**Document Type: Detail Design**

**Document Name:**

**ADSP FRAMEWORK: SSI DRIVER**

Renesas Electronics Corporation

Renesas Design Vietnam Co. Ltd

R-Car Software Solution 2 Group

Middleware 1 Team

Table of Contents

[1. Overview 3](#_Toc529886401)

[2. Function list 4](#_Toc529886402)

[3. Detail information 5](#_Toc529886403)

[3.1 Data type definition 5](#_Toc529886404)

[3.1.1 SSI\_MODULES 5](#_Toc529886405)

[3.1.2 ADG\_CLK\_TYPE 5](#_Toc529886406)

[3.1.3 SSI\_DATA\_MODE 6](#_Toc529886407)

[3.1.4 SSI\_CLK\_MODE 6](#_Toc529886408)

[3.1.5 SSI\_ERR\_CODE 6](#_Toc529886409)

[3.1.6 SSI\_PARAMS 6](#_Toc529886410)

[3.2 Function definition 8](#_Toc529886411)

[3.2.1 ssi\_pcm\_width\_get 8](#_Toc529886412)

[3.2.2 ssi\_setup 9](#_Toc529886413)

[3.2.3 ssi\_start 12](#_Toc529886414)

[3.2.4 ssi\_stop 13](#_Toc529886415)

[3.2.5 ssi\_check\_available\_module 14](#_Toc529886416)

[3.2.6 ssi\_get\_register\_base 15](#_Toc529886417)

[3.2.7 ssi\_master\_get 17](#_Toc529886418)

[4. Revision history 18](#_Toc529886419)

List of Figures

[Figure 1‑1 The software architecture 3](#_Toc529886378)

[Figure 3‑1 ssi\_pcm\_width\_get flowchart 8](#_Toc529886379)

[Figure 3‑2 ssi\_setup flowchart 11](#_Toc529886380)

[Figure 3‑3 ssi\_start flowchart 12](#_Toc529886381)

[Figure 3‑4 ssi\_stop flowchart 13](#_Toc529886382)

[Figure 3‑5 ssi\_check\_available\_module flowchart 14](#_Toc529886383)

[Figure 3‑6 ssi\_get\_register\_base flowchart 16](#_Toc529886384)

[Figure 3‑7 ssi\_master\_get flowchart 17](#_Toc529886385)

List of Table

[Table 2‑1 Function list 4](#_Toc529886391)

[Table 3‑1 SSI\_MODULES type information 5](#_Toc529886392)

[Table 3‑2 ADG\_CLK\_TYPE type information 5](#_Toc529886393)

[Table 3‑3 SSI\_DATA\_MODE type information 6](#_Toc529886394)

[Table 3‑4 SSI\_CLK\_MODE type information 6](#_Toc529886395)

[Table 3‑5 SSI\_ERR\_CODE type information 6](#_Toc529886396)

[Table 3‑6 SSI\_PARAMS type information 6](#_Toc529886397)

[Table 3‑7 Setting of clock supplier for SSI 9](#_Toc529886398)

[Table 3‑8 Setting SSICR based on PCM width 10](#_Toc529886399)

[Table 3‑9 Array ssi\_register\_base[SSIMAX] 15](#_Toc529886400)

[Table 2‑1 Function list 4](#_Toc534277152)

[Table 3‑1 SSI\_MODULES type information 5](#_Toc534277153)

[Table 3‑2 ADG\_CLK\_TYPE type information 5](#_Toc534277154)

[Table 3‑3 SSI\_DATA\_MODE type information 6](#_Toc534277155)

[Table 3‑4 SSI\_CLK\_MODE type information 6](#_Toc534277156)

[Table 3‑5 SSI\_ERR\_CODE type information 6](#_Toc534277157)

[Table 3‑6 SSI\_PARAMS type information 6](#_Toc534277158)

[Table 3‑7 SSI\_CONTROL type information 7](#_Toc534277159)

[Table 3‑8 Setting of clock supplier for SSI 9](#_Toc534277160)

[Table 3‑9 Setting SSICR based on PCM width 10](#_Toc534277161)

[Table 3‑9 Array ssi\_register\_base[SSIMAX] 15](#_Toc534277162)

# Overview

This section provides an overview of software architecture.

User Application

ADSP Interface

Kernel Space

User Space

ADSP Driver

ARM

Audio HW

**ADSP Framework**

TDM class

Renderer class

Equalizer class

Capture class

TDM Plugin

ADSP

Plugin

Equalizer Plugin\*

Capture Plugin

Renderer Plugin

This document’s target is in side of red square

\* not connect to hardware modules

DAC/

ADC

SCU

PDMA

FIFO

SSI

SSIU

ADMA

Figure 1‑1 ****The software architecture****

# Function list

The following is list of functions:

Table 2‑1 Function list

|  |  |  |
| --- | --- | --- |
| **Type** | **Function Name** | **Outline** |
| API | ssi\_pcm\_width\_get | This API is to get the PCM width based on values stored in bits 21 to 19 of Control Register SSICR |
| ssi\_setup | This API is to set up registers necessary for SSI module execution |
| ssi\_start | This API is to start SSI module |
| ssi\_stop | This API is to stop SSI module |
| ssi\_check\_available\_module | This API is to check availability of SSI module |
| ssi\_get\_register\_base | This API is to get base address of registers of SSI module |
| ssi\_master\_get | This API is to get master SSI module of the current one |

# Detail information

This section describes detail information of data types, macro definitions, implemented APIs and internal function units, global variable.

## Data type definition

### SSI\_MODULES

The data type SSI\_MODULES is a type-defined enumeration that lists all supported SSI modules. It starts with SSI0 to SSIMAX.

Table 3‑1 SSI\_MODULES type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SSI0 | 0 | SSI0 |
| SSI1 | 1 | SSI1 |
| SSI2 | 2 | SSI2 |
| SSI3 | 3 | SSI3 |
| SSI4 | 4 | SSI4 |
| SSI5 | 5 | SSI5 |
| SSI6 | 6 | SSI6 |
| SSI7 | 7 | SSI7 |
| SSI8 | 8 | SSI8 |
| SSI9 | 9 | SSI9 |
| SSIMAX | 10 | Number of SSI modules |

### ADG\_CLK\_TYPE

The data type ADG\_CLK\_TYPE is a type-defined enumeration that lists all supported ADG Clocks.

Table 3‑2 ADG\_CLK\_TYPE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| ADG\_CLKA | 0 | ADG Clock A, frequency is 22.5792MHz |
| ADG\_CLKB | 1 | ADG Clock B, frequency is 24.576MHz |
| ADG\_CLKC | 2 | ADG Clock C, frequency is 14.7456MHz |

### SSI\_DATA\_MODE

The data type SSI\_DATA\_MODE is a type-defined enumeration that lists all supported SSI modes: Transfer mode/Receive mode.

Table 3‑3 SSI\_DATA\_MODE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SSI\_RCV\_MODE | 0 | SSI’s Receive mode |
| SSI\_TR\_MODE | 1 | SSI’s Transfer mode |

### SSI\_CLK\_MODE

The data type SSI\_CLK\_MODE is a type-defined enumeration that lists all supported SSI clock modes: Master/Slave.

Table 3‑4 SSI\_CLK\_MODE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SSI\_MASTER\_MODE | 0 | SSI clock mode Master |
| SSI\_SLAVE\_MODE | 1 | SSI clock mode Slave |

### SSI\_ERR\_CODE

The data type SSI\_ERR\_CODE is a type-defined enumeration that lists all error-codes used for SSI.

Table 3‑5 SSI\_ERR\_CODE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SSI\_ERR\_NONE | 0 | No errors |
| SSI\_ERR\_OUT\_RANGE | -1 | SSI module is out of range |
| SSI\_ERR\_BUSY | -2 | SSI module is busy |

### SSI\_PARAMS

The data type SSI\_PARAMS is a type-defined structure that possesses necessary parameters for SSI module.

Table 3‑6 SSI\_PARAMS type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Data type** | **Outline** |
| channel | UWORD32 | channel of data used for SSI module |
| pcm\_width | UWORD32 | PCM width of data used for SSI module |
| fs | UWORD32 | Sampling rate of data used for SSI module |
| clock\_type | ADG\_CLK\_MODE | Clock type (Clock A, B, C) |
| data\_mode | SSI\_DATA\_MODE | SSI data mode |
| clock\_mode | SSI\_CLK\_MODE | SSI clock mode |

### SSI\_CONTROL

The data type SSI\_CONTROL is a type-defined structure that lists registers for SSI modules.

Table 3‑7 SSI\_CONTROL type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Data type** | **Registers** |
| SSICR | volatile UWORD32 | SSICR |
| SSISR | volatile UWORD32 | SSISR |
| SSITDR | volatile UWORD32 | SSITDR |
| SSIRDR | volatile UWORD32 | SSIRDR |
| dummy[4] | volatile UWORD32 | No registers |
| SSIWSR | volatile UWORD32 | SSIWSR |
| SSIFMR | volatile UWORD32 | SSIFMR |
| SSIFSR | volatile UWORD32 | SSIFSR |
| SSICRE | volatile UWORD32 | SSICRE |

## Function definition

### ssi\_pcm\_width\_get

DD\_PLG\_RDR\_05\_001

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | inline UWORD32 ssi\_pcm\_width\_get(const SSI\_MODULES module) | | | |
| **Function** | This function is to get the PCM width based on values stored in bits 21 to 19 of Control Register SSICR. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SSI\_MODULES | module | I | SSI module  Valid range: [0: 9] |
| **Return value** | pcm\_width | | PCM width’s value | |
| **Description** | * ssi\_pcm\_width\_get command processing:   - SSICR’s bits [21:19] is 1 and 5, then PCM width is 16 and 24, respectively. Otherwise, PCM width is 0. | | | |

[Covers: FD\_PLG\_RDR\_005]

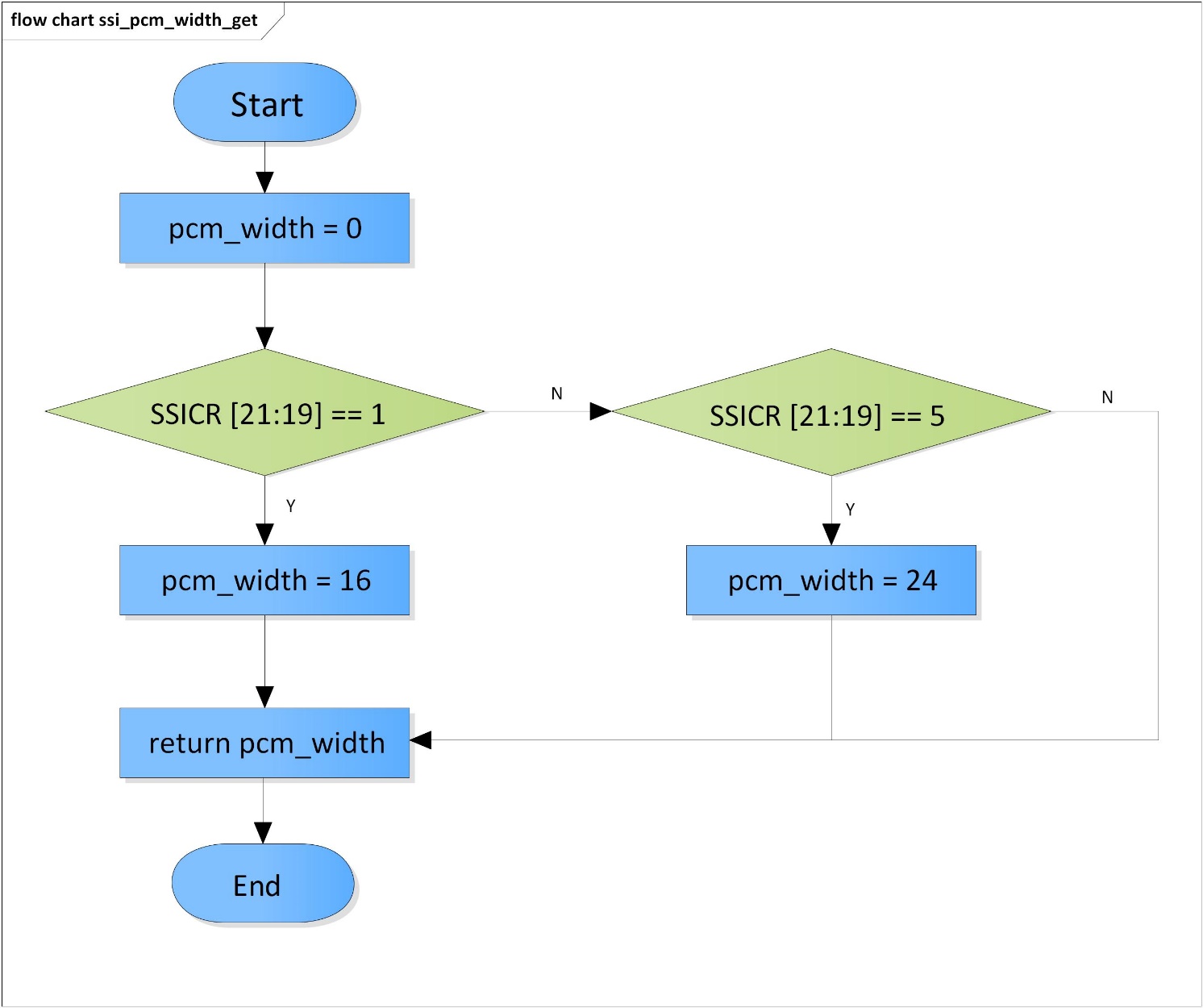


Figure 3‑1 ssi\_pcm\_width\_get flowchart

### ssi\_setup

DD\_PLG\_RDR\_05\_002

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SSI\_ERR\_CODE ssi\_setup(SSI\_MODULES module, SSI\_PARAMS params) | | | |
| **Function** | This function is to set up registers necessary for SSI module execution. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SSI\_MODULES | module | I | SSI module  Valid range: [0: 9] |
| SSI\_PARAMS | params | I | Struct of parameters to set |
| **Return value** | SSI\_ERR\_OUT\_RANGE | | SSI module is invalid  Clock type is invalid  Sampling rate is invalid  PCM width is invalid  Clock mode is invalid  Data mode is invalid | |
| SSI\_ERR\_NONE | | Normal end | |
| **Description** | * ssi\_setup command processing:   - Set register AUDIO\_CLK\_SEL0/1/2 based on which SSI module and sampling rate  - Set register SSIWSR (in master mode)  - Set register SSICR | | | |

[Covers: FD\_PLG\_RDR\_005]

Table 3‑8 Setting of clock supplier for SSI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SSI module** | **Sampling rate** | **Register** | **Bits** | **Value** |
| SSI0 | 32000 | AUDIO\_CLK\_SEL0 | [7:0] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI1 | 32000 | AUDIO\_CLK\_SEL0 | [15:8] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI2 | 32000 | AUDIO\_CLK\_SEL0 | [23:16] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI3 | 32000 | AUDIO\_CLK\_SEL0 | [31:24] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI4 | 32000 | AUDIO\_CLK\_SEL1 | [7:0] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI5 | 32000 | AUDIO\_CLK\_SEL1 | [15:8] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI6 | 32000 | AUDIO\_CLK\_SEL1 | [23:16] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI7 | 32000 | AUDIO\_CLK\_SEL1 | [31:24] | 0x20 |
| 48000 |
| 44100 | 0x10 |
| SSI8 | No clock supplier | | | |
| SSI9 | 32000 | AUDIO\_CLK\_SEL2 | [15:8] | 0x20 |
| 48000 |
| 44100 | 0x10 |

Table 3‑9 Setting SSICR based on PCM width

|  |  |
| --- | --- |
| **PCM width** | **SSICR[21:19]** |
| 8 | b’000 |
| 16 | b’001 |
| 18 | b’010 |
| 20 | b’011 |
| 22 | b’100 |
| 24 | b’101 |
| 32 | b’110 |

Note: setting SSI\_Control register base on the specific codec

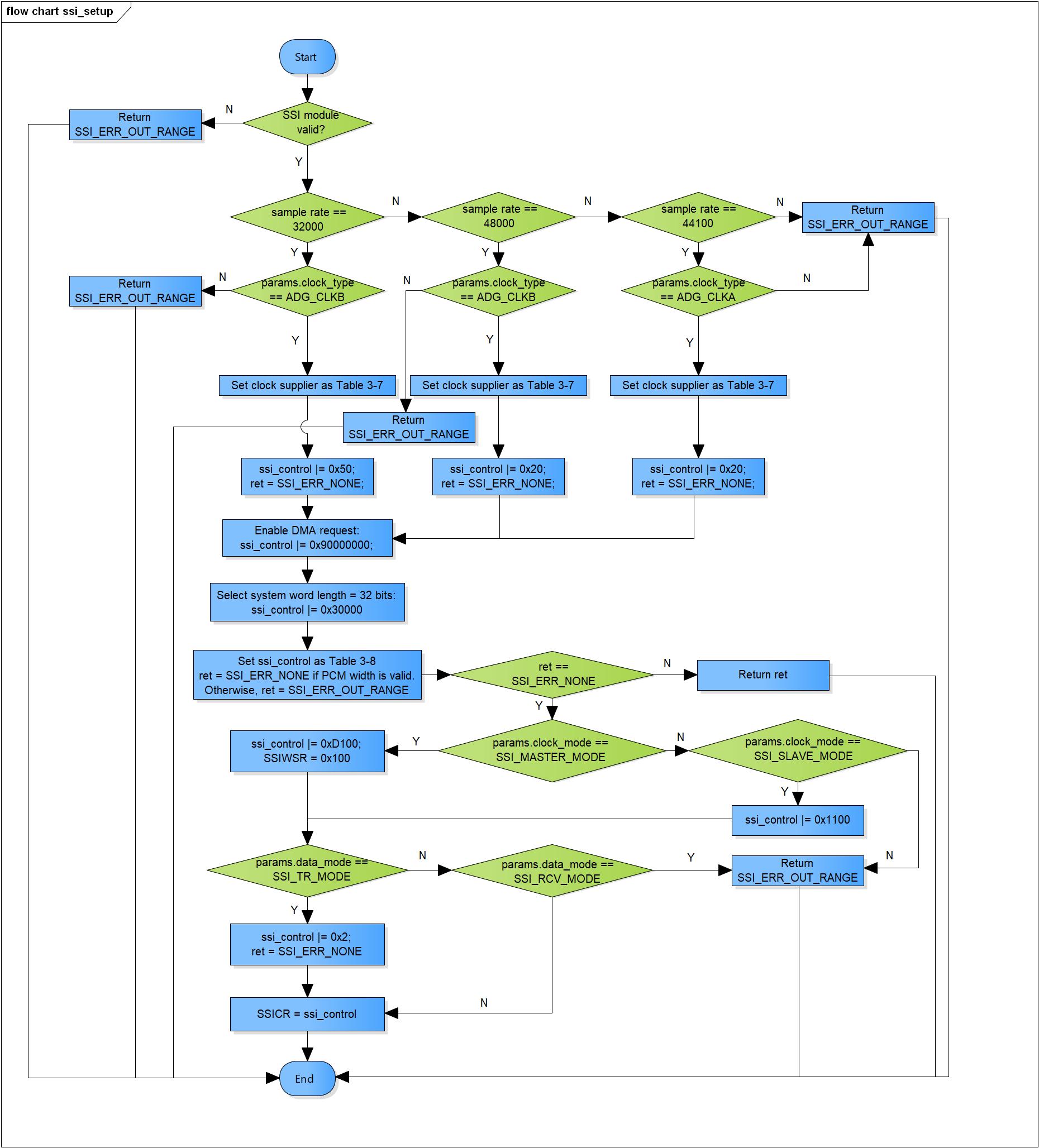


Figure 3‑2 ssi\_setup flowchart

### ssi\_start

DD\_PLG\_RDR\_05\_003

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SSI\_ERR\_CODE ssi\_start(SSI\_MODULES module) | | | |
| **Function** | This function is to start SSI module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SSI\_MODULES | module | I | SSI module  Valid range: [0: 9] |
| **Return value** | SSI\_ERR\_OUT\_RANGE | | SSI module is invalid | |
| SSI\_ERR\_NONE | | Normal end | |
| **Description** | * ssi\_start command processing:   - Set ‘1’ to SSICR’s bit 0 to start SSI | | | |

[Covers: FD\_PLG\_RDR\_005]

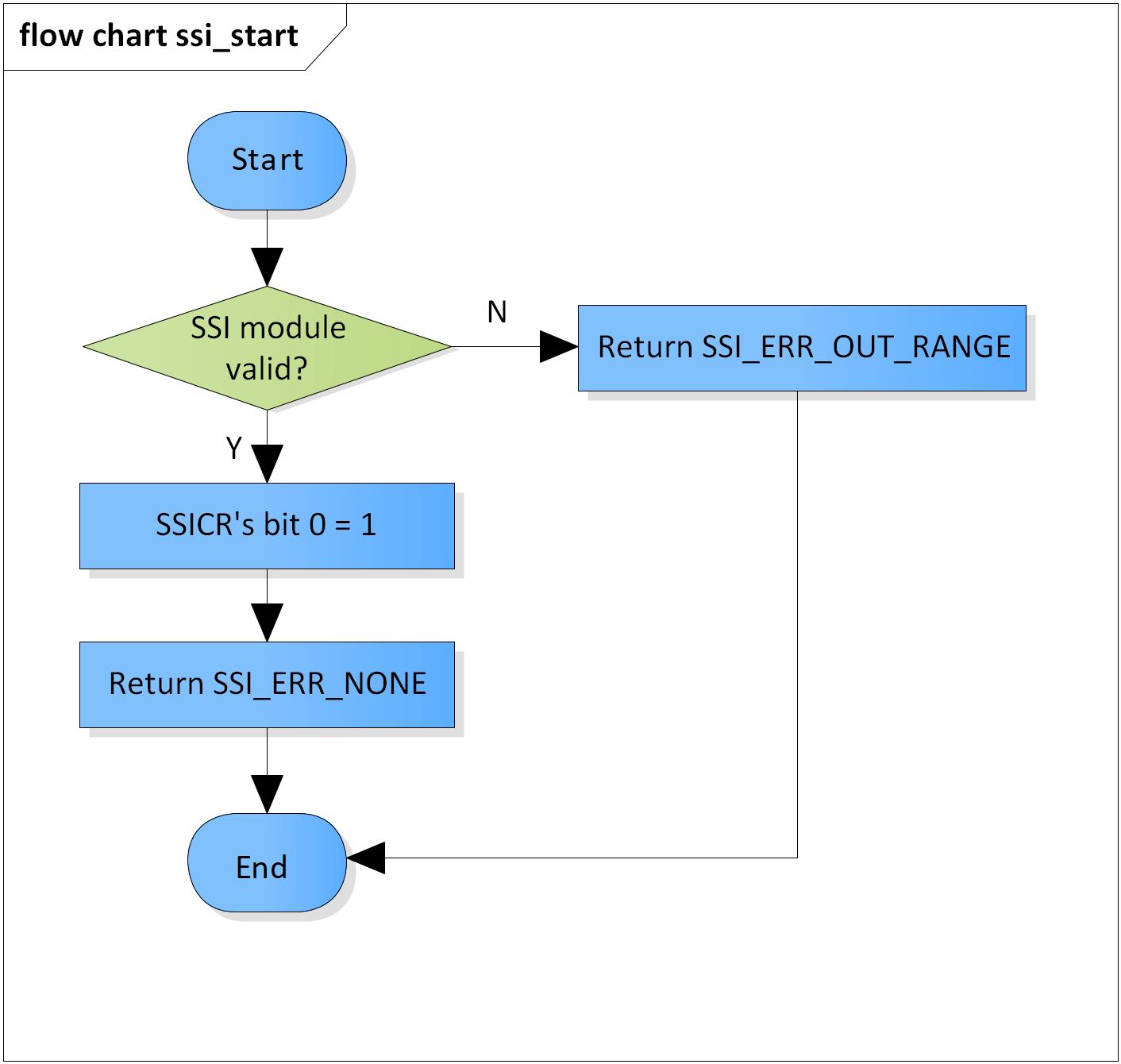


Figure 3‑3 ssi\_start flowchart

### ssi\_stop

DD\_PLG\_RDR\_05\_004

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SSI\_ERR\_CODE ssi\_stop(SSI\_MODULES module) | | | |
| **Function** | This function is to stop SSI module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SSI\_MODULES | module | I | SSI module  Valid range: [0: 9] |
| **Return value** | SSI\_ERR\_OUT\_RANGE | | SSI module is invalid | |
| SSI\_ERR\_NONE | | Normal end | |
| **Description** | * ssi\_start command processing:   - Set ‘0’ to SSICR’s bit 0 to stop SSI | | | |

[Covers: FD\_PLG\_RDR\_005, FD\_PLG\_RDR\_014, FD\_PLG\_RDR\_032, FD\_PLG\_RDR\_043]

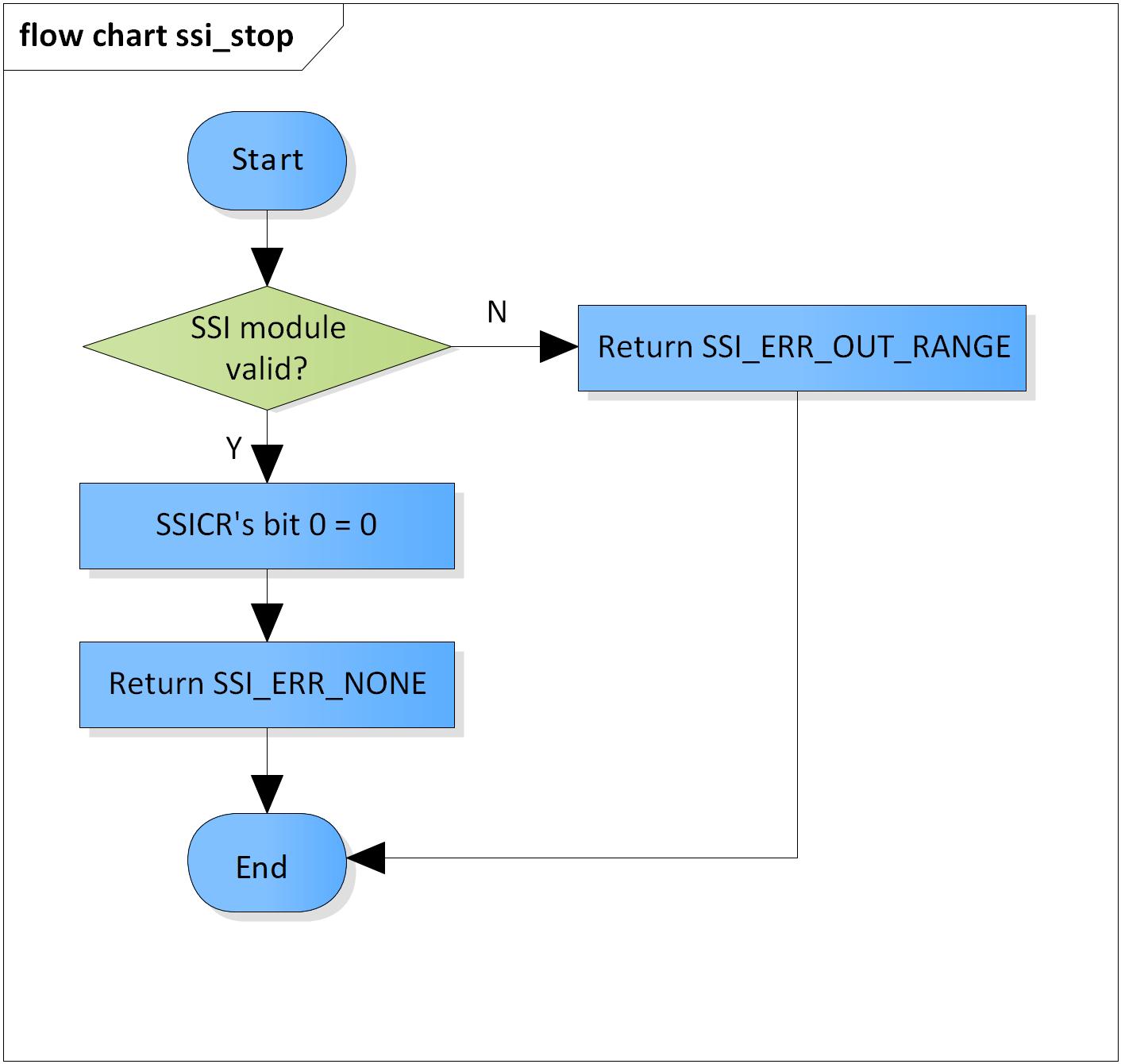


Figure 3‑4 ssi\_stop flowchart

### ssi\_check\_available\_module

DD\_PLG\_RDR\_05\_005

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SSI\_ERR\_CODE ssi\_check\_available\_module(UWORD32 module) | | | |
| **Function** | This function is check availability of SSI module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| UWORD32 | module | I | SSI module  Valid range: [0: 9] |
| **Return value** | SSI\_ERR\_OUT\_RANGE | | SSI module is invalid | |
| SSI\_ERR\_BUSY | | SSI module is busy | |
| SSI\_ERR\_NONE | | Normal end | |
| **Description** | * ssi\_check\_available\_module command processing:   - Check if the current SSI module is running or not | | | |

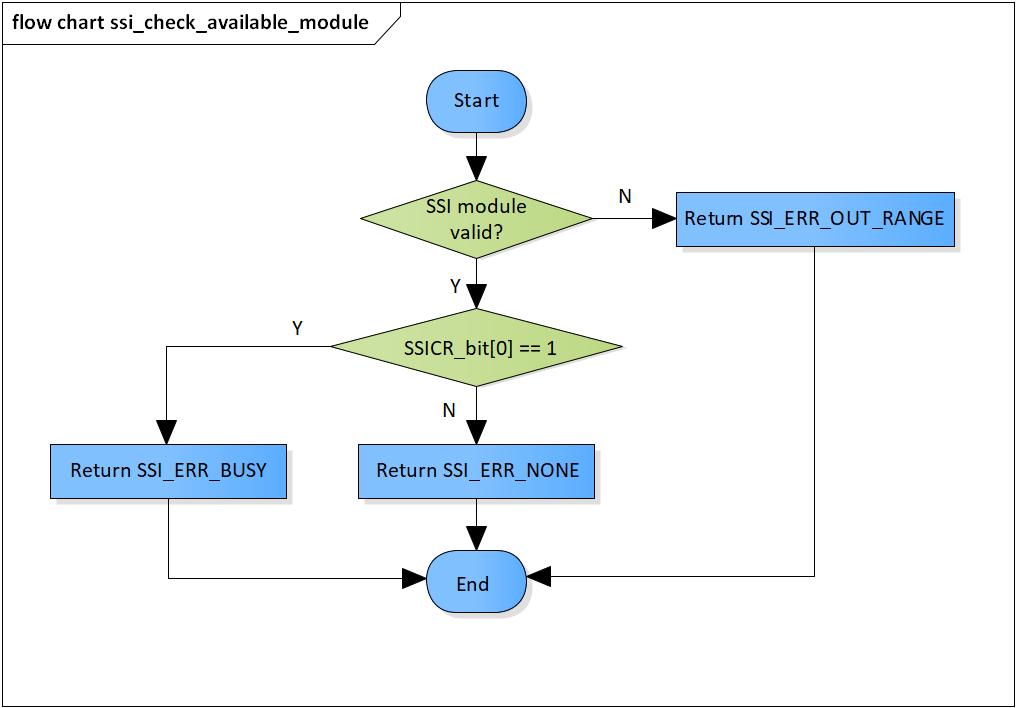
[Covers: FD\_PLG\_RDR\_005]

Figure 3‑5 ssi\_check\_available\_module flowchart

### ssi\_get\_register\_base

DD\_PLG\_RDR\_05\_006

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | static volatile inline SSI\_CONTROL \*ssi\_get\_register\_base(SSI\_MODULES module) | | | |
| **Function** | This function is to get base address of registers of SSI module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SSI\_MODULES | module | I | SSI module  Valid range: [0: 9] |
| **Return value** | 0 | | SSI module is invalid | |
| ssi\_register\_base[module] | | Base address of register | |
| **Description** | * ssi\_get\_register\_base command processing:   - Get the base address of registers of the current SSI module | | | |

[Covers: FD\_PLG\_RDR\_005, FD\_PLG\_RDR\_014, FD\_PLG\_RDR\_032, FD\_PLG\_RDR\_043]

Table 3‑10 Array ssi\_register\_base[SSIMAX]

|  |  |
| --- | --- |
| **Index** | **Values** |
| SSI0 | XF\_RCAR\_REG\_SSICR0 |
| SSI1 | XF\_RCAR\_REG\_SSICR1 |
| SSI2 | XF\_RCAR\_REG\_SSICR2 |
| SSI3 | XF\_RCAR\_REG\_SSICR3 |
| SSI4 | XF\_RCAR\_REG\_SSICR4 |
| SSI5 | XF\_RCAR\_REG\_SSICR5 |
| SSI6 | XF\_RCAR\_REG\_SSICR6 |
| SSI7 | XF\_RCAR\_REG\_SSICR7 |
| SSI8 | XF\_RCAR\_REG\_SSICR8 |
| SSI9 | XF\_RCAR\_REG\_SSICR9 |

[Note]: The Values column of Table 3-9 are macros for registers defined in xf-register.h

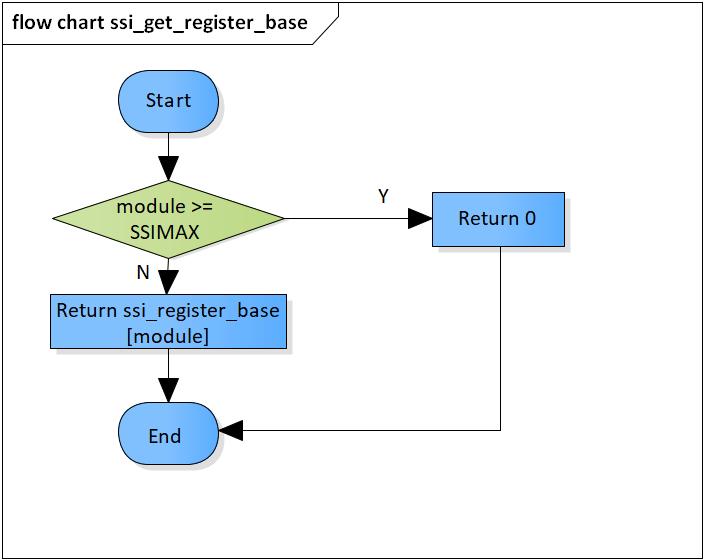


Figure 3‑6 ssi\_get\_register\_base flowchart

### ssi\_master\_get

DD\_PLG\_RDR\_05\_007

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SSI\_MODULES ssi\_master\_get(const SSI\_MODULES module) | | | |
| **Function** | This function is to get the PCM width based on values stored in bits 21 to 19 of Control Register SSICR. | | | |
| **Arguments** | Type | Name | I/O | Description |
| const SSI\_MODULE | module | I | SSI module  Valid range: [0: 9] |
| **Return value** | ssi\_master | | Master SSI module | |
| **Description** | * ssi\_master\_get command processing:   - Get master module based on current SSI module | | | |

[Covers: FD\_PLG\_RDR\_005, FD\_PLG\_RDR\_014, FD\_PLG\_RDR\_032, FD\_PLG\_RDR\_043]

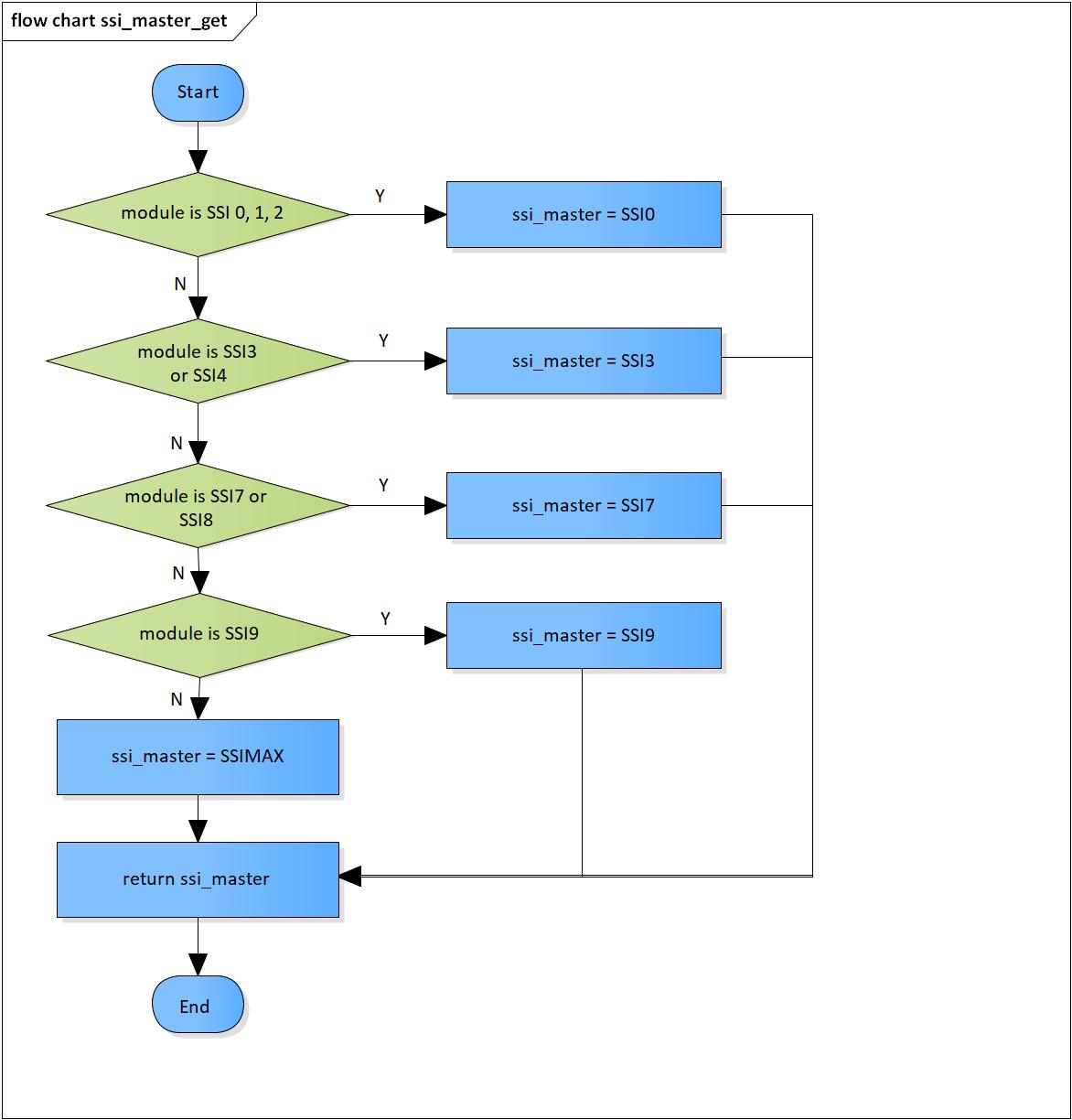
****

Figure 3‑7 ssi\_master\_get flowchart

# Revision history

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Date** | **Page** | **Content** | **Approved** | **Changed** |
| 1.0.0 | Nov 14 2018 | - | First Edition issued | Vu Phan | Vu Phan |
| 1.1.0 | Dec 10 2018 | - | Add traceability ID | Vu Phan | Ngu Pham |
| 1.2.0 | Jan 03, 2019 | - | Add range for input parameters | Vu Phan | Tien Tran |
| 1.3.0 | Mar 20, 2019 | 11 | Modify SSI control register correct to swap channel issue |  | Ngu Pham |