SIGMASTUDIO FOR SHARC (ADSP-SC5XX/ADSP-215XX) - INTEGRATION GUIDE

Document Status	Approved
Approved by	ASH

ANALOG DEVICES, INC.

www.analog.com

Revision List

Revision	Date	Description
0.1	01.06.2015	Initial draft for M3
0.2	05.06.2015	Incorporated review corrections
1.0	13.07.2015	Approved and Base-lined.
1.1	01.02.2016	Updated for Release 3.5.0 (API for parameter update, New memory block for user assisted load balancing).
1.2	02.02.2016	Updated for review comments.
2.0	02.02.2016	Approved and Baselined for 3.5.0
2.1	19.04.2016	Updated for Release 3.6.0.
2.2	21.04.2016	Updated reference section.
2.3	21.04.2016	Removed track changes.
3.0	21.04.2016	Approved and Base-lined for 3.6.0
3.1	22.06.2016	Updated for Release 3.7.0. • Updated document for ADSP-SC573 processor. • Added sections ”Target Framework Status” and “Input/Output Buffers for Data Buffering”
3.2	27.06.2016	Incorporated review comments.
4.0	29.06.2016	Approved and Base-lined for 3.7.0
4.1	04.10.2016	Updated for Release 3.8.0
4.2	06.10.2016	Review comments incorporated
5.0	07.10.2016	Approved and Base-lined for 3.8.0
5.1	28.05.2017	Updated for release 3.10.0 Beta
5.2	27.06.2017	Added GMAP and SMAP details.
6.0	28.06.2017	Approved and Baselined for 3.10.0 beta
6.1	26.09.2017	Updated for release 3.11.0
6.2	29.09.2017	Review comments incorporated
7.0	29.09.2017	Approved and Baselined for 3.11.0
7.1	19.12.2017	Updated for release 3.12.0
8.0	21.12.2017	Approved and Baselined for 3.12.0
8.1	16.03.2018	Updated for release 4.0.0

9.0	21.03.2018	Approved and Baselined for 4.0.0
9.1	22.05.2018	Updated for 4.1.0 Release
9.2	30.05.2018	Minor changes
9.3	30.05.2018	Addressed review comments
9.4	01.06.2018	Addressed review comments
10.0	04.06.2018	Approved and Baselined for 4.1.0
10.1	10.08.2018	Updated for 4.2.0 Release
10.2	29.08.2018	Addressed review comments
11.0	04.09.2018	Approved and Baselined for 4.2.0
11.1	19.06.2019	Updated for release 4.4.0
11.2	27.06.2019	Addressed review comments
12.0	02.07.2019	Approved and Baselined for 4.4.0
12.1	05.11.2019	Updated for release 4.5.0
12.2	07.11.2019	Updated for release 4.5.0 and submitted for Review
12.3	08.11.2019	Addressed review comments
13.0	08.11.2019	Approved and Baselined for 4.5.0
13.1	18.12.2020	Updated for release 4.6.0
13.2	22.12.2020	Addressed review comments for release 4.6.0
14.0	23.12.2020	Approved and baselined for release 4.6.0
14.1	11.04.2022	Updated for release 4.7.0
14.2	13.04.2022	Addressed the review comments.
15.0	14.04.2022	Approved and baselined for release 4.7.0

Table 1: Revision List

Copyright, Disclaimer Statements

Copyright Information

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

SigmaStudio™ is a development environment from Analog Devices for graphically programming ADI's DSPs. SigmaStudio for SHARC (ADSP-SC5xx[i]) includes an extensive set of algorithms to perform audio processing tasks such as filtering and mixing, as well as basic low-level DSP functions, optimized to run on the SHARC family of processors.

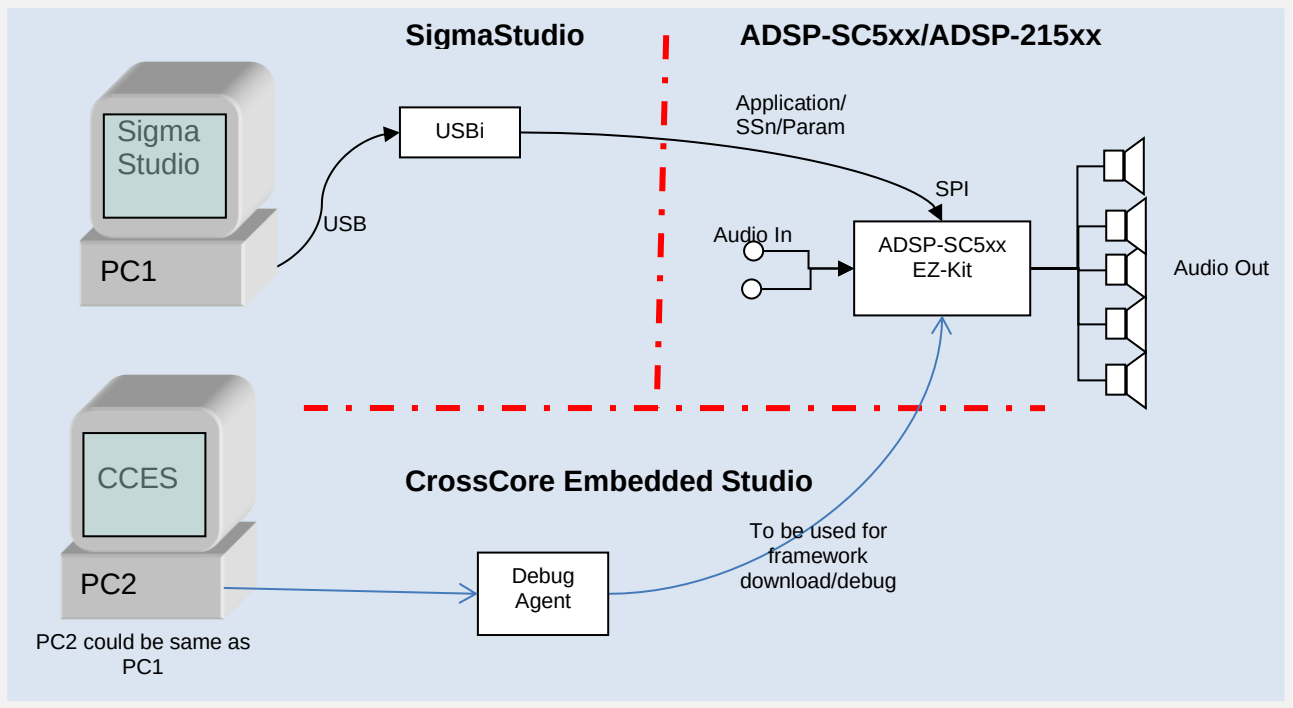


Figure 1: Connecting SigmaStudio Host PC with SHARC Board

The environment also extends parameter export and filter coefficient generation support for a SigmaStudio host microcontroller. Automation API support is provided to connect with many other tools, such as Python, .NET application, Matlab®, and LabVIEW. An easy-to-use graphical interface allows users to create custom filters, compressors and other audio-shaping algorithms to improve or change the characteristics of the audio. SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Algorithm Designer is provided to convert existing Software Modules or other SHARC libraries into SigmaStudio Plug-Ins. The environment is integrated with CrossCore® Embedded Studio.

The development environment for the SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) is shown in the figure above. As part of booting, the Application DXE, that can initialize the audio codec, setup the required peripherals and communicate with the SigmaStudio Host through SPI is loaded and run using the CrossCore Embedded Studio IDE, with the help of ICE-2000. After successful booting and initialization, the code corresponding to the designed Schematic (SSn) is downloaded to SHARC program memory. Once the SSn is loaded, the Application running on SHARC can call the SSn entry point to execute the algorithm described in SigmaStudio.

1.1 Scope

This document is intended to assist design engineers and advanced Tuning engineers. This document describes how SigmaStudio interfaces with the SHARC Target. This document provides a detailed description on the Application creation and instructions that software engineers can use to create new Applications with SigmaStudio for SHARC Target Library APIs or integrate the SigmaStudio functionality to existing SHARC applications. This document also provides a detailed description of the default framework supplied within the package.

1.2 Organization of the Guide

Information supplied in this guide is organised in the following way:

Section 1 : this section contains the introduction

Section 2 : this section lists the specifications of the product

Section 3 : this section describes the hardware setup required

Section 4 : this section discusses different use cases of the product and how to create Application for each of them.

Section 5 : this section describes about the components of SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) product.

Section 6 : this section details about system integration

2 Specifications

2.1 Version Information

Refer to the Release Notes for version information of SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx).

2.2 Target Platforms

The SHARC Target platform used is ADSP-SC584, ADSP-SC589, ADSP-SC573, ADSP-21569 EZ-KIT, ADSP-21569 SOMCRR, ADSP-SC594 SOMCRR and ADSP-21593 SOMCRR. ADSP-21584 and ADSP-21573 processors use the ADSP-SC584 and ADSP-SC573 EZ-KIT platform.

2.3 Input Formats

Input samples to the SSN target library should be in 32-bit Floating point format. Audio samples should have absolute amplitude of 1 representing 0 dBFS.

2.4 Output Formats

Output samples are in 32-bit Floating point format. Audio samples have absolute amplitude of 1 representing 0 dBFS.

3 Hardware Setup

Refer to section 3 of AE_42_SS4G_QuickStartGuide.docx **[1]** for the Hardware setup.

4 Quick Start Guide

The SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) can be used in the following scenarios:

1. Design Mode and Tuning

   SigmaStudio is used to create the Schematic (SSn) and tune it. In this mode, the SHARC can't run any algorithm other than the Schematics downloaded by SigmaStudio.

   In this case, no SHARC programming is required. An application along with a framework is downloaded from the SigmaStudio Host. This Application is run using CrossCore Embedded Studio IDE and after booting, it establishes the links between the SigmaStudio Host and the SHARC Target.

2. Advanced Design Mode and Tuning

   In this mode, SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) functionality can be added to existing Applications/framework. For example, the SigmaStudio functionality can be used along with an existing AC3 decoder Application. The decoder output can be fed to the SSn for performing post processing. Section 5 describes the details of the API to be used for integrating SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) to an existing Application.

   This mode also enables users to create new Applications using the library APIs. The Application can have many other features based on the requirement and is limited only by the SHARC's resources.

3. Deployment Mode (also known as Micro-controller mode)

   Once the Tuning is finalized, SigmaStudio can be used to save the binary containing Schematics and parameters so that the SigmaStudio uC Host can perform mode switching and parameter control. This step is applicable for both cases above. Note that, there will be differences/limitation in tuning certain Algorithms.

4.1 Step-By-Step Usage Guide

4.1.1 Design Mode and Tuning

This section contains a step-by-step guide for creating an Application that can be connected to SigmaStudio through USB-SPI connectivity. The figure below shows the overall flow of the process happening in the SigmaStudio Host and the Target.

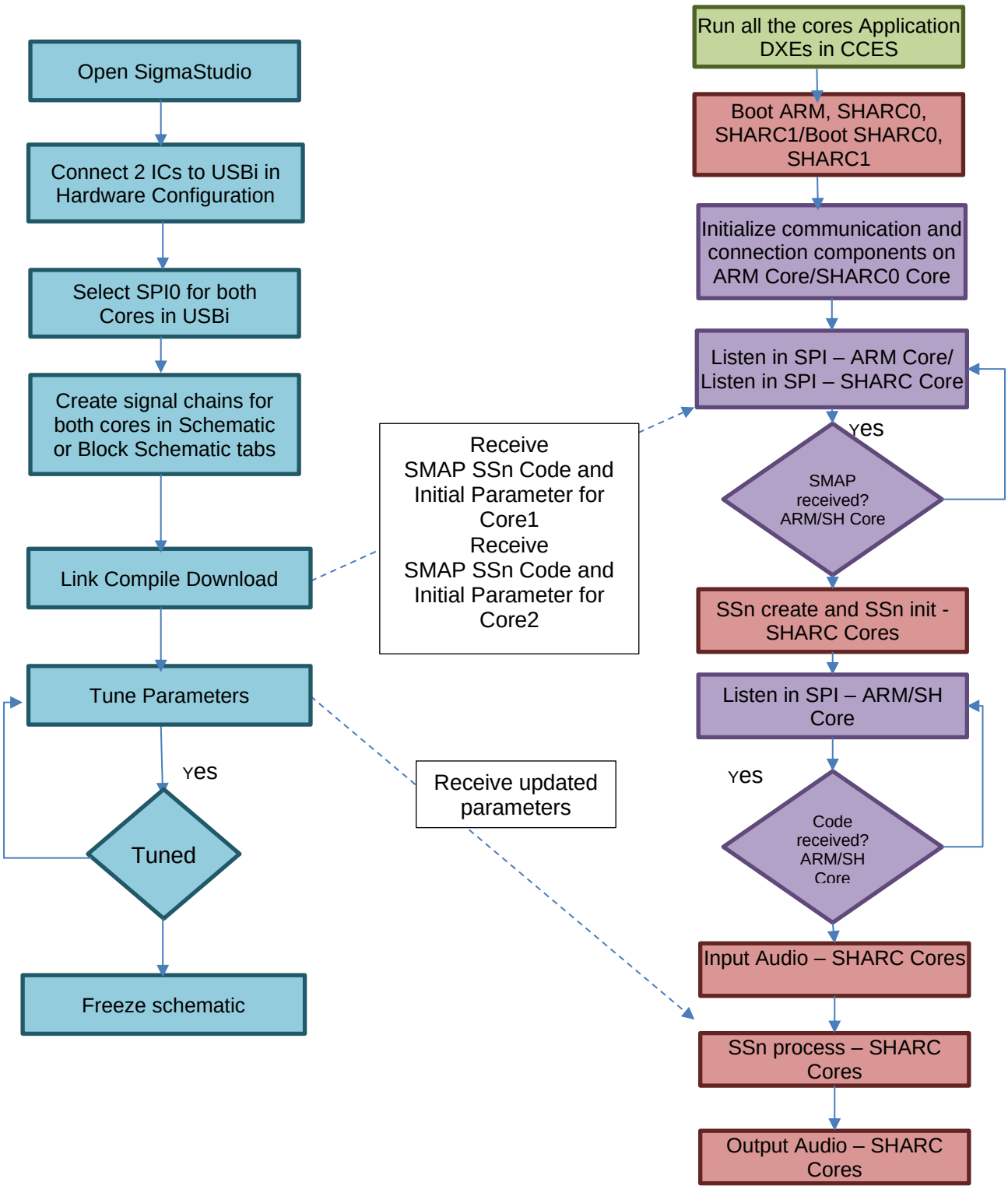


Figure 2: Communication and SSN Host Control for ADSP-SC5xx, ADSP-2157x, ADSP-2158x and ADSP-2159x

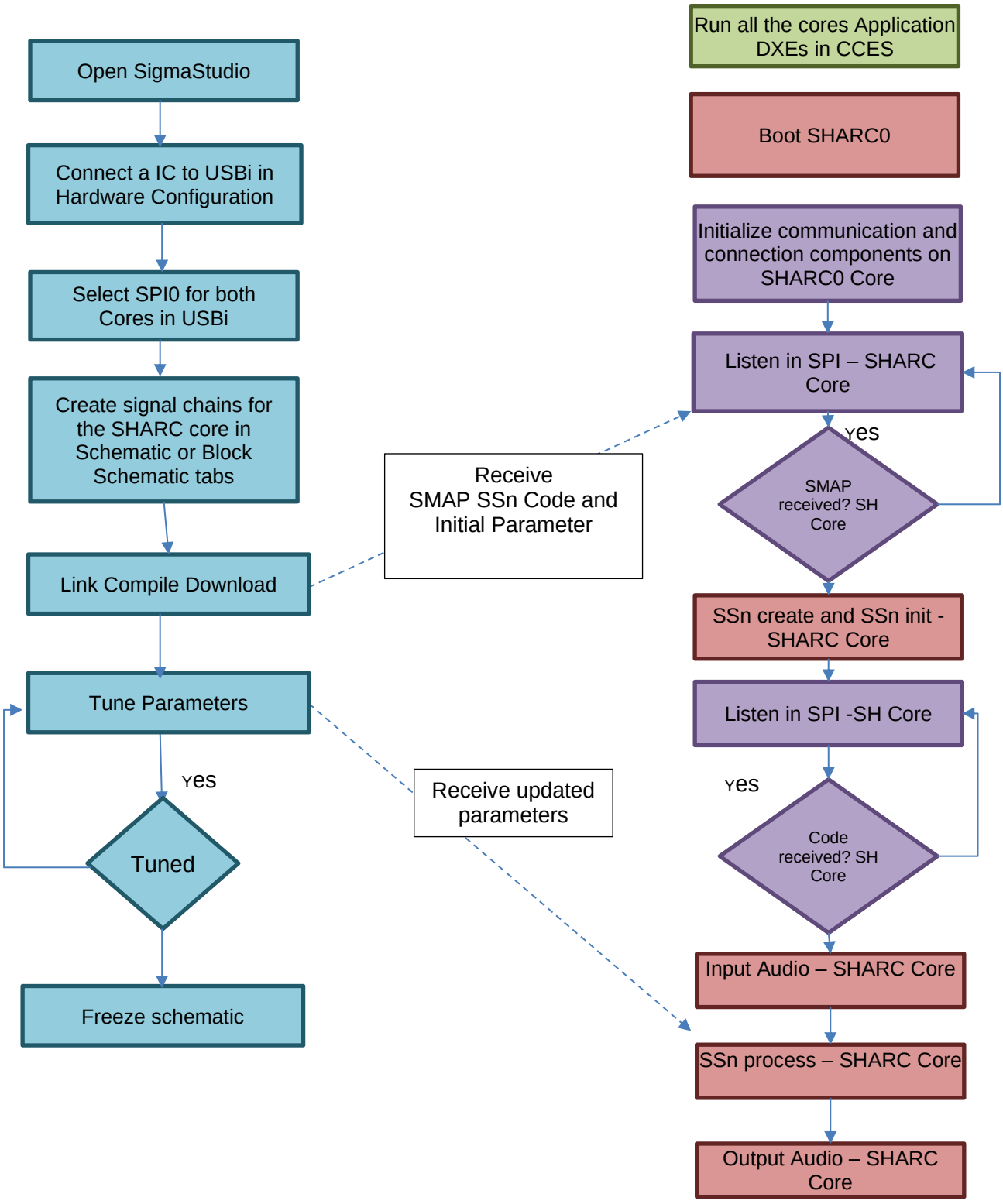


Figure 3: Communication and SSN Host control for ADSP-2156x

4.1.1.1 Selecting Hardware Configuration

The SHARC EZ-Kit is connected to the SigmaStudio Host PC through SPI using the USB to SPI adaptor card. The connection can be established by dragging and dropping the processor and USBi communication Module into the Schematics in the SigmaStudio "Hardware Configuration" window. Figure 4 shows the connection for ADSP-SC5xx Processors and Figure 5 shows the connection for ADSP-2156x processors.

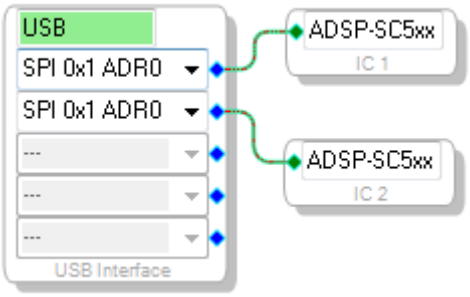


Figure 4: SSn Host Control Connectivity for ADSP-SC5xx/ADSP-2158x/ADSP-2157x/ADSP-2159x

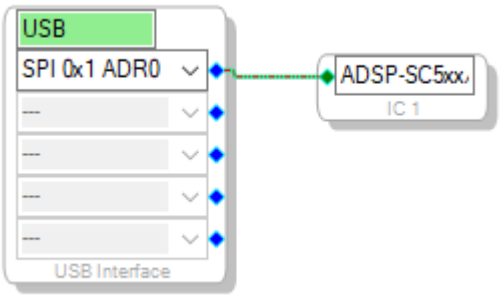


Figure 5: SSn Host Control Connectivity for ADSP-2156x

4.1.1.2 Creating Schematics

Refer to the section with respect to *'Creating a New Schematic'* in AE_42_SS4G_QuickStartGuide.docx **[1]**.

4.1.1.3 Compile and Run the Schematics

Refer to the section *'Running a New Schematic'* in AE_42_SS4G_QuickStartGuide.docx **[1]**.

4.1.1.4 Tune the Cells in the Schematics

The Modules in the schematic can be tuned by modifying the graphical controls on it. The tuning parameters are sent to the target each time a control value is changed. The SPI connectivity is used for sending the tuning parameters to the target.

4.1.2 Advanced Design Mode

In the advanced design mode, the user can write their own Application using the APIs. This flexibility allows the user to integrate the SigmaStudio Tuning feature to existing Applications. This method also enables the user to have SSn and other algorithms in a single Application. The section below explains how to write a custom Application.

4.1.2.1 Integrating target and communication libraries into custom real-time application

In order to create an Application (SHARC in this case), running the communication library and target libraries, users must do the following:

1. In CrossCore Embedded Studio, start a new CrossCore Project. Enter the name of the project, type of project, processor etc. when requested, and proceed by pressing 'Next'.
2. Add the SigmaStudio for SHARC communication library and target library to the project from the installed Add-ins for the project.
3. Include the following header files in the Application code.
    a. *adi_ss_smap.h*
    b. *adi_ss_connection.h*
    c. *adi_ss_communication.h*
    d. *adi_ss_ssn.h*
4. Add the Backchannel information structure present in Framework\Include\ adi_ss_ipc.h to the application code.

```
/* Backchannel info structure.*/
typedef struct ADI_SS_BACKCH_INFO
{
    float32_t    nPeakMIPS[ADI_SS_FW_MAX_PROC_BLOCKS];
    float32_t    nAvgMIPS[ADI_SS_FW_MAX_PROC_BLOCKS];
    uint32_t     nVersionInfo;
    uint32_t     nSSnDownloadStatus;
}ADI_SS_BACKCH_INFO;
```

5. Update the GMAP structure to allocate the total memory for each the memory blocks, to be sent to the SigmaStudio Host. An example is as shown below. Refer 7.1 for details on GMAP.

```
#define NUM_BLOCKS 7
#define SIZE_GMAP (NUM_BLOCKS*2 + 1)
.global _GMAP;

.ALIGN 4;
_GMAP:
    .var = NUM_BLOCKS;    /* Number of memory blocks */
    .var = _Block0_L1_space;     /* Mem Block 0 L1 in LDF */
    .var = _Block0_L1_space_length;    /* Total Mem Size available for SigmaStudio
Host from L1 Block0 */
    .var = _Block0_L1_space_flag;      /* Currently not used */
    .var = _Block1_L1_space;     /* Mem Block 1 L1 in LDF */
    .var = _Block1_L1_space_length;    /* Total Mem Size available for SigmaStudio
Host from L1 Block1 */
    .var = _Block1_L1_space_flag;      /* Currently not used */
    .var = _Block2_L1_space;     /* Mem Block 2 L1 in LDF */
    .var = _Block2_L1_space_length;    /* Total Mem Size available for SigmaStudio
Host from L1 Block2 */
    .var = _Block2_L1_space_flag;      /* Currently not used */
    .var = _Block3_L1_space;     /* Mem Block 3 L1 in LDF */
    .var = _Block3_L1_space_length;    /* Total Mem Size available for SigmaStudio
Host from L1 Block3 */
    .var = _Block3_L1_space_flag;      /* Currently not used */
    .var = _Block_L3_code_space;       /* Mem L3 Code (external) in LDF */
    .var = _Block_L3_code_space_length; /* Total Mem Size available for SigmaStudio
Host from L3 code */
    .var = _Block_L3_code_space_flag;  /* Currently not used */
    .var = _Block_L3_data_space;       /* Mem L3 Data (external) in LDF */
    .var = _Block_L3_data_space_length; /* Total Mem Size available for SigmaStudio
Host from L3 data */
    .var = _Block_L3_data_space_flag;  /* Currently not used */
    .var = _Block_L2_data_space;       /* Mem L2 Data (data cache) in LDF */
    .var = _Block_L2_data_space_length; /* Total Mem Size available for SigmaStudio
Host from L2 data cache */
    .var = _Block_L2_data_space_flag;  /* Currently not used */
._GMAP.end:
```

6. Allocate instance memory for connection and communication component within the application as shown below

```
/* Connection and Communication Instance Mem */
ALIGN(4)
SECTION("ss_app_data0_fast")
uint8_t adi_ss_commn_mem[ADI_SS_COMM_MEMSIZE];
ALIGN(4)
SECTION("ss_app_data0_fast")
uint8_t adi_ss_connection_mem[ADI_SS_CONNECTION_MEMSIZE];
```

7. Initialize memory block structure for the connection instance and call the "*adi_ss_connection_init()*" function to initialize the connection instance as shown below.

```
/* Connection Instance Initialization */
pConnConfig = &oConnConfig;
pConnConfig->eConnectionType = ADI_SS_CONNECTION_SPI;
pConnConfig->nDevId = 1;
pConnConfig->eProcID = PROCESSOR_SH0;

pConnectionMemBlk = &oConnectionMemBlk;
pConnectionMemBlk->nSize = ADI_SS_CONNECTION_MEMSIZE;
pConnectionMemBlk->pMem = adi_ss_connection_mem;
eConnRet = adi_ss_connection_Init(pConnectionMemBlk, pConnConfig, &hConnHandle);
```

8. Initialize memory block structure for the communication instance and call the "*adi_ss_comm_init()*" function to initialize the communication instance as shown below

```
      /* Communication Instance Initialization */
pCommnConfig = &oCommnConfig;
pCommnConfig->bCRCBypass = false;
pCommnConfig->bFullPacketCRC = true;
pCommnConfig->pfCommCmd4CallBack = (ADI_SS_COMM_CMD4_CB)
adi_ss_comm_callback_cmd4;
pCommnConfig->pfCommSMAPCallBack = (ADI_SS_COMM_SMAP_CB)
adi_SMAP_Application_Callback;
pCommnConfig->hConnectionHandle = hConnHandle;
pCommnConfig->pMemSMap[PROCESSOR_SH0] = &oSMAP;
pCommnConfig->pMemSMap[PROCESSOR_SH1] = &oSMAP1;
pCommnConfig->pBkChnlInfo[PROCESSOR_SH0] = &oBkChannelInfo;
pCommnConfig->pBkChnlInfo[PROCESSOR_SH1] = &oBkChannelInfo1;

pCommMemBlk = &oCommMemBlk;
pCommMemBlk->nSize = ADI_SS_COMM_MEMSIZE;
pCommMemBlk->pMem = adi_ss_commn_mem;
adi_ss_comm_Init(pCommMemBlk, pCommnConfig,&hCommHandle);
```

9. Wait for the SMAP information to be received from the host by waiting on a flag which is set in the SMAP callback.

```
/* Wait for SMAP to be received */
while(!nSMAPReceived)
{
      ;
}

void adi_SMAP_Application_Callback(ADI_SS_PROC_ID eCoreID)
{
     if(eCoreID==PROCESSOR_ARM)
     {
        nSMAPReceived = 1;
     }
}
```

10. Populate the memory addresses of SSn memory blocks from the SMAP received from the host

```
 /*Populating Memory addresses of different blocks from SMAO*/
oSSnMemMap.nMemBlocks = MAX_SMAP_SSN_BUFFERS;
pSSnInfo=&oSMAP.oSSnInfo[0];
for(i=0;i<oSSnMemMap.nMemBlocks;i++)
{
  oSSnMemMap.pMemBlocks[i] =&pSSnInfo->oSSnBuff[i];
}
```

11. Create the SSn instance

```
/*Creating SSn handle*/
adi_ss_create(&hSSnHandle, &oSSnMemMap);
```

12. Populate the SSn handle in the Communication properties structure in the communication library using the "*adi_ss_comm_SetProperties*" API.

```
  /*Populating the SSn handle  in Comm Properties structure*/
pCommProp.haSSnHandle[PROCESSOR_SH0][0] = hSSnHandle;
pCommProp.nNumProcBlocks = 1;
pCommProp.nProcId = PROCESSOR_ARM;
eCommRet =
adi_ss_comm_SetProperties(hCommHandle,ADI_COMM_PROP_SSN_HANDLE,&pCommProp);
```

13. Initialize the SSn instance using the "*adi_ss_init*" API. All default configurations are sent through the oSSnConfig structure

```
/*SSnConfig initialization */
pSSnConfig=&oSSnConfig;
pSSnConfig->hSSComm = hCommHandle;
pSSnConfig->nBlockSize = BLOCK_SIZE;
pSSnConfig->nInChannels = NUM_INPUT_CHANNELS;
pSSnConfig->nOutChannels = NUM_OUTPUT_CHANNELS;
pSSnConfig->bSkipProcessOnCRCError = 0;
pSSnConfig->bSkipInitialDownload = 0U;
pSSnConfig->nInitNoWait = 0;
pSSnConfig->eProcID = 0;
pSSnConfig->bClearUnusedOutput = 1;
eSSRes =  adi_ss_init(hSSnHandle, pSSnConfig);
```

14. Assign the input and output buffers to the pointer array.

```
for(idx=0;idx<NUM_INPUT_CHANNELS;idx++)
{
    input_buff[idx]=&Block_In[idx * BLOCK_SIZE];
}

for(idx=0;idx<NUM_OUTPUT_CHANNELS;idx++)
{
    output_buff[idx]=&Block_Out[idx * BLOCK_SIZE];
}
```

15. Call the "*adi_ss_schematic_process()"* function to perform the SSn operation. One or more extra audio processing modules may be added to the system either before or after the *adi_ss_schematic_process()* function.

```
adi_ss_schematic_process(hSSnHandle,BLOCK_SIZE,input_buff,output_buf
f,pSSnProperties);
```

5 Programmer's Reference

This section describes the installation files and the components of SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) product.

5.1 Installers

The installer required for using SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) is

- SigmaStudioForSHARC-ADSPSC5xx-Rel4.7.0_Bundle.exe.

Please refer "Hardware and Software Requirements" and "Installation Information" sections of **[4]** for details on the dependent software for using this product.

Note: This bundle contains the SigmaStudio Installer as well. There is no need to install SigmaStudio separately

5.2 Files

The product consists of the SHARC target library, communication library, target framework and SigmaStudio Host DLLs. Also included in the product are a User Manual and other documentation.

5.3 SSn Target Library

This section provides a detailed description of the SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Library API structures and functions.

The sections describing the SSn target library is organized as follows:

- Section 5.3.1 Target library and header file details
- Section 5.3.2 API Functions
- Section 5.3.3 API Data Types
- Section 5.3.4 API Macros

5.3.1 Target library and header file details

The SSn target library for ADSP-SC5xx processors is

- libadi_sigma_sharc_SC5xx.dlb: SHARC Target Library generated with Short Word Code (VISA) and byte-addressing mode for ADSP-SC5xx processors.

The SSn target library for ADSP-215xx processors is

- libadi_sigma_sharc_215xx.dlb: SHARC Target Library generated with Short Word Code (VISA) and byte-addressing mode for ADSP-215xx processors.

Note: The Framework Source files, which are available in Target/Framework/Source and Target/Sys/Source can be used along with the libraries, to create a whole Application.

5.3.1.1 Include Files

The include files required for integration of this library is

- *adi_ss_ssn.h*
  This public file contains all the API definitions, structures and macros of the SSn target library.
- *adi_ss_common.h*.
  This public file contains structures for memory definition.

5.3.1.2 Linker Files

The linker files required to be used along with this library is

- adi_ss_app.ldf: This .ldf file should be used along with the application .ldf file. This .ldf file reserves sections for GMAP and for IPC.

5.3.2 API Functions

The SSn Target Library API functions are described in the sections below.

5.3.2.1 adi_ss_create

Prototype

```
ADI_SS_RESULT adi_ss_create(
ADI_SS_SSN_HANDLE *phSSn,
ADI_SS_MEM_MAP    *pMemMap
);
```

Description

The function `adi_ss_create` creates an instance of the SSn module, with the memory blocks passed to the function, using the memory map structure.

Parameters

Name: phSSn

Type: ADI_SS_SSN_HANDLE *

Direction: Output

Description: Pointer to the handle of SSn instance.

Name: pMemMap

Type: ADI_SS_MEM_MAP *

Direction: Input

Description: Used to pass information regarding the memory blocks that can be used by the SSn module.

Return value

An appropriate error code of type ADI_SS_RESULT is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.2.2 adi_ss_init

Prototype

```
ADI_SS_RESULT adi_ss_init(
      ADI_SS_SSN_HANDLE  hSSn,
      ADI_SS_CONFIG     *pConfig
);
```

Description

The function adi_ss_init initializes an instance of the SSn with the specified configuration.

This function also performs other initializations such as:

- Validate the configuration parameters against the configure information in SSn block.
- Set the function pointer for SSn process.
- Set status to ADI_SSN_STATE_INITED.
- Listen in communication ISR.

Once the initialization is completed, the function either waits for the code and data to be received or immediately comes out depending upon the communication configuration. When data and code are received, they are placed in appropriate locations based on the memory locations specified by adi_ss_create.

Parameters

Name: hSSn

Type: ADI_SS_SSN_HANDLE

Direction: Input

Description: Handle to the SSn instance.

Name: pConfig

Type: ADI_SS_CONFIG *

Direction: Input

Description: Used to pass configuration parameters to the SSn.

Return value

An appropriate error code of type ADI_SS_RESULT is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.2.3 adi_ss_schematic_process

Prototype

```
ADI_SS_RESULT adi_ss_schematic_process(
      ADI_SS_SSN_HANDLE       hSSn,
      int32_t                 num_input_samples_per_chan,
      adi_ss_sample_t        *input_data[],
      adi_ss_sample_t        *output_data[],
      ADI_SS_SSNPROPERTIES   *pProperties)
);
```

Description

This is the modified process API. This function calls the function pointer of SSn. The function pointer of SSn is the main entry point in the SSn block which is created by SigmaStudio during the linking process. This process API is used in the Default Application provided in the package.

Parameters

Name: hSSn

Type: ADI_SS_SSN_HANDLE

Direction: Input

Description: Handle to the SSn instance.

Name: num_input_samples_per_channel

Type: `int32_t`

Direction: Input

Description: Number of samples per channel passed in the buffer in case of Linear PCM samples.

Name: `input_data`

Type: `adi_ss_sample_t *`

Direction: Input

Description: Pointer to the input audio data buffer. Audio samples should be in floating point format and multiple channels can be saved in the buffer as either block or interleaved. The same setting should be used in the SigmaStudio Host application

Name: `output_data`

Type: `adi_ss_sample_t *`

Direction: Input

Description: Pointer to the output audio data buffer. Audio samples are in floating point format and multiple channels are arranged in the buffer either as block or interleaved based on the setting in the SigmaStudio Host.

Name: `pProperties`

Type: `ADI_SS_SSNPROPERTIES *`

Direction: Output

Description: Used to pass properties from the SSn to Application.

Return value

An appropriate error code of type `ADI_SS_RESULT` is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.2.4 adi_ss_getProperties

Prototype

```
ADI_SS_RESULT adi_ss_getProperties(
      ADI_SS_SSN_HANDLE       hSSn,
      ADI_SS_SSNPROPERTIES   *pProperties
);
```

Description

The function is to get the current state and other properties of SSn instance from SigmaStudio.

Parameters

Name: hSSn

Type: ADI_SS_SSN_HANDLE

Direction: Input

Description: Handle to the SSn instance.

Name: pProperties

Type: ADI_SS_SSNPROPERTIES *

Direction: Output

Description: Used to pass properties from the SSn to Application.

Return value

An appropriate error code of type ADI_SS_RESULT is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.2.5 adi_ss_reset

Prototype

```
ADI_SS_RESULT adi_ss_reset(
      ADI_SS_COMM_HANDLE  hSSComm,
      ADI_SS_COMM_CONFIG *pConf
);
```

Description

The function is to get the reset the SSn instance to its initial state.

Parameters

Name: phSSn

Type: ADI_SS_SSN_HANDLE *

Direction: Output

Description: Pointer to the handle of SSn instance.

Name: pConf

Type: ADI_SS_COMM_CONFIG *

Direction: Input

Description: Used to pass the configuration parameters to the communication module.

Return value

An appropriate error code of type `ADI_SS_RESULT` is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.2.6 adi_ss_clearState

Prototype

```
ADI_SS_RESULT adi_ss_clearState(
      ADI_SS_SSN_HANDLE       hSSn,
);
```

Description

The function is to clear the state memory buffers used by the SSn instance.

Parameters

Name: `hSSn`

Type: `ADI_SS_SSN_HANDLE`

Direction: Input

Description: Handle to the SSn instance.

Return value

An appropriate error code of type `ADI_SS_RESULT` is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.2.7 adi_ss_updateParam

Prototype

```
ADI_SS_RESULT adi_ss_updateParam(ADI_SS_SSN_HANDLE        hSSn,
                                 uint32_t                 *pParamDataAddr,
                                 uint32_t                 nSSnParamMemOffset,
                                 uint32_t                 nNumParams);
```

Description

This function is used to update a single or a set SSn parameters into the SSn parameter memory space.

Parameters

Name: `hSSn`
Type: `ADI_SS_SSN_HANDLE`
Direction: Input
Description: Handle to the SSn instance.

Name: `pParamDataAddr`
Type: `uint32_t *`
Direction: Input
Description: Pointer to memory holding the new parameter values.

Name: `nSSnParamMemOffset`
Type: `uint32_t`
Direction: Input
Description: Offset from the parameter memory base into which the new parameters need to be updated.

Name: `nNumParams`
Type: `uint32_t`
Direction: Input
Description: Number of parameters that needs to be updated.

Return value

An appropriate error code of type `ADI_SS_RESULT` is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.2.8 adi_ss_readParam

Prototype

```
ADI_SS_RESULT adi_ss_readParam(ADI_SS_SSN_HANDLE        hSSn,
                               uint32_t                 *pBuffer,
                               uint32_t                 nSSnParamMemOffset,
                               uint32_t                 nNumParams);
```

Description

This function is used to read a single or a set of parameters from the SSn parameter memory space.

Parameters

Name: `hSSn`

Type: `ADI_SS_SSN_HANDLE`

Direction: Input

Description: Handle to the SSn instance.

Name: `pBuffer`

Type: `uint32_t *`

Direction: Input

Description: Pointer to memory into which the SSn parameters are read.

Name: `nSSnParamMemOffset`

Type: `uint32_t`

Direction: Input

Description: Offset into the parameter memory base from which the SSn parameters are to be read.

Name: `nNumParams`

Type: `uint32_t`

Direction: Input

Description: Number of parameters that needs to be read.

Return value

An appropriate error code of type `ADI_SS_RESULT` is returned. For the list of supported error codes, refer to section 5.3.3.2.1 .

5.3.3 API Data Types

The SSn Target Library API data types are described in the sections below.

5.3.3.1 Structures

5.3.3.1.1 ADI_SS_MEM_BLOCK

```
typedef struct __MemBlock {
      int32_t  nSize;
      int32_t  nFlags;
      void    *pMem;
} ADI_SS_MEM_BLOCK;
```

Description

The structure `ADI_SS_MEM_BLOCK` captures the information about the individual memory blocks in the `ADI_SS_MEM_MAP` structure.

Fields

- `nSize`
  Indicates the size of the memory block.
- `nFlags`
  Flag indicating the status of the memory block. This field is reserved for future use.
- `pMem`
  Pointer to the memory block.

5.3.3.1.2 ADI_SS_MEM_MAP

```
typedef struct __MemMap {
      int32_t           nMemBlocks;
      ADI_SS_MEM_BLOCK *pMemBlocks[ADI_SS_MAX_MEM_BLOCKS];
} ADI_SS_MEM_MAP;
```

Description

The `ADI_MEM_MAP` structure passes the memory blocks that can be used by the SSn to the `adi_ss_create` function.

Fields

- `nMemBlocks`
  Indicates number of memory blocks.

- pMemBlocks
  Array of pointers to the memory blocks.

5.3.3.1.3 ADI_SS_CONFIG

```
typedef struct __SSnConfig {
      uint32_t                nBlockSize;
      uint32_t                nInChannels;
      uint32_t                nOutChannels;
      uint32_t                bSkipProcessOnCRCError;
      uint32_t                bSkipInitialDownload;
      ADI_SS_COMM_HANDLE      hSSComm;
#if defined(__ADSP2158x__)
      uint32_t               nInitNoWait;
      bool                   bClearUnusedOutput;
#endif
} ADI_SS_CONFIG;
```

Description

The structure ADI_SS_CONFIG is used by the Application to configure the SSn.

Fields

- nBlockSize
  Size of the input sample block passed to the SSn for processing. This is equal to the Application Block Size and is expressed in samples per channel.

- nInChannels
  Indicates the number of input audio buffer passed to the SSn.

- nOutChannels
  Indicates the number of output audio data received from the SSn.

- bSkipProcessOnCRCError
  Indicates whether to skip further processing of audio upon encountering a CRC or packet error. 1 indicates skip and 0 indicates continue audio processing.

- bSkipInitialDownload
  Set this to 1 if the code and parameters are not downloaded from the SigmaStudio Host through the communication channel. If this field is not set to ‘1’, the SSn instance internally skips the process stage until the code and parameter are received through the communication channel.

- hSSComm
  Handle to SigmaStudio communication instance.

- `nInitNoWait`
  Flag indicating whether the adi_ss_Init() API needs to wait for the SSn code and parameters to be received or not.

- `bClearUnusedOutput`
  Boolean flag indicating whether the unused output channels need to be cleared or not.

5.3.3.1.4 ADI_SS_SSNPROPERTIES

```
typedef struct __SSnProperties{
      int32_t                bSSnUpdate;
      uint32_t               nSSnCodeSize;
      uint32_t               nStatusField;
      uint32_t               nSSBlockSize;
      uint32_t               nSSInChannels;
      uint32_t               nSSOutChannels;
} ADI_SS_SSNPROPERTIES;
```

Description

The structure `ADI_SS_SSNPROPERTIES` holds the current state and other properties of SigmaStudio.

Fields

- `bSSnUpdate`
  This field indicates whether the SigmaStudio process code is currently being updated. This field has a value 1 if the code is being updated and 0 otherwise.

- `nSSnCodeSize`
  Size of the SigmaStudio process function generated by the SigmaStudio Host based on the Schematic.

- `nStatusField`
  This indicates the current status of the SSn.

- `nSSBlockSize`
  This indicates the current BlockSize used by the SSn.

- `nSSInChannels`
  This indicates the number of input channels used in the SSn.

- `nSSOutChannels`
  This indicates the number of output channels used in the SSn.

5.3.3.2 Enumerations

5.3.3.2.1 ADI_SS_RESULT

```
typedef enum __SSResult {
      ADI_SS_SUCCESS              = E_ADI_SS_SUCCESS,
      ADI_SS_FAILED               = E_ADI_SS_FAILED,
      ADI_SS_INSUFFICIENT_MEMORY = E_ADI_SS_INSUFFICIENT_MEMORY,
      ADI_SS_PAUSED               = E_ADI_SS_PAUSED,
      ADI_SS_INVALID_SCHEMATIC    = E_ADI_SS_INVALID_SCHEMATIC,
      ADI_SS_PROCESS_SKIP         = E_ADI_SS_PROCESS_SKIP
} ADI_SS_RESULT;
```

Description

Represents the different error codes, returned by the SSn.

5.3.3.2.2 ADI_SS_STATE

```
typedef enum __SSResult {
      ADI_SSN_STATE_CREATED       = E_ADI_SSN_STATE_CREATED,
      ADI_SSN_STATE_INITED        = E_ADI_SSN_STATE_INITED,
      ADI_SSN_STATE_CODE_READY    = E_ADI_SSN_STATE_CODE_READY,
      ADI_SSN_STATE_ERROR         = E_ADI_SSN_STATE_ERROR,
      ADI_SSN_STATE_PROGRESSING   = E_ADI_SSN_STATE_PROGRESSING,
      ADI_SSN_STATE_PARAM_READY   = E_ADI_SSN_STATE_PARAM_READY
} ADI_SS_RESULT;
```

Description

Represents the internal state of the SSn.

5.3.3.3 Typedefs

5.3.3.3.1 adi_ss_sample_t

```
typedef float32_t adi_ss_sample_t;
```

Description

This represents the input and output audio samples of SSn

5.3.3.3.2 ADI_SS_SSN_HANDLE

```
typedef void* ADI_SS_SSN_HANDLE;
```

Description

ADI_SS_SSN_HANDLE represents the handle to the SSn instance.

Refer to the AE_42_SS4G_DetailedDesign_Target.chm **[2]**, for the communication library API data types.

5.3.4 API Macros

The SSn Target Library API macros are described in the sections below.

5.3.4.1 Macros

5.3.4.1.1 E_ADI_SS_SUCCESS

Alternative to the enumeration ADI_SS_SUCCESS, for use in assembly.

5.3.4.1.2 E_ADI_SS_FAILED

Alternative to the enumeration ADI_SS_FAILED, for use in assembly.

5.3.4.1.3 E_ADI_SS_INSUFFICIENT_MEMORY

Alternative to the enumeration ADI_SS_INSUFFICIENT_MEMORY, for use in assembly.

5.3.4.1.4 E_ADI_SS_PAUSED

Alternative to the enumeration ADI_SS_PAUSED, for use in assembly.

5.3.4.1.5 E_ADI_SS_INVALID_SCHEMATIC

Alternative to the enumeration ADI_SS_INVALID_SCHEMATIC, for use in assembly.

5.3.4.1.6 E_ADI_SS_PROCESS_SKIP

Alternative to the enumeration ADI_SS_PROCESS_SKIP, for use in assembly.

5.3.4.1.7 E_ADI_SSN_STATE_CREATED

Alternative to the enumeration ADI_SSN_STATE_CREATED, for use in assembly.

5.3.4.1.8 E_ADI_SSN_STATE_INITED

Alternative to the enumeration ADI_SSN_STATE_INITED, for use in assembly.

5.3.4.1.9 E_ADI_SSN_STATE_CODE_READY

Alternative to the enumeration `ADI_SSN_STATE_CODE_READY`, for use in assembly.

5.3.4.1.10 E_ADI_SSN_STATE_ERROR

Alternative to the enumeration `ADI_SSN_STATE_ERROR`, for use in assembly.

5.3.4.1.11 E_ADI_SSN_STATE_PROGRESSING

Alternative to the enumeration `ADI_SSN_STATE_PROGRESSING`, for use in assembly.

5.3.4.1.12 E_ADI_SSN_STATE_PARAM_READY

Alternative to the enumeration `ADI_SSN_STATE_PARAM_READY`, for use in assembly.

Refer to the AE_42_SS4G_DetailedDesign_Target.chm **[2]**, for the communication library API macros.

5.4 SSn communication Library

This section provides a detailed description of the API structures and functions of SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) communication library.

The sections describing the SSn communication library is organized as follows:

- Section 5.4.1 : Details pertaining to the communication library and its interface header files.
- Section 5.4.2 : API Functions of communication library.
- Section 5.4.3 : API data structures of communication library.
- Section 5.4.4 : API enumerations and type defines macros of communication library.

5.4.1 Communication library and interface header file details

The communication library handles the parsing of the SPI packets received from the Host, handling SMAP, code and parameter download, parameter tuning, read-back communication etc.

The SSn communication library for ADSP-SC5xx processors are listed below:

- libadi_sigma_sharc_comm_SC58x_Core0.a or
  libadi_sigma_sharc_comm_SC57x_Core0.a or
  libadi_sigma_sharc_comm_SC59x_Core0.a

  This is the Communication Library which is built to run on ARM core of ADSP-SC58x/ ADSP-SC57x/ADSP-SC594 processors respectively.

- libadi_sigma_sharc_comm_2158x_Core1.dlb or
  libadi_sigma_sharc_comm_2157x_Core1.dlb or
  libadi_sigma_sharc_comm_2156x_Core1.dlb or
  libadi_sigma_sharc_comm_2159x_Core1.dlb

  This is the Communication Library which is built to run on SHARC core of ADSP-SC58x and ADSP-2158x/ADSP-SC57x, ADSP-2157x, ADSP-2156x and ADSP-2159x processors respectively. This Library is generated with Short Word Code (VISA) and byte-addressing mode.

5.4.1.1 Include Files

The include files required for integration of this library is

- ***adi_ss_communication.h***

  This public file contains interfaces for communication module (between Host and Target)

- ***adi_ss_connection.h***

  This public file contains interfaces for physical SPI connection.

- ***adi_ss_common.h***.
  This is a common public header file which contains structures for memory definition.

5.4.2 API Functions

5.4.2.1 Communication Component

5.4.2.1.1 adi_ss_comm_Init

Prototype

```
ADI_SS_COMM_RESULT adi_ss_comm_Init(ADI_SS_MEM_BLOCK *pMemBlk,
                                    ADI_SS_COMM_CONFIG *pCommConfig,
                                    ADI_SS_COMM_HANDLE *phSSComm)
```

Description

This function creates an instance of SigmaStudio Communication Module and initializes it based on the config parameters supplied by the application.

Parameters

Name: pMemBlk

Type: ADI_SS_MEM_BLOCK

Direction: Input

Description: Pointer to a memory block.

Name: pCommConfig

Type: ADI_SS_COMM_CONFIG

Direction: Input

Description: Pointer to communication config structure to be populated by the application.

Name: phSSComm

Type: ADI_SS_COMM_HANDLE

Direction: Output

Description: Pointer to communication handle.

Return value

An error code of type ADI_SS_COMM_RESULT is returned.

- ADI_SS_COMM_SUCCESS : Communication init successful.
- ADI_SS_COMM_INSUFFICIENT_MEMORY : Communication init failed due to insufficient memory.
- ADI_SS_COMM_FAILED : Communication init failed due to general error.

5.4.2.1.2 adi_ss_comm_Packetize

Prototype

```
ADI_SS_COMM_RESULT adi_ss_comm_Packetize(ADI_SS_COMM_HANDLE hSSComm,
                                         ADI_SS_COMM_BACKCH_INFO *pBkChInfo,
                                         uint32_t *pPacketSzInWords
                                         )
```

Description

This API packetizes the back-channel data for transmission.

Parameters

Name: hSSComm

Type: ADI_SS_COMM_HANDLE

Direction: Input

Description: Handle to communication component instance.

Name: pBkChInfo
Type: ADI_SS_COMM_BACKCH_INFO
Direction: Input
Description: Pointer to backchannel information structure.

Name: pPacketSzInWords
Type: uint32_t *
Direction: Output
Description: Pointer holding the size of the packet in words.

Return value

An error code of type ADI_SS_COMM_RESULT is returned.

- ADI_SS_COMM_SUCCESS : Function executed successfully
- ADI_SS_COMM_FAILED: Function execution resulted in error.

5.4.2.1.3 adi_ss_comm_Parse

Prototype

```
ADI_SS_COMM_RESULT adi_ss_comm_Parse(ADI_SS_COMM_HANDLE hSSComm,
                                     uint32_t nData
                                     )
```

Description

This function calls the library parse function to parse a single word from the host.

Parameters

Name: hSSComm
Type: ADI_SS_COMM_HANDLE
Direction: Input
Description: Handle to communication component instance.

Name: nData
Type: uint32_t

Direction: Input

Description: 32-bit received word.

Return value

An error code of type `ADI_SS_COMM_RESULT` is returned.

- ADI_SS_COMM_SUCCESS: Function executed successfully
- ADI_SS_COMM_FAILED: Function execution resulted in error.

5.4.2.1.4 adi_ss_comm_Reset

ADI_SS_COMM_RESULT adi_ss_comm_Reset(ADI_SS_COMM_HANDLE hSSComm)

Prototype

```
ADI_SS_COMM_RESULT adi_ss_comm_Reset(ADI_SS_COMM_HANDLE hSSComm)
```

Description

SigmaStudio Communication Module Reset.

Parameters

Name: `hSSComm`

Type: `ADI_SS_COMM_HANDLE`

Direction: Input

Description: Handle to communication component instance.

Return value

An error code of type `ADI_SS_COMM_RESULT` is returned.

- ADI_SS_COMM_SUCCESS : Function executed successfully
- ADI_SS_COMM_FAILED: Function execution resulted in error.

5.4.2.1.5 adi_ss_comm_GetProperties

Prototype

```
ADI_SS_COMM_RESULT adi_ss_comm_GetProperties(ADI_SS_COMM_HANDLE  hSSComm,
                                     ADI_SS_COMM_PROPERTIES *pCommProperties
                                                )
```

Description

API for getting current communication properties.

Parameters

Name: hSSComm

Type: ADI_SS_COMM_HANDLE

Direction: Input

Description: Handle to communication component instance.

Name: pCommProperties

Type: ADI_SS_COMM_PROPERTIES

Direction: Input

Description: Pointer to structure of type ADI_SS_COMM_PROPERTIES. The fields of this structure is updated by the communication component.

Return value

An error code of type ADI_SS_COMM_RESULT is returned.

- ADI_SS_COMM_SUCCESS : Function executed successfully
- ADI_SS_COMM_FAILED: Function execution resulted in error.

5.4.2.1.6 adi_ss_comm_SetProperties

Prototype

```
ADI_SS_COMM_RESULT adi_ss_comm_SetProperties(ADI_SS_COMM_HANDLE  hSSComm,
                                        ADI_COMM_PROPERTY_ID ePropId
                                        ADI_SS_COMM_PROPERTIES *pCommProperties
                                              )
```

Description

API for Setting communication properties.

Parameters

Name: hSSComm

Type: ADI_SS_COMM_HANDLE

Direction: Input

Description: Handle to communication component instance.

Name: ePropId

Type: ADI_SS_PROPERTY_ID

Direction: Input

Description: Enumeration for Setting communication properties.

Name: pCommProperties

Type: ADI_SS_COMM_PROPERTIES

Direction: Input

Description: Pointer to structure of type ADI_SS_COMM_PROPERTIES. The fields of this structure are updated by the communication component.

Return value

An error code of type ADI_SS_COMM_RESULT is returned.

- ADI_SS_COMM_SUCCESS: Function executed successfully
- ADI_SS_COMM_FAILED: Function execution resulted in error.

5.4.2.2 Connection Component

5.4.2.2.1 adi_ss_connection_Init

Prototype

```
ADI_SS_CONNECTION_RESULT adi_ss_connection_Init(ADI_SS_MEM_BLOCK *pMemBlk,
                                    ADI_SS_CONNECTION_CONFIG *pConnectionConfig,
                                       ADI_SS_CONNECTION_HANDLE *phSSConnection
                                             )
```

Description

Connection component API for creating and initializing a physical connection.

Parameters

Name: pMemBlk

Type: ADI_SS_MEM_BLOCK

Direction: Input

Description: Pointer to a block of memory.

Name: pConnectionConfig

Type: ADI_SS_CONNECTION_CONFIG*

Direction: Input

Description: Configuration structure populated by the application.

Name: phSSConnection

Type: ADI_SS_CONNECTION_HANDLE *

Direction: Input

Description: Pointer to the Handle to the Connection Component populated by this function.

Return value

Return code of ADI_SS_CONNECTION_RESULT type.

- ADI_SS_CONNECTION_SUCCESS : Connection Init successful.
- ADI_SS_CONNECTION_INSUFF_MEM : Connection Init failed due to insufficient memory.
- ADI_SS_CONNECTION_INVALID_CONFIG : Connection Init failed due to invalid config.
- ADI_SS_CONNECTION_FAILED : Connection Init failed due to general error.

5.4.2.2.2 adi_ss_connection_Reconfigure

Prototype

```
ADI_SS_CONNECTION_RESULT adi_ss_connection_Reconfigure(
                                    ADI_SS_CONNECTION_HANDLE hSSConnection,
                                    ADI_SS_CONNECTION_CONFIG_ITEM eConfigItem,
                                    ADI_SS_CONNECTION_CONFIG *pConnectionConfig
                                   )
```

Description

This API reconfigures the connection component based on the chosen config item.

Parameters

Name: hSSConnection

Type: ADI_SS_CONNECTION_HANDLE

Direction: Input

Description: Handle to the connection component instance.

Name: eConfigItem

Type: ADI_SS_CONNECTION_CONFIG_ITEM

Direction: Input

Description: Configuration parameter to be reconfigured.

Name: pConnectionConfig

Type: ADI_SS_CONNECTION_CONFIG*

Direction: Input

Return value

Return code of ADI_SS_CONNECTION_RESULT type.

- ADI_SS_CONNECTION_SUCCESS : Connection Init successful.
- ADI_SS_CONNECTION_INVALID_CONFIG : Connection Init failed due to invalid config.
- ADI_SS_CONNECTION_FAILED : Connection Init failed due to general error.

5.4.2.2.3 adi_ss_connection_set_CommHandle

Prototype

```
ADI_SS_CONNECTION_RESULT adi_ss_connection_set_CommHandle(
                                        ADI_SS_CONNECTION_HANDLE hSSConnection,
                                        ADI_SS_COMM_HANDLE hSSComm
                                        )
```

Description

This API provides the communication handle to the connection component.

Parameters

Name: hSSConnection

Type: ADI_SS_CONNECTION_HANDLE

Direction: Input

Description: Handle to the connection component instance.

Name: hSSComm

Type: ADI_SS_COMM_HANDLE

Direction: Input

Description: Handle to the communication component which is to be passed to the connection component.

Return value

Return code of ADI_SS_CONNECTION_RESULT type.

- ADI_SS_CONNECTION_SUCCESS : Connection Init successful.
- ADI_SS_CONNECTION_FAILED : Connection Init failed due to general error.

5.4.2.2.4 adi_ss_connection_Enable

Prototype

```
ADI_SS_CONNECTION_RESULT adi_ss_connection_Enable(
                                        ADI_SS_CONNECTION_HANDLE hSSConnection
                                                )
```

Description

This API to enable the physical connection.

Parameters

Name: hSSConnection

Type: ADI_SS_CONNECTION_HANDLE

Direction: Input

Description: Handle to the connection component instance.

Return value

Return code of ADI_SS_CONNECTION_RESULT type.

- ADI_SS_CONNECTION_SUCCESS : Connection Init successful.
- ADI_SS_CONNECTION_FAILED : Connection Init failed due to general error.

5.4.3 API data structures

5.4.3.1 Communication component

5.4.3.1.1 ADI_SS_COMM_CONFIG

```
typedef struct ADI_SS_COMM_CONFIG
{
    bool                    bCRCBypass;
    bool                    bFullPacketCRC;
    ADI_SS_COMM_CMD4_CB     pfCommCmd4CallBack;
    ADI_SS_COMM_APP_ISR_CB  pfSPIRxIsrCallBack;
    ADI_SS_COMM_APP_ISR_CB  pfSPITxIsrCallBack;
    ADI_SS_COMM_SMAP_CB     pfCommSMAPCallBack;
    void*                   hConnectionHandle;
    ADI_SS_MEM_SMAP        *pMemSMap[MAX_NUMBER_OF_PROCESSORS];
    ADI_SS_BACKCH_INFO     *pBkChnlInfo[MAX_NUMBER_OF_PROCESSORS];
}ADI_SS_COMM_CONFIG;
```

Description

The communication component configuration structure to be populated by the application.

Fields

- `bCRCBypass`
  A boolean flag to indicate bypassing of CRC check.
- `bFullPacketCRC`
  A boolean flag to indicate whether CRC check is required for the entire packet.
- `pfCommCmd4CallBack`
  Command 4 callback function pointer.
- `pfSPIRxIsrCallBack`
  ISR callback for Rx. The callback won't be registered if this is NULL.
- `pfSPITxIsrCallBack`
  ISR callback for Tx. The callback won't be registered if this is NULL.
- `pfCommSMAPCallBack`
  Application callback on receiving SMAP.
- `hConnectionHandle`
  Handle to connection component.
- `pMemSMap`
  SMAP pointers for each of the cores.

- `pBkChnlInfo`
  Pointers to Backchannel info structure for each of the cores.

5.4.3.1.2 ADI_SS_COMM_BACKCH_INFO

```
typedef struct ADI_SS_COMM_BACKCH_INFO
{
    uint32_t     nProc;
    uint32_t     nCommandName;
    uint32_t     *pInData;
    uint32_t     nInDataSize;
    uint32_t     *pOutPacket;
}ADI_SS_COMM_BACKCH_INFO;
```

Description

Backchannel information structure which needs to be populated by the function which calls `adi_ss_comm_Packetize()`.

Fields

- `nProc`
  Processor ID for which this back channel data belongs. This field is currently ignored.
- `nCommandName`
  Back channel command.
- `pInData`
  Pointer to payload data which needs to be sent back to the host (backchannel data).
- `nInDataSize`
  Size of the back channel payload data in words.
- `pOutPacket`
  Pointer to back channel packet which should be transmitted.

5.4.3.1.3 ADI_SS_COMM_PROPERTIES

```
typedef struct ADI_SS_COMM_PROPERTIES
{
    bool                bCommError;
    bool                bCustomCmdRcvd;
    uint32_t            *pSSnBuf;
    ADI_SS_MEM_SMAP     *pMemSMap;

    /* Fileds used for adi_ss_comm_SetProperties() API */
    uint32_t            nProcId;
    uint32_t            nNumProcBlocks;
    void*               hasSSnHandle[MAX_NUMBER_OF_PROCESSORS]
                        [ADI_SS_FW_MAX_PROC_BLOCKS]

}ADI_SS_COMM_PROPERTIES;
```

Description

Properties structure which provides information about the Communication component.

Fields

- `bCommError`
  Flag indicating communication error.
- `bCustomCmdRcvd`
  This flag indicates that a custom command is received.
- `pSSnBuf`
  This pointer contains the data received via custom command.
- `pMemSMap`
  Pointer to SMAP structure.
- `nProcId`
  This field is used to specify the Processor ID
- `nNumProcBlocks`
  This field is used to specify the number of Process Blocks running on a processor.
- `haSSnHandle`
  SSn handle for each of the Process blocks.

5.4.3.2 Connection component

5.4.3.2.1 ADI_SS_CONNECTION_CONFIG

```
typedef struct ADI_SS_CONNECTION_CONFIG
{
    /* One time configurable items */
    ADI_SS_CONNECTION_TYPE  eConnectionType;
    uint32_t                nDevId;
    ADI_SS_PROC_ID          eProcID;

    CONN_ISR                pfISRRx;
    CONN_ISR                pfISRTx;

    /* Reconfigurable items */
    uint32_t                *pRxAddr;
    uint32_t                nRxPayloadCnt;
    uint32_t                *pTxAddr;
    uint32_t                nTxPayloadCnt;
}ADI_SS_CONNECTION_CONFIG;
```

Description

Configuration structure for connection component. This must be populated by the application before calling `adi_ss_connection_init()` API.

Fields

- `eConnectionType`
  Type of the physical connection. The supported physical connections are defined by the `ADI_SS_CONNECTION_TYPE` enumeration
- `nDevId`
  Device number (id) of the physical connection.
- `eProcID`
  Processor ID on which the connection component must execute from.
- `pfISRRx`
  Rx interrupt service routine function pointer. Currently unused.
- `pfISRTx`
  Tx interrupt service routine function pointer. Currently unused.
- `pRxAddr`
  Payload receive address. This pointer is a reconfigurable parameter. It can be reconfigured and used along with `adi_ss_connection_Reconfigure()` API.

- `nRxPayloadCnt`
  Receive payload count in words. This count is a reconfigurable parameter. It can be reconfigured and used along with `adi_ss_connection_Reconfigure()` API.

- `pTxAddr`
  Payload transmit address. This pointer is a reconfigurable parameter. It can be reconfigured and used along with `adi_ss_connection_Reconfigure()` API.

- `nTxPayloadCnt`
  Receive payload count in words. This count is a reconfigurable parameter. It can be reconfigured and used along with `adi_ss_connection_Reconfigure()` API.

5.4.4 API enumerations and type defines

5.4.4.1 Communication component

5.4.4.1.1 ADI_SS_COMM_RESULT

```
typedef enum ADI_SS_COMM_RESULT
{
    ADI_SS_COMM_SUCCESS               = E_ADI_SS_COMM_SUCCESS,
    ADI_SS_COMM_FAILED                = E_ADI_SS_COMM_FAILED,
    ADI_SS_COMM_INSUFFICIENT_MEMORY   = E_ADI_SS_COMM_INSUFF_MEMORY,
    ADI_SS_COMM_CRC_ERROR             = E_ADI_SS_COMM_CRC_ERROR
}ADI_SS_COMM_RESULT;
```

Description

Enumeration for communication result type.

5.4.4.1.2 ADI_COMM_PROPERTY_ID

```
typedef enum ADI_COMM_PROPERTY_ID
{
  ADI_COMM_PROP_SSN_HANDLE
} ADI_COMM_PROPERTY_ID;
```

Description

Enumeration for communication result type

5.4.4.1.3 ADI_SS_COMM_HANDLE

```
typedef void *ADI_SS_COMM_HANDLE;
```

Description

Handle to the communication component.

5.4.4.1.4 ADI_SS_COMM_APP_ISR_CB

```
typedef void (*ADI_SS_COMM_APP_ISR_CB)(ADI_SS_COMM_HANDLE hSSComm);
```

Description

Type define for application ISR callback.

5.4.4.1.5 ADI_SS_COMM_CMD4_CB

```
typedef void (*ADI_SS_COMM_CMD4_CB)();
```

Description

Type define for CMD4 callback.

5.4.4.1.6 ADI_SS_COMM_SMAP_CB

```
typedef void (*ADI_SS_COMM_SMAP_CB)(ADI_SS_PROC_ID eProcID);
```

Description

Type define for callback on receiving SMAP.

5.4.4.2 Connection component

5.4.4.2.1 ADI_SS_CONNECTION_RESULT

```
typedef enum ADI_SS_CONNECTION_RESULT
{
    ADI_SS_CONNECTION_SUCCESS      = E_ADI_SS_CONNECTION_SUCCESS,
    ADI_SS_CONNECTION_FAILED       = E_ADI_SS_CONNECTION_FAILED,
    ADI_SS_CONNECTION_INSUFF_MEM   = E_ADI_SS_CONNECTION_INSUFF_MEM,
    ADI_SS_CONNECTION_INVALID_CONFIG  = E_ADI_SS_CONNECTION_INVALID_CONFIG
}ADI_SS_CONNECTION_RESULT;
```

Description

Enumeration for connection result type.

5.4.4.2.2 ADI_SS_CONNECTION_TYPE

```
typedef enum ADI_SS_CONNECTION_TYPE
{
    ADI_SS_CONNECTION_SPI = E_ADI_SS_CONNECTION_SPI
}ADI_SS_CONNECTION_TYPE;
```

Description

Enumeration for type of connection.

5.4.4.2.3 ADI_SS_CONNECTION_CONFIG_ITEM

```
typedef enum ADI_SS_CONNECTION_CONFIG_ITEM
{
    ADI_SS_CONFIG_SPI_RX_DMA = E_ADI_SS_CONFIG_SPI_RX_DMA,
    ADI_SS_CONFIG_SPI_TX_DMA = E_ADI_SS_CONFIG_SPI_TX_DMA
}ADI_SS_CONNECTION_CONFIG_ITEM;
```

Description

Enumeration for indicating which item is to be reconfigured using the `adi_ss_connection_Reconfigure()` API.

5.4.4.2.4 ADI_SS_CONNECTION_HANDLE

```
typedef void* ADI_SS_CONNECTION_HANDLE;
```

Description

Handle type to connection component.

5.4.4.2.5 CONN_ISR

```
typedef void* CONN_ISR;
```

Description

Typedefine for connection ISR type.

5.5 Target Framework

This section briefs about the implementation and configuration details of different components of the SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Framework. Refer **[2]** for Target Framework API functions, Data Structures and Enumerations.

5.5.1 Target Framework Architecture

The SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Framework is as shown in Figure 6. The six components of the Target Framework architecture are:

1. Audio Control Framework
2. Audio Process Framework
3. Connection
4. Communication
5. System
6. IPC

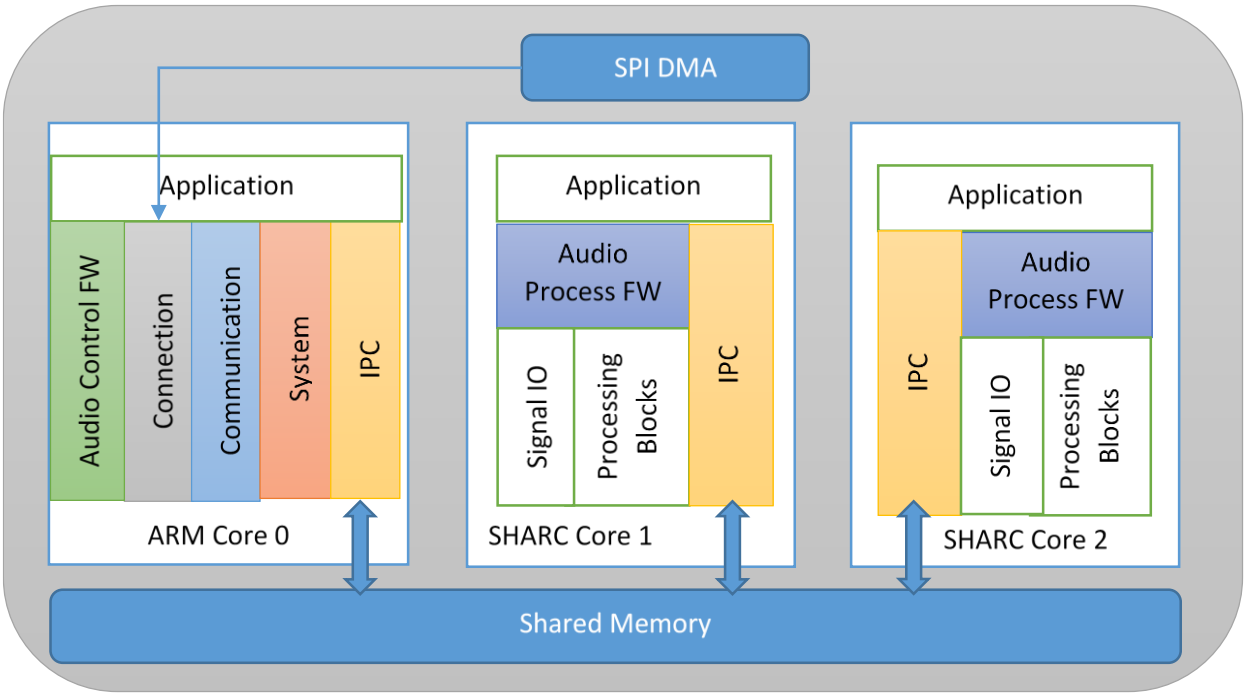


Figure 6: SigmaStudio for SHARC (ADSP-SC5xx) Target Framework

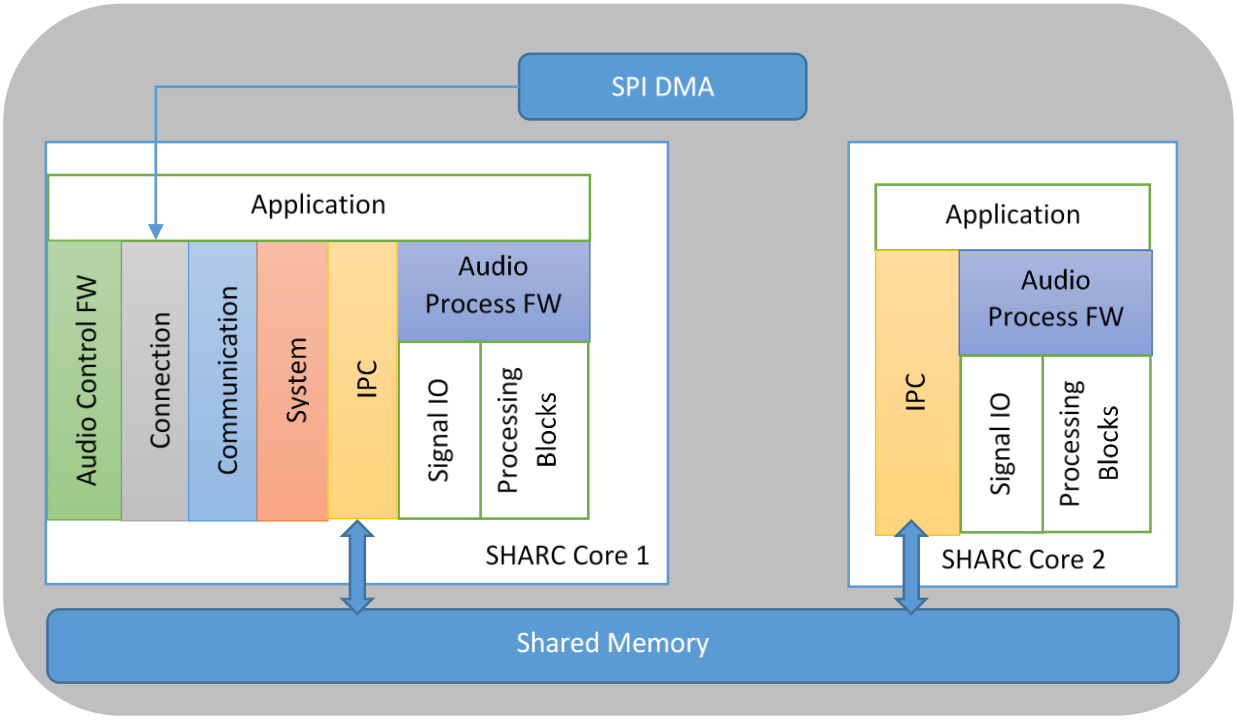


Figure 7: SigmaStudio for SHARC (ADSP-2157x/ADSP-2158x/ADSP-2159x) Target Framework

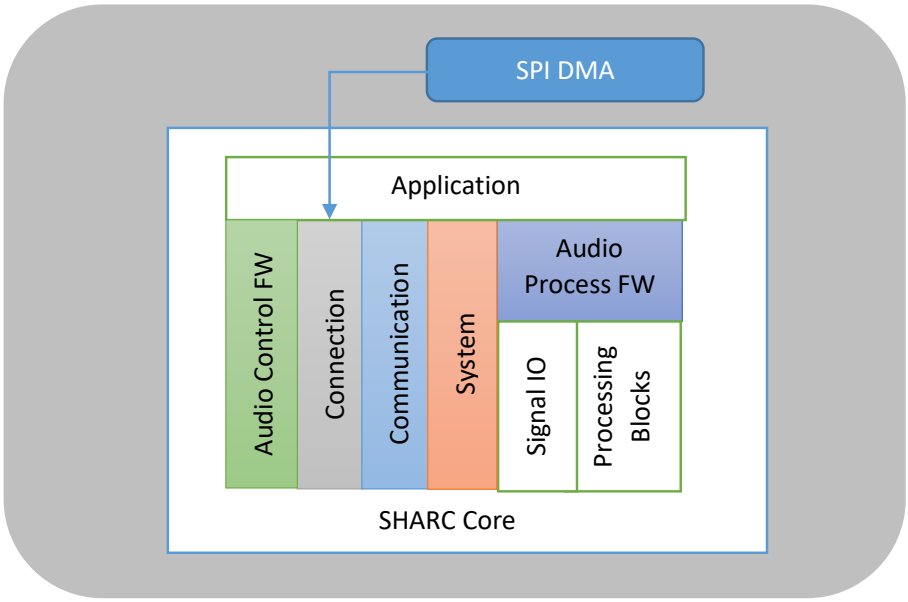


Figure 8: SigmaStudio for SHARC (ADSP-2156x) Target Framework

5.5.1.1 Audio Control Framework

The Audio Control Framework component of the Target Framework is responsible for configuring the various audio peripherals of ADSP-SC5xx/ADSP-215xx processors, receiving the audio data and rendering it out after processing. It also handles i/o data buffering. In the provided demo code, Audio Control Framework executes out of the ARM core on ADSP-SC5xx processors while it executes out of SHARC core 1 on ADSP-215xx processors. Audio Control Framework component has two interface header files namely adi_ss_arm_fw.h and adi_ss_control_fw.h. adi_ss_arm_fw.h interface header file is used while Audio Control Framework executes out of ARM core of ADSP-SC5xx variants while adi_ss_control_fw.h interface header file is used while Audio Control Framework executes out of SHARC core of ADSP-215xx variant.

5.5.1.2 Audio Process Framework

The Audio Process Framework component of the Target Framework is responsible for format conversion and for processing the data as per the different algorithms in the schematic signal chain. The Audio Process Framework component runs on each of the SHARCX1 cores of ADSP-SC5xx and ADSP-215xx processors.

5.5.1.3 Connection

The Connection component of the Target Framework architecture is responsible for the physical connection with a host or microcontroller for communication. The connection component initializes the communication peripheral for data reception and transmission. SPI is used as the physical connection for communication with host/microcontroller.

5.5.1.4 Communication

The Communication component of the Target Framework architecture is responsible for the protocol parsing of the connection packets obtained from or being sent to the host/microcontroller. Refer to **[3]** for more information on SigmaStudio SPI communication protocol packetization.

5.5.1.5 System

The System component of the Target Framework architecture includes the usage of off chip resources. The following off chip resources are configured by the System component.

- ADC
- DAC
- GPIO
- Power

5.5.1.6 IPC

The IPC component of the Target Framework architecture is used for inter-core communication between ARM and SHARC cores of ADSP-SC5xx processor. IPC between cores is required for core synchronization as well as data exchange such as back channel data. IPC is accomplished

using message passing mechanism between cores using shared memory. L2 memory is used as shared memory. The shared memory for IPC will be allocated at compile time with in the LDF for each of the cores.

5.5.2 Target Framework Status

The SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Framework has five states and these states are core agnostic, meaning, it is ensured that status of both the cores are considered for the overall Target status. For example, if core 2 reaches an erred state then the overall target status reported by primary core is “error” even if its status is success. The states are explained below.

1. IDLE: The Target will be in this state upon hardware reset.
2. BOOTED: The Target reaches this state upon successful boot, system and IPC initialization.
3. INITED: The Target reaches this state upon successful initialization of connection, communication and Framework components.
4. PROCESSING: The Target reaches this state upon successful Framework initialization after SSn code and parameters have been downloaded.
5. ERROR: The Target ends up in this state if any of the APIs returns an error.

5.5.2.1 LED Pattern for indicating Target Framework Status

Six LEDs are used within the application to indicate the Target status. The LED pattern for the various states are described in the table below. The rest of the LED’s will be off.

Note:

1. ADSP-SC589 EZ-Board does not have on board LED’s and hence Target Framework and processing errors are not indicated through LED’s.
2. LED 15 of ADSP-SC573 EZ-Board will always be turned ‘On’ when USBi is connected to ADSP-SC573 EZ-Board and cannot be used as the GPIO pin used for LED 15 and SPI 1 are same. Hence, LED 16 is used instead of LED 15 to indicate the Target Framework and processing errors for ADSP-SC573 EZ-Board.

State	LED 10	LED 11	LED 12	LED 13	LED 14	LED 15
IDLE	Off	Off	Off	Off	Off	Off
BOOTED	On	On	Off	Off	Off	Off
INITED	On	On	On	Off	Off	Off
PROCESSING	On	On	On	On	Off	Off
ERROR	Off	Off	Off	Off	On	Off

Table 2: LED Pattern for indicating Target Framework Status

State	LED 9	LED 10	LED 7
IDLE	Off	Off	Off
BOOTED	On	Off	Off
INITED	On	On	Off
PROCESSING	On	On	Off
ERROR	Off	Off	On

Table 3: LED Pattern for indicating Target Framework Status for ADSP-2156x EZ-Board or SOMCRR

Note: ADSP-SC589 MINI board follows the same LED pattern for indicating target framework status, except that the LEDs 9, 10 and 7 are replaced with LED's 10, 11 and 12 respectively.

The LED patterns indication for ADSP-21593 and ADSP-SC594 is same as ADSP-21569 SOM CRR board.

5.5.2.2 LED Pattern for indicating non-terminal processing error

While processing, there can be non-terminal errors such as data clipping and NAN errors. These are indicated by the following LED patterns.

Processing errors	LED 10	LED 11	LED 12	LED 13	LED 14	LED 15
Data clipped while processing	On	On	On	On	On	Off
NAN occurred	On	On	On	On	Off	On (30 sec)
Buffer Overflow	On	On	On	Off	On	Off
Buffer Under flow	On	On	On	Off	Off	On

Table 4: LED pattern for indicating non-terminal processing error

Processing errors	LED 9	LED 10	LED 7
Data clipped while processing	On	On	On
NAN occurred	On	On	On (blinks 5sec)
Buffer Overflow	On	On (blinks 5sec)	Off
Buffer Under flow	On (blinks 5sec)	On	Off

Table 5: LED pattern for indicating non-terminal processing error for ADSP-2156x EZ-Board or SOMCRR

Note: ADSP-SC589 MINI board follows the same LED pattern for indicating target framework status, except that the LEDs 9, 10 and 7 are replaced with LED's 10, 11 and 12 respectively.

The LED patterns indication for ADSP-21593 and ADSP-SC594 is same as ADSP-21569 SOM CRR board.

5.5.3 Audio Input-Output Modes

This section describes the different types of inputs and outputs which are supported by SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Framework.

5.5.3.1 Analog\Digital Co-existence

Analog\Digital Co-existence Audio I/O mode supports Analog/Digital Inputs and Analog Outputs.

5.5.3.1.1 Routing scheme for ADSP-SC58x/ADSP-2158x/ADSP-SC59x/ADSP-2159x

The routing scheme for Analog\Digital Co-existence mode for ADSP-SC58x\ADSP-2158x\ADSP-SC59x\ADSP-2159x processors is illustrated in Figure 9.

The analog data path consisting of CODEC and SPORT's are configured in TDM mode. The digital S/PDIF path is in I2S mode. ASRC and PCG's are used by the framework to facilitate this configuration.

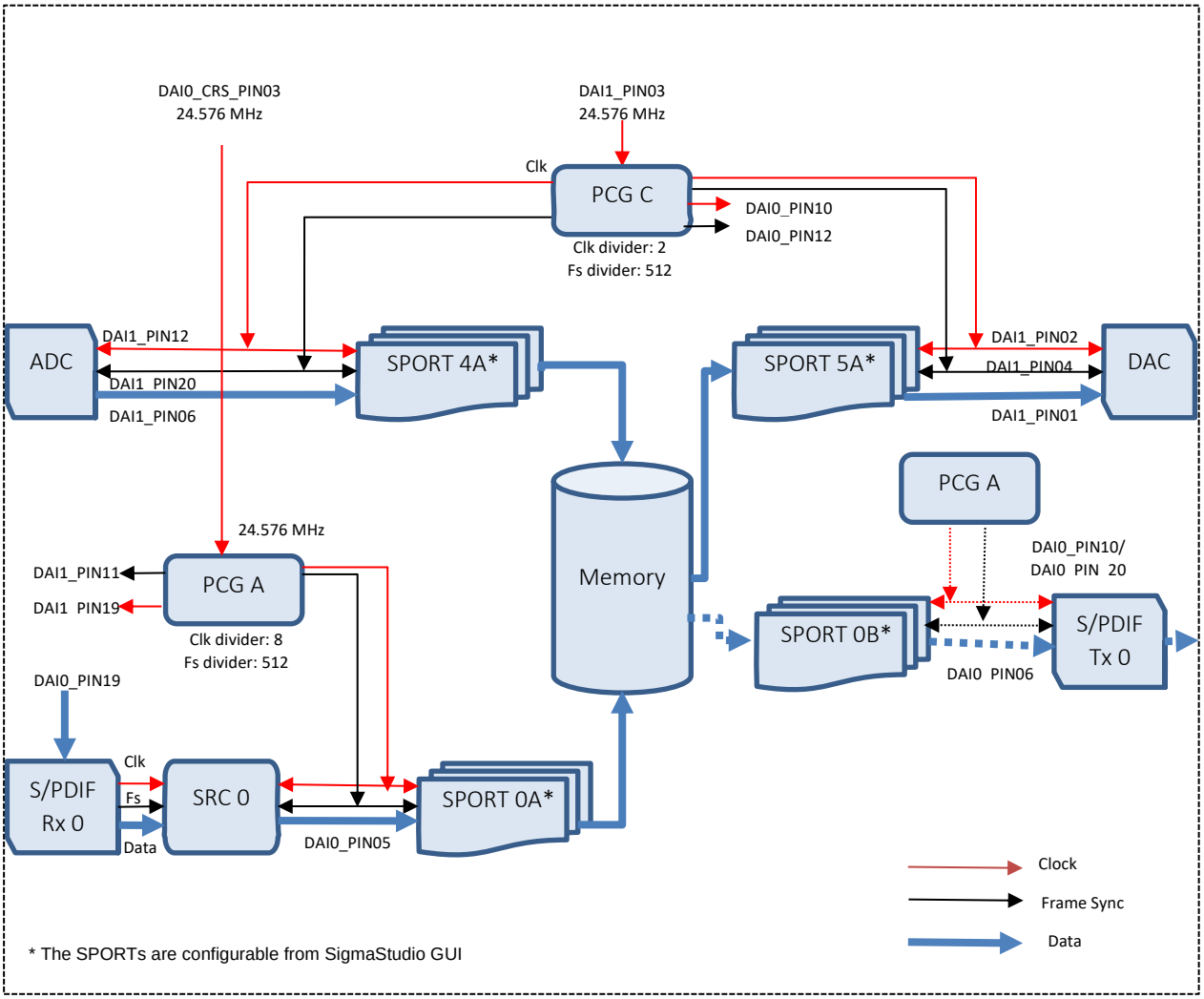


Figure 9: Default Audio I/O mode Configuration - Analog\Digital Co-existence for ADSP-SC58x\ADSP-2158x/ADSP-SC59x/ADSP-2159x

The master clock for PCG C is derived from 24.576 MHz which is available through DAI1_PIN03. The bit clock and frame sync for CODEC and the SPORTs are derived from PCG C using appropriate clock and frame sync dividers. By default, PCG C is configured to generate clock and frame sync signals for TDM8 configuration at 48 kHz sample rate. The clock and frame sync from PCG C are also available on DAI0_PIN10 and DAI0_PIN12 respectively. These pins are used for clocking the SPORTs of DAI0.

The master clock for PCG A is derived from 24.576 MHz which is available through DAI0_CRS_PIN03. The bit clock and frame sync for SRC 0 and SPORT used for obtaining the data from S/PDIF are derived from PCG A using appropriate clock and frame sync dividers. SRC 0 is used to de-jitter the S/PDIF recovered clock. By default, PCG A is configured to generate clock and frame sync signals for I2S configuration at 48 kHz sample rate. The clock and frame sync from

PCG A are also available on DAI1_PIN11 and DAI1_PIN19 respectively. These pins are used for clocking the SPORTs of DAI1.

The S/PDIF Tx feature can be enabled to ADSP-SC58x/ADSP-2158x/ADSP-SC59x/ADSP-2159x target application by referring section 10 of **[1]**. The S/PDIF Tx I/O mode configuration shown in dotted lines, refer Figure 9. The SPORT and S/PDIF Tx uses the bit clock and frame sync from PCG A for I2S configuration at 48 kHz sample rate. For ADSP-SC59x/ADSP-2159x, the SPORT data available in DAI0_PIN06 and the S/PDIF Tx Data available in DAI0_PIN10. For ADSP-SC58x/ADSP-2158x, the SPORT data available in DAI0_PIN06 and the S/PDIF Tx Data available in DAI0_PIN20.

5.5.3.1.2 Routing scheme for ADSP-SC57x\ADSP-2157x

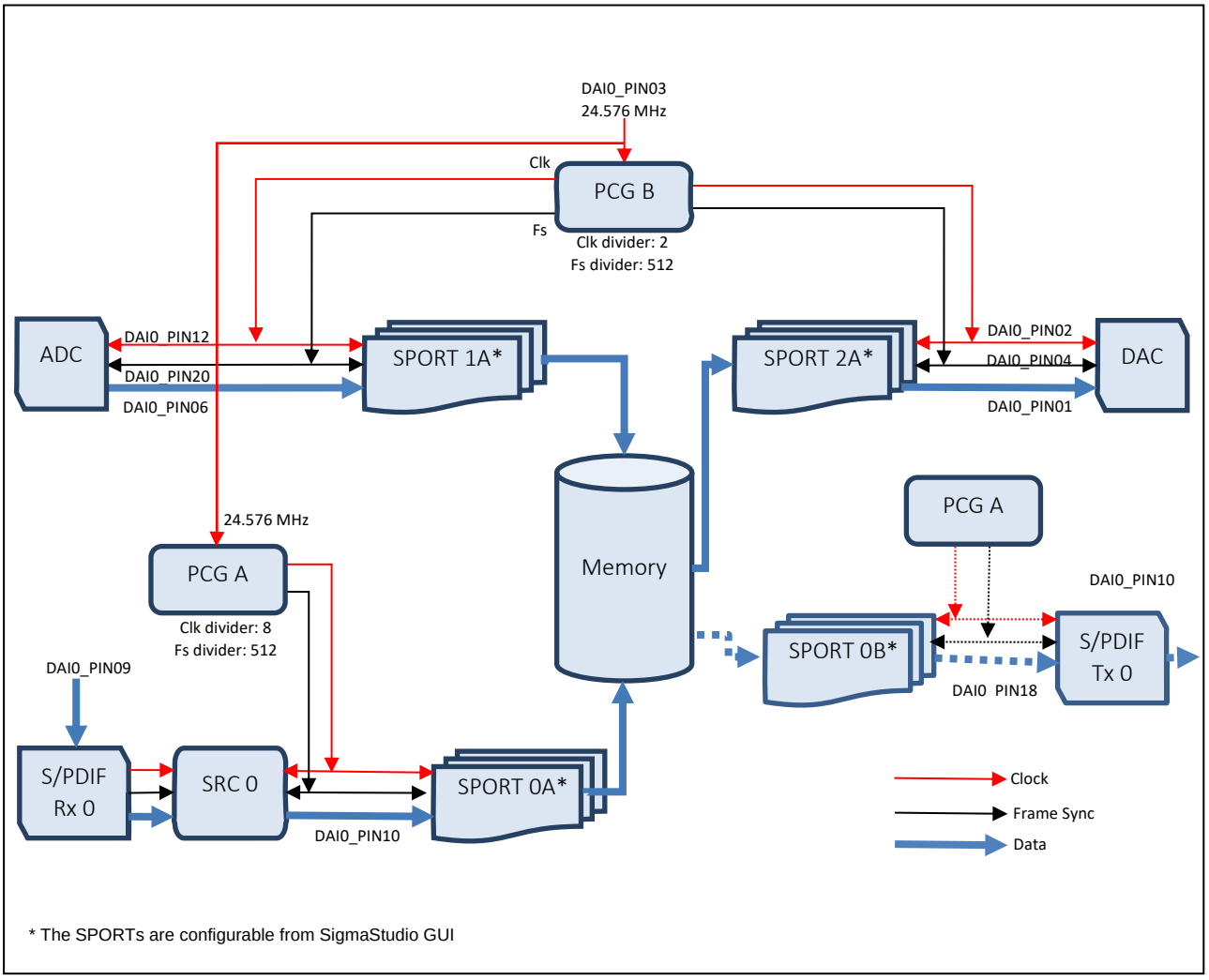


Figure 10: Default Audio I/O mode - Analog\Digital Co-existence for ADSP-SC57x\ADSP-2157x processors

The master clock for PCG A and PCG B is derived from 24.576 MHz which is available through DAI0_PIN03. By default, PCG B is configured to generate clock and frame sync signals for TDM8 configuration at 48 kHz sample rate and PCG A is configured to generate clock and frame sync signals for I2S configuration at 48 kHz sample rate.

The bit clock and frame sync for CODEC and SPORTs are derived from PCG B. The bit clock and frame sync for SRC 0 and SPORT used for obtaining the data from S/PDIF are derived from PCG A. SRC 0 is used to de-jitter the S/PDIF recovered clock.

The S/PDIF Tx feature can be enabled to ADSP-SC57x/ADSP-2157x target application by refereeing section 10 of **[1]**. The S/PDIF Tx I/O mode configuration shown in dotted lines refer Figure 10. The SPORT and S/PDIF Tx uses the bit clock and frame sync from PCG A for I2S configuration at 48 kHz sample rate. For ADSP-SC57x/ADSP-2157x, the SPORT data available in DAI0_PIN18 and the S/PDIF Tx Data available in DAI0_PIN10.

5.5.3.1.3 Routing scheme for ADSP-SC589 SAM

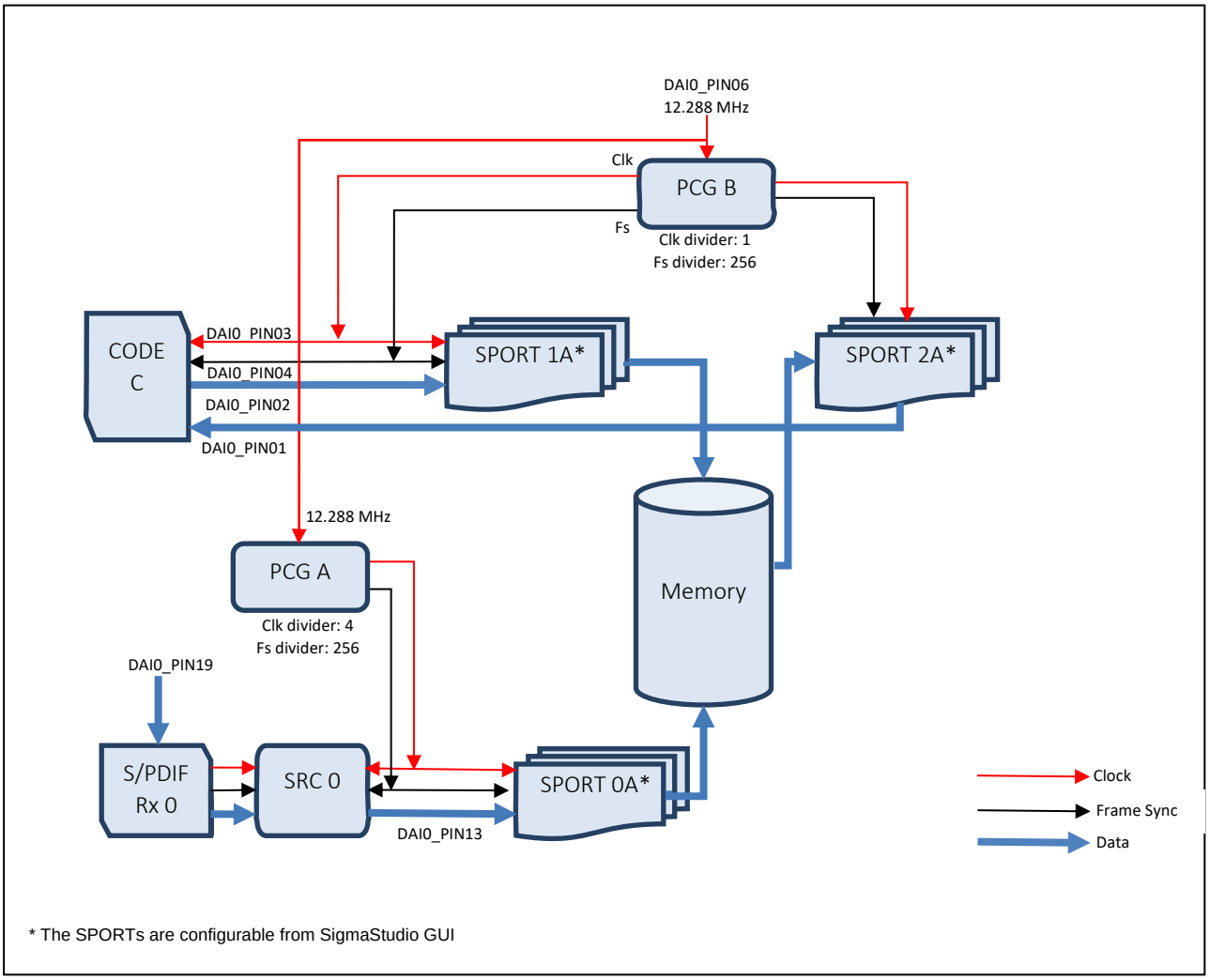


Figure 11: Default Audio I/O mode - Analog\Digital Co-existence for ADSP-SC589 SAM processors

The master clock for PCG A and PCG B is derived from 12.288 MHz which is available through DAI0_PIN06. By default, PCG B is configured to generate clock and frame sync signals for TDM8 configuration at 48 kHz sample rate and PCG A is configured to generate clock and frame sync signals for I2S configuration at 48 kHz sample rate.

The bit clock and frame sync for CODEC and SPORTs are derived from PCG B. The bit clock and frame sync for SRC 0 and SPORT used for obtaining the data from S/PDIF are derived from PCG A. SRC 0 is used to de-jitter the S/PDIF recovered clock.

Note: -
The S/PDIF Tx feature not supported for the ADSP-SC589 Mini application.

5.5.3.1.4 Routing scheme for ADSP-2156x

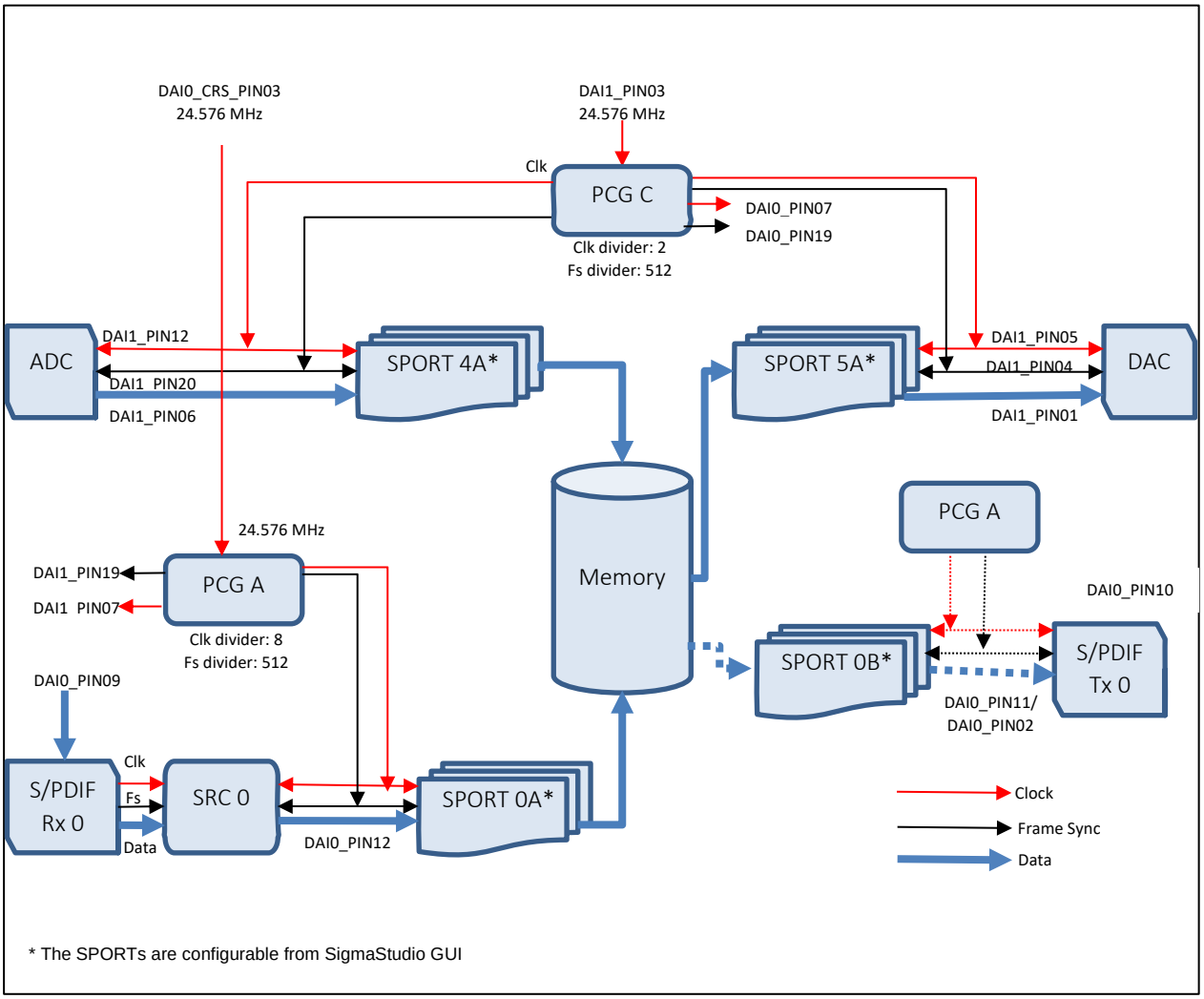


Figure 12: Default Audio I/O mode - Analog\Digital Co-existence for ADSP ADSP-2156x processors

For the ADSP-2156x processors, the default IO routing is as shown in Figure 9. The master clock for PCG C is derived from 24.576 MHz which is available through DAI1_PIN03. The bit clock and frame sync for CODEC and the SPORTs are derived from PCG C using appropriate clock and frame sync dividers. By default, PCG C is configured to generate clock and frame sync signals for TDM8 configuration at 48 kHz sample rate. The clock and frame sync from PCG C are also available on DAI0_PIN07 and DAI0_PIN19 respectively. These pins are used for clocking the SPORTs of DAI0.

The master clock for PCG A is derived from 24.576 MHz which is available through DAI0_CRS_PIN03. The bit clock and frame sync for SRC 0 and SPORT used for obtaining the data from S/PDIF are derived from PCG A using appropriate clock and frame sync dividers. SRC 0 is used to de-jitter the S/PDIF recovered clock. By default, PCG A is configured to generate clock and frame sync signals for I2S configuration at 48 kHz sample rate. The clock and frame sync from

PCG A are also available on DAI1_PIN07 and DAI1_PIN19 respectively. These pins are used for clocking the SPORTs of DAI1.

The S/PDIF Tx feature can be enabled to ADSP-2156x target application by referring section 10 of **[1]**. The S/PDIF Tx I/O mode configuration shown in dotted lines refer Figure 12. The SPORT and S/PDIF Tx uses the bit clock and frame sync from PCG A for I2S configuration at 48 kHz sample rate. For ADSP-21569 SOM-CRR EZ-KIT, the SPORT data available in DAI0_PIN11 and the S/PDIF Tx Data available in DAI0_PIN10. For ADSP-21569 EZ-KIT, the SPORT data available in DAI0_PIN02 and the S/PDIF Tx Data available in DAI0_PIN10.

5.5.4 Multi Core Processing and SSn Multi Instancing

SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Framework supports schematic processing on both SHARC cores. It also supports processing of multiple instances of SSn's per SHARC core.
Multi core processing is not supported by ADSP-2156x processors as it has a single SHARC core.

5.5.4.1 Multi Core Processing

Multi core processing involves executing SSn's on each of the SHARC cores of ADSP-SC5xx/ADSP-2157x/ADSP-2158x processors. Multi core processing is supported in two modes namely,

1. **Single signal chain within a schematic for each of the cores:** In this mode an IC in the 'Hardware configuration' tab of the GUI corresponds to an instance running on both the SHARC cores. Refer to section 5.6.5 for more information on this mode.

2. **Multiple signal chains within a schematic for each of the cores:** In this mode an IC in the 'Hardware configuration' tab of the GUI corresponds to an instance running on either of the SHARC cores. This is the default mode for Multi core processing.

The below sections describe the different ways of processing the input data using both the SHARC cores of ADSP-SC5xx processor in 'Multiple signal chain' mode.

5.5.4.1.1 Serial Data Operation from SHARC Cores

By default, both the SHARC cores of ADSP-SC5xx/ADSP-215xx processors process the input data serially i.e., the second SHARC core processes the output data of the first SHARC core.

5.5.4.1.2 Parallel Data Operation from SHARC Cores

SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Framework also support parallel data processing by SHARC cores of ADSP-SC5xx/ADSP-215xx processor. In this mode of operation, each of the SHARC cores process the same input data and produce mutually exclusive outputs.

Follow the steps below for modifying the demo Application to enable parallel data processing by SHARC cores

1. Open CrossCore Embedded Studio and browse the required Demo Application project from "<SigmaStudioForSHARC installation folder>\Target\Demo\ADSP-SC58x\" or "<SigmaStudioForSHARC installation folder>\Target\Demo\ADSP-SC57x\"
2. Change the processing mode on the input data by the SHARC cores from serial to parallel by changing the enumeration type of the SHARC framework configuration field 'eFwConfigShCoreProcessMode' from 'ADI_SS_SHCORE_PROCESSMODE_SERIAL' to 'ADI_SS_SHCORE_PROCESSMODE_PARALLEL'. This configuration field is set in function 'adi_ss_FW_Config()' in files "adi_ss_app_sh0.c" and "adi_ss_app_sh1.c"
3. Rebuild and run the applications.

Note that in 'Parallel Data Operation' from SHARC Cores, the output channels selected from SigmaStudio GUI for each of the cores must be mutually exclusive and the sum of the output channels across the cores must not be greater than 'ADI_SS_FW_MAX_NUM_OUT_CHANNELS'.

5.5.4.2 SSn Multi Instance

A maximum of 3 SSn instances are supported in each of the SHARC cores of ADSP-SC5xx processor. Both SHARC cores can have a single SSn instances or a combination of single/serial/ parallel instances in each of the SHARC cores. All instances have same Block Size and operates at a constant fixed Sampling Rate.

5.5.4.2.1 SSn Single Instance

This mode is for running a single SSn instance on the chosen SHARC core. This is the default 'Process Mode'.

5.5.4.2.2 SSn Serial Instance

This mode can be chosen by setting the 'Process Mode' field in the IC control window of all the instances running on a SHARC core to 'Serial' as shown in Figure 13. In serial mode of SSn's, the output of the first instance is the input to the subsequent SSn running on a given SHARC core.

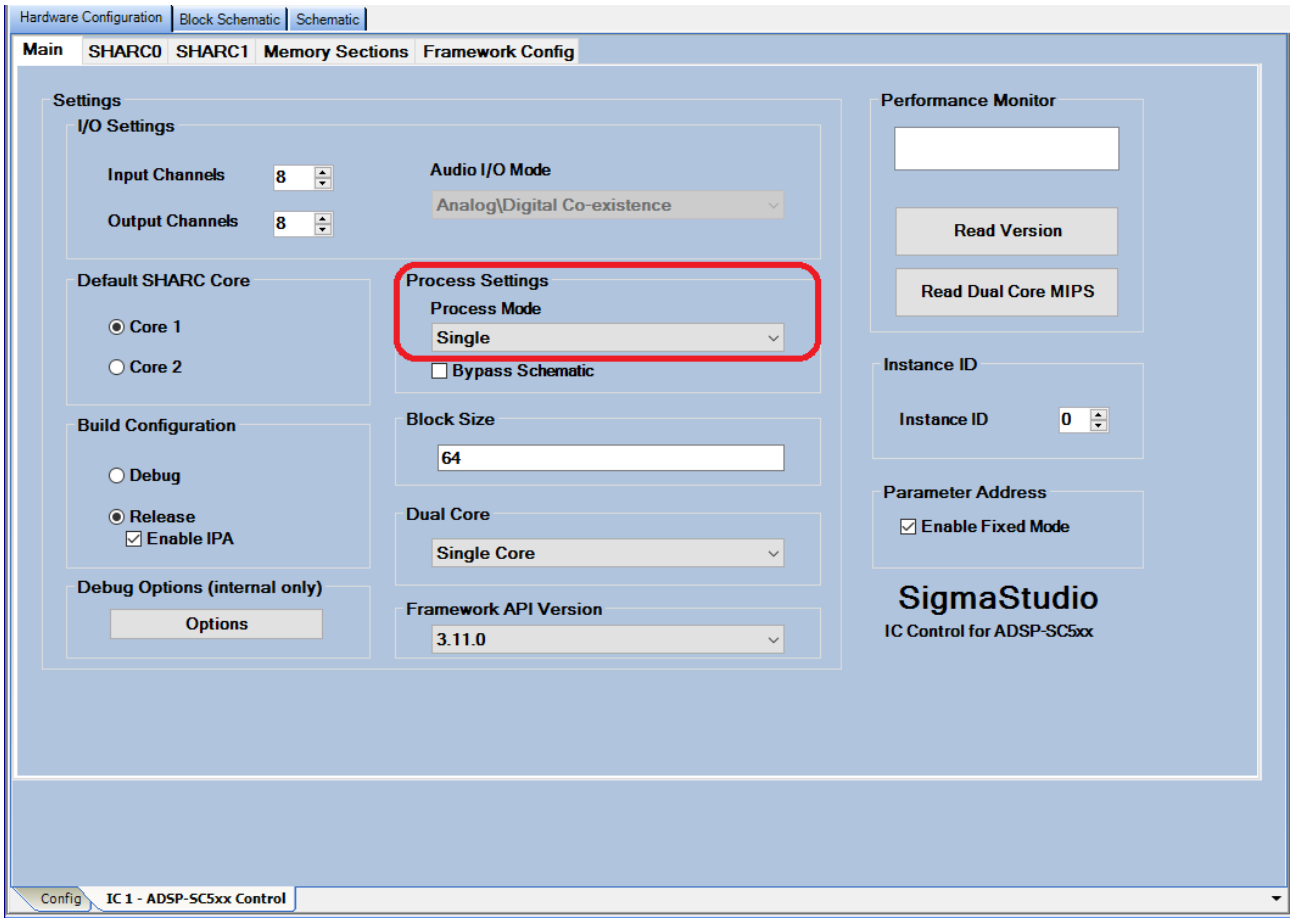


Figure 13: Process Mode - Serial

5.5.4.2.3 SSn Parallel Instance

This mode can be chosen by setting the 'Process Mode' field in the IC control window of all the instances running on a SHARC core to 'Parallel' as shown in Figure 14. In parallel mode of SSn's, the same input is fed to all parallel SSn instances. The output channels selected from SigmaStudio GUI for each of the instances must be mutually exclusive and the sum of the output channels across the instances and cores must not be greater than 'ADI_SS_FW_MAX_NUM_OUT_CHANNELS'.

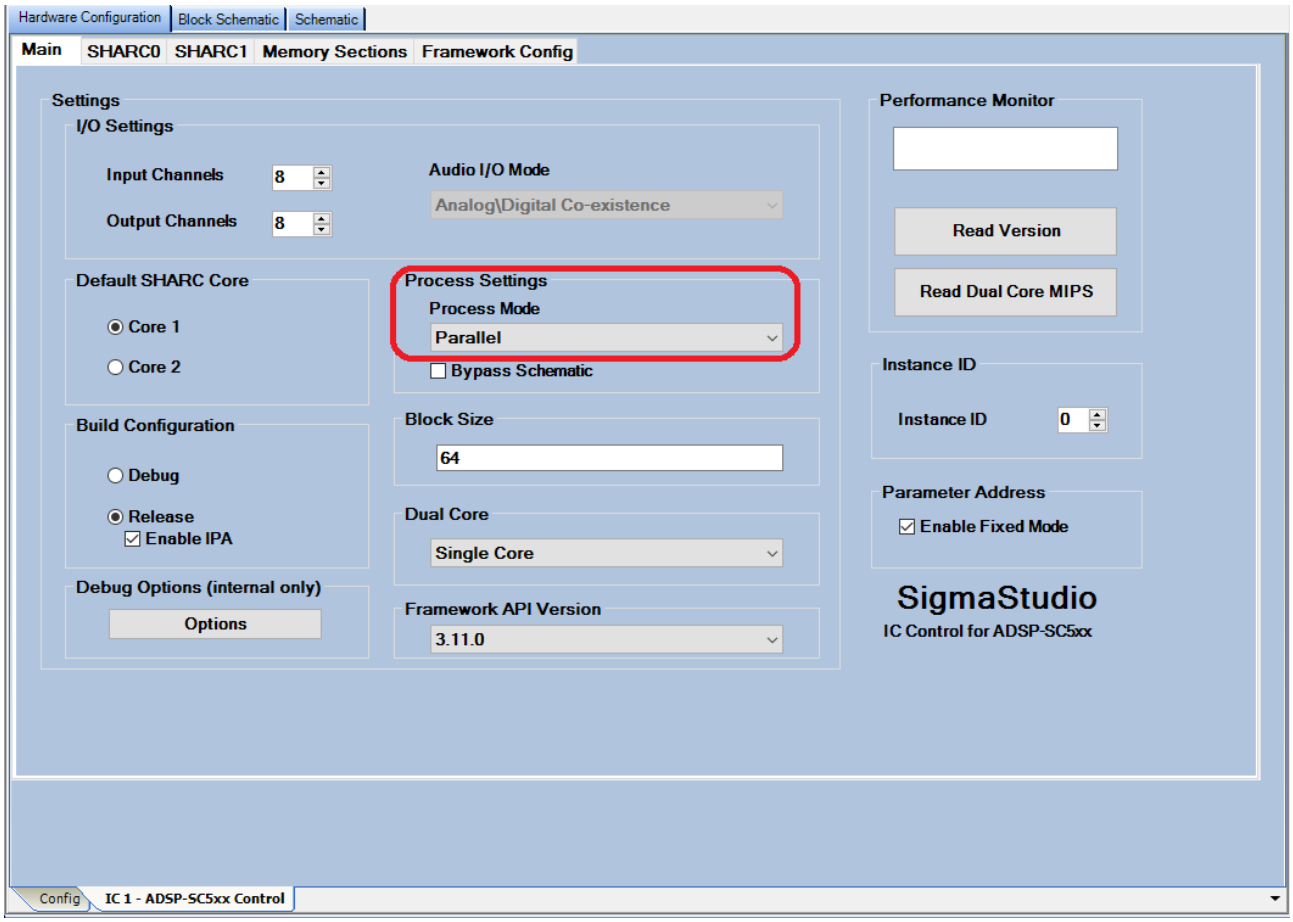


Figure 14: Process Mode - Parallel

5.5.5 Input/Output Buffers for Data Buffering

SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Target Framework uses the number of Input/Output buffers each of size “BlockSize” set in SigmaStudio GUI for input/output data buffering. This information is communicated to the SigmaStudio Target Framework via SMAP. By default, the number of Input/Output buffers is set to 3.

The minimum number of I/O buffers supported by the Framework is 2 and the maximum number of I/O buffers supported is 64. Refer to section 5.6.1.14.1.5 on how to set the number of I/O buffers in SigmaStudio GUI and on how to generate a config file and rebuild the target application with the updated number of I/O buffers.

5.5.6 Clocking Scheme

ADSP-SC58x and ADSP-SC57x processor clocks are configured for functional Mode 0.

Clock	Computation	Frequency
XTAL	= CLK_	25 MHz
CCLK0_0/1	= CLK_/CSEL	400 MHz
CCLK1_0/1	= CLK_/CSEL	400 MHz
SYSCLK_0/1	= CLK_/SYSSEL	200 MHz

Table 6: Processing Clocks

The clock sources for different peripherals in ADSP -SC58x and ADSP-2158x processor are as tabulated below:

Peripheral	Source	Frequency
SHARC-XI 0	CCLK0_0	400 MHz
SHARC-XI 1	CCLK0_0	400 MHz
ARM	CCLK1_0	400 MHz
SPORT BCLK(TDM8 Configuration)	PCG0_CLKC	12.288 MHz
SPORT FSCLK(TDM8 Configuration)	PCG_FSC	48 kHz
SPORT BCLK(I2S Configuration)	PCG0_CLKA	3.072 MHz
SPORT FSCLK(I2S Configuration)	PCG_FSA	48 kHz
L2	SYSCLK_0	200 MHz
ADC BCLK	PCG0_CLKC	12.288 MHz
ADC LRCLK	PCG_FSC	48 kHz
DAC BCLK	PCG0_CLKC	12.288 MHz
DAC LRCLK	PCG_FSC	48 kHz
DAI	SCLK0_0	100MHz
TWI	SCLK0_0	100 MHz
SPI1 TX	Host	385 kHz
SPI1 RX	Host	385 kHz

Table 7: Peripheral Clocks – ADSP-SC58x and ADSP-2158x

The clock sources for different peripherals in ADSP-SC57x and ADSP-2157x processor are as tabulated below:

Peripheral	Source	Frequency
SHARC-XI 0	CCLK0_0	400 MHz
SHARC-XI 1	CCLK0_0	400 MHz
ARM	CCLK1_0	400 MHz
SPORT BCLK(I2S Configuration)	PCG0_CLKA	3.072 MHz
SPORT FSCLK(I2S Configuration)	PCG_FSA	48 kHz
SPORT BCLK(TDM8 Configuration)	PCG0_CLKB	12.288 MHz
SPORT FSCLK(TDM8 Configuration)	PCG_FSB	48 kHz
L2	SYSCLK_0	200 MHz
ADC BCLK	PCG0_CLKB	12.288 MHz
ADC LRCLK	PCG_FSB	48 kHz
DAC BCLK	PCG0_CLKB	12.288 MHz
DAC LRCLK	PCG_FSB	48 kHz
DAI	SCLK0_0	100MHz
TWI	SCLK0_0	100 MHz
SPI1 TX	Host	385 kHz
SPI1 RX	Host	385 kHz

Table 8: Peripheral Clocks – ADSP-SC57x, ADSP-SC589 SAM and ADSP-2157x

The clock sources for different peripherals in ADSP -2156x processor are as tabulated below:

Peripheral	Source	Frequency
SHARC-XI 0	CCLK0_0	1000 MHz
SPORT BCLK(TDM8 Configuration)	PCG0_CLKC	12.288 MHz
SPORT FSCLK(TDM8 Configuration)	PCG_FSC	48 kHz
SPORT BCLK(I2S Configuration)	PCG0_CLKA	3.072 MHz
SPORT FSCLK(I2S Configuration)	PCG_FSA	48 kHz
L2	SYSCLK_0	500 MHz
ADC BCLK	PCG0_CLKC	12.288 MHz
ADC LRCLK	PCG_FSC	48 kHz
DAC BCLK	PCG0_CLKC	12.288 MHz
DAC LRCLK	PCG_FSC	48 kHz

DAI	SCLK0_0	125 MHz
TWI	SCLK0_0	125 MHz
SPI1 TX	Host	385 kHz
SPI1 RX	Host	385 kHz

Table 9: Peripheral Clocks – ADSP-2156x

ADSP-SC59x and ADSP-2159x processor clocks are configured for functional Mode 0.

Clock	Computation	Frequency
XTAL	= CLK_	25 MHz
CCLK0_0/1	= CLK_/CSEL	1 GHz
CCLK1_0/1	= CLK_/CSEL	1 GHz
SYSCLK_0/1	= CLK_/SYSSEL	500 MHz

Table 10: Processing Clocks

The clock sources for different peripherals in ADSP -SC59x and ADSP-2159x processor are as tabulated below:

Peripheral	Source	Frequency
SHARC-XI 0	CCLK0_0	1 GHz
SHARC-XI 1	CCLK0_0	1 GHz
ARM	CCLK1_0	1 GHz
SPORT BCLK(TDM8 Configuration)	PCG0_CLKC	12.288 MHz
SPORT FSCLK(TDM8 Configuration)	PCG_FSC	48 kHz
SPORT BCLK(I2S Configuration)	PCG0_CLKA	3.072 MHz
SPORT FSCLK(I2S Configuration)	PCG_FSA	48 kHz
L2	SYSCLK_0	500 MHz
ADC BCLK	PCG0_CLKC	12.288 MHz
ADC LRCLK	PCG_FSC	48 kHz
DAC BCLK	PCG0_CLKC	12.288 MHz
DAC LRCLK	PCG_FSC	48 kHz
DAI	SCLK0_0	125MHz
TWI	SCLK0_0	125 MHz
SPI1 TX	Host	385 kHz
SPI1 RX	Host	385 kHz

Table 11: Peripheral Clocks – ADSP-SC59x and ADSP-2159x

5.5.7 Microcontroller mode support

In microcontroller mode, the Target Framework shall process the incoming audio data based on the exported set of code and parameters from SigmaStudio. The ARM core of ADSP-SC5xx processor acts as a microcontroller, downloading the SMAP, SSn code and parameters on to the SHARC cores of ADSP-SC5xx.

A utility is provided with the package by which the exported code and parameters by SigmaStudio can be converted into a 'C' compiler readable form. This utility creates arrays of SMAP, SSn code and parameters as part of a 'C' file which can directly be included in the SigmaStudio ARM application for ADSP-SC5xx. Refer to "DemoUc application" and "Utility for formatting Exported data from SigmaStudio" sections of **[1]** for more information on the usage of ARM core of ADSP-SC5xx as microcontroller in the final deployment mode of the application developed using SigmaStudio.

5.5.8 Target framework default parameters

Some of the default Target Framework parameters are as below:

- Number of SSn Instances in each SHARC core of ADSP-SC5xx = 1
- Block Size = 64 samples,
- Sampling Rate = 48000 Hz,
- Number of input channels = 8,
- Number of output channels = 8,
- Number of input buffers = 3,
- Number of output buffers = 3,
- Audio I/O Mode = Analog\Digital Co-existence

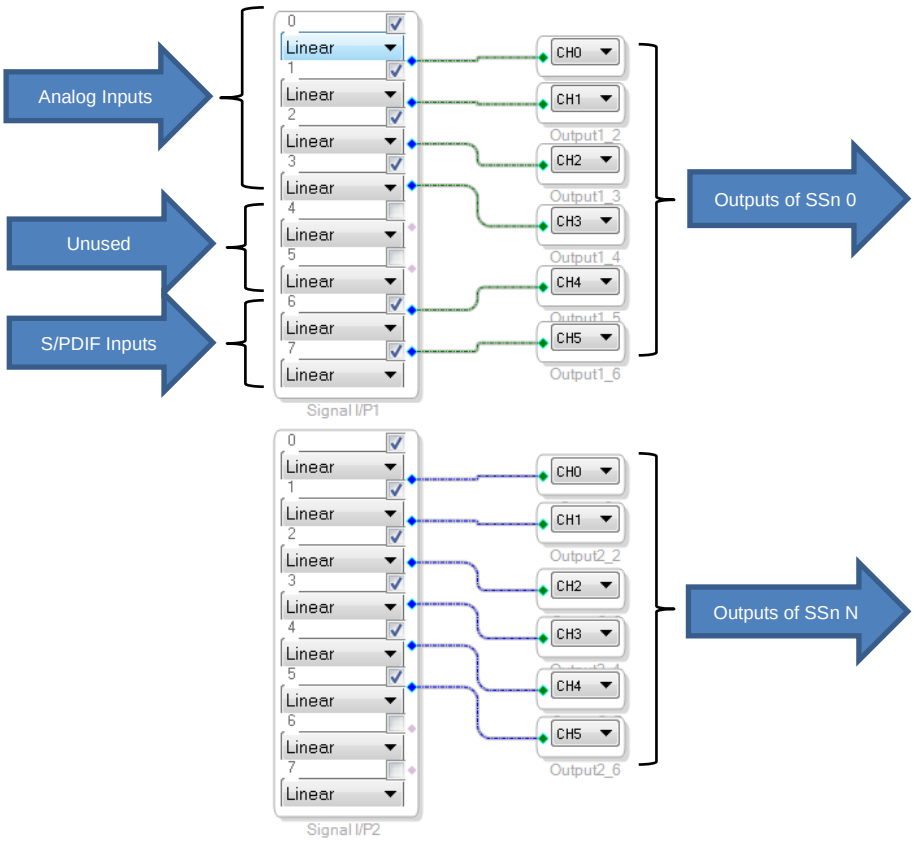


Figure 15: SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) inputs/outputs

5.6 SigmaStudio Host

This section briefs about the details of different components of the SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Host DLLs. SigmaStudio Host contains the following DLLs:

1. SharcPubLib.dll: has the SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Host Framework implementation.
2. SharcModules.dll: has the SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) module implementations.
3. SharcDesigner.dll: has the SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) Algorithm Designer implementation.

5.6.1 SigmaStudio IC Control Window

Figure 16 shows the 'Main Control Window' for the ADSP-SC5xx series of SHARC processors. This window can be launched by clicking on the 'IC x- ADSP-SC5xx Control' tab at the bottom of the 'Hardware Configuration' window.

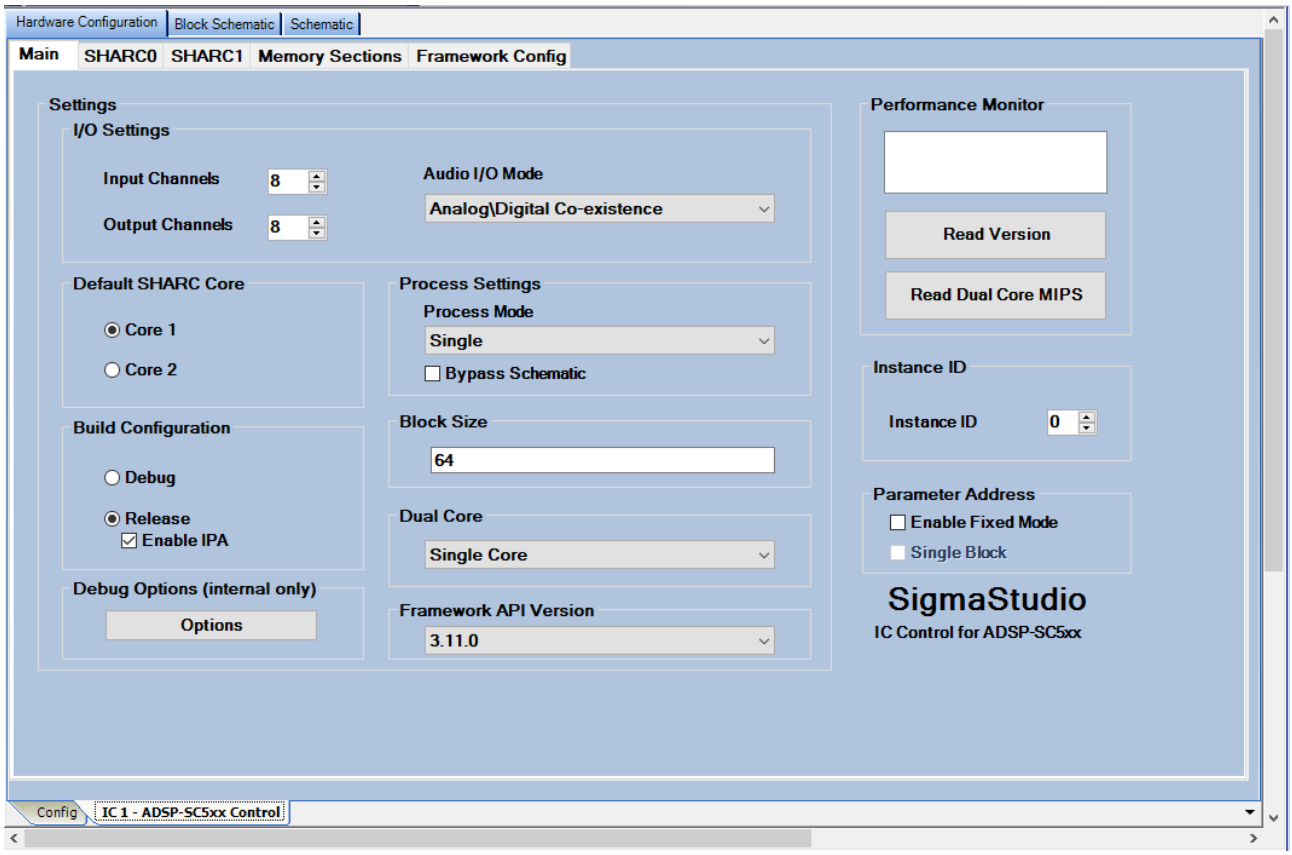


Figure 16: Main tab of ADSP-SC5xx IC Control Window

5.6.1.1 I/O Settings

5.6.1.1.1 Input Channels

This field indicates the maximum number of input audio channels in the Schematic. This number should match the number of input channels used in the Application. The minimum and maximum supported values of this field are 2 and 128. Usage of this field is outlined below.

- When an *Input Cell/Signal Input* Cell is inserted in the Schematic, it has as many channels as the value of this field.
- The value of this field (number of input channels) can be increased or decreased after inserting the input Cell, but it cannot be decreased below the number of input channels used in the Schematic.
- After modifying the value of this field, the *Input/Signal Input* Cell can be grown or reduced to match the number of channels to the value of the field. To grow/reduce the *Input Cell/Signal Input* Cell, right-click over the *Input Cell/Signal Input* Cell and select Growth or Reduce. *Note: If the number of channels is matching with that configured in the IC Control Window, the Growth or Reduce option won't be visible.*

For running the Default Application, this value should be set to 8.

5.6.1.1.2 Output Channels

This field indicates the maximum number of output audio channels in the Schematic. This number should match the number of output channels used in the Application. The minimum and maximum supported values of this field are 2 and 128. The following explains the usage of this field.

- The maximum number of Output Cells that can be inserted on to the Schematic is the value of this field.
- The value of this field can be increased at any point but cannot be decreased below the highest Output Cell present in the Schematic.

For running the Default Application, this value should be set to 8.

5.6.1.1.3 Audio I/O Mode

This will set the input-output mode of the application. This is currently fixed to “Analog\Digital Co-existence” mode.

5.6.1.2 Default SHARC Core

This value determines whether the schematic should be processed on SHARC Core 1 (Core 1) or SHARC Core 2, (Core 2). This field is not applicable in case of a schematic with ‘Dual Core’ option set. Refer to section 5.6.1.6 for more details on ‘Dual Core’ option.

5.6.1.3 Build Configuration

This field is an input to the CCES compiler, indicating whether the SigmaStudio Host Code should be compiled in Debug or Release mode.

Note: When compiling after changing the build configuration, the user must use the Clean-Link-compile-Download option. This option is explained in section 5.6.8 .

5.6.1.3.1 Enable IPA

This will enable inter-procedural optimization in the CCES compiler. This can be enabled or disabled in the Release mode only. More information on this option is present in section 5.6.8.4 .

5.6.1.4 Process Settings

5.6.1.4.1 Process Mode

This field indicates as to how the signal block instances are connected and processed on the SHARC core. This field is not applicable in case of a dual-core schematic. The following Process Modes are supported:

1. Single
2. Serial
3. Parallel

Refer section 5.5.4.2 for more information.

5.6.1.4.2 Bypass Schematic

Enabling this checkbox, will bypass the schematic processing and copy the input to the respective outputs.

5.6.1.5 Set Block Size

This field indicates the Block Size of the Schematic. This is also used by the Target application for the setting the SPORT and interface buffers for the selected core. Note that for the default application, the Block Size should be the same for both the SHARC cores.

5.6.1.6 Dual Core

This field indicates whether the schematic is required to have a single signal chain for each of the SHARC cores or the schematic is required to have two independent signal chains for each of the SHARC cores.

5.6.1.7 Framework API version

This field is used to achieve framework backward compatibility. If the Framework version is selected as '3.10.0 or before', SPORT configuration tab will not be displayed, and sport parameters will not be passed in SMAP. Hence schematics will be compatible with target frameworks 3.10.0 and before. Schematics generated with older version of SS4G will have Framework API version as '3.10.0 or before' by default. These schematics can be made compatible with the latest target framework by selecting '3.11.0' as the Framework API version. Any new schematic created will have Framework API version as 3.11.0 by default.

Figure 17: Framework API version for Backward compatibility

Note: Since there are no Framework API updates as part of 4.7.0 Release, the Framework API version to be chosen for the latest framework is *3.11.0.*

5.6.1.8 Performance Monitor

5.6.1.8.1 Read Version

The Target Library Version for the selected core can be obtained by clicking the *'Read Version'* button.

Note: *'Read Version'* will be 4.7.0 for the current release.

5.6.1.8.2 Read Dual Core MIPS

Average MIPS for both cores can be obtained by clicking the *'Read Dual Core MIPS'*. Note that the MIPS displayed is inclusive of the MIPS consumed by the framework.

5.6.1.9 Instance ID

This field indicates the ID of the instance in a multi-instance schematic. There can be a maximum of 3 instances per SHARC core. Hence, this field accepts a minimum value of 0 and a maximum value of 2. Also, the instances are processed in the order of 0 to 2. Multiple instances of the same SHARC core cannot have the same instance ID. An error message pops up during compilation of the schematic in such a case. Note that this field is not applicable in case of a dual-core schematic since dual core mode is not supported in case of a multi-instance schematic.

5.6.1.10 Fixed Address mode

This field enables the fixed address mode in the schematic. Refer to section 5.6.6 for more details.

5.6.1.11 SHARC 0 Tab

Figure 18 shows the SHARC 0 Control Window tab in the ADSP-SC5xx IC control Window.

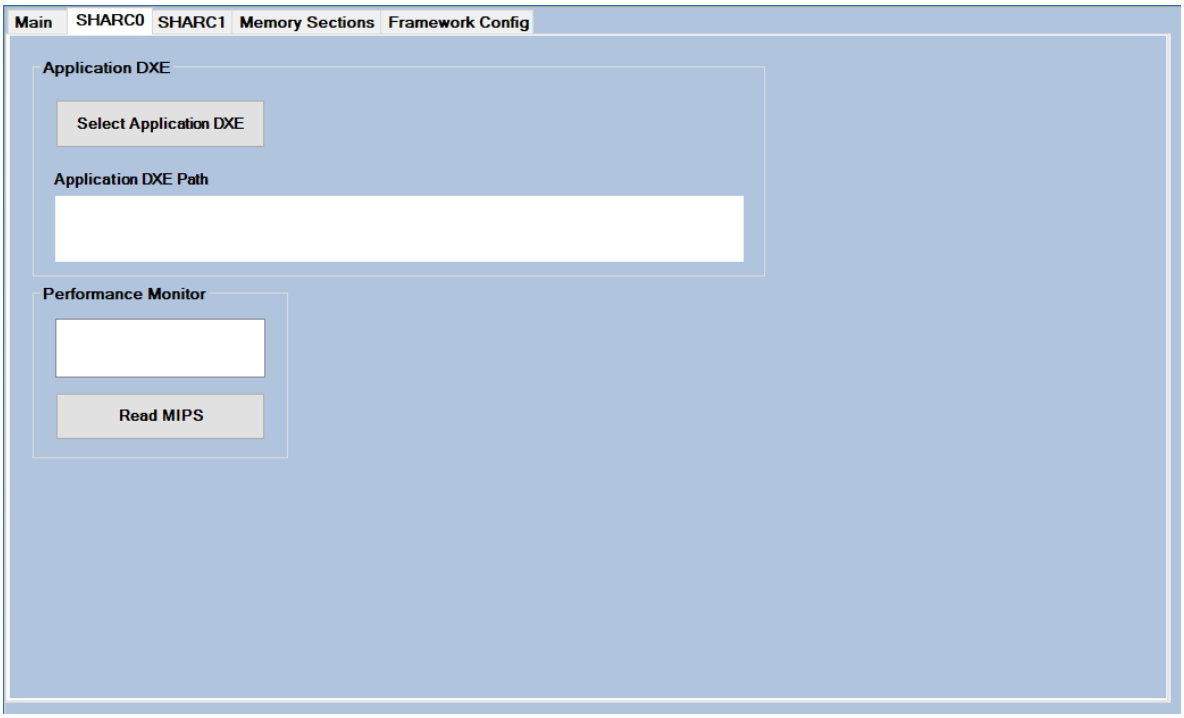


Figure 18: SHARC0 tab in ADSP-SC5xx IC Control Window

5.6.1.11.1 Application DXE

The Global Memory map (GMAP) contains information corresponding to available physical memory for the schematic (remaining unused memory on a per block basis after consumption by the application). GMAP is part of the application and this information is available in the Application DXE. The Application DXE can be loaded to the schematic using this *'Select Application DXE'* button for the selected core in the corresponding SHARC tab. When Core 0 is selected in the Main Control Window tab, the Application DXE for SHARC Core 0 can be loaded to the schematic using the '*Select Application DXE'* present in the SHARC 0 tab. When Core 1 is selected in the Main Control Window tab, the '*Select Application DXE'* button in the SHARC 0 tab will be disabled since it is not applicable. In case of a multi-instance schematic, the Application DXE needs to be loaded for the first instance only, i.e., only for the instance whose "*Instance ID*" is set to 0. Hence, the '*Select Application DXE'* button will be disabled for all instances other than the first instance. In other words, the '*Select Application DXE'* button will be disabled for all instances other than the instance for which the '*Instance ID'* is set as 0. However, if Dual Core mode is set to "Dual Core", the '*Select Application DXE'* button will be active on both the SHARC 0 and SHARC 1 tabs and the Application DXEs for SHARC 0 and SHARC 1 cores have to be selected using '*Select Application DXE'* buttons in SHARC 0 and SHARC 1 tabs.

The full path of the selected application DXE gets displayed in the "Application DXE Path" text box. The text "Invalid Path!!" is displayed if the path of the application DXE is invalid.

5.6.1.11.2 Performance Monitor

5.6.1.11.2.1 Read MIPS

MIPS for the SHARC core 0 can be obtained by clicking the *'Read MIPS'* button. In case of a multi-instance schematic, this MIPS indicates the MIPS of the individual instance running on SHARC core 0. When Core 1 is selected in the Main Control Window tab, the '*Read MIPS'* button in the SHARC 0 tab will be disabled since it is not applicable. However, if core mode is set to "Dual Core", the '*Read MIPS'* button will be active on both the SHARC 0 and SHARC 1 tabs.

5.6.1.12 SHARC 1 Tab

Figure 19 shows the SHARC 1 Control Window tab in the ADSP-SC5xx IC control Window.

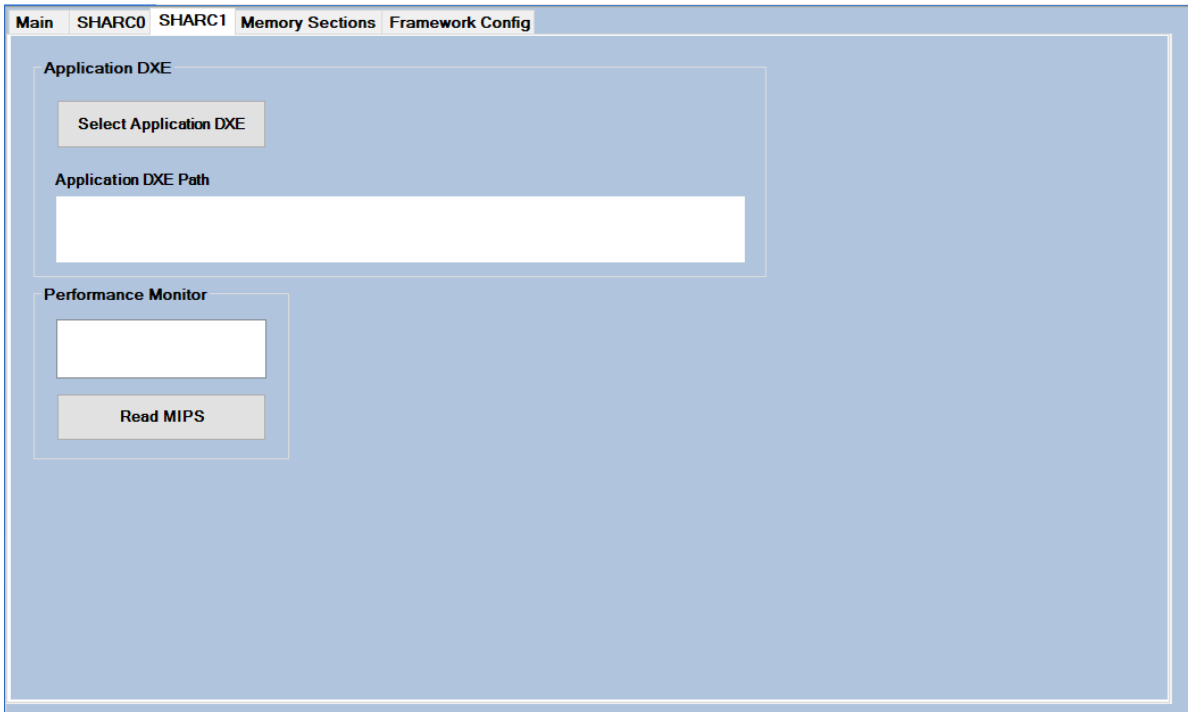


Figure 19: SHARC1 tab in ADSP-SC5xx IC Control Window

5.6.1.12.1 Application DXE

The Global Memory map (GMAP) contains information corresponding to available physical memory for the schematic (remaining unused memory on a per block basis after consumption by the application). GMAP is part of the application and this information is available in the Application DXE. The Application DXE can be loaded to the schematic using this *'Select Application DXE'* button for the selected core in the corresponding SHARC tab. When Core 1 is selected in the Main Control Window tab, the Application DXE for SHARC Core 1 has to be loaded to the schematic using the '*Select Application DXE'* present in the SHARC 1 tab. When Core 0 is selected in the Main Control Window tab, the '*Select Application DXE'* button in the SHARC 1 tab will be disabled

since it is not applicable. In case of a multi-instance schematic, the Application DXE needs to be loaded for the first instance only, i.e., only for the instance whose "*Instance ID*" is set to 0. Hence, the '*Select Application DXE'* button will be disabled for all instances other than the first instance. In other words, the '*Select Application DXE'* button will be disabled for all instances other than the instance for which the '*Instance ID'* is set as 0. However, if Dual Core mode is set to "Dual Core", the '*Select Application DXE'* button will be active on both the SHARC 0 and SHARC 1 tabs and the Application DXEs for SHARC 0 and SHARC 1 cores have to be selected using '*Select Application DXE'* buttons in SHARC 0 and SHARC 1 tabs.

The full path of the selected application DXE gets displayed in the "Application DXE Path" text box. The text "Invalid Path!!" is displayed if the path of the application DXE is invalid.

5.6.1.12.2 Performance Monitor

5.6.1.12.2.1 Read MIPS

MIPS for the SHARC core 1 can be obtained by clicking the *'Read MIPS'* button. In case of a multi-instance schematic, this MIPS indicates the MIPS of the individual instance running on SHARC core 1. When Core 0 is selected in the Main Control Window tab, the '*Read MIPS'* button in the SHARC 1 tab will be disabled since it is not applicable. However, if Dual Core mode is set to "Dual Core", the '*Read MIPS'* button will be active on both the SHARC 0 and SHARC 1 tabs.

5.6.1.13 Memory Sections Tab

This tab is used to map Code/ State/ Parameter sections to required buffers of SigmaStudio for SHARC.

5.6.1.13.1 Code section mapping

Figure 20 shows the Code sub-tab in Memory Sections tab in the ADSP-SC5xx IC control Window.

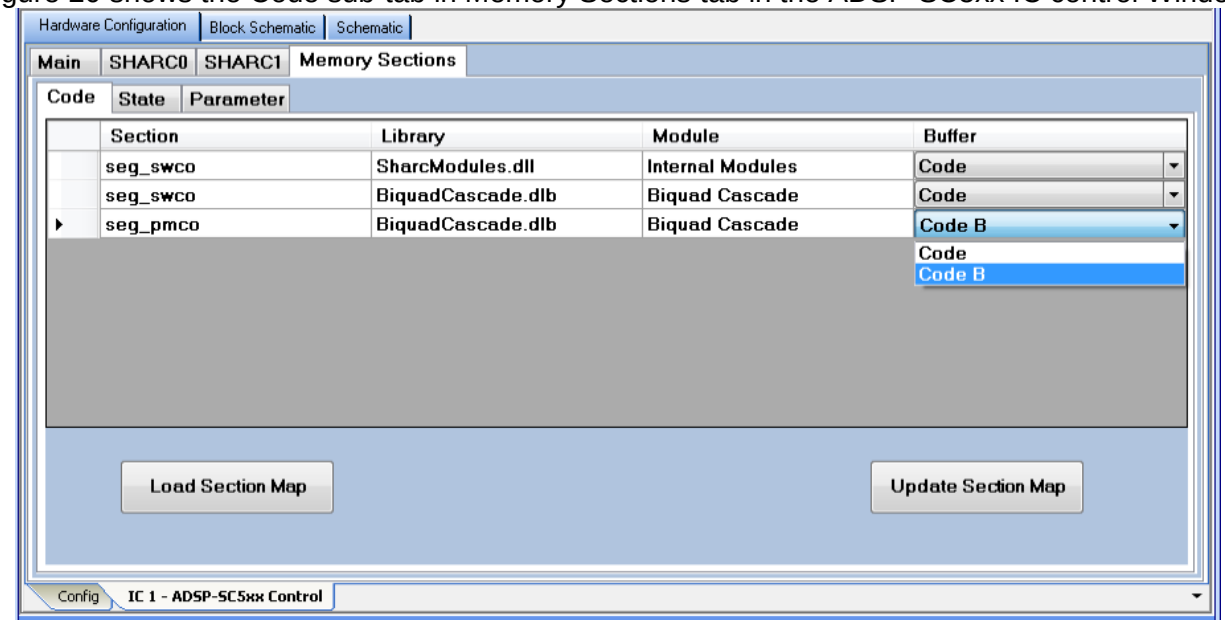


Figure 20 Code section mapping tab in ADSP-SC5xx IC Control Window

The input sections used in Plug-In modules can be mapped to SigmaStudio buffers using the 'Code' tab in 'Memory Sections' tab of 'IC Control Window'. There are two instruction buffers, Code and Code B, used by SigmaStudio for SHARC for storing schematic instructions. The input code sections in the Plug-In modules can be mapped to either 'Code' or 'Code B' buffers using a 'Code' tab. All the internal modules are mapped to the code section 'seg_swco' by default. This can also be mapped to Code/ Code B.

- 'Load Section Map' button is used to load the list of code sections used by modules in the schematic.
- 'Update Section Map' button is used to update the section mapping information in the schematic after selecting appropriate buffer for each section listed in the table. The section mapping will not get reflected in the code generation unless this button is pressed.
- When the same section is used in multiple modules or libraries, selecting a buffer for one of the section entries will update the buffer against all other entries of the same section.

5.6.1.13.2 State Section mapping

Figure 21 shows the State sub-tab in Memory Sections tab in the ADSP-SC5xx IC control Window.

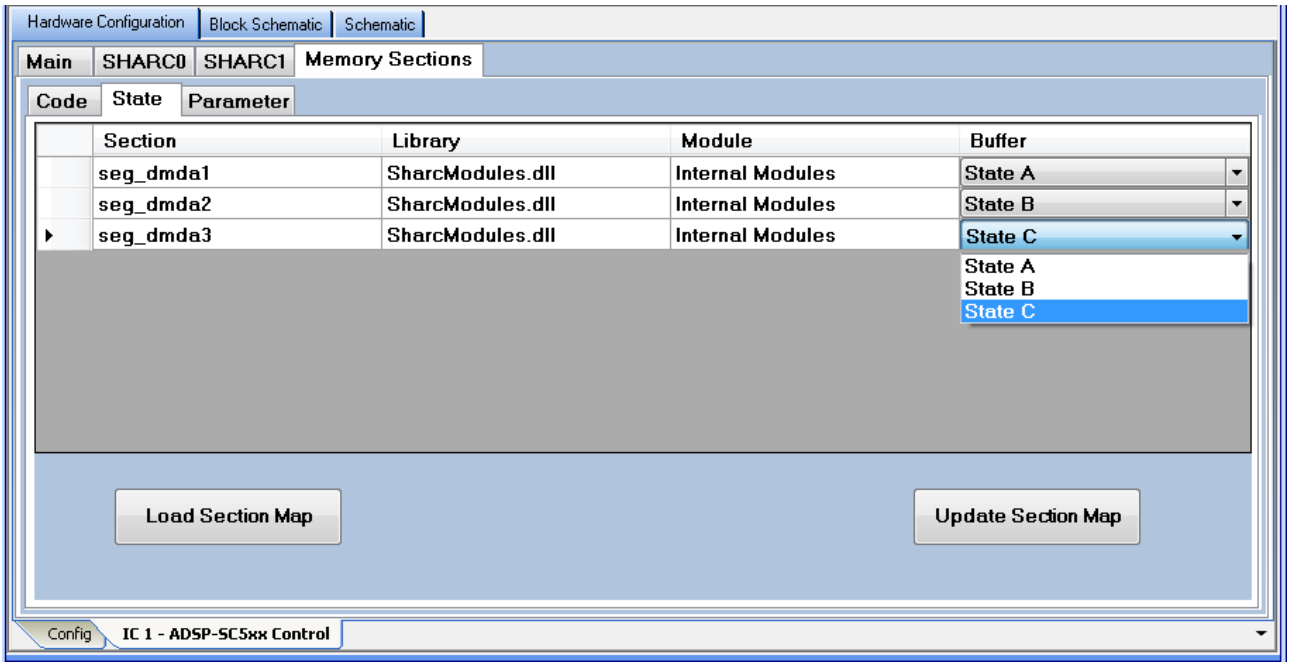


Figure 21 State section mapping tab in ADSP-SC5xx IC Control Window

The State memory used by the Internal Modules in the schematic can be mapped to 3 SigmaStudio state buffers using the 'State' tab in 'Memory Sections' window of 'IC Control Window'. There are 3 state buffers, State A, State B and State C, used by SigmaStudio for SHARC for storing schematic state information. The entire State used by all internal modules present in the schematic will be grouped in to 3 sections – seg_dmda1, seg_dmda2, seg_dmda3. These three sections can be mapped to any of the 3 State buffers: State A, State B and State C in the 'State' tab.

- 'Load Section Map' button displays the various state sections and the buffer to which they are mapped. By default, the sections are mapped to State A. They may be changed to State B or State C as per the requirement.
- 'Update Section Map' button updates the state section mapping information in the schematic after the user has selected appropriate buffer for each state section. The changed section mapping will not get reflected in the code generation unless this button is pressed.

5.6.1.13.3 Parameter section mapping

Figure 22 shows the Parameter sub-tab in Memory Sections tab in the ADSP-SC5xx IC control Window.

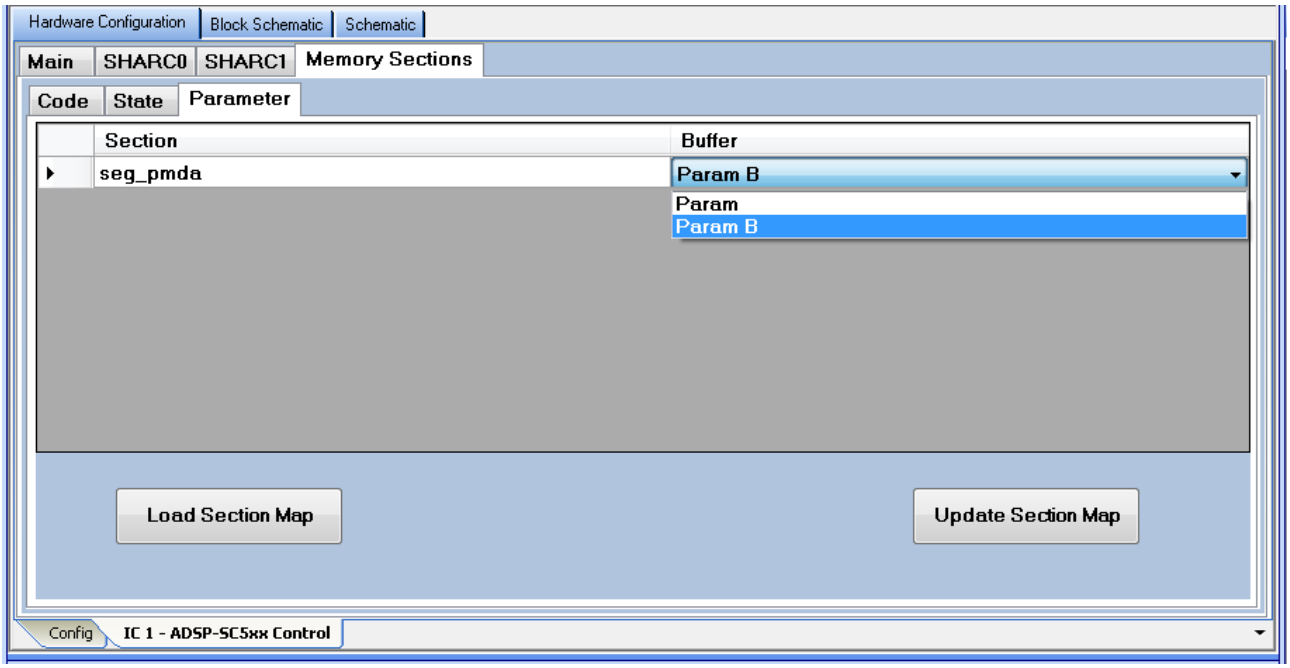


Figure 22 Parameter section mapping tab in ADSP-SC5xx IC Control Window

The Parameter memory used by all the modules in the schematic can be mapped to 2 SigmaStudio parameter buffers using the 'Parameter' tab in 'Memory Sections' window of 'IC Control Window'. There are 2 parameter buffers Param and Param B, used by SigmaStudio for SHARC for storing schematic parameter information. The entire Parameter buffer used by all modules present in the schematic will be mapped to the data section 'seg_pmda'. This 'seg_pmda' section can be mapped to any of the 2 Parameter buffers Param/Param B in the 'Parameter' tab.

- 'Load Section Map' button displays the parameter section and the buffer to which it is mapped. By default, the parameter section is mapped to 'Param'. It may be changed to 'Param B' as per the requirement.
- 'Update Section Map' button updates the parameter section mapping information in the schematic after the user has selected appropriate buffer for the section. The changed section mapping will not get reflected in the code generation unless this button is pressed.

5.6.1.14 Framework Config tab

This tab is used to configure framework parameters to be used in the SigmaStudio for SHARC target framework.

5.6.1.14.1 SPORTs configuration

Figure 23 shows the SPORTs sub-tab in Framework Config tab in the ADSP-SC5xx IC control Window. This can be used to select the number of sources and sinks used in the framework and

set the required SPORTs for them.

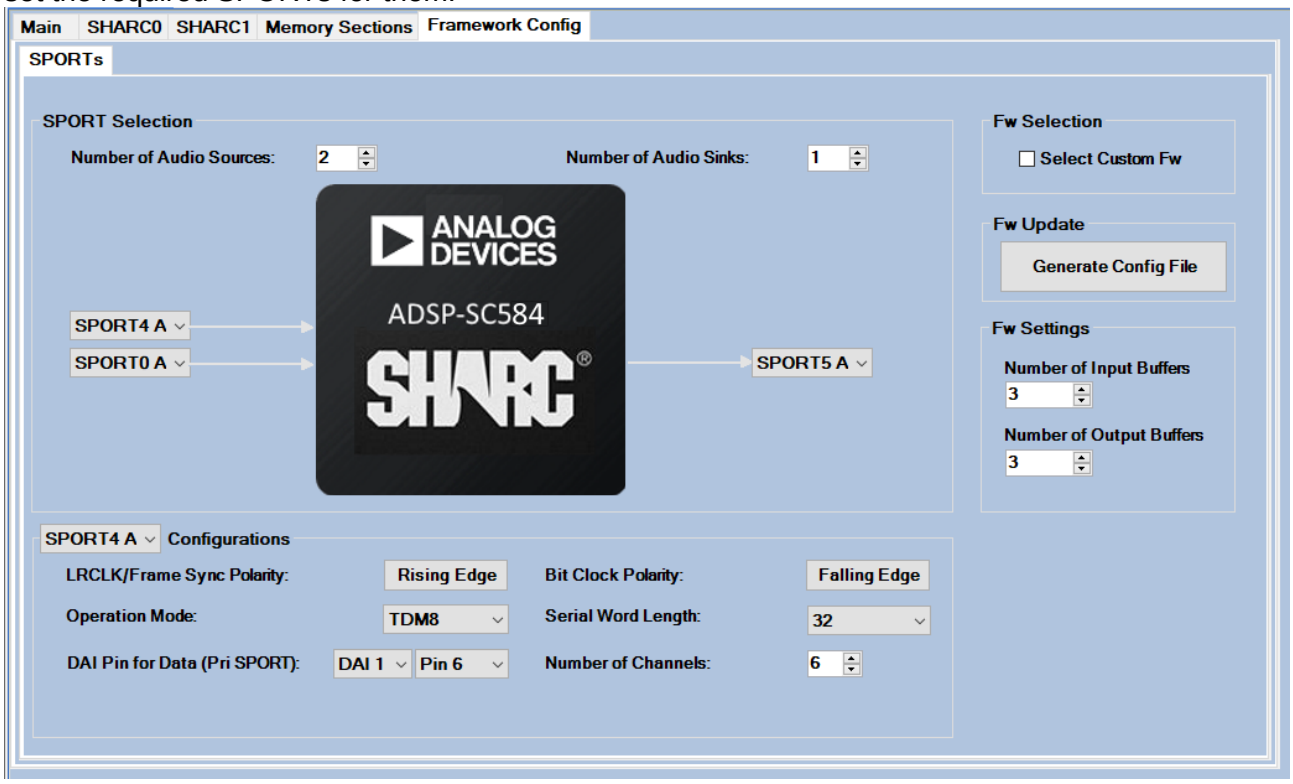


Figure 23: SPORT Configuration tab

5.6.1.14.1.1 SPORT Selection

5.6.1.14.1.1.1. Number of Audio Sources

This field is used to set the number of audio sources (input pins) from which audio data is obtained.

5.6.1.14.1.1.2. Number of Audio Sinks

This field is used to set the number of audio sinks (output pins) to which audio data is rendered.

Figure 24: Number of Sources and Sinks Selection

5.6.1.14.1.1.3. SPORT Input Selection

This field is used to set the SPORTs to be used as input for each of the audio sources in the system. The number of controls for SPORT input selection depends on the value set in the “Number of Audio Sources” field and appears as a graphical representation in the tab. The graphical representation gets updated dynamically for any change in the number of Audio Sources.

5.6.1.14.1.1.4. SPORT Output Selection

This field is used to set the SPORTs to be used as output for each of the audio sinks in the system. The number of controls for SPORT output selection depends on the value set in the “Number of Audio Sinks” field and appears as a graphical representation in the tab. The graphical representation gets updated dynamically for any change in the number of Audio Sinks.

Figure 25: SPORT Selection for Inputs and Outputs

5.6.1.14.1.2 SPORT Configuration

This field is used to select the SPORT (Input or Output) for which the configurations have to be set.

5.6.1.14.1.2.1. LRCLK/Frame Sync Polarity

This field is used to set the polarity of the Frame Sync signal. Frame Sync Polarity can be set to Rising Edge or Falling edge.

5.6.1.14.1.2.2. Bit Clock Polarity

This field is used to set the polarity of the Bit Clock. Bit Clock polarity can be set to Rising Edge or Falling edge.

5.6.1.14.1.2.3. Operation mode

This field is used to set the SPORT operation mode. SPORT operation mode can be set to I2S, TDM4, TDM8 or TDM16.

5.6.1.14.1.2.4. Serial Word length

This field is used to set the word length for SPORT data. 8, 16, 24 and 32 are available options for serial word length.

5.6.1.14.1.2.5. DAI Selection Primary SPORT

This field is used to select the DAI to be used for Primary SPORT Data.

5.6.1.14.1.2.6. DAI Pin Selection Primary SPORT

This field is used to select the DAI pin to be used for Primary SPORT Data.

5.6.1.14.1.2.7. Number of Channels

This field is used to set the number of channels for a particular SPORT operation mode.

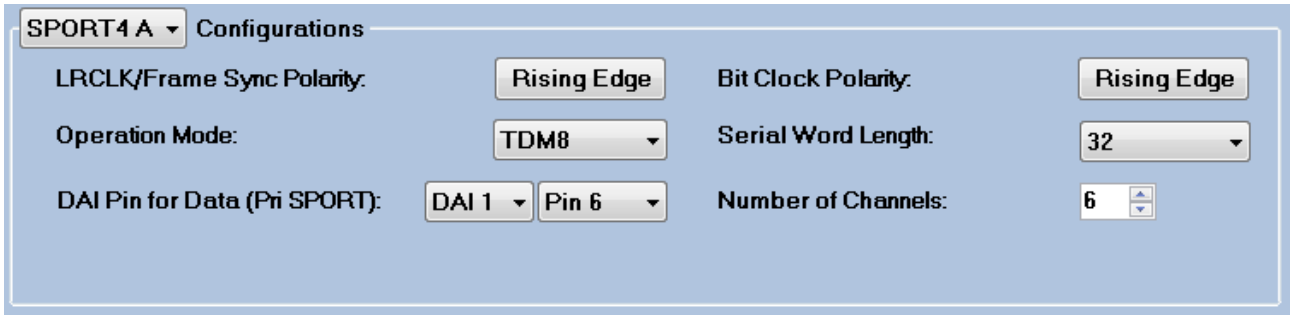


Figure 26: Configurations for selected Input or Output SPORT

5.6.1.14.1.3 Fw Selection

5.6.1.14.1.3.1. Select Custom Fw

This field must be enabled, if the user is planning to use a Custom target Framework instead of the default Framework available with the package. This disables the allocation of Framework buffers by the SigmaStudio Host, required for the default Framework.

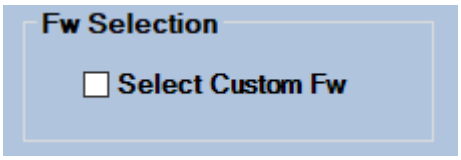


Figure 27: Custom Framework Selection option

5.6.1.14.1.4 Fw Update

5.6.1.14.1.4.1. Generate Config File

This field is used to generate a header file based on the Framework configuration set by the user in SigmaStudio Host. Memory requirements for ARM and SHARC target frameworks are also populated in the generated header file based on the Framework Settings. The Config file must be regenerated whenever the user changes any of the Framework settings mentioned below:

- Number of Audio Source/Sinks
- Number of Input/output Buffers
- Number of Input/output Channels
- Sport Operation Mode

The target framework must be rebuilt by including the regenerated Framework Config file, each time the Framework setting is modified.

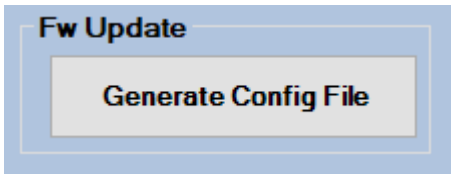


Figure 28: Config file Generation option

```
/*==============  D E F I N E S  ===============*/
#define ADI_SS_NUM_AUDIO_SOURCES             (2)
#define ADI_SS_NUM_AUDIO_SINKS               (1)
#define ADI_SS_TOTAL_IP_CHANNELS             (8)
#define ADI_SS_TOTAL_OP_CHANNELS             (8)
#define ADI_SS_MAX_CHANNELS_PER_PIN          (8)
#define ADI_SS_NUM_FW_BUFFERS                (3)
#define ADI_SS_CONTROL_FW_ARM_MEMORY         (10828)
#define ADI_SS_SHARC_FW_MEMORY               (6380)
#define ADI_SS_CONTROL_FW_SHARC_MEMORY       (15572)
#define ADI_SS_MAX_IP_CHANNELS_PIN0          (8)

#define ADI_SS_MAX_IP_CHANNELS_PIN1          (2)

#define ADI_SS_MAX_OP_CHANNELS_PIN0          (8)

#define ADI_SS_TOTAL_PHY_IP_CHANNELS        \
(ADI_SS_MAX_IP_CHANNELS_PIN0 +ADI_SS_MAX_IP_CHANNELS_PIN1)
#define ADI_SS_TOTAL_PHY_OP_CHANNELS      \
(ADI_SS_MAX_OP_CHANNELS_PIN0)
```

Config file generated for the default framework settings set within the 'Framework Config' tab is shown above. The default framework config file, 'adi_ss_fw_config.h' is available in "\Target\Framework\Include" folder. Given below is a description of each of the macros within the framework config file:

- ADI_SS_NUM_AUDIO_SOURCES: Corresponds to "Number of Audio Sources" selection in the host.
- ADI_SS_NUM_AUDIO_SINKS: Corresponds to "Number of Audio Sinks" selection in the host.
- ADI_SS_TOTAL_IP_CHANNELS: Corresponds to "Input Channels" selection in the host.
- ADI_SS_TOTAL_OP_CHANNELS: Corresponds to "Output Channels" selection in the host.
- ADI_SS_MAX_CHANNELS_PER_PIN: Corresponds to Maximum number of Channels per pin across all the Input and Output selection.
- ADI_SS_NUM_FW_BUFFERS: Corresponds to the Maximum of the "Number of Input Buffers" and "Number of Output Buffers" selection in the host.
- ADI_SS_CONTROL_FW_ARM_MEMORY: Corresponds to the memory requirement of the Audio Control framework when executed out of ARM core.
- ADI_SS_SHARC_FW_MEMORY**:** Corresponds to the memory requirement of the audio process framework that runs out of the SHARC core.
- ADI_SS_CONTROL_FW_SHARC_MEMORY: Corresponds to the memory requirement of the Audio Control framework when executed out of SHARC core.

- ADI_SS_MAX_IP_CHANNELS_PINx: Represents the maximum number of channels corresponding to each of the input pins, based on the SPORT Operation mode. Here 'x' varies from 0 to (Number of Audio Sources – 1).
- ADI_SS_MAX_OP_CHANNELS_PINx: Represents the maximum number of channels corresponding to each of the output pins, based on the SPORT Operation mode. Here 'x' varies from 0 to (Number of Audio Sinks – 1).
- ADI_SS_TOTAL_PHY_IP_CHANNELS: This macro represents the sum of ADI_SS_MAX_IP_CHANNELS_PINx, where 'x' varies from 0 to (Number of Audio Sources – 1).
- ADI_SS_TOTAL_PHY_OP_CHANNELS: This macro represents the sum of ADI_SS_MAX_OP_CHANNELS_PINx, where 'x' varies from 0 to (Number of Audio Sinks – 1).

Note: -
The "SOURCE_SPDIF" macro and the "nDataSink" variable in SHRAC core projects must be redefined when there is a change in the source or sink order from the default one for setting up "FIX2FLOAT_SHIFT_VAL_SPDIF" shift value.

5.6.1.14.1.5 Framework Settings

This field controls the number of I/O buffers each of size "BlockSize" to be used by the SigmaStudio Target Framework for input/output data buffering.

5.6.1.14.1.5.1. Number of Input Buffers

This field indicates the number of input buffers each of size "BlockSize" which the SigmaStudio Target Framework will use for input data buffering. By default, the number of Input buffers is set to 3. If a user enters a value which is more than the default value in SigmaStudio GUI, then the Audio control framework needs to be rebuilt after regenerating the framework config file as described in section 5.6.1.14.1.4.1

5.6.1.14.1.5.2. Number of Output Buffers

This field indicates the number of output buffers each of size "BlockSize" which the SigmaStudio Target Framework will use for output data buffering. By default, the number of Output buffers is set to 3. If a user enters a value which is more than the default value in SigmaStudio GUI, then the Audio control framework needs to be rebuilt after regenerating the framework config file as described in section 5.6.1.14.1.4.1

5.6.1.14.2 Locking the 'Framework Config' tab

The framework config tab can be locked which will prevent inadvertent changes to the framework configuration while working with the schematic. The framework config tab can be locked by right clicking within the tab and clicking 'Lock' context menu item as shown in Figure 29.

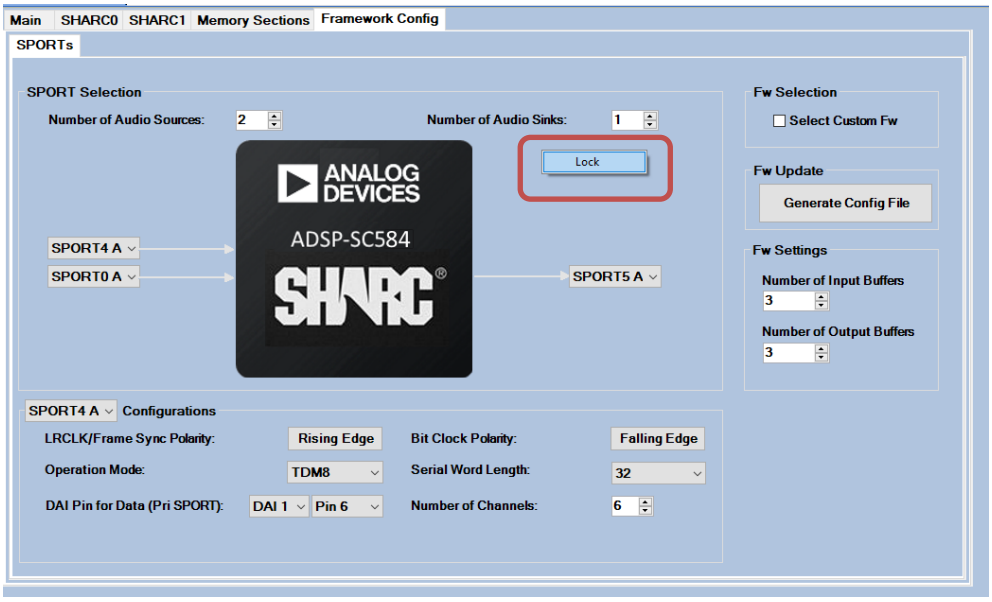


Figure 29: Locking the framework config tab

The ‘Framework Config’ tab can be unlocked by right clicking again within the tab and clicking ‘Unlock’ context menu item as shown in Figure 30.

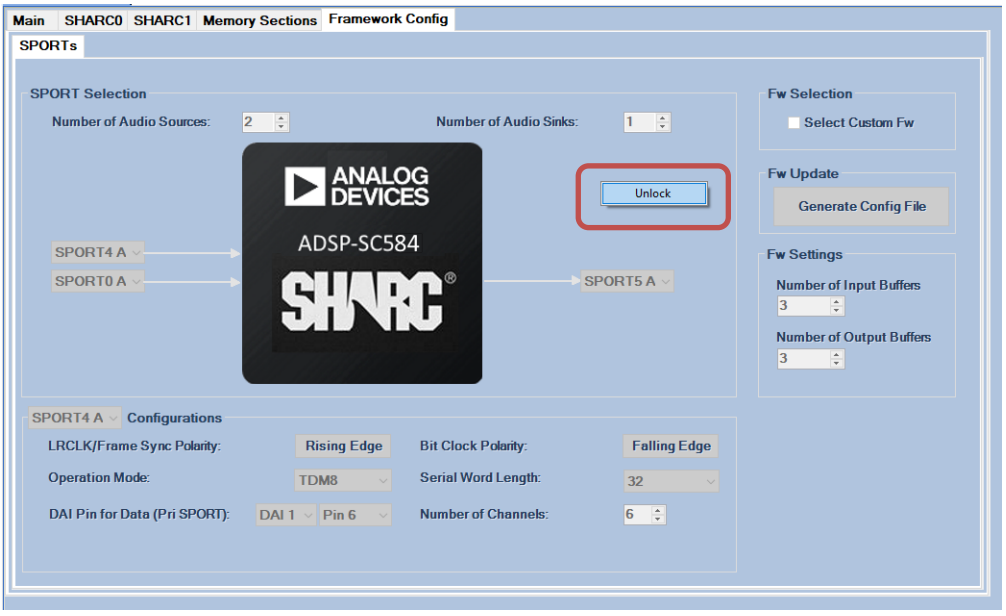


Figure 30: Unlocking the framework config tab

Note that it is not possible to load a different application dxe from within the ‘SHARC0’ and ‘SHARC1’ tab when the ‘Framework Config’ tab is locked.

5.6.1.14.3 Default SPORT Configuration

Default SPORT configurations are set when the DXE for a specific processor is selected. The default configurations for different processors are shown below.

Parameter		Default Value
Number of Audio Sources		2
Number of Audio Sinks		1
Source SPORTs		SPORT 4A (Connected to Analog Sources) SPORT 0A (Connected to SPDIF)
Sink SPORTs		SPORT 5A (Connected to Analog Sinks)
SPORT 4A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	TDM 8
	Serial Word Length	32 Bit
	DAI Pin for Primary Sport	DAI1_PIN06
	Number of Channels	6
SPORT 0A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	I2S
	Serial Word Length	24 Bit
	DAI Pin for Primary Sport	DAI0_PIN05 DAI0_PIN12 for ADSP-2156x
	Number of Channels	2
SPORT 5A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	TDM 8
	Serial Word Length	32 Bit
	DAI Pin for Primary Sport	DAI1_PIN01
	Number of Channels	8

Figure 31: Default SPORT Configuration ADSP-SC58x/ ADSP-SC59x /ADSP-2158x/ ADSP-2159x /ADSP-2156x processors

Parameter		Default Value
Number of Audio Sources		2
Number of Audio Sinks		1
Source SPORTs		SPORT 1A, SPORT 0A
Sink SPORTs		SPORT 2A
SPORT 1A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	TDM8
	Serial Word Length	32bit
	DAI Pin for Primary Sport	DAI0_PIN02
	Number of Channels	6
SPORT 0A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	I2S
	Serial Word Length	24 bit
	DAI Pin for Primary Sport	DAI0_PIN19
	Number of Channels	2
SPORT 2A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	TDM8
	Serial Word Length	32 Bit
	DAI Pin for Primary Sport	DAI0_PIN01
	Number of Channels	8

Figure 32: Default SPORT Configuration ADSP-SC589 SAM processor

Parameter	Default Value
Number of Audio Sources	2
Number of Audio Sinks	1
Source SPORTs	SPORT 1A, SPORT 0A

Sink SPORTs		SPORT 2A
SPORT 1A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	TDM8
	Serial Word Length	32bit
	DAI Pin for Primary Sport	DAI0_PIN06
	Number of Channels	6
SPORT 0A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	I2S
	Serial Word Length	24 bit
	DAI Pin for Primary Sport	DAI0_PIN10
	Number of Channels	2
SPORT 2A Configurations	LR Clock Polarity	Rising Edge
	Bit Clock Polarity	Falling Edge
	Operation Mode	TDM8
	Serial Word Length	32 Bit
	DAI Pin for Primary Sport	DAI0_PIN01
	Number of Channels	8

Figure 33: Default SPORT Configuration ADSP-SC57x/ADSP-2157x processors

5.6.2 SigmaStudio Settings

The SigmaStudio ‘Settings’ window can be used to configure the SigmaStudio Host tool. The “Settings” window can be launched by clicking **Tools → Settings**.

5.6.2.1 System Files Export

Settings under the “System Files Export” tab are:

5.6.2.1.1 Pre-Export Command

The Pre-export command is executed before the export operation. For example, this can be a batch file containing Windows command-line instructions. In this case, the name of the batch file along with the path should be entered in this field.

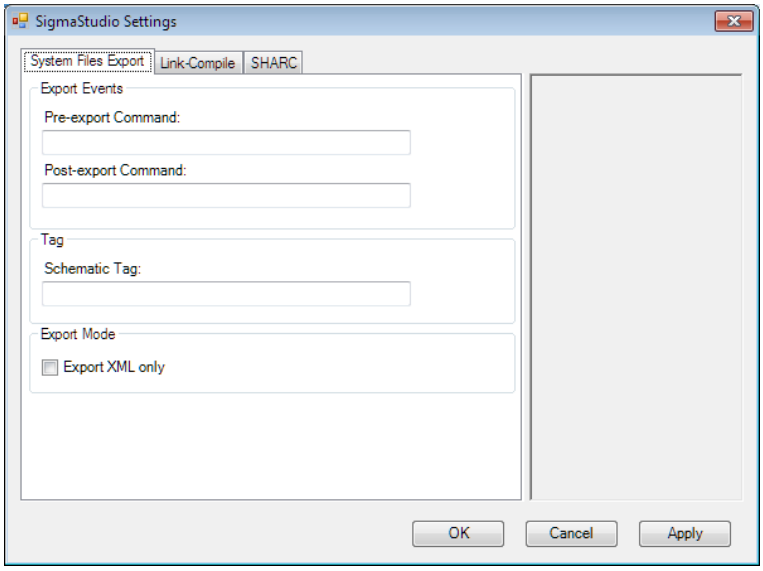


Figure 34: SigmaStudio Settings Window – System Files Export

5.6.2.1.2 Post-Export Command

The Post-export command is executed after the export operation. For example, this can be a batch file containing Windows command-line instructions. In this case, the name of the batch file along with the path should be entered in this field.

5.6.2.1.3 Schematic Tag

This is a user-defined label that can be inserted to tag the Schematic with a unique identification. This tag is saved with the Schematic and can be restored and modified upon re-opening the Schematic.

5.6.2.1.4 Export Mode

The user can select whether, on export, all files are to be generated or only the XML file is to be generated. If 'XML Only' option is checked, only the XML file is generated on export. Otherwise, all export system files including the XML file is generated.

5.6.2.2 Link-Compile

Settings under the "Link-Compile" tab are:

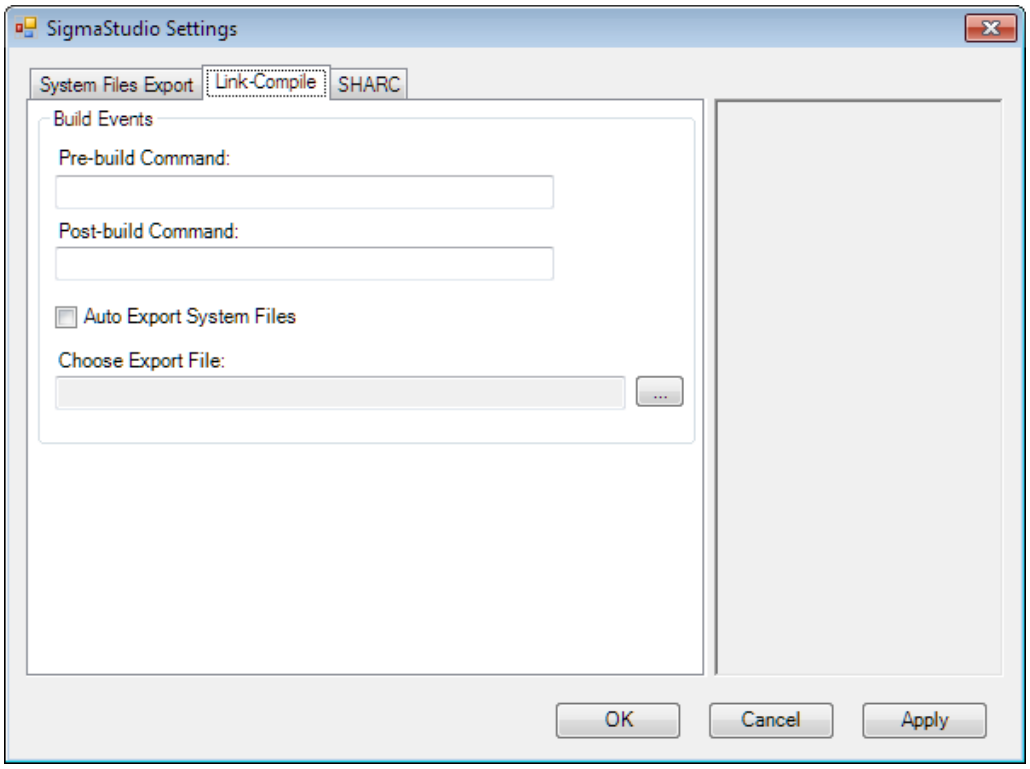


Figure 35: SigmaStudio Settings Window – Link Compile

5.6.2.2.1 Pre-Build Command

The Pre-build command is executed before “Link, Compile, Download” operation. For example, this can be a batch file containing Windows command-line instruction. In this case, the name of the batch file along with the path should be entered in this field.

5.6.2.2.2 Post-Build Command

The Post-build command is executed after a successful “Link, Compile, Download” operation. For example, this can be a batch file containing Windows command-line instructions. In this case, the name of the batch file along with the path should be mentioned in this field.

5.6.2.2.3 Auto Export System Files

System files are exported after every “Link, Compile, Download” operation if the 'Auto Export' option is enabled.

5.6.2.2.4 Choose Export File

The export file prefix and path can be selected using the 'Choose Export File' option in the same tab.

5.6.2.3 SHARC

Settings under the “SHARC” tab are:

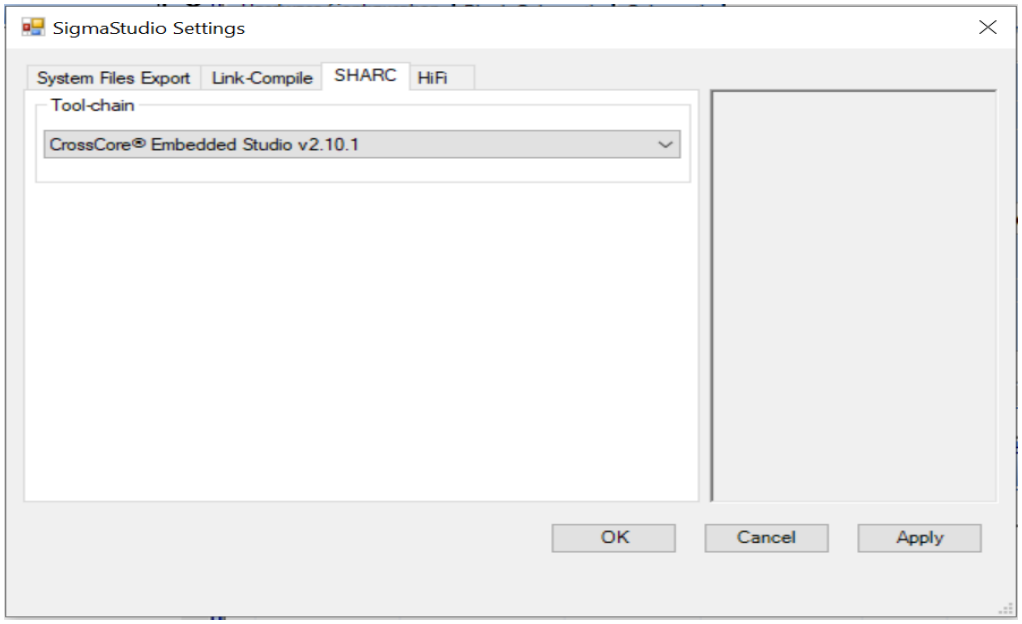


Figure 36: SigmaStudio Settings Window – SHARC Tool-chain

5.6.2.3.1 Tool Chain

All the supported tool-chains which are installed on the PC are listed in this drop-down box. The user can select a tool-chain from this drop-down box for every Schematic. The latest installed version of CrossCore Embedded Studio is the default selection. The selected tool-chain is used by SigmaStudio to compile the Schematics and process the Plug-Ins. The selected tool-chain information is stored in the Schematics after a successful compilation. While opening a Schematic, if the tool-chain required for the Schematic is not installed, SigmaStudio shows a pop-up message *“<tool-chain which is not found>, which was used for saving the current schematic, is not a valid installed Tool-Chain. Choosing <Default_Tool-Chain> as the active Tool-Chain.”* SigmaStudio then uses the latest installed version of CrossCore Embedded Studio. Consult **[4]** for the minimum supported version.

5.6.3 Code and Buffer sharing

5.6.3.1 Code and Buffer Sharing between Application DXE and the Schematic

SigmaStudio for SHARC uses differential DXE mechanism for schematic compilation and linking. In differential DXE mechanism, all the source files and libraries are compiled and linked to form the differential DXE which is called the schematic DXE. All the global symbols in the target application are defined as symbols in the schematic DXE LDF file. Hence the linker will take care of reusing

the functions and tables from the application instead of re-inserting them in the schematic DXE. Hence, code and buffers can be shared between the target Application and the schematic.

5.6.3.2 Code and Buffer sharing between the modules in the Schematic

SigmaStudio for SHARC uses differential DXE mechanism for schematic compilation and linking. In differential DXE mechanism, all the source files and libraries are compiled and linked to form the differential DXE which is called the schematic DXE. If there are 2 or more modules in a schematic that use the same function/ tables, the linker will take care of reusing the functions and tables instead of inserting them for every module which accesses the function/ tables. Hence, code and buffers can be shared between the modules in the schematic.

5.6.4 Multi-Rate Processing

Multi-rate processing is achieved with the help of multiple Block Sizes within a schematic. This means that having modules with multiple Block Sizes in a schematic implies having multiple Sampling rates in the same schematic. In order to obtain multiple Block Sizes which are factors of the schematic Block Size, Up-Sampler/ Down-Sampler modules are to be used. The user can have intermediate Block Sizes which are only lesser than or equal to the Schematic Block Size. Up-Sampler and Down-Sampler modules would essentially be changing the Block Size and Sampling rate of the modules at its output side.

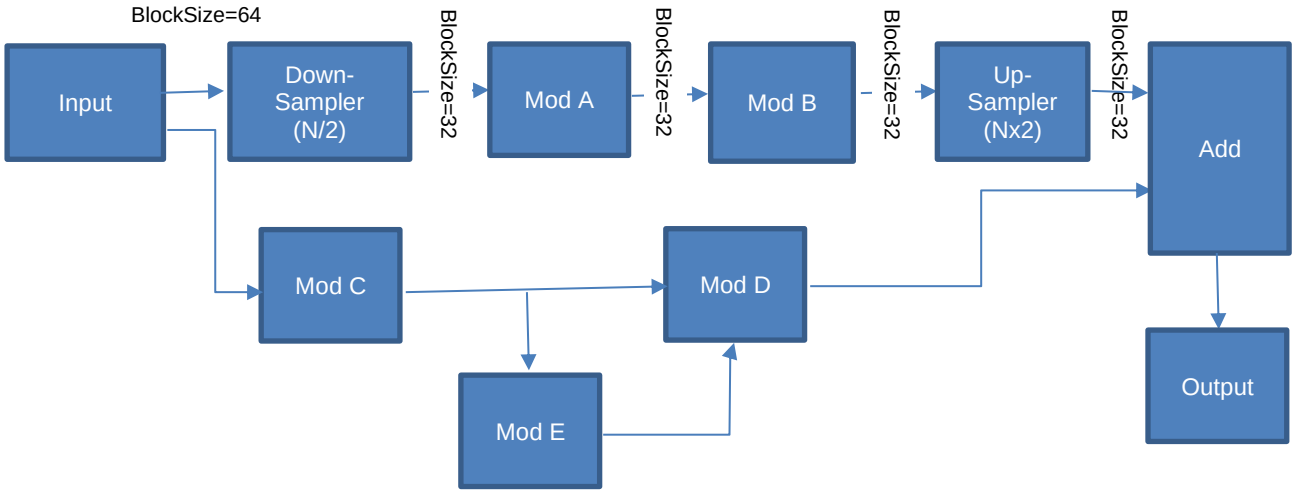


Figure 37: Multi-rate processing

- In the above example, ModC, ModD and ModE operates on BlockSize=64. ModA and ModB operates on BlockSize=32. Effectively ModA and ModB are operating at half the sampling rate as ModC, ModD and ModE.
- SigmaStudio includes up-sampler and down-sampler modules which changes the Block Size of the modules connected to it.

- Up-sampling and down-sampling will operate only on factors which are powers of 2. Also, the Up-Sampling/ Down-Sampling factor should be a factor of the Schematic BlockSize.

5.6.4.1 Up-Sampler

This module increases the sampling rate by a required factor 'N' which is a multiple of 2. This interpolating factor 'N' can be selected in the drop-down control present in the cell as shown in the figure below.

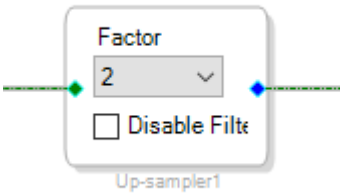


Figure 38: Up-Sampler cell

The up-sampled signal is then passed through an interpolating filter which is essentially a low-pass filter. The interpolation filter can be disabled by checking the "Disable Filter" checkbox in the cell.

5.6.4.2 Down-Sampler

This module decreases the sampling rate by a required decimating factor 'L' which is a multiple of 2. This decimating factor 'L' can be selected in the drop-down control present in the cell as shown in the figure below.

Figure 39: Down-Sampler cell

The input signal can be optionally passed through an anti-aliasing/ decimating filter (a low-pass filter) to avoid aliasing. This can be achieved by selecting the anti-aliasing check box present in the down-sampler cell.

5.6.5 Load Balancing

In a “Dual Core” schematic execution mode, a single schematic uses both the SHARC cores on ADSP-SC5xx for the processing. A subset of the modules on the schematic executes on the first SHARC core and the remaining modules execute on the second SHARC core. User has the flexibility to choose which all modules executes on each of the SHARC cores, thus the processing load of the schematic is balanced across the SHARC cores.

5.6.5.1 Enable Dual Core Mode

Set the ‘Dual Core’ drop down option to “Dual Core” in order to enable dual core execution mode. “Default SHARC Core”, “Processing Mode” and “Instance ID” will be disabled when Dual Core mode is enabled. This is because multi-instancing is currently not supported in “Dual Core” mode and both the SHARC Cores will be used for schematic processing in this mode.

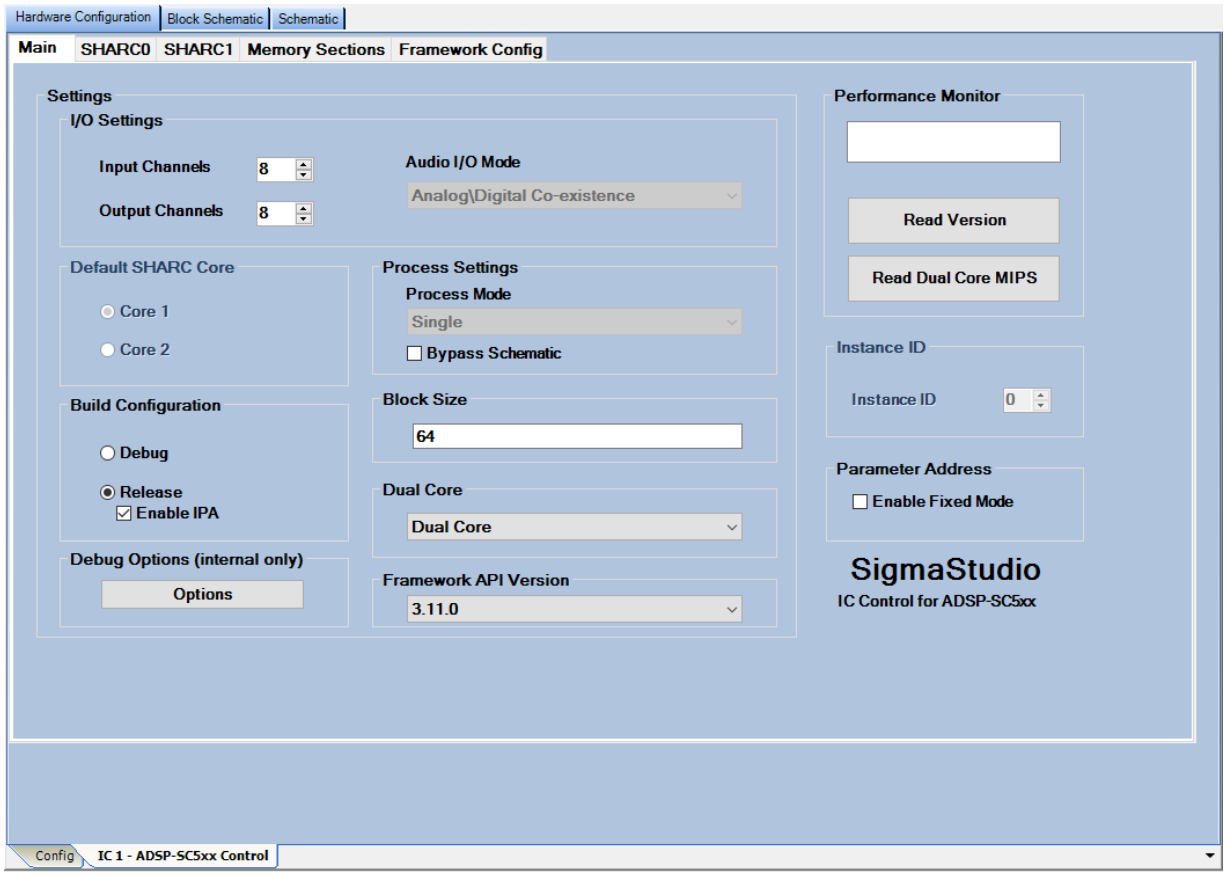


Figure 40: “Dual Core” Mode in Control Window

Both “SHARC0” and “SHARC1” tabs will be enabled in “Dual Core” mode. User will have to select application DXE for both the SHARC Cores when Dual Core mode is selected.

5.6.5.2 Mapping Modules to SHARC Cores

By default, all modules inserted on to a Dual Core schematic will get mapped to SHARC 0. In the case of ADSP-2156x, all modules are always mapped to SHARC0, since only a single SHARC core is present.

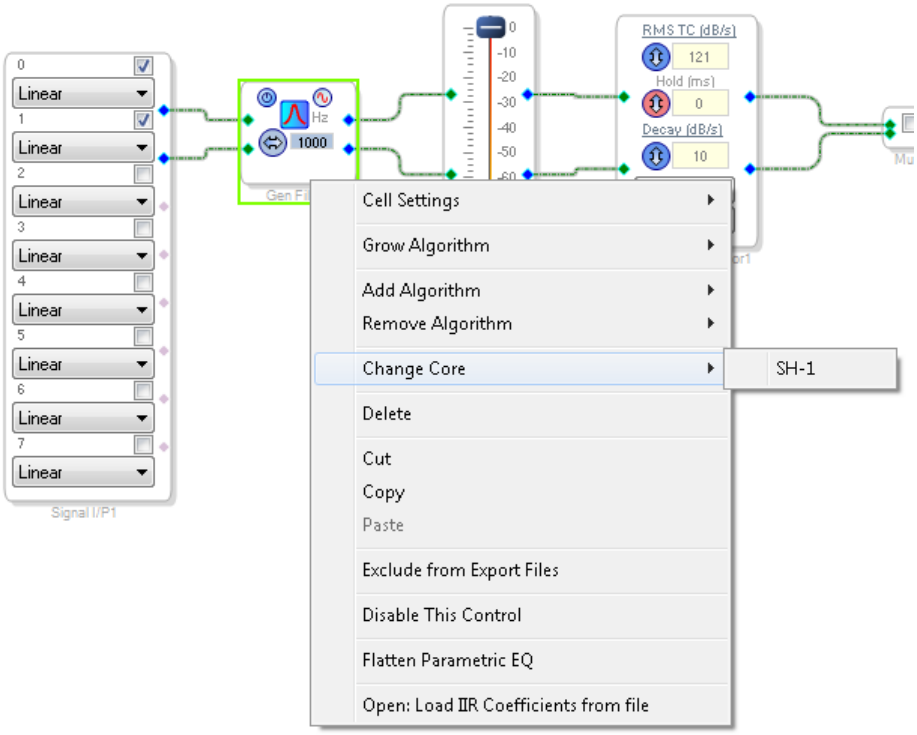


Figure 41: Mapping Modules to SHARC Cores

User may right click on the module and select “Change Core” option from the Context Menu to change the SHARC Core on which the module executes. It is also possible to right click on a “Hierarchy Board” and SHARC Core of all the modules inserted inside the “Hierarchy Board”.

Following are few exceptional cases with respect to the dual core schematics:

1. The “Signal Input” module will always execute on the SHARC0. It is not possible to map “Signal Input” to SHARC1. “Change Core” option is disabled on “Signal Input” module.
2. “Output” module when inserted on the schematic will execute on both the SHARC cores. This means that the same output will be made available at the output buffers of the instances running on both the SHARC Cores. “Change Core” option is disabled on “Output” module.

“Feedback” module is currently not supported on “Dual Core” schematics.

5.6.5.3 Compile Dual Core Schematic

Once a Dual Core schematic is compiles, there will be separate set of SMAP, Core, Parameter and other buffers sent from the host to each of the SHARC cores. The output window will display the memory and schematic load details of both the SHARC cores separately.

5.6.6 Fixed Address Mode

SigmaStudio supports locking the parameter offset of one or more modules in the Schematic and the parameter buffer base address of the schematic. Once, the module's parameter offset is locked, addition of any new modules will not alter the parameter offset of the module.

To Enable the Fixed address mode in the schematic, the "Enable Fixed Mode" option in the "Main" tab of IC control form has to be enabled as shown below:

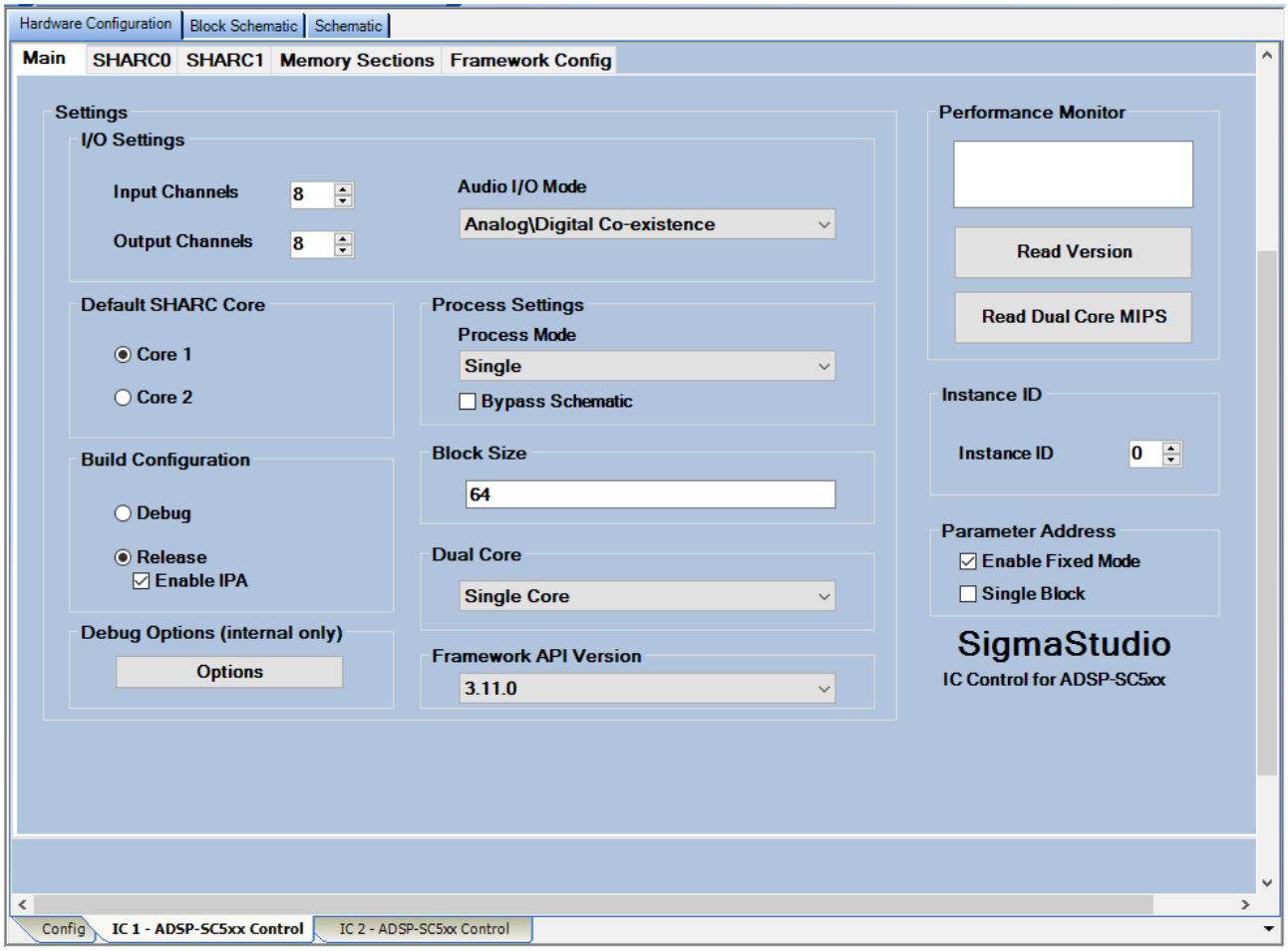


Figure 42: Enable Fixed Addressed mode for a schematic

Once this option is enabled, “Lock Parameter Address” can be selected in the context menu of any module whose Parameter offset must be locked. The context menu option is as shown below:

Figure 43: Lock Parameter address of a module

Once the parameter offset for a module or a set of modules is locked and the schematic is link-compile-downloaded, a text file “Param.txt” is generated which gives the details of the locked Parameter buffer base address of the schematic and the parameter offset addresses of each of the modules for which parameter offset is locked.

Note: If a module whose parameter offset is locked is moved to the other core in a dual core schematic, then the parameter offset address of this module will be different in the other core. There will be no warning displayed since, it is as good as inserting a new module in the other core with fixed address mode enabled.

The Parameter offset of a fixed addressed module can be released by selecting the “Release Parameter Address” option in context menu of the module and link-compile-downloading the schematic. This option is shown below:

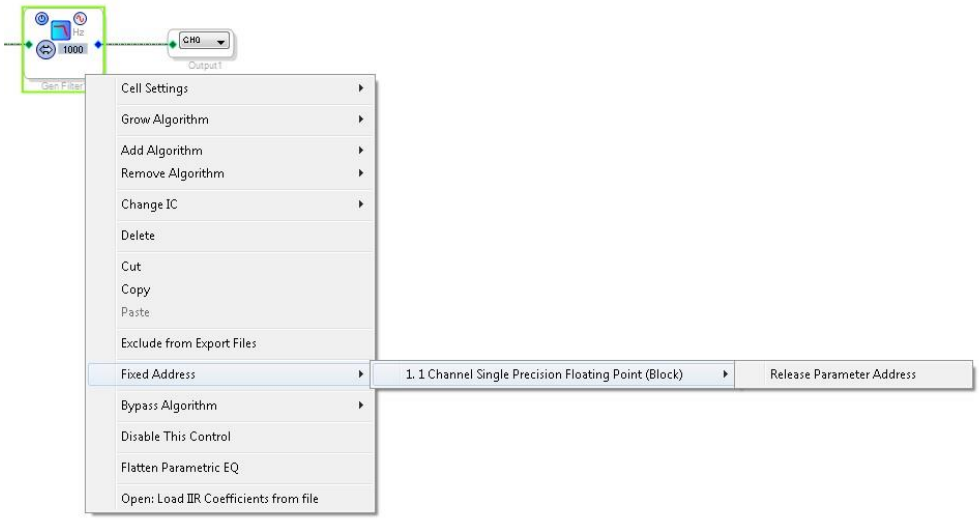


Figure 44: Release Parameter address of a module

5.6.6.1 Single Block of Fixed Parameters

SigmaStudio allocates parameters of fixed address modules into a single block. However, the following actions can result in gaps getting introduces in the fixed address memory block and as a result there can be two or more fixed address blocks.

- Removing a fixed address module from the schematic
- Reducing the parameter memory size (e.g., reduce the Filter Taps in FIR filter) of a fixed address module
- Increasing the parameter memory size (e.g., increase the Filter Taps in FIR filter) of a fixed address module. This may also result in the module getting assigned a different parameter address

When ‘Single Block’ is enabled, SigmaStudio will check whether the fixed address parameters are assigned as a single block. If not, the modules will be reallocated to a single block after taking user confirmation through a dialog box. The “Single Block” option can be enabled for the schematic in “Hardware Configuration → Main tab” refer Figure 42.

5.6.7 Bypass Module

SigmaStudio supports Bypass of modules in a schematic. Bypass mode can be selectively enabled on one or more modules in the schematic. There is a provision to bypass those modules in run-time for which the Bypass option is enabled.

To enable the bypass option for a module, the option “Include Bypass Option” must be enabled in the context menu of the module and the schematic has to be link-compile-downloaded. The context menu option is as shown below:

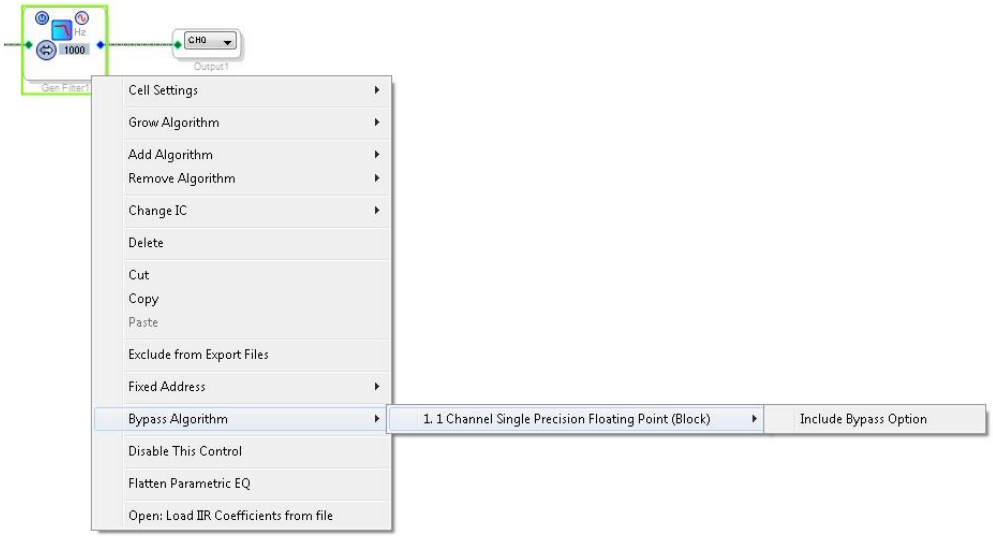


Figure 45: Enable Bypass option for a module

Once the schematic is downloaded, the module for which bypass option was included/ enabled can be bypassed during run-time. This can be done by selecting the option “Bypass Module” in the context menu of the module as shown below:

Figure 46: Bypass the module

The bypassed modules can be processed by changing the context menu item to “Process Module” as shown below:

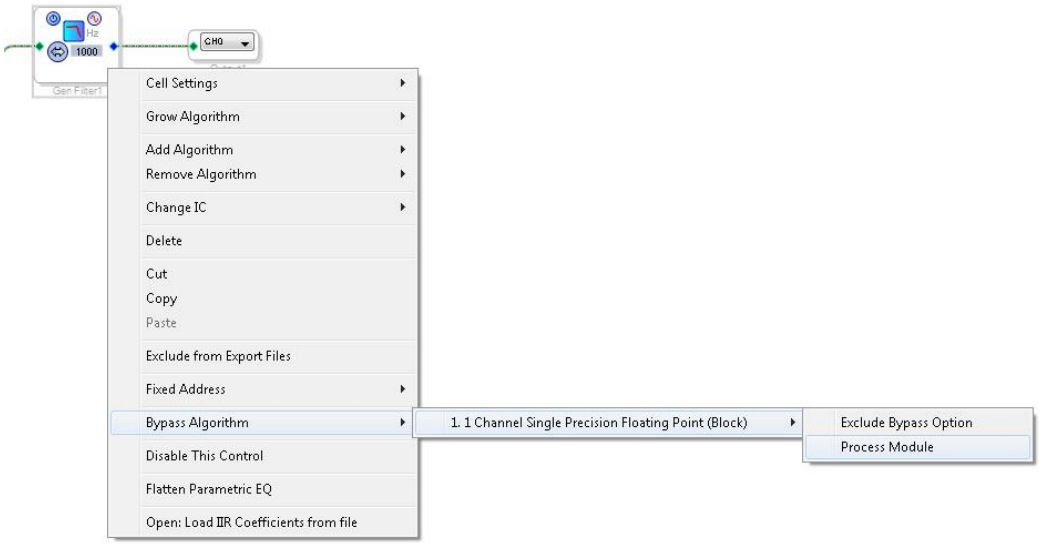


Figure 47: Process the module

5.6.8 Schematic Compilation Optimizations

This section talks about the following features added for optimizing schematic compilation:

- Differential Compilation: This is explained in detail in section 5.6.8.1
- SSn framework improvements for lesser compilation time: The framework improvements implemented increases the MIPS consumption. The trade-off between the time reduction and MIPS consumption is explained in section 5.6.8.5

5.6.8.1 Link Compile Download

This option compiles only the files that are changed due to changes in the schematic. The toolbar option for this feature is shown in the figure below:

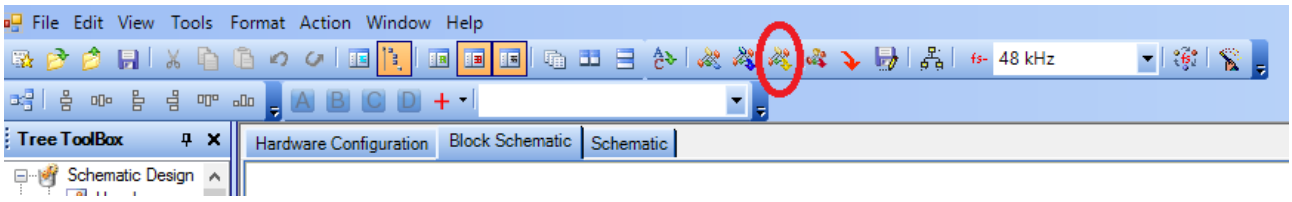


Figure 48: Link Compile Download

After compilation, it downloads the code and parameter data to the target. If the schematic is unchanged, no files get compiled and only the code and parameter data gets downloaded onto the target.

The differential compilation feature is added to reduce the overall time it takes for a schematic to get compiled and downloaded onto the target.

5.6.8.2 Clean Link Compile Download

This option deletes all temporary files and compiles all the files regardless of whether there is a change in schematic. Once compiled, it downloads the code and parameter data onto the target.

The toolbar option for this feature is shown in the figure below:

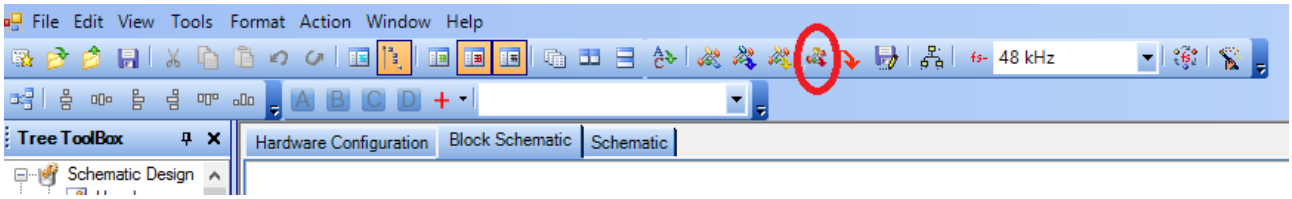


Figure 49: Clean Link Compile Download

Clean Link compile download option should be used in the following scenarios:

- Application DXEs are changed
- Build configuration is changed
- Sampling rate is changed
- Any settings changed in the IC control window.

5.6.8.3 Connect to Target

SigmaStudio for SHARC (ADSP-SC5xx/ADSP-215xx) supports connecting to target for tuning when a schematic enters design mode due to a switch in view. A toolbar option has been added for this as shown in the figure below:

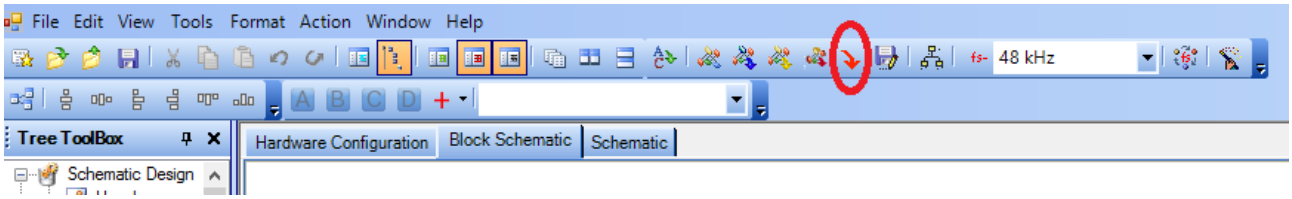


Figure 50: Connect to Target

This option can be used when the user is switching between schematics and wants to connect to the target to tune the parameters. The user must ensure that the schematic that is running on the target is same as the schematic that is being connected to target from the SigmaStudio IDE. The behavior may be undefined if the schematic running on the target and the one being attempted to be connected through GUI are different.

5.6.8.4 Inter-procedural optimization

Inter-procedural optimization can be enabled to reduce the MIPS consumption for a schematic. This will affect the time taken to compile and link the schematic. The tradeoff between compilation time and MIPS with and without IPA is as described in the 5.6.8.5

This option can be enabled as described in section 5.6.1.3.1 .

5.6.8.5 Compilation time vs MIPS trade-off

The Ssn framework improvements reduce the compilation time by around 50% when IPA is disabled and around 30% when IPA is enabled. The MIPS consumption increase by around 2-5%.

6 System Integration

Communication between the SigmaStudio Host and Target is handled completely by the ARM Core Application. The 'SigmaStudio for SHARC Target Library' when integrated to a SHARC Application acts as one among the many processing modules in the SHARC Core Application. It takes audio sample and/or bit-stream data as input from one or more Sources/Modules and gives out processed audio samples as output. These output audio samples can be fed as input to another module in the Application.

6.1 Memory Allocations

This section describes the memory sections for code and data buffers.

Block	Section/Purpose	Type
Block 0 in ADI_SS_MEM_MAP for communication	Communication Instance	Unsigned Integer
Block 1 in ADI_SS_MEM_MAP for communication	Reserved	Unsigned Integer
Block 0 in ADI_SS_MEM_MAP	SSn Instance	Unsigned Integer
Block 1 in ADI_SS_MEM_MAP	Code - SSn Instance. The address of the declared memory block should be assigned in the ADI_SS_MEM_BLOCK structure in byte addressing mode and should be aligned by 2	Unsigned Integer
Block 2 in ADI_SS_MEM_MAP	Receive Commands	Unsigned Integer
Block 3 in ADI_SS_MEM_MAP	Receive parameters from the SigmaStudio Host for update	Unsigned Integer
Block 4 in ADI_SS_MEM_MAP	Data - SSn Instance	Unsigned Integer
Block 5 in ADI_SS_MEM_MAP	Parameter - SSn Instance	Unsigned Integer
Block 6 in ADI_SS_MEM_MAP	Extended Precision Data[1] – SSn Instance. This block should be declared as 48-bit memory section. The address of the declared block should be passed to the ADI_SS_MEM_BLOCK structure in 48-bit mode	Unsigned Integer

[1] Extended precision memory buffer should always be mapped to L1 memory irrespective of the processor. Extended precision memory buffer should not be mapped to the same hardware memory block as any other data objects, except for blocks 8 and 9 in ADI_SS_MEM_MAP, which are only used in Block Processing Schematics.

	and should be aligned by 2.	
Block 7 in ADI_SS_MEM_MAP	Code Memory B – intended only for use by Plug-Ins. Code sections in Plug-Ins can be mapped to this memory block using the LDF file used in the Plug-In DLM generation. The address of the declared memory block should be assigned in the ADI_SS_MEM_BLOCK structure in byte addressing mode and should be aligned by 2	Unsigned Integer
Block 8 in ADI_SS_MEM_MAP	State Memory Block B - intended only for use by Plug-Ins. This State Memory is not cleared by the SigmaStudio during initialization or using the API to clear the state memory. If needed, user should explicitly clear this memory as part of the Application.	Unsigned Integer
Block 9 in ADI_SS_MEM_MAP	State Memory Block C - intended only for use by Plug-Ins. This State Memory is not cleared by the SigmaStudio during initialization or using the API to clear the state memory. If needed, user should explicitly clear this memory as part of the Application.	Unsigned Integer
Block 10 in ADI_SS_MEM_MAP	Memory for inter core communication for ADSP-SC5xx processors. This memory must be allocated from L2 shared memory block.	Unsigned Integer

Table 12: Memory Sections

For optimal performance, Block 4 in `ADI_SS_MEM_MAP` (Data- SSn Instance) and Block 5 in `ADI_SS_MEM_MAP` (Parameter- SSn Instance) should be mapped (in the .ldf file) to different memory blocks. This can be done by assigning “dm” for Block 4 and “pm” for Block 5 when declaring them.

SigmaStudio supplies a scratch DM memory buffer pointer and a scratch PM memory buffer pointer to Plug-Ins (available only in Block Processing). The DM scratch is allocated from Block 4 in `ADI_SS_MEM_MAP` and PM scratch is allocated from Block 5 in `ADI_SS_MEM_MAP`.

For target processors not containing an L3 memory section, the L3 data and code sections are mapped from the L2 section. The memory start address for this L2 code section is 0x200c0000 and the end address is 0x200D7FFF (96 kB) and for L2 data section is 0x200D8000 and the end address is 0x200F9FFF (160kB). Thus, the user must take care to accordingly set the memory ranges for mem_L2_bw memory section in app.ldf file present in the project. If L3 memory is present, then the start address for mem_L2_bw section is 0x20000000 and the end address is 0x200F9FFF. If L3 is not present then the end address of the mem_L2_bw section must be set to 0x200BFFFF.

7 GMAP and SMAP

7.1 GMAP

GMAP is the Global memory MAP. This structure holds information corresponding to the available physical memory in each of the SHARC memory blocks. This information is communicated by the target library to the SigmaStudio host. The figure below shows the arrangement of information within the GMAP structure.

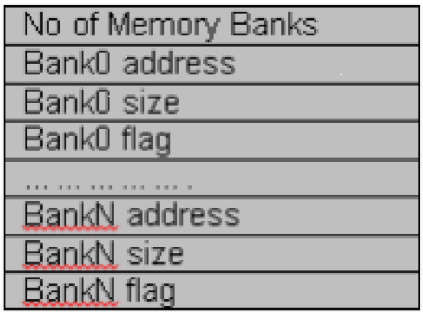


Figure 51: GMAP structure format

The table below lists the blocks made available to host by target through GMAP

Sl No	Physical Block	Description	Use for
1	Block 0	L1 block 0	Allocation of SigmaStudio schematic code
2	Block 1	L1 block 1	Allocation of SigmaStudio schematic 32-bit data (State)
3	Block 2	L1 block 2	Allocation of SigmaStudio schematic parameters (Coef)
4	Block 3	L1 block 3	Allocation of SigmaStudio schematic 48-bit data (data-48) and 32-bit data (State B)
5	Block L3 code	L3 SW code memory block	Allocation of SigmaStudio schematic code (Code-B)
6	Block L3 data	L3 data memory block	Allocation of SigmaStudio schematic 32-bit data (State C)
7	Block L2 data	L2 cached memory block	For SSn-target library and for inter core communication in dual core mode.

Table 13: GMAP blocks

The GMAP structure values updated with memory blocks allocated in “adi_ss_app.ldf” file for each target application. The “adi_ss_app.ldf” available in respective target application source folder. For example, the “adi_ss_app.ldf” file for ADSP-2156x target can be found in *“< Software Modules folder > \SigmaStudioForSHARC-SH-Rel*4.7.0*Target\Examples*\Demo\ADSP-2156x\Source” folder.

```
/*
** SigmaStudio for Griffin application linker description file for GMAP
*/


/* Default for SSn code */
SS4G_block0
{
   RESERVE(_Block0_L1_space, _Block0_L1_space_length = 2, 2)
   RESERVE_EXPAND(_Block0_L1_space, _Block0_L1_space_length, 0, 2)
} > mem_block0_bw

/* Default for SSn data */
SS4G_block1
{
   RESERVE(_Block1_L1_space, _Block1_L1_space_length = 2, 2)
   RESERVE_EXPAND(_Block1_L1_space, _Block1_L1_space_length, 0, 2)
} > mem_block1_bw

/* Default for SSn parameter */
SS4G_block2
{
   RESERVE(_Block2_L1_space, _Block2_L1_space_length = 8, 8)
   RESERVE_EXPAND(_Block2_L1_space, _Block2_L1_space_length, 0, 8)
} > mem_block2_bw

/* Default for SSn data B and extended precision */
SS4G_block3
{
   RESERVE(_Block3_L1_space, _Block3_L1_space_length = 2, 2)
   RESERVE_EXPAND(_Block3_L1_space, _Block3_L1_space_length, 0, 2)
} > mem_block3_bw

#if defined(MY_SDRAM_SWCODE_MEM)
/* Default for SSn code B */
SS4G_L3_Code
{
   RESERVE(_Block_L3_code_space, _Block_L3_code_space_length = 2, 2)
   RESERVE_EXPAND(_Block_L3_code_space, _Block_L3_code_space_length, 0, 2)
} > MY_SDRAM_SWCODE_MEM
#else
/* Default for SSn code B */
SS4G_L2_Code
{
   RESERVE(_Block_L3_code_space, _Block_L3_code_space_length = 2, 2)
```

```
   RESERVE_EXPAND(_Block_L3_code_space, _Block_L3_code_space_length, 0, 2)
} > mem_L2_bw_SS4G_Code
#endif

#if defined(MY_SDRAM_DATA1_MEM)
/* Default for SSn data C */
SS4G_L3_Data
{
   RESERVE(_Block_L3_data_space, _Block_L3_data_space_length = 8, 8)
   RESERVE_EXPAND(_Block_L3_data_space, _Block_L3_data_space_length, 0, 8)
} > MY_SDRAM_DATA1_MEM
#else
/* Default for SSn data C */
SS4G_L2_Data1
{
   RESERVE(_Block_L3_data_space, _Block_L3_data_space_length = 8, 8)
   RESERVE_EXPAND(_Block_L3_data_space, _Block_L3_data_space_length, 0, 8)
} > mem_L2_bw_SS4G_Data
#endif

/* Default for SSn L2 data for buffer sharing */
/* This is allocated from the cached portion of L2 */
SS4G_L2_Data
{
   RESERVE(_Block_L2_data_space, _Block_L2_data_space_length = 2, 2)
   RESERVE_EXPAND(_Block_L2_data_space, _Block_L2_data_space_length, 0, 2)
} > MY_L2_CACHED_MEM

/* Sections for inter core handshaking */
/* This is allocated from the uncached portion of L2 for core 1 (SHARC 0) */
#if !defined(__ADSP2156x__)
#if defined(__ADSPSC5xx__)
SS4G_L2_Core0_Handshake BW
{
   INPUT_SECTIONS($OBJS_LIBS(ss4g_l2_core0_handshake) )
} > mem_L2B1P2_bw

/* This is allocated from the uncached portion of L2 for core 1 (SHARC 0) */
SS4G_L2_Core1_Handshake BW
{
   INPUT_SECTIONS($OBJS_LIBS(ss4g_l2_core1_handshake) )
} > mem_L2B1P4_bw

/* This is allocated from the uncached portion of L2 for core 2 (SHARC 1) */
SS4G_L2_Core2_Handshake BW
{
   INPUT_SECTIONS($OBJS_LIBS(ss4g_l2_core2_handshake) )
} > mem_L2B1P3_bw
#elif defined(__ADSP215xx__) && (__NUM_ARM_CORES__==0)
/* This is allocated from the uncached portion of L2 for core 1 (SHARC 0) */
SS4G_L2_Core1_Handshake BW
```

```
{
   INPUT_SECTIONS($OBJS_LIBS(ss4g_l2_core1_handshake) )
} > mem_L2B1P2_bw

/* This is allocated from the uncached portion of L2 for core 2 (SHARC 1) */
SS4G_L2_Core2_Handshake BW
{
   INPUT_SECTIONS($OBJS_LIBS(ss4g_l2_core2_handshake) )
} > mem_L2B1P3_bw
#endif
#endif
```

The GMAP memory blocks start addresses and size of which are allocated by target application and the details of GMAP blocks can be seen in the generated linker map file (SS_App_Core1.map.xml) under corresponding build output folder (Release/Debug).

LDF symbols

Symbol	Address
heaps_and_system_stack_in_L1	0x26e400
heaps_and_system_stack_in_L1_length	0x1c00
___ldf_dmcachesize	0xffffffff
___ldf_pmcachesize	0xffffffff
___ldf_icachesize	0xffffffff
___l2_uncached_start	0x200fa000
___l2_uncached_end	0x200fdfff
__ctors	0x10000000
_ctors.	0x80000008
_Block0_L1_space	0x252180
_Block0_L1_space_length	0x1c280
_Block1_L1_space	0x2c5514
_Block1_L1_space_length	0x2aaec
_Block2_L1_space	0x300000
_Block2_L1_space_length	0x20000
_Block3_L1_space	0x38b1cc
_Block3_L1_space_length	0x14e34
_Block_L3_code_space	0x80a00000
_Block_L3_code_space_length	0xf600000
_Block_L3_data_space	0x80000010
_Block_L3_data_space_length	0x9fff0
_Block_L2_data_space	0x20003600
_Block_L2_data_space_length	0xf6a00
ldf_stack_space	0x26e400
ldf_stack_end	0x26fbf8
ldf_stack_length	0x17f8
ldf_heap_space	0x26fc00
ldf_heap_end	0x26fff8
ldf_heap_length	0x3f8

Figure 52: GMAP block allocation in Map file

7.2 SMAP

SMAP is the Schematic memory MAP. This information is communicated to the target library by the SigmaStudio host. SMAP contains information on certain framework parameters such as sampling rate, block size, Sport configuration etc. It also contains memory information for the SSn instances. The SMAP structure is detailed below.

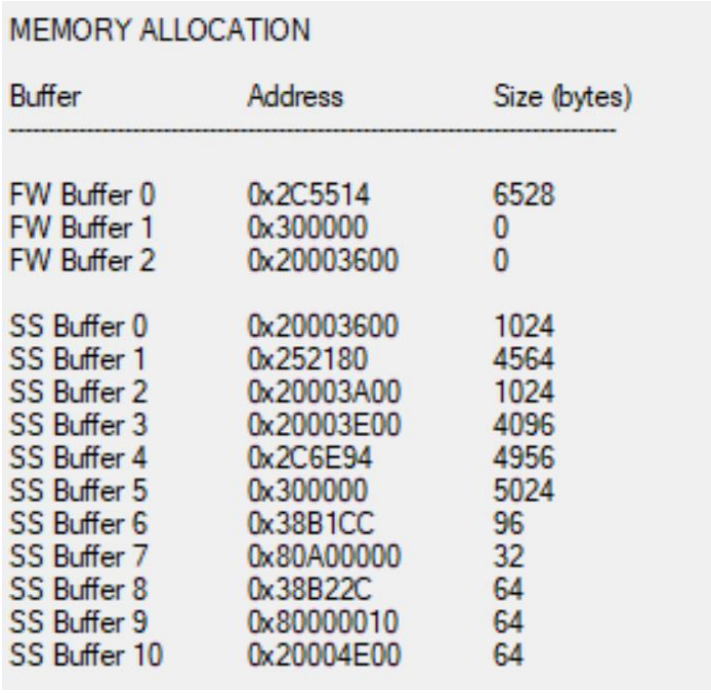
7.2.1 SMAP

```
struct SS_SMAP
{
    SS_SMAP_FW_INFO        oFwInfo;
    uint32_t               nNumSSn;
    ADI_SS_FW_PROCESS_MODE eProcessMode;
    SS_SMAP_SSN_INFO       oSSnInfo[ADI_SS_FW_MAX_PROC_BLOCKS];
};
```

Fields

- `oFwInfo`
  Instance of framework structure. Refer 7.2.2 for details.
- `nNumSSn`
  Number of SSn instances.
- `eProcessMode`
  SSn processing mode. Refer **[2]** for more details on ADI_SS_FW_PROCESS_MODE.
- `oSSnInfo`
  Array of SSn info structure type. Refer 7.2.6 for details.

The SMAP buffer information for the target framework can be found in SigmaStudio schematic compiler output window.



MEMORY ALLOCATION

Buffer	Address	Size (bytes)
FW Buffer 0	0x2C5514	6528
FW Buffer 1	0x300000	0
FW Buffer 2	0x20003600	0
SS Buffer 0	0x20003600	1024
SS Buffer 1	0x252180	4564
SS Buffer 2	0x20003A00	1024
SS Buffer 3	0x20003E00	4096
SS Buffer 4	0x2C6E94	4956
SS Buffer 5	0x300000	5024
SS Buffer 6	0x38B1CC	96
SS Buffer 7	0x80A00000	32
SS Buffer 8	0x38B22C	64
SS Buffer 9	0x80000010	64
SS Buffer 10	0x20004E00	64

Figure 53: SMAP block allocations in schematic compilation

The SMAP buffer information and corresponding GMAP blocks are shown in below table.

GMAP Blocks	SMAP Buffers	SMAP Memory Section Names	Purpose
Block0_L1_space	SS Buffer 1	Code	SigmaStudio instance code memory
Block1_L1_space	FW Buffer 0	NA	SigmaStudio framework buffers
	SS Buffer 4	StateA	SigmaStudio instance Data32 memory
Block2_L1_space	SS Buffer 5	Param	SigmaStudio instance parameter memory
Block3_L1_space	SS Buffer 6	NA	SigmaStudio instance extended precision state memory
	SS Buffer 8	StateB	SigmaStudio instance Data32 B memory
Block_L3_code_space	SS Buffer 7	CodeB	SigmaStudio instance code B

			memory
Block_L3_data_space	SS Buffer 9	StateC	SigmaStudio instance Data32 C memory
	SS Buffer 5	ParamB	SigmaStudio instance parameter B memory
Block_L2_data_space	SS Buffer 0	NA	SigmaStudio instance handle
	SS Buffer 10	NA	Memory for inter core communication

Table 14: SMAP buffer allocation in GMAP blocks

Note: *The target application doesn't have any control other than GMAP section mapping to create SMAP buffers.*

7.2.2 SS_SMAP_FW_INFO

```
typedef struct SS_SMAP_FW_INFO
{
    ADI_SS_FW_HOST_CONFIG  oFwHostConfig;
    uint32_t               nNumFwBuffers;
    ADI_SS_MEM_BLOCK       oFwBuff[MAX_SMAP_FW_BUFFERS];
}SS_SMAP_FW_INFO;
```

Framework info structure within SMAP. The structure elements are described below.

- `oFwHostConfig`
  Structure instance for host configurable framework parameters. Refer 7.2.3 for details.
- `nNumFwBuffers`
  Part of memory required for the framework is provided by the host. This field indicates the number of buffers required for the framework.
- `oFwBuff`
  Array of mem blocks for the framework buffers. Refer 5.3.3.1.1 for details on ADI_SS_MEM_BLOCK structure.

7.2.3 ADI_SS_FW_HOST_CONFIG

```
typedef struct ADI_SS_FW_HOST_CONFIG
{
    uint32_t                  nBlockSize;
    uint32_t                  nInInterfaceBuffSz;
    uint32_t                  nNumInInterfaceBuff;
    uint32_t                  nOutInterfaceBuffSz;
    uint32_t                  nNumOutInterfaceBuff;
    uint32_t                  nOutputPreroll;
    uint32_t                  nInSamplingRate;
    uint32_t                  nOutSamplingRate;
    ADI_SS_FW_AUDIOMODE       eAudioMode;
    ADI_SS_FW_MULTICORE_MODE  eFwMultiCoreMode;
    uint32_t                   nPeripheralIOBuffSz;
    uint32_t                   nNumPeripheralIOBuff;
    uint32_t                   nNumSources;
    ADI_SS_FW_DATA_PERI_TYPE   eSourcePeriType[ADI_SS_MAX_SOURCES];
    ADI_SS_FW_DATA_PERI_CONFIG oSourcePeriConfig[ADI_SS_MAX_SOURCES];
    uint32_t                   nNumSinks;
    ADI_SS_FW_DATA_PERI_TYPE   eSinkPeriType[ADI_SS_MAX_SINKS];
    ADI_SS_FW_DATA_PERI_CONFIG oSinkPeriConfig[ADI_SS_MAX_SINKS];

}ADI_SS_FW_HOST_CONFIG;
```

The framework parameters in this structure may be configured by the host.

- `nBlockSize`
  Schematic processing block size.
- `nInInterfaceBuffSz`
  Total input interface buffer size reserved by host for all pins. This has to be a multiple of nBlockSize. This field will be used for input buff size validation.
- `nNumInInterfaceBuff`
  Number of input interface buffers of size nBlockSize for the i/o data buffering by the framework.
- `nOutInterfaceBuffSz`
  Total output interface buffer size reserved by host for all pins. This has to be a multiple of nBlockSize. This field will be used for output buff size validation.
- `nNumOutInterfaceBuff`
  Number of output interface buffers of size nBlockSize for the i/o data buffering by the framework.
- `nOutputPreroll`
  Output preroll as a multiple of nBlockSize.
- `nInSamplingRate`
  Input sampling rate.

- `nOutSamplingRate`
  Output sampling rate.

- `eAudioMode`
  Audio mode. Refer **[2]** for more details on ADI_SS_FW_AUDIOMODE enumeration type.

- `eFwMultiCoreMode`
  Parameter for different types of signal flows designed in host for multicore mode. Refer **[2]** for more details on ADI_SS_FW_MULTICORE_MODE enumeration type.

- `nPeripheralIOBuffSz`
  Total Peripheral i/o buffer size for all pins including input and output.This field is currently not used.

- `nNumPeripheralIOBuff`
  Number of peripheral i/o buffers. This field is currently not used.

- `nNumSources`
  Number of Data sources

- `eSourcePeriType`
  Source peripheral type enumeration for each of the sources

- `oSourcePeriConfig`
  SPORT configuration for each of the sources received from the host

- `nNumSinks`
  Number of Data sinks

- `eSinkPeriType`
  Sink peripheral type enumeration for each of the sinks

- `eSinkPeriConfig`
  SPORT configuration for each of the sinks received from the host

7.2.4 ADI_SS_FW_DATA_PERI_CONFIG

```
typedef struct ADI_SS_FW_DATA_PERI_CONFIG
{
    ADI_SS_FW_DAI_PIN_GROUP   eDataDAIPinGroup
    ADI_SS_FW_DAI_PIN         eDataDAIPin;
    uint32_t                  nEnSecChannel;
    ADI_SS_FW_DAI_PIN_GROUP   eDataDAIPinGroupSec;
    ADI_SS_FW_DAI_PIN         eDataDAIPinSec;
    uint32_t                  nChannels;
    ADI_SS_FW_SPORT_CONFIG    oSPORTPeriConfig;
}ADI_SS_FW_HOST_CONFIG;
```

The framework parameters in this structure may be configured by the host.

- `eDataDAIPinGroup`
  Primary in/out DAI pin group for a source or sink
- `eDataDAIPin`
  Primary in/out DAI pin number for a source or sink
- `nEnSecChannel`
  Flag indicating whether secondary channel of the peripheral is enabled or not. This field is currently unused.
- `eDataDAIPinGroupSec`
  Secondary in/out DAI pin group for a source or sink. This field is currently unused.
- `eDataDAIPinSec`
  Secondary in/out DAI pin number for a source or sink. This field is currently unused.
- `nChannels`
  Number of audio channels from/to the peripheral
- `oSPORTPeriConfig`
  SPORT peripheral configuration

7.2.5 ADI_SS_FW_SPORT_CONFIG

```
typedef struct ADI_SS_FW_SPORT_CONFIG
{
    ADI_SS_FW_SPORT_NUM     eSportNum;          /*!< SPORT number  */
    ADI_SS_FW_SPORT_HALF    eSportHalf;         /*!< SPORT Half */
    ADI_SS_FW_SPORT_CLK_FS_POL  eSportClkPol;   /*!< SPORT clock polarity */
    ADI_SS_FW_SPORT_CLK_FS_POL  eSportFsPol;    /*!< SPORT frame sync polarity */
    ADI_SS_FW_SPORT_MODE    eSportMode;         /*!< SPORT operation mode */
    uint32_t                nSportWordLen;      /*!< SPORT word length */
}ADI_SS_FW_SPORT_CONFIG;
```

- `eSportNum`:
  Enumeration for SPORT number for a source or a sink.
- eSportHalf:
  Enumeration for SPORT half for a source or a sink.
- eSportClkPol:
  Enumeration for SPORT clock polarity. Rising/Falling.
- eSportFSPol:
  Enumeration for SPORT frame sync polarity. Rising/Falling.
- eSportMode:
  Enumeration for SPORT operation mode indicating I2S/TDM4/TDM8/TDM16
- nSportWordLen:
  SPORT data word length.

7.2.6 SS_SMAP_SSN_INFO

```
typedef struct SS_SMAP_SSN_INFO
{
      SMAP_HOST2TGT_INFO      oHostInfo;
      uint32_t                nNumSSnBuffers;
      ADI_SS_MEM_BLOCK        oSSnBuff[MAX_SMAP_SSN_BUFFERS];
}SS_SMAP_SSN_INFO;
```

SSn info structure which provides SSn specific details including memory details for the target library.

- `oHostInfo`
  SSn information which has to be communicated to target framework from host. Refer 7.2.7 for details.
- `nNumSSnBuffers`
  Number of buffers required for SSn target library.
- `oSSnBuff`
  Array of mem blocks for the SSn buffers. Refer 5.3.3.1.1 for details on ADI_SS_MEM_BLOCK structure.

7.2.7 SMAP_HOST2TGT_INFO

```
typedef struct SMAP_HOST2TGT_INFO
{
    uint32_t          nInPhyChannels;
    uint32_t          nOutPhyChannels;
    int32_t           aInChMap[ADI_SS_FW_MAX_NUM_IN_CHANNELS_SMAP];
    int32_t           aOutChMap[ADI_SS_FW_MAX_NUM_OUT_CHANNELS_SMAP];
}SMAP_HOST2TGT_INFO;
```

This structure provides SSn information which has to be communicated to target framework.

- `nInPhyChannels`
  Number of physical input channels configured from the IC control form of the SigmaStudio GUI. This field is used for total input channel count validation from within the framework.

- `nOutPhyChannels`
  Number of physical output channels configured from the IC control form of the SigmaStudio GUI. This field is used for total output channel count validation from within the framework.

- `aInChMap`
  Array indicating the indices of the input channels used within the SSn. Rest must be -1.

- `aOutChMap`
  Array indicating the indices of the output channels used within the SSn. Rest must be -1.

Terminology

Term	Description
ASRC	Asynchronous Sample Rate Converter
Core	CORE 0 = ARM Cortex A5 CORE 1 = SHARC Core 1 CORE 2 = SHARC Core 2
CCES	CrossCore Embedded Studio
CODEC	Coder / Decoder
ADC	Analog to Digital Converter
ADI	Analog Devices Inc.
API	Application Program Interface
DAC	Digital to Analog Converter
dBFS	Decibels relative to Full Scale
DLL	Dynamic Link Library
GMAP	Global Memory Map
GPIO	General Purpose Input Output
GUI	Graphical User Interface
I2S	Inter-IC Sound
IPC	Inter Processor Communication
ISR	Interrupt Service Routine
LDR	Executable Loader file format
LP	Link Port
PCG	Precision Clock Generator
SMAP	Schematic Memory Map
S/PDIF	Sony/Philips Digital Interface
SPI	Serial Peripheral Interface
SPORT	Serial Port
SSn	SHARC machine code corresponds to SigmaStudio Schematic
SWM	Software Modules
TDM	Time Division Multiplexing
USB	Universal Serial Bus
VISA	Variable Instruction Set Architecture

Table 15: Terminology

References

Reference No.	Description
[1]	AE_42_SS4G_QuickStartGuide.pdf
[2]	AE_42_SS4G_Framework_API_Reference.chm
[3]	AE_42_SS4G_HostControllerGuide.pdf
[4]	AE_42_SS4G_ReleaseNotes.pdf

Table 16: References

[i] ADSP-SC5xx refers to ADSP-SC589/ADSP-SC584/ADSP-SC573/ADSP-SC594 processor in the entire document.