  It will free shared memory area. | |
| Buffer structure | void \*buffer | |
| Parameters | buffer | Pointer to point to the shared memory. |

FD\_FWK\_CMN\_008

|  |  |  |
| --- | --- | --- |
| XF\_SET\_PARAM | | |
| Synopsis | This opcode sends message to .  It can send multiple sub-commands and parameters at a time by adding the number of items. | |
| Buffer structure | typedef struct xf\_set\_param\_item  {  u32 id;  u32 value;  } xf\_set\_param\_item\_t;  typedef struct xf\_set\_param\_msg  {  xf\_set\_param\_item\_t item[0];  } xf\_set\_param\_msg\_t; | |
| Parameters | id | Sub-command to ADSP Plugin |
| value | Parameters to ADSP Plugin |

[Covers: RD\_010]

FD\_FWK\_CMN\_009

|  |  |  |
| --- | --- | --- |
| XF\_GET\_PARAM | | |
| Synopsis | This opcode sends XF\_GET\_PARAM message to .  It can send / receive multiple sub-commands and parameters at a time by adding the number of ids. | |
| Buffer structure | typedef union xf\_get\_param\_msg  {  struct  {  u32 id[0];  } c;  struct  {  u32 value[0];  } r;  } xf\_get\_param\_msg\_t; | |
| Parameters | id | Sub-command to ADSP Plugin |
| value | Parameters received from ADSP Plugin by callback. |

[Covers: RD\_010]

### XF\_EMPTY\_THIS\_BUFFER

FD\_FWK\_CMN\_010

|  |  |  |
| --- | --- | --- |
| XF\_EMPTY\_THIS\_BUFFER | | |
| Synopsis | This opcode sends XF\_EMPTY\_THIS\_BUFFER message to .  There are 2 usages for this message.  (1) It sends the data in buffer as input buffer. The size is defined by length parameter.  (2) It notifies the end of data when buffer is set to NULL and length parameter in xf\_command is set to 0. | |
| Buffer structure | void \*buffer; | |
| Parameters | buffer | Buffer to send to ADSP Plugin |

[Covers: RD\_010]

### XF\_FILL\_THIS\_BUFFER

FD\_FWK\_CMN\_011

|  |  |  |
| --- | --- | --- |
| XF\_FILL\_THIS\_BUFFER | | |
| Synopsis | This opcode sends XF\_FILL\_THIS\_BUFFER message to .  There are 2 usages for this message.  (1) It sends the data in buffer as output buffer. The size is defined by length parameter.  (2) It shifts the state of to Runtime Initialization Stage and notifies initial information by callback function when length parameter in xf\_command is set to 0. | |
| Buffer structure | void \*buffer; | |
| Parameters | buffer | Buffer to send to ADSP Plugin |

[Covers: RD\_010]

### XF\_FLUSH

FD\_FWK\_CMN\_012

|  |  |
| --- | --- |
| XF\_FLUSH | |
| Synopsis | This opcode sends XF\_FLUSH message to .  It is mainly used for trick-play.  It deletes the buffer send by XF\_EMPTY\_THIS\_BUFFER immediately and call callback functions of each buffer.  When output port of is connected to the input port of another plugin, the command propagates to other plugins successively. If output port is not connected to other plugins, it calls callback function in this message. |
| Buffer structure | None |
| Parameters | None |

[Covers: RD\_010]

### XF\_PAUSE

|  |  |
| --- | --- |
| XF\_PAUSE | |
| Synopsis | This opcode sends XF\_PAUSE message to .  It stops consumption of input buffer and transmitting of output buffer in . While pausing, it is possible to send new input buffer and output buffer to . |
| Buffer structure | None |
| Parameters | None |

### XF\_RESUME

|  |  |
| --- | --- |
| XF\_RESUME | |
| Synopsis | This opcode sends XF\_RESUME message to .  It resumes pause. |
| Buffer structure | None |
| Parameters | None |

### XF\_MMAP\_THIS\_BUFFER

FD\_FWK\_CMN\_013

|  |  |  |
| --- | --- | --- |
| XF\_MMAP\_THIS\_BUFFER | | |
| Synopsis | This opcode sends XF\_MMAP\_THIS\_BUFFER message to .  This sends a buffer’s address to plugin so that plugin will store that address for use. | |
| Buffer structure | void \*buffer; | |
| Parameters | buffer | Buffer to send to ADSP Plugin |

[Covers: RD\_015]

# Notes

This section describes the notice of developing user programs.

## Function Call

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

## Other notes

### Allocation of memory

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

### Out of range memory access

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

### Combination with other applications

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

### Monitoring on Performance

The products embedding this shall observe performance of the periodically with Watch Dog timer or such functions in order not to damage system performance.