**Document Type: Detail Design**

**Document Name:**

**ADSP FRAMEWORK: SCU DRIVER**

Renesas Electronics Corporation

Renesas Design Vietnam Co. Ltd

R-Car Software Solution 2 Group

Middleware 1 Team

Table of Cotents

[1. Overview 3](#_Toc530075084)

[2. Function list 4](#_Toc530075085)

[3. Detail information 5](#_Toc530075086)

[3.1 Data type definition 5](#_Toc530075087)

[3.1.1 SRC\_MODULE 5](#_Toc530075088)

[3.1.2 SRC\_DATA\_LENGTH 5](#_Toc530075089)

[3.1.3 SRC\_CH\_NUM 6](#_Toc530075090)

[3.1.4 SRC\_SYNC\_MODE 6](#_Toc530075091)

[3.1.5 SRC\_UF\_MODE 6](#_Toc530075092)

[3.1.6 SRC\_START\_MODE 7](#_Toc530075093)

[3.1.7 SRC\_ERROR\_CODE 7](#_Toc530075094)

[3.1.8 SRC\_FLAG 7](#_Toc530075095)

[3.1.9 DVC\_CH\_NUM 8](#_Toc530075096)

[3.1.10 DVC\_MODES 8](#_Toc530075097)

[3.1.11 DVC\_ERROR\_CODE 8](#_Toc530075098)

[3.1.12 DVC\_FLAG 9](#_Toc530075099)

[3.1.13 CMD\_MODULES 9](#_Toc530075100)

[3.1.14 SRC\_CONFIG 9](#_Toc530075101)

[3.1.15 SRC\_CTRL 10](#_Toc530075102)

[3.1.16 SRC\_REG 10](#_Toc530075103)

[3.2 Macro definition 11](#_Toc530075104)

[3.3 Global variable definition 12](#_Toc530075105)

[3.4 Function definition 13](#_Toc530075106)

[3.4.1 xa\_src\_check\_module\_index 13](#_Toc530075107)

[3.4.2 xa\_src\_config 14](#_Toc530075108)

[3.4.3 xa\_src\_start 18](#_Toc530075109)

[3.4.4 xa\_src\_stop 19](#_Toc530075110)

[3.4.5 xa\_src\_check\_available\_module 20](#_Toc530075111)

[3.4.6 xa\_dvc\_dig\_setting 21](#_Toc530075112)

[3.4.7 xa\_dvc\_config 22](#_Toc530075113)

[3.4.8 xa\_dvc\_start 24](#_Toc530075114)

[3.4.9 xa\_dvc\_stop 25](#_Toc530075115)

[3.4.10 xa\_dvc\_reset 26](#_Toc530075116)

[3.4.11 xa\_dvc\_check\_available 27](#_Toc530075117)

[3.4.12 xa\_src\_get\_config 28](#_Toc530075118)

[3.4.13 xa\_src\_get\_control 29](#_Toc530075119)

[4. Revision history 30](#_Toc530075120)

List of Figures

[Figure 1‑1 The software architecture 3](#_Toc530075121)

[Figure 3‑1 xa\_src\_check\_module\_index flowchart 13](#_Toc530075122)

[Figure 3‑2 xa\_src\_config flowchart 17](#_Toc530075123)

[Figure 3‑3 xa\_src\_start flowchart 18](#_Toc530075124)

[Figure 3‑4 start\_stop flowchart 19](#_Toc530075125)

[Figure 3‑5 xa\_src\_check\_available\_module flowchart 20](#_Toc530075126)

[Figure 3‑6 xa\_dvc\_dig\_setting flowchart 21](#_Toc530075127)

[Figure 3‑7 xa\_dvc\_config flowchart 23](#_Toc530075128)

[Figure 3‑8 xa\_dvc\_start flowchart 24](#_Toc530075129)

[Figure 3‑9 xa\_dvc\_stop flowchart 25](#_Toc530075130)

[Figure 3‑10 xa\_dvc\_reset flowchart 26](#_Toc530075131)

[Figure 3‑11 xa\_dvc\_check\_available flowchart 27](#_Toc530075132)

[Figure 3‑12 xa\_src\_get\_config flowchart 28](#_Toc530075133)

[Figure 3‑13 xa\_src\_get\_control flowchart 29](#_Toc530075134)

List of Table

[Table 2‑1 Function list 4](#_Toc530075135)

[Table 3‑1 SRC\_MODULE type information 5](#_Toc530075136)

[Table 3‑2 SRC\_DATA\_LENGTH type information 5](#_Toc530075137)

[Table 3‑3 SRC\_DATA\_MODE type information 6](#_Toc530075138)

[Table 3‑4 SRC\_SYNC\_MODE type information 6](#_Toc530075139)

[Table 3‑5 SRC\_UF\_MODE type information 6](#_Toc530075140)

[Table 3‑6 SRC\_START\_MODE type information 7](#_Toc530075141)

[Table 3‑7 SRC\_ERROR\_CODE type information 7](#_Toc530075142)

[Table 3‑8 SRC\_FLAG type information 7](#_Toc530075143)

[Table 3‑9 DVC\_CH\_NUM type information 8](#_Toc530075144)

[Table 3‑10 DVC\_MODES type information 8](#_Toc530075145)

[Table 3‑11 DVC\_ERROR\_CODE type information 8](#_Toc530075146)

[Table 3‑12 DVC\_FLAG type information 9](#_Toc530075147)

[Table 3‑13 CMD\_MODULES type information 9](#_Toc530075148)

[Table 3‑14 SRC\_CONFIG type information 9](#_Toc530075149)

[Table 3‑15 SRC\_CTRL type information 10](#_Toc530075150)

[Table 3‑16 SRC\_REG type information 10](#_Toc530075151)

[Table 3‑17 Setting SRC Input Timing Select registers based on input sampling rate 14](#_Toc530075152)

[Table 3‑18 Setting SRC Output Timing Select registers based on output sampling rate 14](#_Toc530075153)

[Table 3‑19 Setting XF\_RCAR\_REG\_CMDOUT\_TIMSEL for CMD execution 22](#_Toc530075154)

[Table 3‑20 Setting for CMD\_ROUTE\_SELECT register 22](#_Toc530075155)

[Table 2‑1 Function list 5](#_Toc534285774)

[Table 3‑1 SRC\_MODULE type information 6](#_Toc534285775)

[Table 3‑2 SRC\_DATA\_LENGTH type information 6](#_Toc534285776)

[Table 3‑3 SRC\_DATA\_MODE type information 7](#_Toc534285777)

[Table 3‑4 SRC\_SYNC\_MODE type information 7](#_Toc534285778)

[Table 3‑5 SRC\_UF\_MODE type information 7](#_Toc534285779)

[Table 3‑6 SRC\_START\_MODE type information 8](#_Toc534285780)

[Table 3‑7 SRC\_ERROR\_CODE type information 8](#_Toc534285781)

[Table 3‑8 SRC\_FLAG type information 8](#_Toc534285782)

[Table 3‑9 DVC\_CH\_NUM type information 9](#_Toc534285783)

[Table 3‑10 DVC\_MODES type information 9](#_Toc534285784)

[Table 3‑11 DVC\_ERROR\_CODE type information 9](#_Toc534285785)

[Table 3‑12 DVC\_FLAG type information 10](#_Toc534285786)

[Table 3‑13 CMD\_MODULES type information 10](#_Toc534285787)

[Table 3‑14 SRC\_CONFIG type information 10](#_Toc534285788)

[Table 3‑15 SRC\_CTRL type information 11](#_Toc534285789)

[Table 3‑16 SRC\_REG type information 11](#_Toc534285790)

[Table 3‑17 Setting SRC Input Timing Select registers based on input sampling rate 15](#_Toc534285791)

[Table 3‑18 Setting SRC Output Timing Select registers based on output sampling rate 15](#_Toc534285792)

[Table 3‑19 Setting XF\_RCAR\_REG\_CMDOUT\_TIMSEL for CMD execution 23](#_Toc534285793)

[Table 3‑20 Setting for CMD\_ROUTE\_SELECT register 23](#_Toc534285794)

# 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** |
| APIs | xa\_src\_check\_module\_index | This API is to check if current SRC supports multichannel stream |
| xa\_src\_config | This API is to set up SRC configuration |
| xa\_src\_start | This API is to start SRC module |
| xa\_src\_stop | This API is to stop SRC module |
| xa\_src\_check\_available\_module | This API is to check availability of SRC modules and choose an available one |
| xa\_dvc\_dig\_setting | This API is to set up DVC digital mode |
| xa\_dvc\_config | This API is to set up DVC module |
| xa\_dvc\_start | This API is to start DVC module |
| xa\_dvc\_stop | This API is to stop DVC module |
| xa\_dvc\_reset | This API is to reset DVC module |
| xa\_dvc\_check\_available | This API is to check availability and select free DVC module |
| Internal functions | xa\_src\_get\_config | This function is to get SRC Config base register |
| xa\_src\_get\_control | This function is to get SRC Control base register |

# Detail information

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

## Data type definition

### SRC\_MODULE

The data type SRC\_MODULE is a type-defined enumeration that lists all supported SRC modules. It starts with SRC0 to SRCMAX.

Table 3‑1 SRC\_MODULE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SRC0 | 0 | SRC0 |
| SRC1 | 1 | SRC1 |
| SRC2 | 2 | SRC2 |
| SRC3 | 3 | SRC3 |
| SRC4 | 4 | SRC4 |
| SRC5 | 5 | SRC5 |
| SRC6 | 6 | SRC6 |
| SRC7 | 7 | SRC7 |
| SRC8 | 8 | SRC8 |
| SRC9 | 9 | SRC9 |
| SRCMAX | 10 | Number of SRC modules |

### SRC\_DATA\_LENGTH

The data type SRC\_DATA\_LENGTH is a type-defined enumeration that lists all supported SRC module’s data bit length.

Table 3‑2 SRC\_DATA\_LENGTH type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SRC\_24BIT | 0 | Data 24-bit width |
| SRC\_22BIT | 2 | Data 22-bit width |
| SRC\_20BIT | 4 | Data 20-bit width |
| SRC\_18BIT | 6 | Data 18-bit width |
| SRC\_16BIT | 8 | Data 16-bit width |
| SRC\_8BIT | 16 | Data 8-bit width |

### SRC\_CH\_NUM

The data type SRC\_CH\_NUM is a type-defined enumeration that lists all supported number of channels for SRC module.

Table 3‑3 SRC\_DATA\_MODE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SRC\_NONE | 0 | No channels |
| SRC\_1CH | 1 | 1 channel |
| SRC\_2CH | 2 | 2 channels |
| SRC\_4CH | 4 | 4 channels |
| SRC\_6CH | 6 | 6 channels |
| SRC\_8CH | 8 | 8 channels |

### SRC\_SYNC\_MODE

The data type SRC\_SYNC\_MODE is a type-defined enumeration that lists all supported SRC synchronization mode.

Table 3‑4 SRC\_SYNC\_MODE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SRC\_ASYNC | 0 | Asynchronous |
| SRC\_SYNC | 1 | Synchronous |
| SRC\_SYNC\_MAX | 2 | Maximum number of synchronization types |

### SRC\_UF\_MODE

The data type SRC\_UF\_MODE is a type-defined enumeration that lists all supported SRC underflow data treatment modes.

Table 3‑5 SRC\_UF\_MODE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SRC\_REPEAT | 0 | Data before underflow occurs is output |
| SRC\_ZERO | 1 | All 0s are output when underflow occurs |

### SRC\_START\_MODE

The data type SRC\_START\_MODE is a type-defined enumeration that lists all supported SRC start mode.

Table 3‑6 SRC\_START\_MODE type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| SRC\_INPUT | 0 | SRC only receives input data from out of SCU module and output within SCU module |
| SRC\_OUTPUT | 1 | SRC only produces output data to modules of of SCU |
| SRC\_BOTH | 2 | SRC receives input data from modules out of SCU and produces output data out of SCU |

### SRC\_ERROR\_CODE

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

Table 3‑7 SRC\_ERROR\_CODE type information

|  |  |
| --- | --- |
| **Member name** | **Value** |
| SRC\_ERROR\_NONE | 0 |
| SRC\_ERROR\_INVALID | -1 |
| SRC\_ERROR\_INTERNAL | -2 |
| SRC\_ERROR\_BUSY | -3 |
| SRC\_ERROR\_OUT\_RANGE | -4 |

### SRC\_FLAG

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

Table 3‑8 SRC\_FLAG type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Data type** | **Outline** |
| pcm\_wdsz | SRC\_DATA\_LENGTH | PCM width of the data input to and output from SRC |
| ch | SRC\_CH\_NUM | The number of channels input to and output from SRC |
| inp\_sync\_mode | SRC\_SYNC\_MODE | Input synchronization mode of SRC |
| out\_sync\_mode | SRC\_SYNC\_MODE | Output synchronization mode of SRC |
| uf\_mode | SRC\_UF\_MODE | Underflow mode |
| fs\_in | WORD32 | Input frequency sampling |
| fs\_out | WORD32 | Output frequency sampling |

### DVC\_CH\_NUM

The data type DVC\_CH\_NUM is a type-defined enumeration that lists all supported number of channels for DVC module.

Table 3‑9 DVC\_CH\_NUM type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| DVC\_NONE | 0 | No channels |
| DVC\_1CH | 1 | 1 channel |
| DVC\_2CH | 2 | 2 channels |
| DVC\_4CH | 4 | 4 channels |
| DVC\_6CH | 6 | 6 channels |
| DVC\_8CH | 8 | 8 channels |
| DVC\_CHMAX | 9 | Maximum number of DVC output channels |

### DVC\_MODES

The data type DVC\_MODES is a type-defined enumeration that lists all supported DVC modes.

Table 3‑10 DVC\_MODES type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| DVC\_DIG | 0 | Digital volume mode |
| DVC\_RAMP | 1 | Ramp volume mode |
| DVC\_MUTE | 2 | Zero cross mute mode |
| DVC\_DIG\_RAM | 3 | Digital and ramp volume mode |

### DVC\_ERROR\_CODE

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

Table 3‑11 DVC\_ERROR\_CODE type information

|  |  |
| --- | --- |
| **Member name** | **Value** |
| DVC\_ERROR\_NONE | 0 |
| DVC\_ERROR\_INVALID | -1 |
| DVC\_ERROR\_INTERNAL | -2 |
| DVC\_ERROR\_BUSY | -3 |
| DVC\_ERROR\_OUT\_RANGE | -4 |

### DVC\_FLAG

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

Table 3‑12 DVC\_FLAG type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Data type** | **Outline** |
| CMD\_module | CMD\_MODULES | CMD module |
| fs | UWORD32 | Sampling rate for DVC |
| sync\_mode | SRC\_SYNC\_MODE | SRC synchronization mode |
| SRC\_module | SRC\_MODULE | SRC module |
| dvc\_mode | DVC\_MODES | DVC operation mode |
| pcm\_width | UWORD32 | PCM width |
| data\_ch | UWORD32 | Channel data |
| volume[DVC\_CHMAX -1] | WORD32 | volume of channels |

### CMD\_MODULES

The data type CMD\_MODULES is a type-defined enumeration that lists all supported CMD modules.

Table 3‑13 CMD\_MODULES type information

|  |  |  |
| --- | --- | --- |
| **Member name** | **Value** | **Outline** |
| CMD0 | 0 | CMD0 module |
| CMD1 | 1 | CMD1 module |
| CMDMAX | 2 | Maximum number of CMD modules |

### SRC\_CONFIG

The data type SRC\_CONFIG is a type-defined structure of config registers of SRC module.

Table 3‑14 SRC\_CONFIG type information

|  |  |
| --- | --- |
| **Member name** | **Data type** |
| SRC\_IN\_BUSIF\_MODE | UWORD32 |
| SRC\_OUT\_BUSIF\_MODE | UWORD32 |
| SRC\_BUSIF\_DALIGN | UWORD32 |
| SRC\_MODE | UWORD32 |
| SRC\_CONTROL | UWORD32 |
| SRC\_STATUS | UWORD32 |
| SRC\_INT\_ENABLE0 | UWORD32 |

### SRC\_CTRL

The data type SRC\_CTRL is a type-defined structure of control registers of SRC module.

Table 3‑15 SRC\_CTRL type information

|  |  |
| --- | --- |
| **Member name** | **Data type** |
| SRC\_SWRSR | UWORD32 |
| SRC\_SRCIR | UWORD32 |
| dummy1[12] | UWORD8 |
| SRC\_ADINR | UWORD32 |
| dummy2[4] | UWORD8 |
| SRC\_IFSCR | UWORD32 |
| SRC\_IFSVR | UWORD32 |
| SRC\_SRCCR | UWORD32 |
| dummy3[4] | UWORD8 |
| SRC\_BSDSR | UWORD32 |
| dummy4[8] | UWORD8 |
| SRC\_BSISR | UWORD32 |

### SRC\_REG

The data type SRC\_REG is a type-defined structure of control registers of SRC module.

Table 3‑16 SRC\_REG type information

|  |  |
| --- | --- |
| **Member name** | **Data type** |
| SRC\_CONFIG\_BASE | UWORD32 |
| SRC\_CONTROL\_BASE | UWORD32 |

## Macro definition

|  |  |  |
| --- | --- | --- |
| **Macros** | **Value** | **Outline** |
| VOLUME\_MAX | 0x7FFFFF | Max volume DVC can support |
| VOLUME\_NONUSE | 0xFFFFFFFF | DVC is not used |
| FALSE | 0 | Checked condition is false |
| TRUE | 1 | Checked condition is true |
| SRC\_SUPPORT\_MULTICHANNEL\_NUM | 4 | Number of SRC modules which support multi-channel |
| SRC\_ONLY\_SUPPORT\_2CHANNEL\_NUM | 6 | Number of SRC modules which only support mono, stereo |
| SRC\_CMD\_SUPPORT\_ONLY\_2CHANNEL\_NUM | 4 | Number of SRC modules which only support mono, stereo and can route CMD modules |
| SRC\_CMD\_NUM | SRC\_SUPPORT\_MULTICHANNEL\_NUM + SRC\_CMD\_SUPPORT\_ONLY\_2CHANNEL\_NUM | Number of SRCs that can route CMD modules |
| SRC\_SUPPORT\_2CHANNEL\_NUM | SRC\_ONLY\_SUPPORT\_2CHANNEL\_NUM + SRC\_CMD\_SUPPORT\_ONLY\_2CHANNEL\_NUM | Number of SRC modules which support 2 channels |
| SRC\_FSO | 1 << 22 | SRC FSO value |
| VOL\_SCALE | 1000 | Volume scale |

## Global variable definition

|  |  |
| --- | --- |
| static const WORD32 xa\_dvc\_register\_base[CMDMAX] | |
| Array’s index | Value |
| CMD0 | XF\_RCAR\_REG\_DVC0\_BASE |
| CMD1 | XF\_RCAR\_REG\_DVC1\_BASE |

|  |  |
| --- | --- |
| static const WORD32 cmd\_register\_base[CMDMAX] | |
| Array’s index | Value |
| CMD0 | XF\_RCAR\_REG\_CMD0OUT\_BUSIF\_MODE |
| CMD1 | XF\_RCAR\_REG\_CMD1OUT\_BUSIF\_MODE |

|  |  |  |
| --- | --- | --- |
| static const SRC\_REG xa\_src\_register\_base[SRCMAX] | | |
| Array’s index | SRC\_CONFIG\_BASE | SRC\_CONTROL\_BASE |
| SRC0 | XF\_RCAR\_REG\_SRC0IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(0) |
| SRC1 | XF\_RCAR\_REG\_SRC1IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(1) |
| SRC2 | XF\_RCAR\_REG\_SRC2IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(2) |
| SRC3 | XF\_RCAR\_REG\_SRC3IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(3) |
| SRC4 | XF\_RCAR\_REG\_SRC4IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(4) |
| SRC5 | XF\_RCAR\_REG\_SRC5IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(5) |
| SRC6 | XF\_RCAR\_REG\_SRC6IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(6) |
| SRC7 | XF\_RCAR\_REG\_SRC7IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(7) |
| SRC8 | XF\_RCAR\_REG\_SRC8IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(8) |
| SRC9 | XF\_RCAR\_REG\_SRC9IN\_BUSIF\_MODE | XF\_RCAR\_REG\_SRC\_SWRSR(9) |

|  |  |
| --- | --- |
| static SRC\_MODULE module\_support\_multichannel[SRC\_SUPPORT\_MULTICHANNEL\_NUM] | |
| Array’s index | Value |
| 0 | SRC0 |
| 1 | SRC1 |
| 2 | SRC3 |
| 3 | SRC4 |

## Function definition

### xa\_src\_check\_module\_index

DD\_PLG\_TDM\_06\_001

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SRC\_ERROR\_CODE xa\_src\_check\_module\_index(SRC\_MODULE module) | | | |
| **Function** | This API is used to check if current SRC supports multichannel stream. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SRC\_MODULE | module | I | SRC module  Valid value: [0: 9] |
| **Return value** | SRC\_ERROR\_OUT\_RAGE | | Current SRC does not support multichannel stream | |
| SRC\_ERROR\_NONE | | Current SRC supports multichannel stream | |
| **Description** | * xa\_src\_check\_module\_index command processing:   - Check if current SRC supports multichannel stream | | | |

[Covers: FD\_PLG\_TDM\_005, FD\_PLG\_TDM\_024, FD\_PLG\_TDM\_026, FD\_PLG\_TDM\_034, FD\_PLG\_TDM\_036]

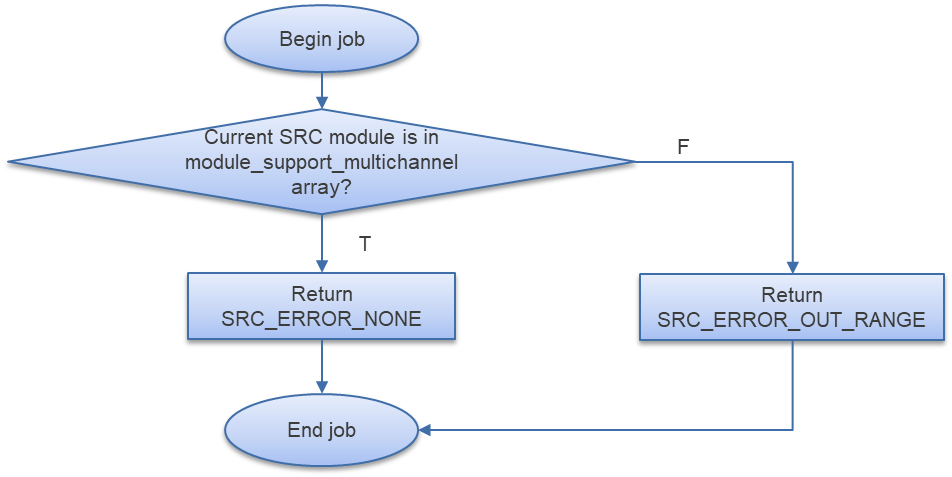


Figure 3‑1 xa\_src\_check\_module\_index flowchart

### xa\_src\_config

DD\_PLG\_TDM\_06\_002

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SRC\_ERROR\_CODE xa\_src\_config(SRC\_MODULE module, const SRC\_FLAG \*flag, SRC\_START\_MODE start) | | | |
| **Function** | This function is to set up SRC configuration. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SRC\_MODULE | module | I | SRC module  Valid value: [0: 9] |
| const SRC\_FLAG \* | flag | I | Pointer to SRC module configuration setting structure |
| SRC\_START\_MODE | start | I | SRC operation mode  Valid values: 0/1/2 |
| **Return value** | SRC\_ERROR\_INVALID | | Channel is invalid  SRC module does not support multichannel stream | |
| SRC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_src\_config command processing:   - Set registers for SRC execution | | | |

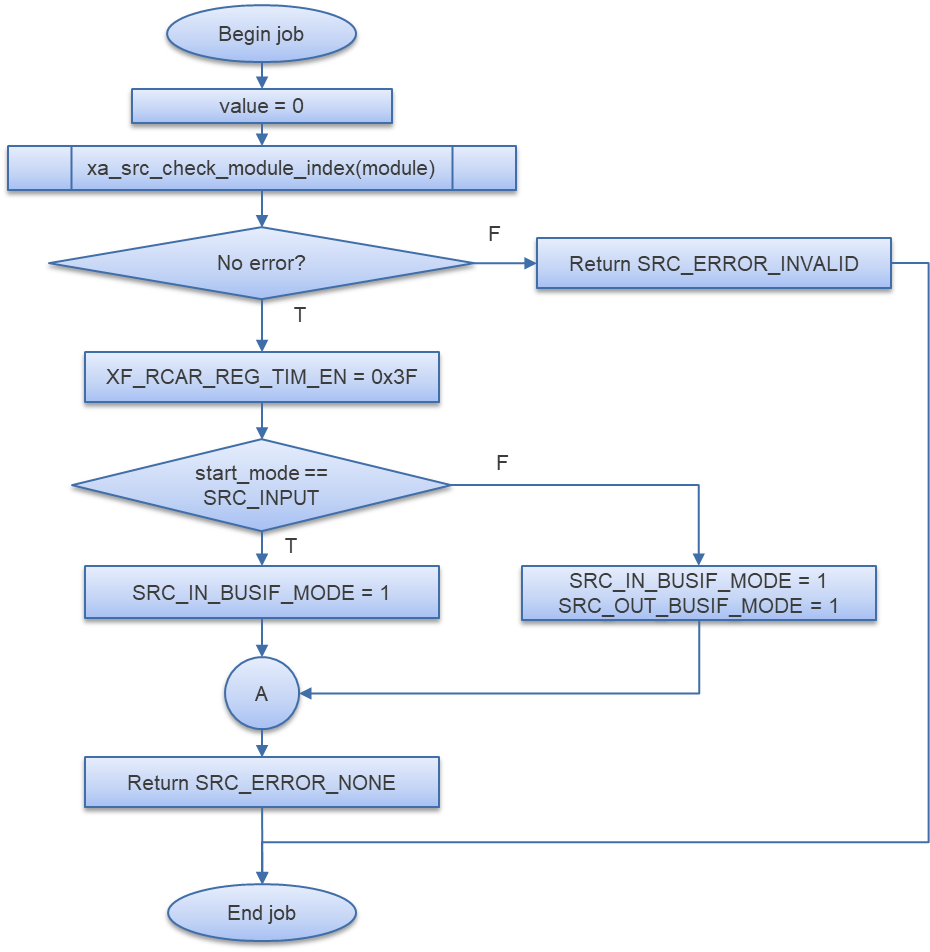
[Covers: FD\_PLG\_TDM\_005]

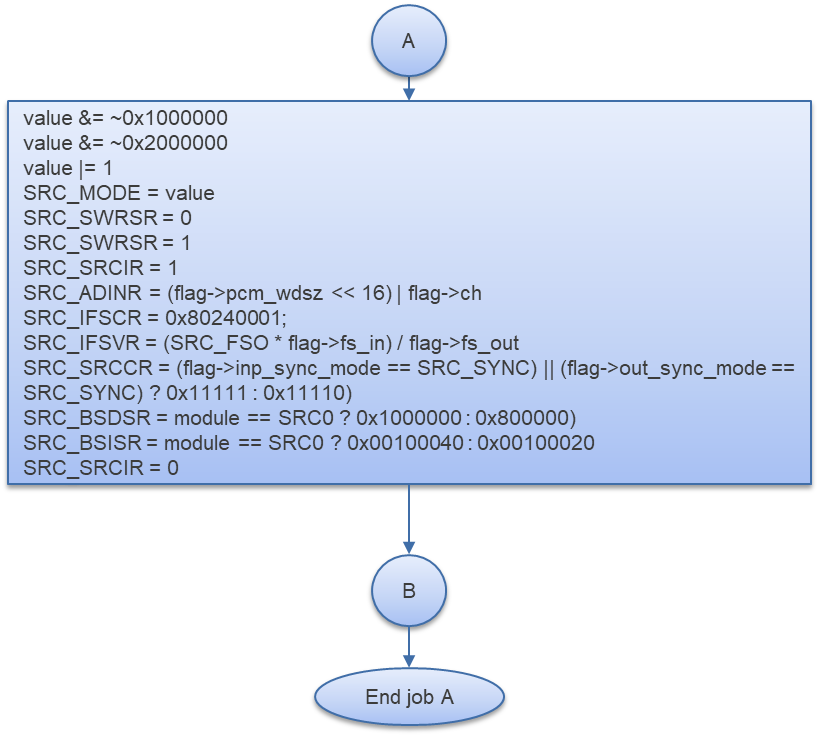
Table 3‑17 Setting SRC Input Timing Select registers based on input sampling rate

|  |  |  |  |
| --- | --- | --- | --- |
| **flag->fs\_in** | **module** | **register** | **value** |
| 32000 | SRC0 | XF\_RCAR\_REG\_SRCIN\_TIMSEL0[15:0] | 0x110 |
| SRC1 | XF\_RCAR\_REG\_SRCIN\_TIMSEL0[31:16] | 0x110 |
| SRC3 | XF\_RCAR\_REG\_SRCIN\_TIMSEL1[31:16] | 0x110 |
| SRC4 | XF\_RCAR\_REG\_SRCIN\_TIMSEL2[15:0] | 0x110 |
| 44100 | SRC0 | XF\_RCAR\_REG\_SRCIN\_TIMSEL0[15:0] | 0xF |
| SRC1 | XF\_RCAR\_REG\_SRCIN\_TIMSEL0[31:16] | 0xF |
| SRC3 | XF\_RCAR\_REG\_SRCIN\_TIMSEL1[31:16] | 0xF |
| SRC4 | XF\_RCAR\_REG\_SRCIN\_TIMSEL2[15:0] | 0xF |
| 48000 | SRC0 | XF\_RCAR\_REG\_SRCIN\_TIMSEL0[15:0] | 0x10F |
| SRC1 | XF\_RCAR\_REG\_SRCIN\_TIMSEL0[31:16] | 0x10F |
| SRC3 | XF\_RCAR\_REG\_SRCIN\_TIMSEL1[31:16] | 0x10F |
| SRC4 | XF\_RCAR\_REG\_SRCIN\_TIMSEL2[15:0] | 0x10F |

Table 3‑18 Setting SRC Output Timing Select registers based on output sampling rate

|  |  |  |  |
| --- | --- | --- | --- |
| **flag->fs\_out** | **module** | **register** | **value** |
| 32000 | SRC0 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL0[15:0] | 0x110 |
| SRC1 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL0[31:16] | 0x110 |
| SRC3 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL1[31:16] | 0x110 |
| SRC4 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL2[15:0] | 0x110 |
| 44100 | SRC0 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL0[15:0] | 0xF |
| SRC1 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL0[31:16] | 0xF |
| SRC3 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL1[31:16] | 0xF |
| SRC4 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL2[15:0] | 0xF |
| 48000 | SRC0 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL0[15:0] | 0x10F |
| SRC1 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL0[31:16] | 0x10F |
| SRC3 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL1[31:16] | 0x10F |
| SRC4 | XF\_RCAR\_REG\_SRCOUT\_TIMSEL2[15:0] | 0x10F |

****

****

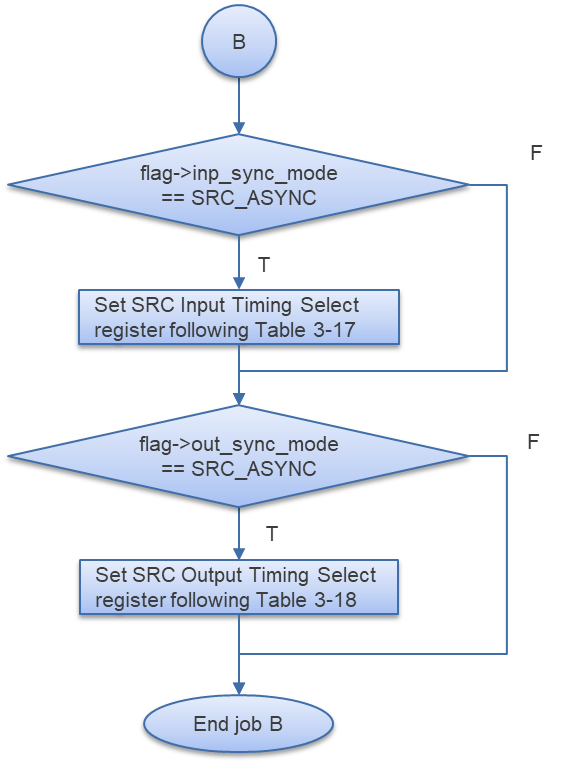
****

Figure 3‑2 xa\_src\_config flowchart

### xa\_src\_start

DD\_PLG\_TDM\_06\_003

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SRC\_ERROR\_CODE xa\_src\_start(SRC\_MODULE module, SRC\_START\_MODE start) | | | |
| **Function** | This function is to start SRC module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SRC\_MODULE | module | I | SRC module  Valid value: [0: 9] |
| SRC\_START\_MODE | start | I | SRC operation mode  Valid values: 0/1/2 |
| **Return value** | SRC\_ERROR\_BUSY | | SRC module is running | |
| SRC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_src\_start command processing:   - Start SRC execution | | | |

[Covers: FD\_PLG\_TDM\_005]

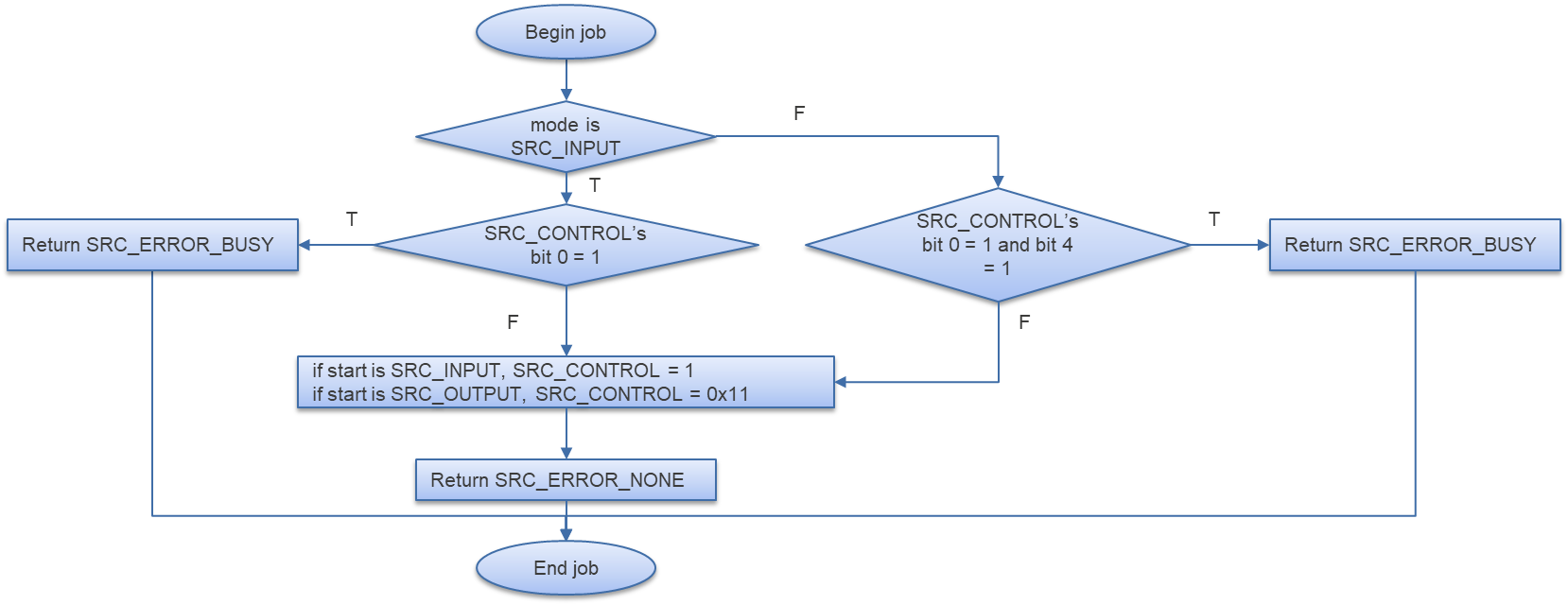
****

Figure 3‑3 xa\_src\_start flowchart

### xa\_src\_stop

DD\_PLG\_TDM\_06\_004

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SRC\_ERROR\_CODE xa\_src\_stop(SRC\_MODULE module) | | | |
| **Function** | This function is to stop SRC module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SRC\_MODULE | module | I | SRC module  Valid value: [0: 9] |
| **Return value** | SRC\_ERROR\_OUT\_RANGE | | SRC module is invalid | |
| SRC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_src\_stop command processing:   - Stop SRC execution | | | |

[Covers: FD\_PLG\_TDM\_017]

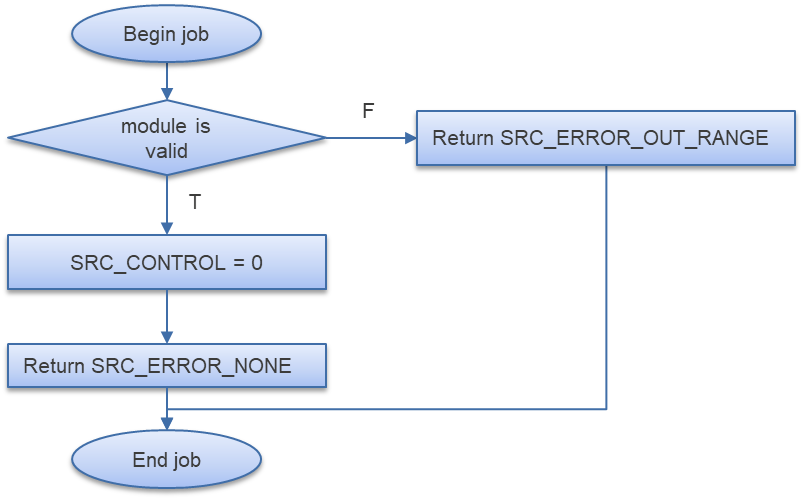
****

Figure 3‑4 start\_stop flowchart

### xa\_src\_check\_available\_module

DD\_PLG\_TDM\_06\_005

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | SRC\_ERROR\_CODE xa\_src\_check\_available\_module(UWORD32 \*pmodule, SRC\_CH\_NUM ch\_num) | | | |
| **Function** | This function is to check availability of SRC modules and choose an available one. | | | |
| **Arguments** | Type | Name | I/O | Description |
| UWORD32 \* | pmodule | I/O | Pointer to selected SRC module  Valid value: [0: 9] |
| SRC\_CH\_NUM | ch\_num | I | Channel number |
| **Return value** | SRC\_ERROR\_OUT\_RANGE | | SRC module is invalid | |
| SRC\_ERROR\_BUSY | | No free SRC modules | |
| SRC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_src\_check\_available\_module command processing:   - Check availability of SRC module and select an available one | | | |

[Covers: FD\_PLG\_TDM\_005]

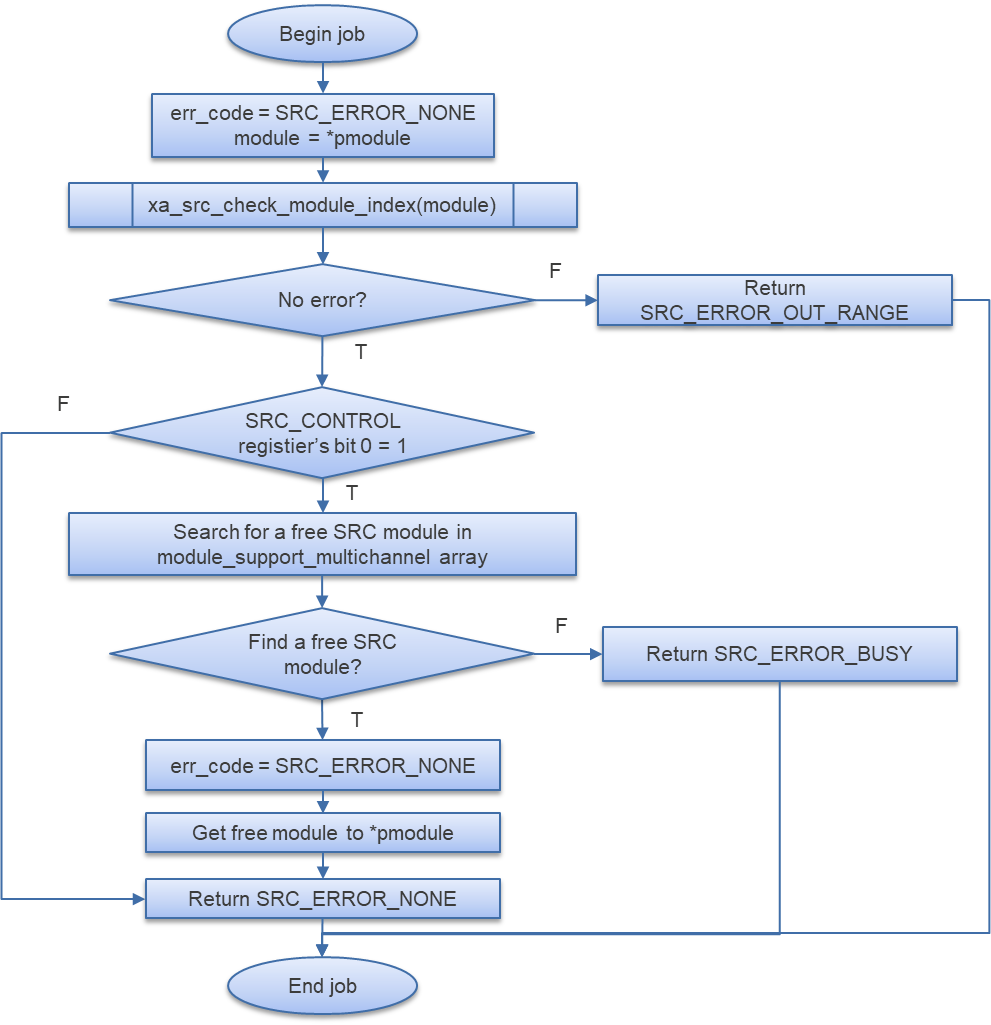
****

Figure 3‑5 xa\_src\_check\_available\_module flowchart

### xa\_dvc\_dig\_setting

DD\_PLG\_TDM\_06\_006

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | static DVC\_ERROR\_CODE xa\_dvc\_dig\_setting(DVC\_CH\_NUM ch, WORD32 vol[], volatile DVC\_CONFIG \*dvcConf) | | | |
| **Function** | This function is to set up DVC module in digital mode. | | | |
| **Arguments** | Type | Name | I/O | Description |
| DVC\_CH\_NUM | ch | I | Number of channels  Valid value: 0/1/2/4/6/8 |
| WORD32 | vol[] | I | Array of volume values to set |
| volatile DVC\_CONFIG \* | dvcConf | I | Pointer to DVC Config registers’ structure |
| **Return value** | DVC\_ERROR\_INVALID | | Channel is invalid | |
| DVC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_dvc\_dig\_setting command processing:   - Set up DVC registers in digital mode | | | |

[Covers: FD\_PLG\_TDM\_005]

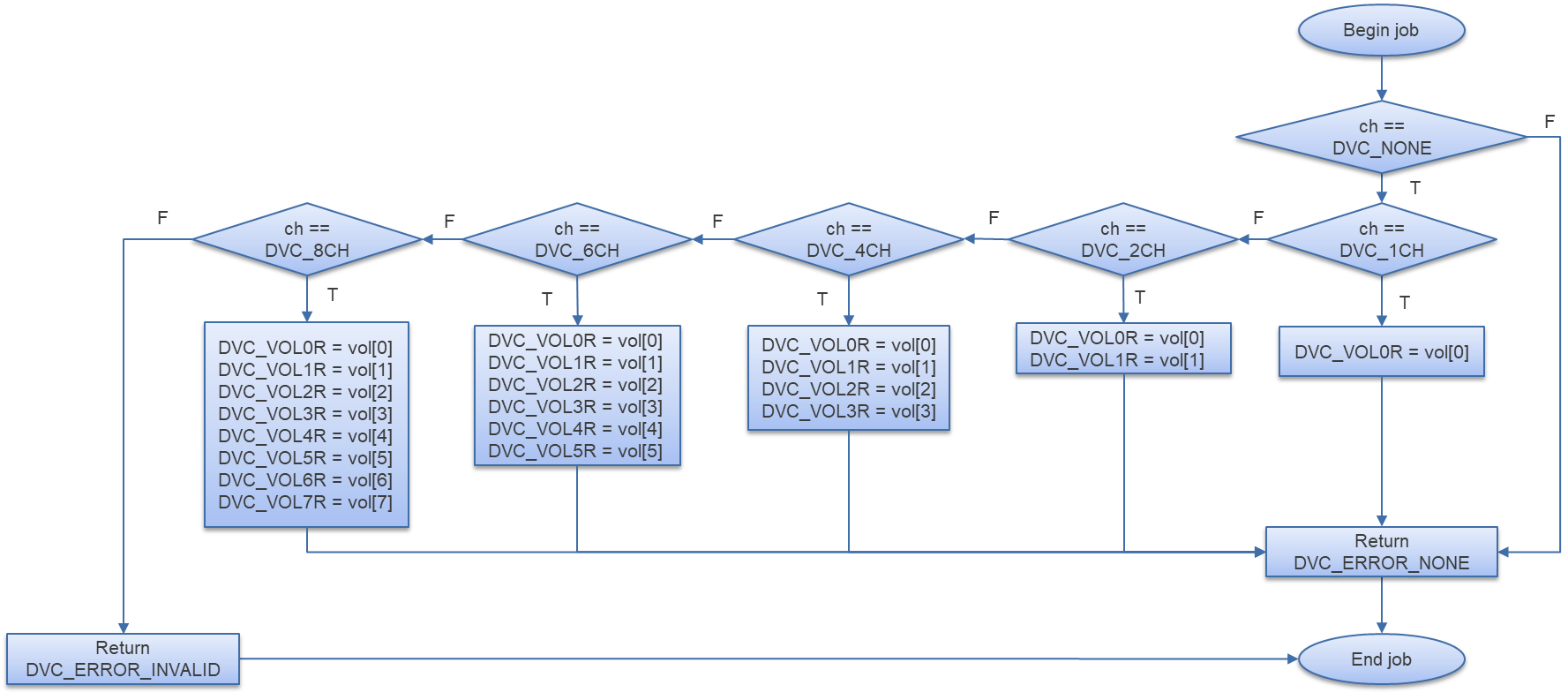


Figure 3‑6 xa\_dvc\_dig\_setting flowchart

### xa\_dvc\_config

DD\_PLG\_TDM\_06\_007

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | DVC\_ERROR\_CODE dvc\_config(CMD\_FLAG \*flag) | | | |
| **Function** | This function is to set up for DVC module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| CMD\_FLAG \* | flag | I/O | Pointer to CMD’s parameter structure |
| **Return value** | DVC\_ERROR\_INVALID | | SRC module is invalid  DVC mode is invalid  Cannot set up DVC digital mode | |
| DVC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_dvc\_config command processing:   - Set up registers for DVC execution | | | |

[Covers: FD\_PLG\_TDM\_005]

Table 3‑19 Setting XF\_RCAR\_REG\_CMDOUT\_TIMSEL for CMD execution

|  |  |  |  |
| --- | --- | --- | --- |
| **flag->fs** | **module** | **register** | **value** |
| 32000 | CMD0 | XF\_RCAR\_REG\_CMDOUT\_TIMSEL[15:0] | 0x110 |
| CMD1 | XF\_RCAR\_REG\_CMDOUT\_TIMSEL[31:16] | 0x110 |
| 44100 | CMD0 | XF\_RCAR\_REG\_CMDOUT\_TIMSEL[15:0] | 0xF |
| CMD1 | XF\_RCAR\_REG\_CMDOUT\_TIMSEL[31:16] | 0xF |
| 48000 | CMD0 | XF\_RCAR\_REG\_CMDOUT\_TIMSEL[15:0] | 0x10F |
| CMD1 | XF\_RCAR\_REG\_CMDOUT\_TIMSEL[31:16] | 0x10F |

Table 3‑20 Setting for CMD\_ROUTE\_SELECT register

|  |  |
| --- | --- |
| **flag->SRC\_module** | **value to set CMD\_ROUTE\_SELECT register** |
| SRC0 | 0x30000 |
| SRC1 | 0x30001 |
| SRC3 | 0x10000 |
| SRC4 | 0x20000 |

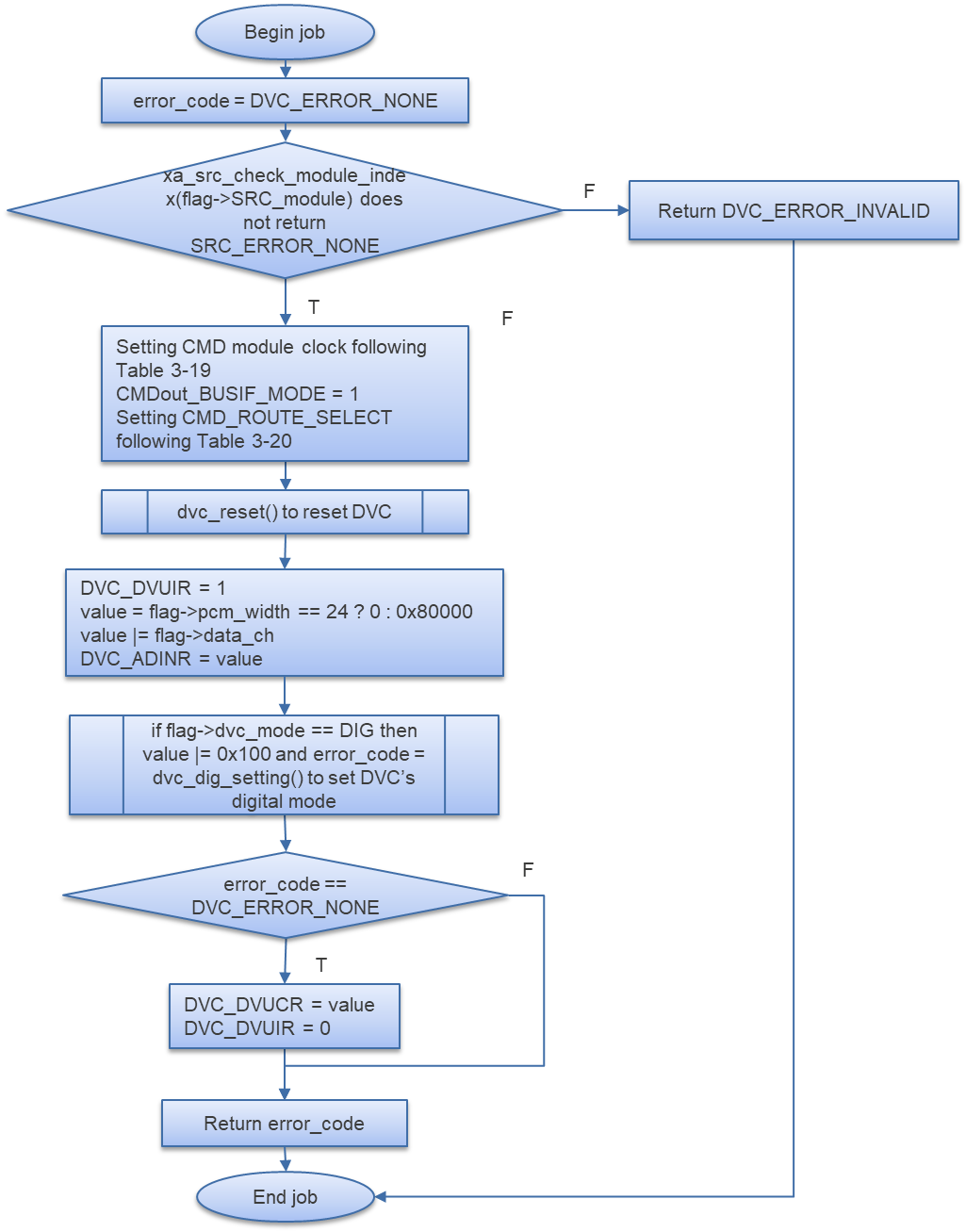


Figure 3‑7 xa\_dvc\_config flowchart

### xa\_dvc\_start

DD\_PLG\_TDM\_06\_008

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | DVC\_ERROR\_CODE xa\_dvc\_start(CMD\_MODULES modules) | | | |
| **Function** | This function is to start CMD module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| CMD\_MODULES | modules | I | CMD module  Valid value: [0: 1] |
| **Return value** | DVC\_ERROR\_BUSY | | CMD module is running | |
| DVC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_dvc\_start command processing:   - Start DVC module by set ‘1’ to bit 4 of CMD\_CONTROL register | | | |

[Covers: FD\_PLG\_TDM\_005]

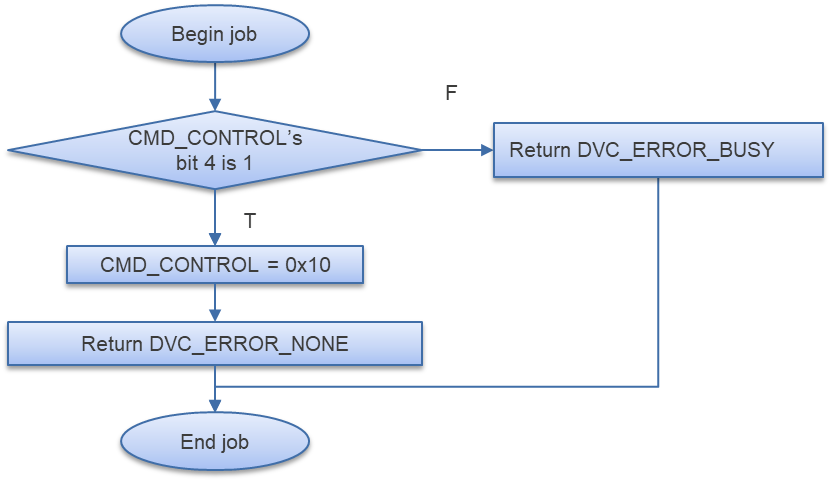


Figure 3‑8 xa\_dvc\_start flowchart

### xa\_dvc\_stop

DD\_PLG\_TDM\_06\_009

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | DVC\_ERROR\_CODE xa\_dvc\_stop(CMD\_MODULES modules) | | | |
| **Function** | This function is to stop CMD module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| CMD\_MODULES | modules | I | CMD module  Valid value: [0: 1] |
| **Return value** | DVC\_ERROR\_INVALID | | CMD module is invalid | |
| DVC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_dvc\_stop command processing:   - Stop DVC module by set ‘0’ to CMD\_CONTROL register | | | |

[Covers: FD\_PLG\_TDM\_017]

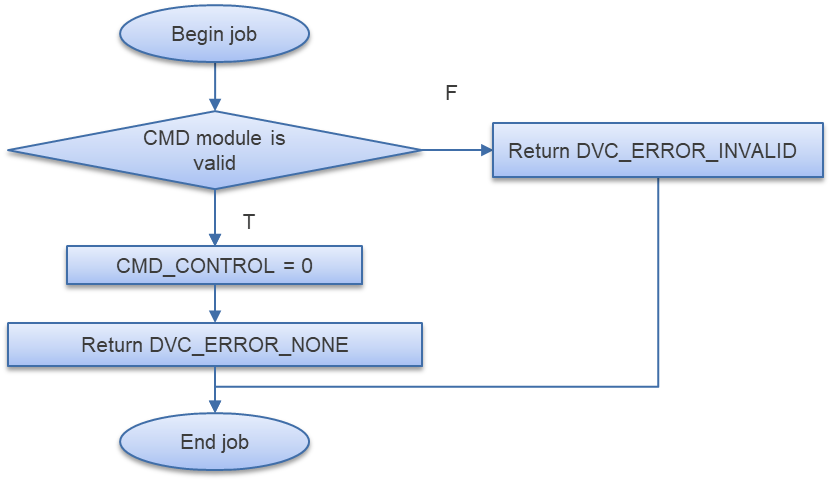
****

Figure 3‑9 xa\_dvc\_stop flowchart

### xa\_dvc\_reset

DD\_PLG\_TDM\_06\_010

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | static VOID xa\_dvc\_reset(CMD\_MODULES modules) | | | |
| **Function** | This function is to reset DVC module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| CMD\_MODULES | modules | I | CMD module  Valid value: [0: 1] |
| **Return value** | None | | | |
| **Description** | * xa\_dvc\_reset command processing:   - Set DVC\_SWRSR to reset DVC module | | | |

[Covers: FD\_PLG\_TDM\_005]

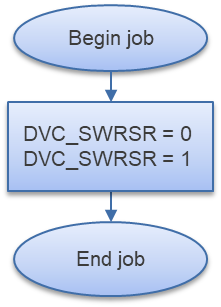


Figure 3‑10 xa\_dvc\_reset flowchart

### xa\_dvc\_check\_available

DD\_PLG\_TDM\_06\_011

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | DVC\_ERROR\_CODE xa\_dvc\_check\_available(UWORD32 \*pcmd\_num) | | | |
| **Function** | This function is to check availability of DVC module and select an available one. | | | |
| **Arguments** | Type | Name | I/O | Description |
| UWORD32 \* | pcmd\_num | I/O | Pointer to DVC module selected  Valid value: [0: 1] |
| **Return value** | DVC\_ERROR\_BUSY | | DVC modules are busy | |
| DVC\_ERROR\_INVALID | | DVC module is invalid | |
| DVC\_ERROR\_NONE | | Normal end | |
| **Description** | * xa\_dvc\_check\_available command processing:   - Check availability of DVC module and select a free one | | | |

[Covers: FD\_PLG\_TDM\_005]

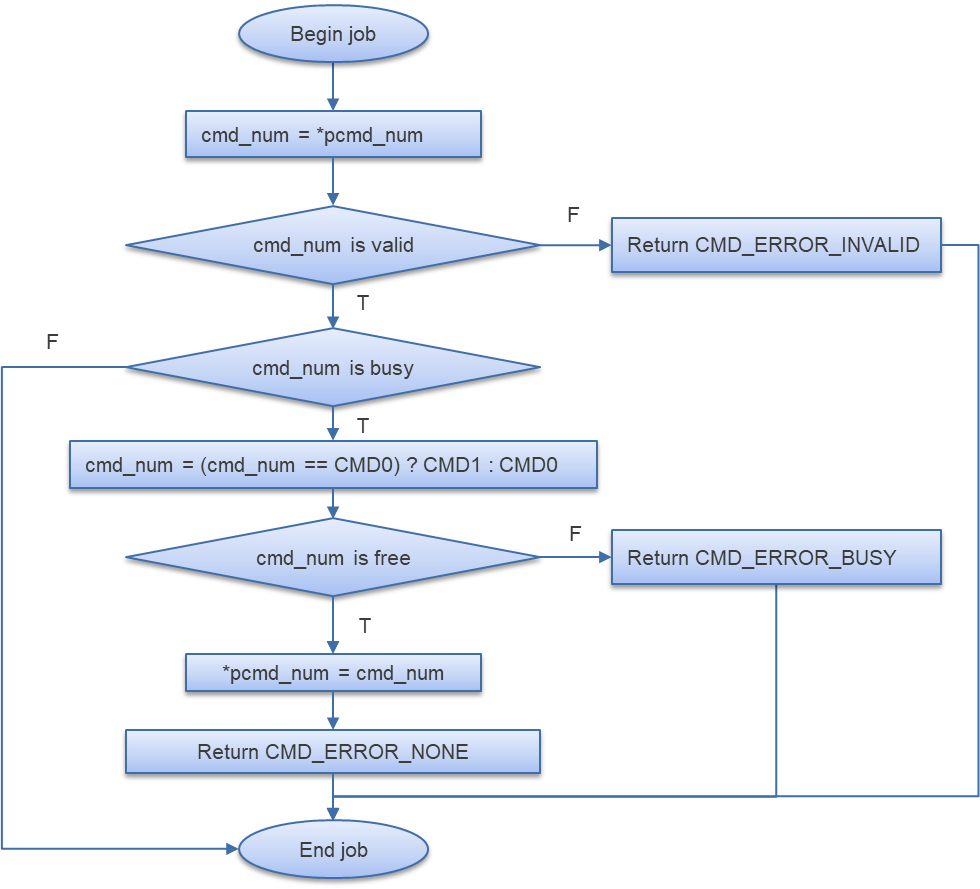


Figure 3‑11 xa\_dvc\_check\_available flowchart

### xa\_src\_get\_config

DD\_PLG\_TDM\_06\_012

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | static volatile inline SRC\_CONFIG \*xa\_src\_get\_config(SRC\_MODULE module) | | | |
| **Function** | This function is to get base address of Config registers of SRC module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SRC\_MODULE | module | I | SRC module  Valid value: [0: 9] |
| **Return value** | 0 | | SRC module is invalid | |
| Base address of Config registers of SRC module | | | |
| **Description** | * xa\_src\_get\_config command processing:   - Get base address of Config registers of SRC module | | | |

[Covers: FD\_PLG\_TDM\_005, FD\_PLG\_TDM\_017]

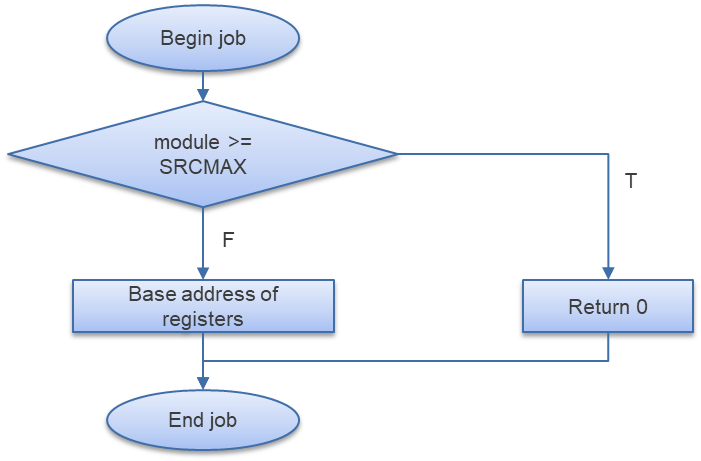


Figure 3‑12 xa\_src\_get\_config flowchart

### xa\_src\_get\_control

DD\_PLG\_TDM\_06\_013

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Syntax** | static volatile inline SRC\_CTRL \*xa\_src\_get\_control(SRC\_MODULE module) | | | |
| **Function** | This function is to get base address of Control registers of SRC module. | | | |
| **Arguments** | Type | Name | I/O | Description |
| SRC\_MODULE | module | I | SRC module  Valid value: [0: 9] |
| **Return value** | Base address of Control registers of SRC module | | | |
| **Description** | * xa\_src\_get\_control command processing:   - Get base address of Control registers of SRC module | | | |

[Covers: FD\_PLG\_TDM\_005]

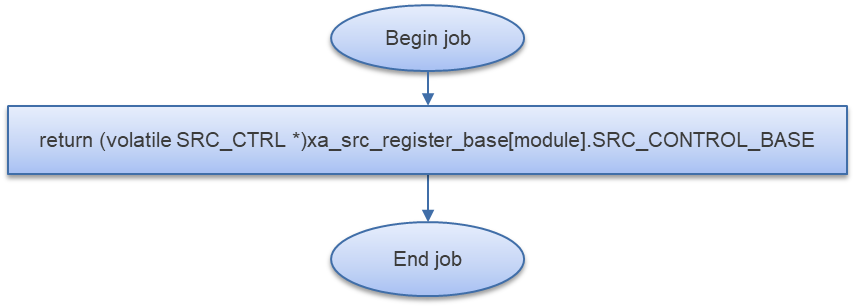


Figure 3‑13 xa\_src\_get\_control flowchart

# Revision history

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Version** | **Date** | **Page** | **Content** | **Approved** | **Changed** |
| 1.0.0 | Nov 14 2018 | - | First Edition issued | Vu Phan | Nguyen Dang |
| 1.1.0 | Dec 10 2018 | - | Add traceability ID | Vu Phan | Nguyen Dang |
| 1.2.0 | Jan 03, 2019 | - | Add range for input parameters | Vu Phan | Tien Tran |