# Audio Framework Specification



Renesas Electronics
www.renesas.com

Rev.1.00 Jun. 2013

# How to Use This Manual

1. Related Manuals

   None

2. Technical Terms and Abbreviations

| Technical term /Abbreviation | Description |
|---|---|
| OS | Operating system |
| CPU | Central processing unit |
| ADSP | Audio digital signal processor |
| OMX IL | OpenMax integration layer |
| OMX MC | OpenMax media component |
| IST | Interrupt service thread |
| OMX | OpenMax |

*Table 1 List of technical terms and abbreviation are used*

**Contents**

## Figures



## Tables

# 1 Introduction

The purpose of this document is to describe the whole image of the audio framework and each of its components as well as to explain the communication between them.

The document has 3 chapters:

- Chapter 1: Introduction
- Chapter 2: Overview of audio framework
- Chapter 3: Communication in the audio framework. This chapter explains the mechanism of execution and communication of the audio framework.

# 2 Overview of audio framework

The audio framework is an audio solution for a whole system, which has the capability to execute multiple codecs as well as easily to customize. The framework is divided into 2 main parts as the below figure: the upper part runs on ARM core and is called CPU side; the lower part is executed on Tensilica Audio DSP (ADSP) and called ADSP side.

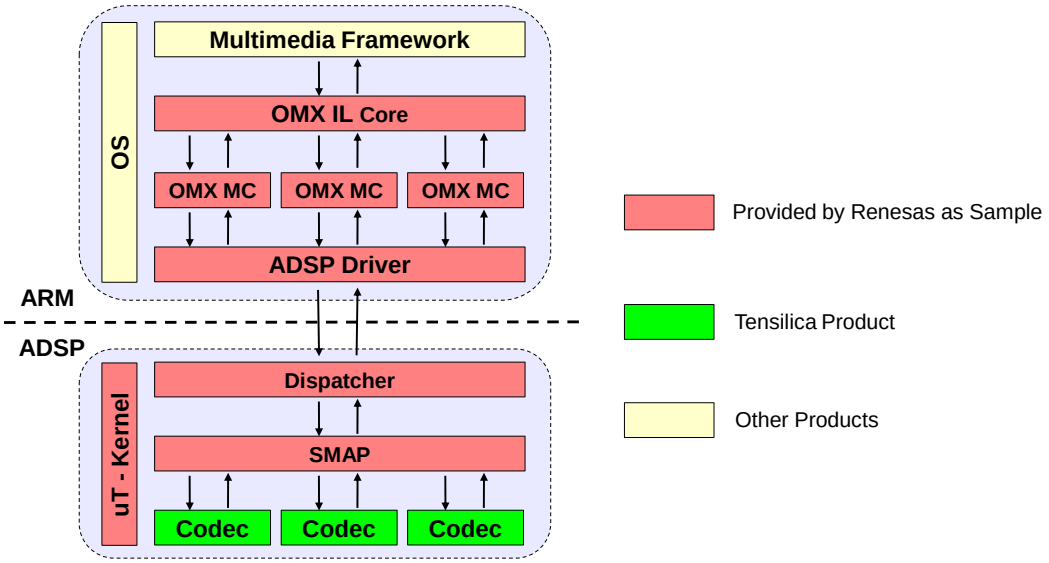


*Figure 2.1 Structure of audio framework*

The audio framework includes following modules:

- **Multimedia Framework:** The framework is provided by OS to support audio playback. For WEA7, DirectShow is the target multimedia framework.
- **OMX IL Core:** OMX interface to communicate between multimedia framework and OMX MC.
- **OMX MC:** OMX IL media component of each codec.
- **ADSP Driver:** Driver to control ADSP module.
- **Dispatcher:** Communicate with ADSP Driver to get the command from CPU side (ARM) and send the response from ADSP side (ADSP).
- **uT-kernel:** Real time OS is used for small embedded system.
- **SMAP:** The framework which utilizes the services provided by uT-Kernel to control the codec execution.
- **Codec:** Tensilica audio codec libraries

# 3 Communication in the Audio Framework

When the audio framework is executed, CPU side will run first. It allocates the working memory for ADSP side, download all parts of ADSP side to the allocated memory and activate ADSP side to run. After that, CPU and ADSP will communicate with each other continuously to complete the encoding / decoding process.

For the detailed memory allocation, please refer to section 4.2.

## 3.1 Communication between CPU and ADSP

In the audio framework, CPU and ADSP communicate with each other by using registers and shared memory.

### 3.1.1 Register

There are 3 types of registers:

- Input interrupt register: used by CPU to raise an interrupt to ADSP.
- Output interrupt register: used by ADSP to raise an interrupt to CPU.
- Communication register: used by CPU in the initialization phase to transfer the memory address to ADSP.

Please refer to table 4.1 for detailed description of each register

### 3.1.2 Shared memory

Shared memory is a memory area which can be read and written by both CPU and ADSP. In the initialization phase, ADSP driver allocates the memory for shared memory and returns the physical and logical address.

Shared memory includes 2 main parts:

- Outgoing data and incoming data contain the message queues
- Memory for parameter, input and output of codec in the encoding / decoding process

There are 2 message queues: command message queue and response message queue. Each queue has 256 messages. Message in the queue has the following structure:

- Session ID of a codec
- Operation code is used to define command type.
- Length of message content buffer
- Physical address of message buffer
- Virtual address (on CPU side) of message buffer

```
struct xa_proxy_message {
    /* ...session ID */
    u32                 session_id;

    /* ...proxy API command/reponse code */
    u32                 opcode;

    /* ...length of attached buffer */
    u32                 length;

    /* ...physical address of message buffer */
    u32                 phy_address;

    /* ...virtual address of message buffer */

    u32                 vir_address;
};
```

Outgoing data contains the command message queue and incoming data contains the response message queue. CPU pushes the command message to the command message queue; then ADSP will read and process it. When finishing processing, ADSP pushes the response message to the response message queue; then CPU will read it and know that the command has been executed. The structure of outgoing data and incoming data is described as below.

```
struct xa_proxy_host_data {
    /* ...command queue */
    struct xa_proxy_message command[XA_PROXY_MESSAGE_QUEUE_LENGTH];

    /* ...writing index into command queue */
    u32                     cmd_write_idx;

    /* ...reading index for response queue */
    u32                     rsp_read_idx;
};
```

The outgoing data has 3 fields:

- *Command message queue*: store the command messages of CPU
- *Command write index*: the writing index of CPU into the command queue
- *Response read index*: the reading index of CPU into the response queue of incoming data

```
struct xa_proxy_dsp_data {
    /* ...response queue */
    struct xa_proxy_message response[XA_PROXY_MESSAGE_QUEUE_LENGTH];

    /* ...writing index into response queue */
    u32                     rsp_write_idx;

    /* ...reading index for command queue */
    u32                     cmd_read_idx;
};
```

The incoming data has 3 similar fields:

- *Response queue*: store the response messages of ADSP
- *Response write index*: the writing index of ADSP into the response queue
- *Command read index*: the reading index of ADSP into the command queue of outgoing data

### 3.1.3 Communication process

Below is the diagram of communication between CPU and ADSP

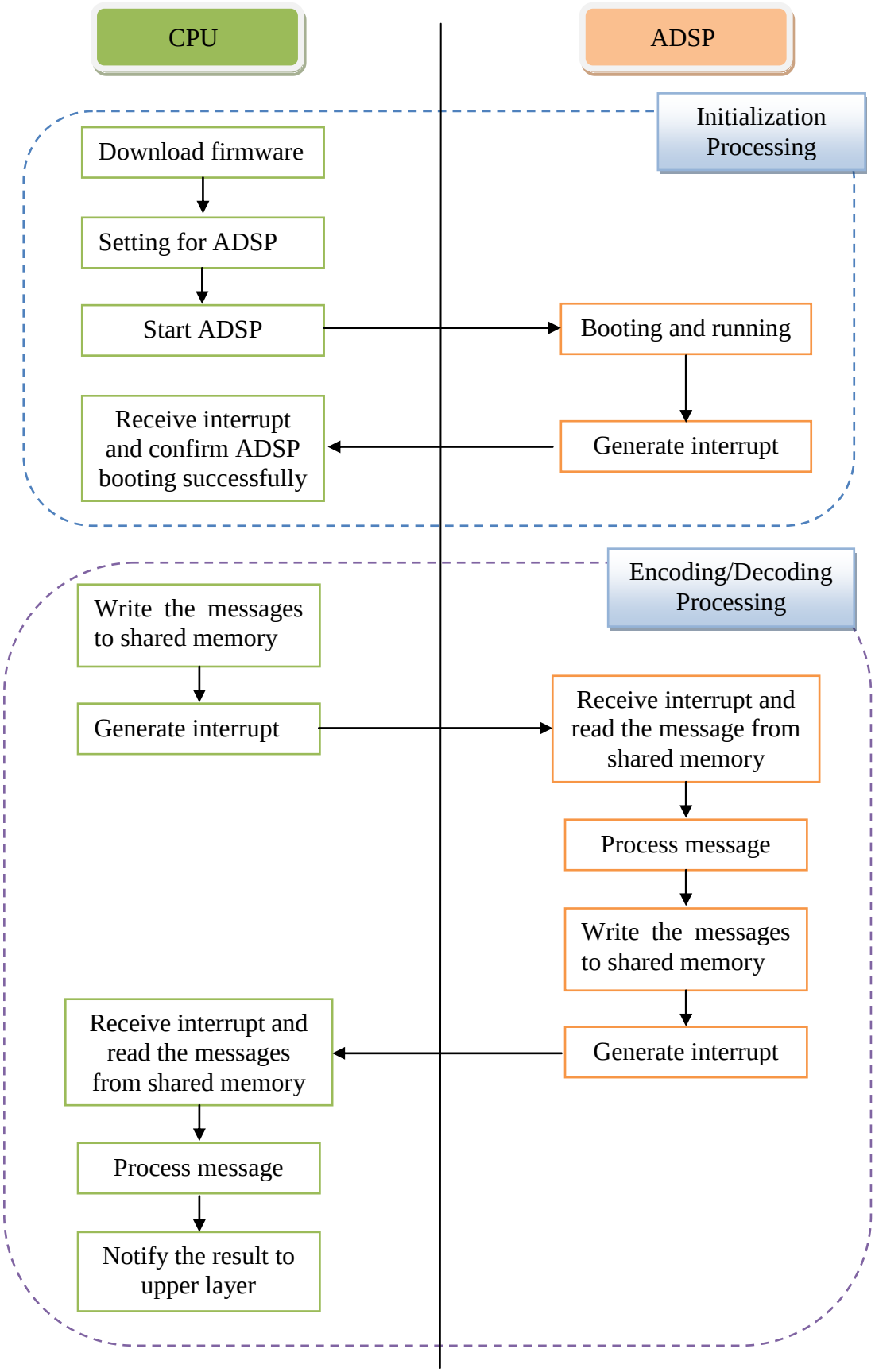


*Figure 3.1 Overview of communication between CPU and ADSP*

## 3.2 Communication inside CPU

This part explains how the parts of CPU side communicate (multimedia framework, OMX IL core, OMX MC, ADSP driver) each other in data transferring.

### 3.2.1 OMX MC

In CPU side, each OMX MC has two message lists, two threads and Proxy. They are created when an OMX MC is created. The following figure is a structure of OMX MC.

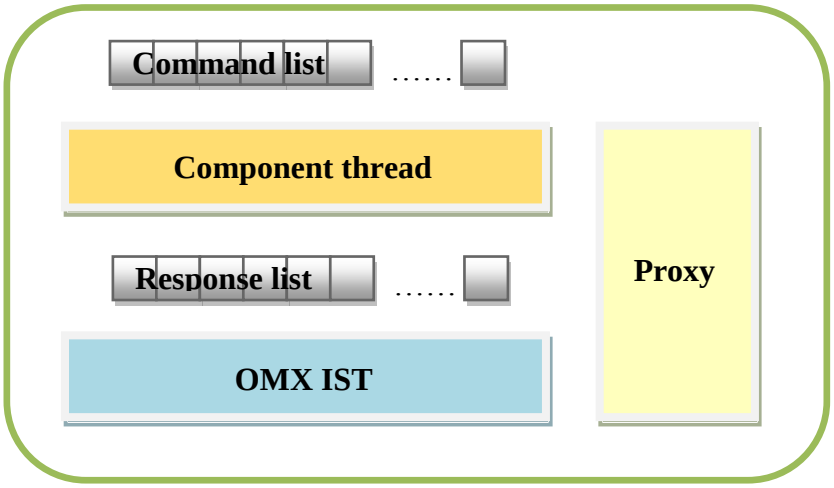


*Figure 3.2 The structure of an OMX MC*

Two message lists are command and response lists.

- Command list is used to receive the messages from application.
- Response list is used to receive the messages from ADSP side.

Two threads are Component thread and OMX interrupt service thread (IST)

- Component thread is used to send data to and receive data from ADSP side.
- OMX IST is used to receive the event (RECEIVE_EVENT) from ADSP driver, then read the message from shared memory and submit this message to response list of OMX MC.

Proxy is responsible to allocate the Firmware memory and the shared memory area of each of codec and write data through the functions of ADSP driver.

### 3.2.2 Send data from CPU to ADSP

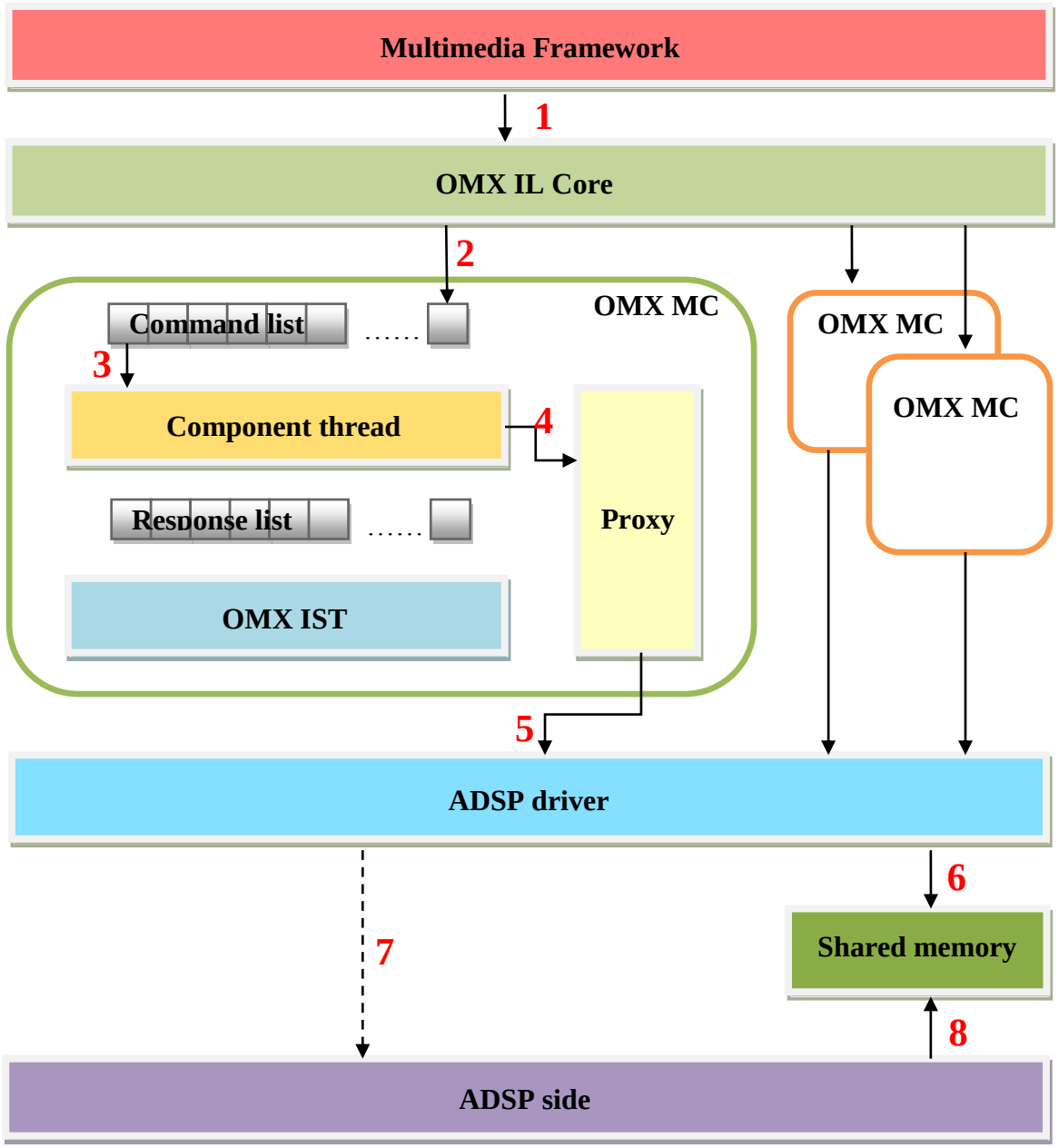


*Figure 3.3 Transferring data to ADSP side*

When the multimedia framework submits the messages (buffers or commands) through the functions or macros of OMX IL core (1), the messages are added to the tail of command list of an OMX MC (2). Besides, the Component thread polls to read the command and response list continuously. If the command list is not empty, the Component thread reads the first element of command list (3) and sends it to Proxy. Proxy writes this message the shared memory through the functions ADSP driver (4, 5, and 6). Then ADSP driver enables interrupt to ADSP side (7) to get data for encoding or decoding of ADSP side (8).

### 3.2.3 Receive data from ADSP side

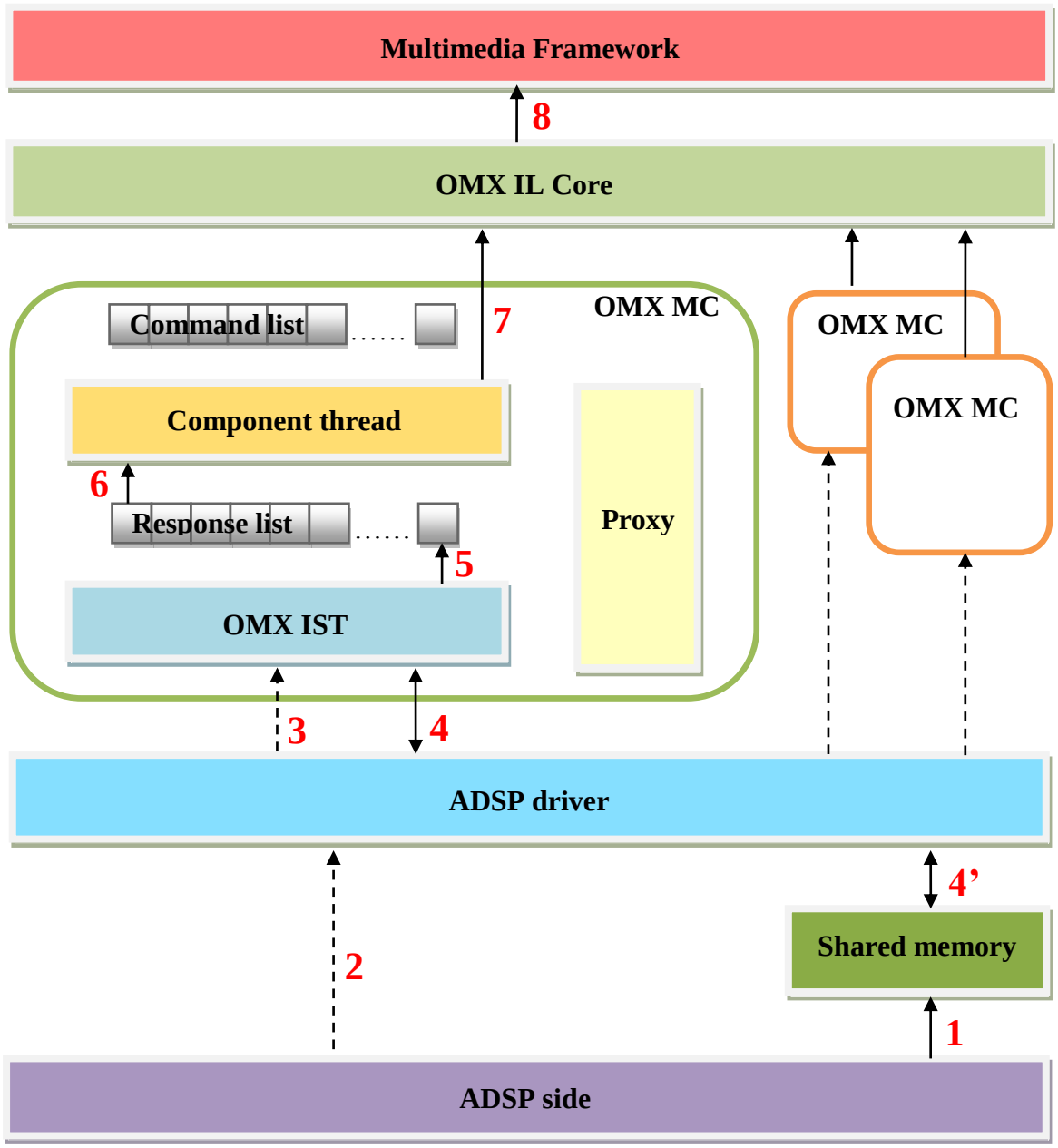


*Figure 3.4 Receive data from ADSP side*

When ADSP side finishes encoding or decoding a buffer, it writes message to shared memory (1) and enables interrupt to CPU side (2). When ADSP driver receives the interrupt from ADSP side, it wakes up all OMX IST of OMX MCs (3) by an event (RECEIVE_EVENT).

When all OMX MCs are waked up, they find the messages appreciated to themselves from shared memory (4, 4') (Means that OMX MC of AAC codec only processes messages related to AAC codec). If they find comfortable messages, these messages are processed and added the tail of their response list (5). All OMX MCs read messages until

the response list is empty, then one of OMX MCs clear RECEIVE_EVENT event and status interrupt sent from ADSP side.

Besides, the Component thread polls to read the command and response list continuously. If the response list is not empty, the Component thread reads the first element of response list (6) and sends it to multimedia framework (7, 8).

### 3.3 Communication inside ADSP

ADSP side is organized as a set of tasks; each task is executed in its own thread. The ISR of shared memory is in charge of receiving interrupts from CPU. For each incoming command message, it signals the *ISR task* to dispatch the message to appropriate recipient task. After the command message is processed and the response message is ready, ISR sends an interrupt to inform CPU.

There are three kinds of task: ISR task, factory task and codec task. These tasks communicate with CPU by interrupt, command message queue and response message queue of shared memory. Besides, they use an internal message queue to contain the response messages before pushing these messages to shared memory.

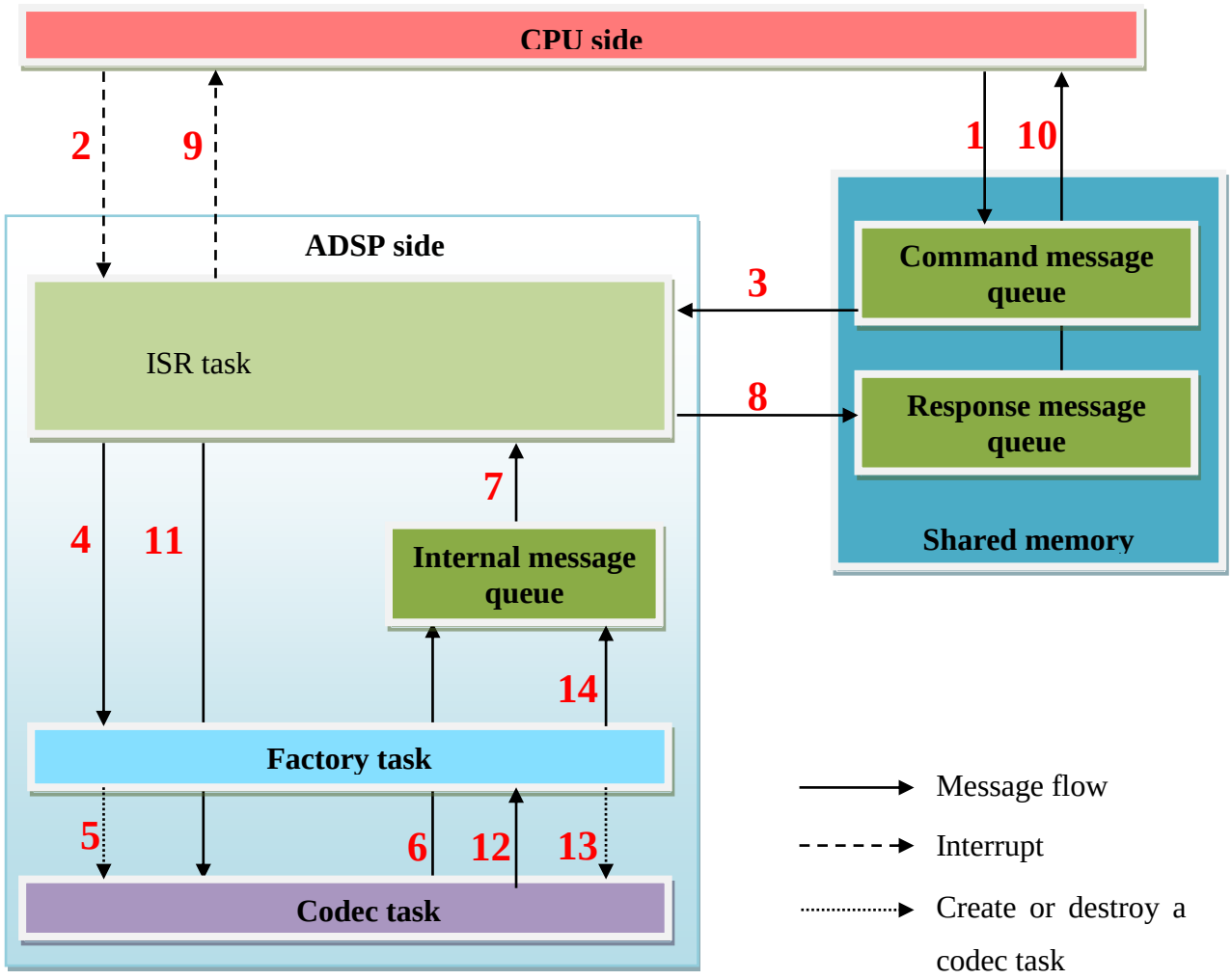


*Figure 3.5 Communication between tasks in ADSP side*

### 3.3.1 Initialize codec task

When CPU wants to initialize a codec, it pushes a register message to the command message queue (1) and raises an interrupt (2). ISR task receives the interrupt and gets the message from command message queue (3), then sends it to factory task (4). Factory task will create codec task (5). After being created, codec task will push a response message to the internal message queue (6). Then ISR task gets the message (7), copies it to the response message queue (8) and raises an interrupt to CPU (9). CPU receives the interrupt, gets the message (10) and knows that the codec is created successfully.

### 3.3.2 Process the commands from CPU

When the encoding / decoding process begins and CPU wants to send a command to ADSP, three first steps (1), (2), (3) are the same with section 3.3.1. Then ISR task will send the message directly to the corresponding codec task based on the codec ID in the

message (11). The codec task processes the message and the last steps (6), (7), (8), (9), (10) are also the same with section 3.3.1.

### 3.3.3 Destroy codec task

When the encoding / decoding process ends and CPU wants to stop a codec task, CPU pushes an unregister message to the command message queue (1) and the steps (2), (3), (11) are the same with section 0. The codec task will sends the message to factory task (12), then factory task will destroy the codec task (13) and push a response message to the internal message queue. The last steps (7), (8), (9), (10) are the same with section 3.3.1 and 3.3.2 and CPU will know that the codec has stopped successfully.

# 4 Appendix

## 4.1 List of used registers of ADSP module

*Table 4.1 List of used registers of ADSP module*

| Register | Description |
|---|---|
| Input Interrupt Set Register (IRQIN_SET16) | Used to raise an interrupt from CPU to ADSP. |
| Input Interrupt Select Register 16 | Used to select the bit of Input Interrupt Set Register that will be used to raise interrupt. |
| Output Interrupt Status Register | Used to check the status (ON/OFF) of Output Interrupt Set Register . |
| Output Interrupt Set Register | Used to raise an interrupt from ADSP to CPU. |
| Output Interrupt Clear Register | Used to clear the interrupt from ADSP to CPU. |
| Communication Register 00 | Used to store the addresses which are shared between OMX MCs on CPU side: command mutex, response mutex, command queue and response queue of shared memory |
| Communication Register 01 | |
| Communication Register 02 | |
| Communication Register 03 | |
| Communication Register 04 | Used to transfer the address of shared memory from CPU to ADSP |
| Communication Register 05 | Used to transfer the address of trace buffer (contain debug log) from CPU to ADSP |

*** Note: Do not use these registers for other modules.**

## 4.2 Memory allocation

The detailed memory allocation is described in the below figure:

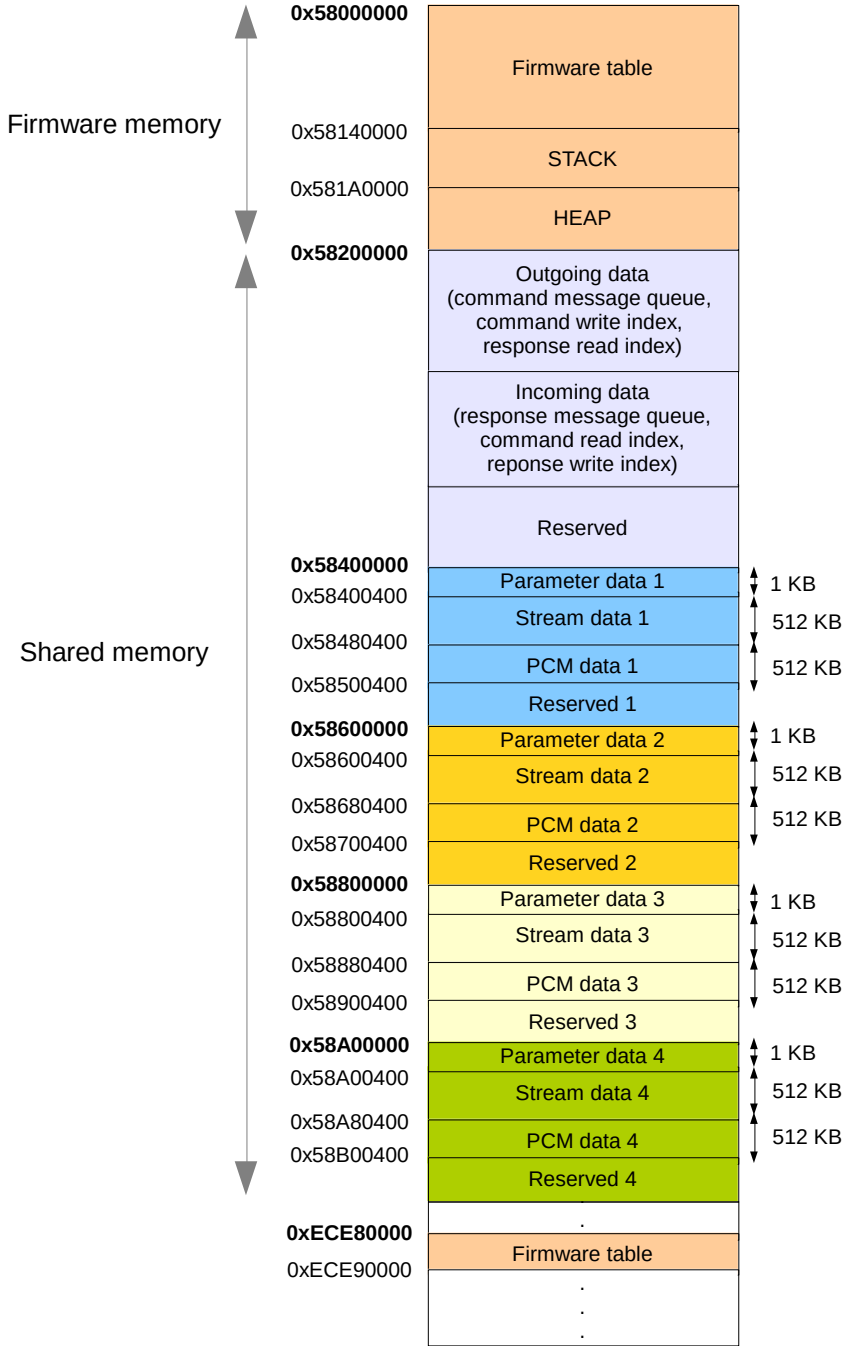


*Figure 4.1 Memory map*

There are 2 main allocated areas:

- Firmware memory:
  - Firmware tables: the executables of ADSP side
  - STACK, HEAP: used in the working process of ADSP side
- Shared memory:

- Outgoing data and incoming data are used for communication between CPU and ADSP
- 4memory areas for parameter data, input data and output data of 4 codecs

* **Note: Do not use these memory areas for other modules.**

| REVISION HISTORY | Audio Framework Specification |
|---|---|

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Jun 05, 2013 | 17 | First Edition issued |

Audio Framework Specification

Publication Date: Rev.1.00 Jun 05, 2013
Published by: Renesas Electronics Corporation

**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.**
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

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

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

**Renesas Electronics Europe GmbH**
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

**Renesas Electronics (China) Co., Ltd.**
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

**Renesas Electronics (Shanghai) Co., Ltd.**
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

**Renesas Electronics Hong Kong Limited**
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

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

**Renesas Electronics Singapore Pte. Ltd.**
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

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

**Renesas Electronics Korea Co., Ltd.**
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141





Colophon 1.1

# Audio Framework Specification

Renesas Electronics Corporation