SIGMASTUDIO FOR SHARC - HOST CONTROLLER GUIDE

Document Status	Approved
Approved by	ASH

ANALOG DEVICES, INC.

www.analog.com

Revision List

Table 1: Revision List

Revision	Date	Description
2.1	01.06.2015	Draft Version for M3
2.2	08.06.2015	Incorporated review comments
2.3	12.06.2015	Incorporated review comments
3.0	13.07.2015	Approved and Base-lined for 3.2.0
3.1	06.10.2015	Updated for 3.4.0 - Added details about dual-core MIPS read back command in table 3
4.0	07.10.2015	Approved and Base-lined for 3.4.0
4.1	19.12.2017	Section 4.1.1 updated for SS_CMD_BEGIN bit field details. Section A.1.2.1 updated for SS_CMD_BK_VERSION_INFO payload data.
5.0	20.12.2017	Reviewed, approved and Base-lined for 3.12.0
5.1	26.06.2019	Updated for release 4.4.0
6.0	02.07.2019	Approved and Base-lined for 4.4.0
6.1	18.12.2020	Updated for release 4.6.0
7.0	23.12.2020	Approved and baselined for release 4.6.0
7.1	08.04.2022	Updated for release 4.7.0
8.0	13.04.2022	Approved and baselined for release 4.7.0

Copyright, Disclaimer Statements

Copyright Information

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Table of Contents

List of Figures

List of Tables

List of Equations

1 Introduction

SigmaStudio™ is a development environment from Analog Devices for graphically programming ADI's DSPs. SigmaStudio for SHARC 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. SigmaStudio for SHARC also provides support for Analog Devices Software Modules such as the Dolby® Digital AC3 Decoder. SHARC Software Modules can be obtained separately along with their respective SigmaStudio Plug-Ins.

The environment also extends parameter export and filter coefficient generation support for a 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 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 production environment for a SigmaStudio system is shown in Figure 1.

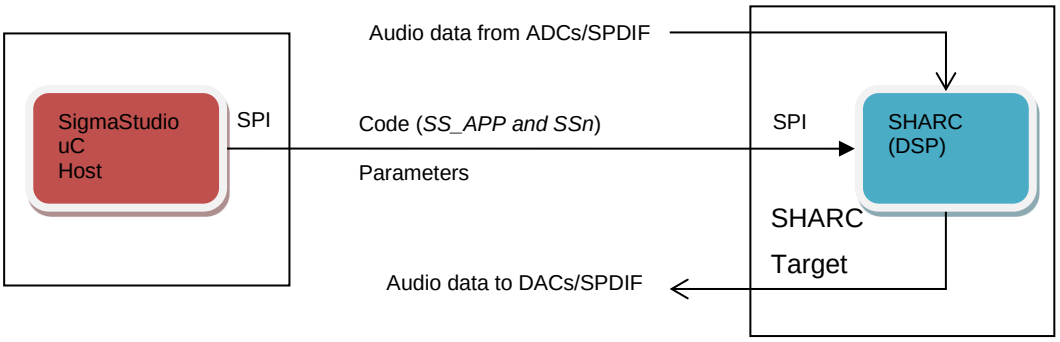

Figure 1: Connecting SigmaStudio uC Host with SHARC Target

As part of booting, the Target Application DXEs, which can initialize audio codec, set up required peripherals and communicate with the SigmaStudio Host through SPI, is loaded in CCES Debug Configurations and run. After successful booting and initialization, the code corresponding to the Schematic and the parameters for the Modules are downloaded to the SHARC Target from the SigmaStudio Host application through SPI. Once the code and parameters are available, the SHARC Target Application can call the API in the SigmaStudio for SHARC Target Library to execute the received code.

The code and parameters that are sent to the SHARC Target Application and the respective memory address offset can be exported from the SigmaStudio development environment. The exported file can be used to program a micro-controller to send the code and parameter control information.

This document describes how to package and send the data from the SigmaStudio Host application so that it can be interpreted by the SHARC Target. Information on how to customize the data for Tuning and specific use cases is given in this document. The Blackfin processor ADSP-BF533 is used to illustrate the micro-controller use cases.

1.1 Scope

The document gives a detailed description on how the code and parameter data can be exported from the SigmaStudio Host application and how micro-controller engineers can use it.

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 SigmaStudio for SHARC product.

Section 3 : this section describes how to export and use system files from SigmaStudio.

Section 4 : this section explains how to packetize the data for communication between the SigmaStudio Host and the SHARC Target.

Section 5 : this section gives details of different type of payloads.

Section 6 : this section details how to use the exported data and thereby control the SHARC Target.

Section 7 : this section explains parameter address calculation done in the Host.

Section 8 : this section details the usage of custom command.

Appendix A: this section contains information on backchannel communication from the SHARC Target to the SigmaStudio Host.

2 Specifications

2.1 Version Information

Refer to the AE_42_SS4G_ReleaseNotes.docx **[2]** for version information of SigmaStudio for SHARC and other software and hardware requirements.

3 SigmaStudio Data Export

The SigmaStudio Host application can export the code and parameters that are to be sent to the SHARC Target once it is booted. This section describes how to export and use the system files.

3.1 Export Settings

The export related settings can be configured through the SigmaStudio Settings window. The "Settings" window can be launched by selecting **Tools → Settings**.

3.1.1 Export Mode

The user can select whether, on export, all files are to be generated or only the XML file is to be generated. 'Export Mode' is present in the 'System Files Export' tab of the Settings window. If 'XML Only' is checked, only the XML file is generated on export. Otherwise, all export system files including the XML file are generated.

3.1.2 Auto Export Mode

Export files can be automatically generated after a successful 'Link-Compile-Download' by enabling 'Auto Export System Files' in the SigmaStudio settings window.

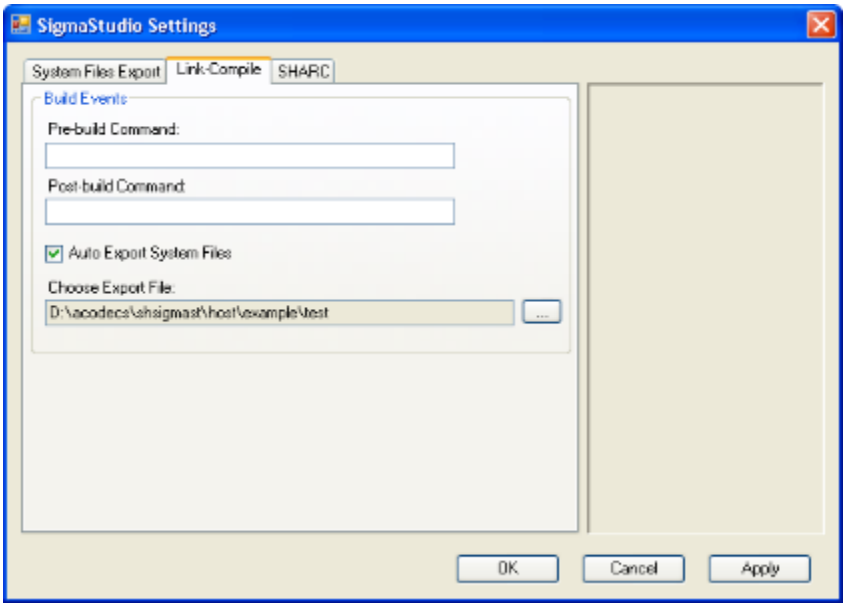

Figure 2: Auto Export option in Settings Window

3.2 Exporting System Files

The system files can be exported after a Schematic is successfully compiled. Once the 'Link-Compile-Download' action is complete, the system files can be exported by selecting **Action → Export System Files**.

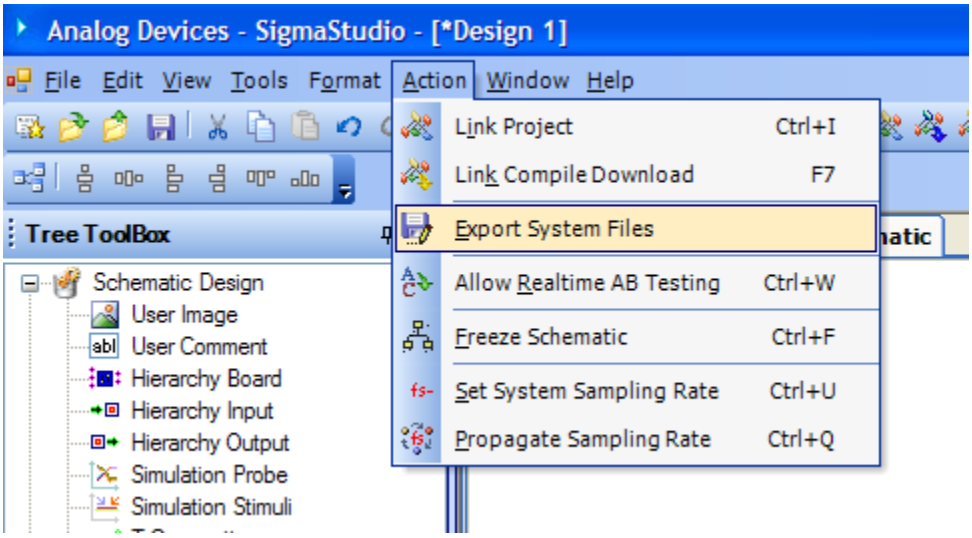

Figure 3: Export System Files

Note: The **Action → Export System Files** menu item is inactive if the Schematic is not compiled after any modifications.

The name of system files and the folder to which it can be saved is to be decided by the user. The files contain code, parameter and version information to be sent to the SHARC Target. The files should be freshly generated every time the Schematic is modified.

3.3 System File Contents and Usage

Examples of the content of the exported files and associated descriptions are given in this section. Files that are named by the user are named "xxx" for ease of representation. The file name also contains information on the IC used and has a text of form "_IC_1" for IC-1. The name can be modified by the user. For ease of representation this is indicated by "_IC_y" in the document.

3.3.1 defines.h

Example Contents

```
#define BufferSize_IC_1       9792
#define NumTransactions_IC_1  4
```

Description

The file contains definitions for the size of data in the system files. The macro `BufferSize_IC_y` gives the total size in bytes of reset, version, code and parameter data. Related data can be found in the file ***TxBuffer_IC_y.dat***. The macro `NumTransactions_IC_y` contains the size of the data contained in ***NumBytes_IC_y.dat***.

3.3.2 xxx.hex

Example Contents

```
0x00 , 0x00 , 0x00 , 0x00 ,
0x3F , 0x19 , 0x99 , 0x9A ,
```

Description

This file contains the parameter data in 8-bit hexadecimal format. The order in the file is such that the most significant byte of a 32-bit hexadecimal word comes first.

3.3.3 xxx.params

Example Contents

```
IC Name               = IC 1
Cell Name             = Tone1
Algorithm Index       = ALG0
Parameter Name = ToneGen1_cos
Parameter Address     = 1
Parameter offset      = 0
Parameter Value       = 0.997858837246895
Parameter Data :
0X3F , 0XDC , 0XEB , 0X50 ,
```

Description

This file lists all parameters used by the individual Cells, the index of the Algorithm within the Cell, parameter name, address, parameter offset, value and 8-bit hexadecimal representation of the parameters. The entire parameter data in 8-bit hexadecimal and corresponding binary representation is also present in the file.

3.3.4 xxx_IC_y.h

Example Contents

```
#include "SigmaStudioFW.h"
#include "inout_IC_1_REG.h"
#define DEVICE_ARCHITECTURE_IC_1                  "ADSP-213xx"
#define DEVICE_ADDR_IC_1                          0x0
#define PARAM_SIZE_IC_1 8192
#define PARAM_ADDR_IC_1 0
ADI_REG_TYPE Param_Data_IC_1[PARAM_SIZE_IC_1] = {
0x00, 0x00, 0x00, 0x00,
--, --,  --,  --,                    };
```

Description

This file lists the IC used, size and offset address for writing code and parameters. Two files are included by this file. ***SigmaStudioFW.h*** has to be taken from the installation folder for SigmaStudio (E.g. "***C:\Program Files\Analog Devices\SigmaStudio 4.76***"). The `Program data` and `Param_Data` buffers declared in the file can be directly used in the Application to access program data and parameter data respectively.

3.3.5 xxx_IC_y_PARAM.h

Example Contents

```
#define MOD_TONE1_COUNT              3
#define MOD_TONE1_DEVICE             "IC1"
#define MOD_TONE1_ALG0_COS_ADDR      1
#define MOD_TONE1_ALG0_COS_VALUE     SIGMASTUDIOTYPE_2_30_CONVERT(0.99785883763)
#define MOD_TONE1_ALG0_COS_TYPE      SIGMASTUDIOTYPE_2_30
#define MOD_TONE1_ALG0_SIN_ADDR      2
#define MOD_TONE1_ALG0_SIN_VALUE     SIGMASTUDIOTYPE_2_30_CONVERT(0.06540443542)
#define MOD_TONE1_ALG0_SIN_TYPE      SIGMASTUDIOTYPE_2_30
```

Description

This file gives all the details regarding parameter data for the individual Cells in the form of macros. The macros used in this header file are defined in the ***SigmaStudioFW.h*** file. The file also lists the parameters to be sent to the SHARC Target processor, in 32-bit hexadecimal representation of floating point or fixed point values.

Note: All the macros in ***SigmaStudioFW.h*** are not defined and therefore the user can customize and use the header file in the Application.

3.3.6 xxx_IC_y_REG.h

Description

This file does not convey any information as such in the current release of SigmaStudio for SHARC. This file can be ignored.

3.3.7 NumBytes_IC_y.dat

Example Contents

```
6,
6,
1682,
8194,
```

Description

The file lists the number of bytes for reset, version information, SHARC Target code and parameters respectively in the file ***TxBuffer_IC_y.dat***.

3.3.8 TxBuffer_IC_y.dat

Example Contents

```
0x00, 0x00,                   /* (0) Reset */
0x00, 0x00, 0x00, 0x00,
0x00, 0x00,                   /* (1) Version Info */
0x20, 0x00, 0x00, 0x00,
0x00, 0x00,                   /* (2) Program Data */
0xFF, 0xFF, 0xFF, 0xFA,
.
.
0x00, 0x00,                   /* (3) Parameter Data */
0x00, 0x00, 0x00, 0x00,
```

Description

The file lists all the parameters to be sent to the SHARC Target in 8-bit hexadecimal format. Note that two zeros each of 1 byte size are added before each new parameter. These two zeros are not required while sending the parameters from a SigmaStudio uC Host to the SHARC Target.

3.3.9 TxMetaBuffer_IC_y.dat

Example Contents

```
0X00, 0X00, 0X00, 0X06,                    /* Size - Reset */
0X00, 0X00, 0X00, 0X06,                    /* Size - Version Info */
0X00, 0X00, 0X06, 0XDA,                    /* Size - Program Data */
0X00, 0X00, 0X06, 0XDA,                    /* Size - Parameter Data */
0X00, 0X00, 0X00, 0X25,                    /* Size - Parameter Metadata */
0x00, 0x00,                                       /* (0) Reset */
0x00, 0x00, 0x00, 0x00,
0x00, 0x00,                                       /* (1) Version Info */
0x20, 0x00, 0x00, 0x00,
0x00, 0x00,                                       /* (2) Program Data */
0x00, 0xFF, 0xFC, 0xFA,
.
0x00, 0x00,                                       /* (2) Parameter Data */
0x00, 0xFF, 0xFC, 0xFA,
.
0x00, 0x08,                                       /* (2) Parameter Meta Data */
```

The format of the content can be represented as shown below

```
/* Beginning of file */
Reset_Data_Size[4]
Version_Data_Size[4]
Program_Data_Size[4]
Parameter_Data_Size[4]
Tuning_Data_Size[4]
Reset_Data[....]
Version_Data[....]
Program_Data[....]
Parameter_Data[....]
/* Tuning Data*/
CellName_AlgoNo_String_Length1, CellName_AlgoNo_String1,Parameter_Base_Address1,
CellName_AlgoNo_String_Length2, CellName_AlgoNo_String2,Parameter_Base_Address2,
CellName_AlgoNo_String_LengthN, CellName_AlgoNo_StringN,Parameter_Base_AddressN
/*End of file*/
```

Description

The data in this file contain Reset, Version, Program Data, Parameter Data and information required for Tuning individual Cells. The data can be flashed or stored to a memory location and then accessed from there. The various fields in the header file are as follows:

Name: `Reset_Data_Size`

Type: 4bytes, big endian unsigned integer

Description: Signifies the size of the Reset data.

Name: `Version_Data_Size`

Type: 4bytes, big endian unsigned integer

Description: Signifies the size of the Version data.

Name: `Program_Data_Size`

Type: 4bytes, big endian unsigned integer

Description: Signifies the size of the Program data.

Name: `Parameter_Data_Size`

Type: 4bytes, big endian unsigned integer

Description: Signifies the size of the Parameter data.

Name: `Tuning_Data_Size`

Type: 4bytes, big endian unsigned integer

Description: Signifies the size of the Tuning meta data.

Name: `Reset_Data`

Type: Byte array, big endian hexadecimal

Description: Data for Reset information. Note that first two bytes need not be sent to SHARC Target.

Name: `Version_Data`

Type: Byte array, big endian hexadecimal

Description: Data for Version information. Note that first two bytes need not be sent to SHARC Target.

Name: `Program_Data`

Type: Byte array, big endian hexadecimal

Description: Data for Program information. Note that first two bytes need not be sent to SHARC Target.

Name: `Parameter_Data`

Type: Byte array, big endian hexadecimal

Description: Data for Parameter information. Note that first two bytes need not be sent to SHARC Target.

Name: `CellName_AlgoNo_String_Length1`

Type: 2bytes, big endian unsigned integer

Description: Number of bytes for string with Cell name and Algorithm number of "Nth" tuneable block in Schematic.

Name: `CellName_AlgoNo_String1`

Type: Character array

Description: Character string without null containing the Cell name and Algorithm number of "Nth" tuneable block in Schematic.

Name: `Parameter_Base_Address1`

Type: 4bytes, big endian unsigned integer

Description: Offset address of the starting parameter of "Nth" tuneable block in the Schematic specified by the Cell name and Algorithm number string.

3.3.10 xxx.xml

Example Contents

Only part of the output XML file is shown below in the example.

```
<?xml version="1.0" encoding="utf-8"?>
<Schematic>
    <IC>
        <Name>IC 1</Name>
       <Meta>
            <Name>SMAP</Name>
            <Size>240</Size>
            <Data>0x00, 0x00, 0x00, 0x12, 0x00, 0x00, 0x00, 0x04,
                  . . . .
                0x00, 0x2C, 0x8E, 0x54, </Data>
        </Meta>
        <Meta>
            <Name>Reset</Name>
            <Size>4</Size>
            <Data>0x00, 0x00, 0x00, 0x00, </Data>
        </Meta>
        <Meta>
            <Name>Version Info</Name>
            <Size>4</Size>
            <Data>0x21, 0x00, 0x00, 0x00, </Data>
        </Meta>
        <Program>
            <Name>Program Data</Name>
            <Address>0</Address>
            <Size>4580</Size>
            <Data>0x00, 0x01, 0x00, 0x01, 0x00, 0x00, 0x07, 0x3E, 0x00, 0x01, 0x07,
                  . . . .
                  0x08, 0x3F, 0x00, 0x01, 0x00, 0x01, </Data>
        </Program>
        <Parameter>
            <Name>Parameter Data</Name>
            <Address>0</Address>
            <Size>376</Size>
            <Data>0x06, 0x0A, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F, 0x73, 0xD5,
                  . . . .
                  0x00, 0x00, </Data>
        </Parameter>
        <Module>
            <CellName>Gen Filter1</CellName>
            <Algorithm>
                <AlgoName>ALG0</AlgoName>
                <Description>1 Channel Single Precision Floating Point( 1 ) :
Gain[ 0 ],Q[ 1.41 ],Boost1[ 0 ],Frequency1[ 1000 ],PhaseShifted[ False ],Type[ 0 ];</Description>
                <Address>6</Address>
                <ModuleParameter>
                    <Name>GenSecOrderFilt10B1</Name>
                    <Type>Float32</Type>
                    <Offset>0</Offset>
```

Description

The exported XML file contains SMAP (only in case ADSP-SC58x), Reset, Version, Program Data, Parameter Data and the information required for Tuning individual Cells. The information can be parsed using an xml parser. The XML file also includes the GUI parameter information of the Modules in the Schematic. The schema of the XML is illustrated below.

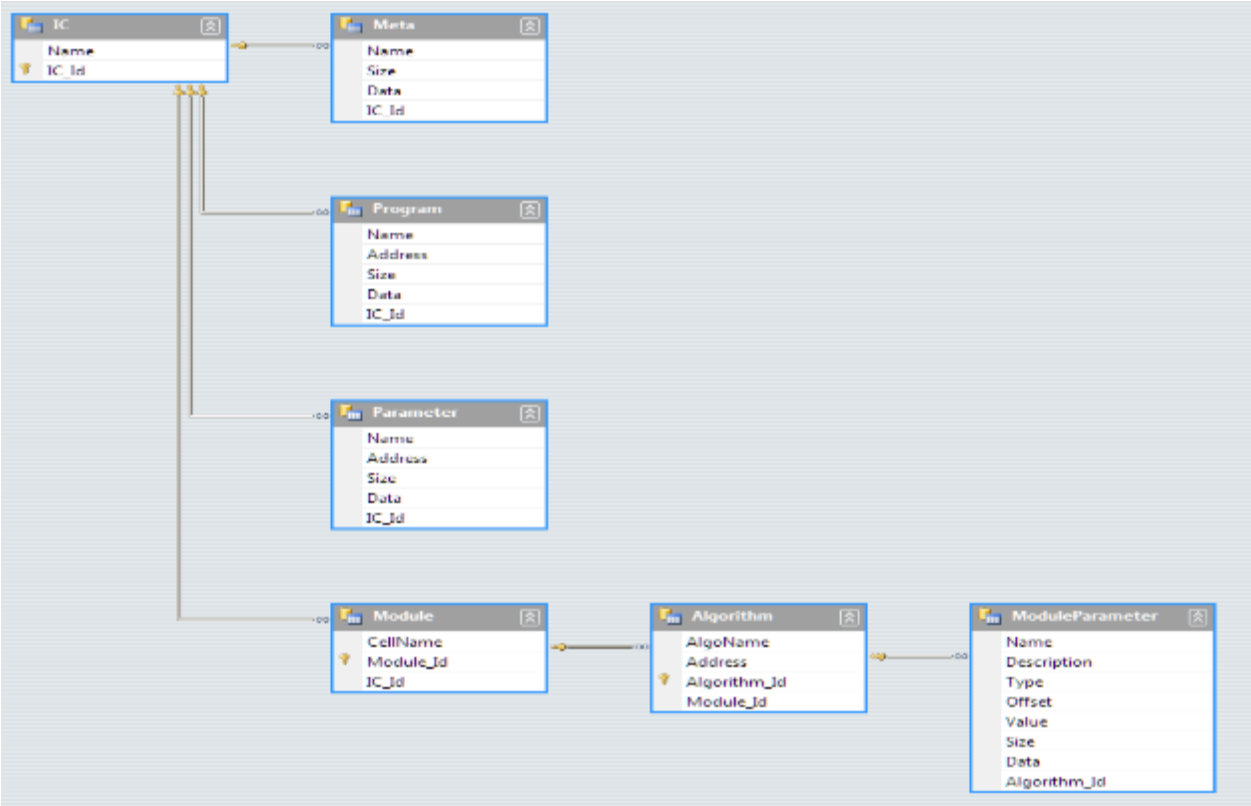

Figure 4: XML Export File Schema Specification

IC:	Name of the *Processor (ICs/DSPs)* in the Hardware Configuration Window of the schematic. There can be more than one IC for a multiple SSn Schematic.
Name:	Name of the SSn (sub Schematics). Same as the Name of the IC. Default IC name is "IC 1".
Meta:	Meta information can be any value from { Reset, Version Info, SchematicVersionTag}
Name:	Name of the Meta information.
Description:	Meta information name can be any value from { Reset, Version Info, SchematicVersionTag}
Program:	Program Memory
Name:	Name of the Program memory. This string can be from {Program Data, Program DataB}
Address:	Address offset to which the program has to be loaded. The start address is always the memory pointer pointed by Block 1/Block7 in ADI_SS_MEM_MAP respectively for Program Data/ Program DataB.
Size:	Size in Bytes (2 bytes less than the size reported in export file).
Data:	Data in Hexadecimal (Most significant byte comes first).

Parameter:	Parameter Memory.
Name:	Name of the Parameter memory. This string can be from {Parameter Data}
Address:	Address offset to which the initial parameter has to be loaded. The start address is always the memory pointer pointed by Block 5 in ADI_SS_MEM_MAP. This is always 0.
Size:	Size in Bytes (2 bytes less than the size reported in export file)
Data:	Size in Bytes (2 bytes less than the size reported in export file).
Module:	Details of individual Cells (A Cell is the basic graphical entity with one or more Algorithms).
CellName:	Name of the Cell
Algorithm:	Details of individual Algorithm (An Algorithm is the basic processing entity with one or more Parameters associated to it)
AlgoName:	Index of the Algorithm within the Cell. There can be multiple Algorithms within a Cell.
Description:	Algorithm Description with Names of Algorithm and the GUI parameters.
Address:	Algorithm Starting address (offset from the beginning of the parameter memory)
ModuleParameter:	Details of the individual parameter (Parameter is the basic Tuning entity with one or more data units).
Name:	Name of the parameter
Type:	Type of the parameter. Type can be {Int32, Float32, Hex}
Offset:	Offset from the Algorithm starting address
Value:	Value of the data element. Note: This is applicable only if the Parameter has a single data unit. In case of multiple data units, only the hexadecimal values are provided. For example, <ModuleParameter> <Name>Taps</Name> <Type>Int32</Type> <Offset>0</Offset> <Value>5</Value>

<Size>4</Size>

<Data>0x00, 0x00, 0x00, 0x05, </Data>

</ModuleParameter>

<ModuleParameter>

<Name>FIRFiltBlock1coeff</Name>

<Type>HexArray</Type>

<Offset>1</Offset>

<Size>8</Size>

<Data>0x3F, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, </Data>

</ModuleParameter>

In the above example, the first parameter 'Taps' is a single data unit with a "Value" equal to 5. Whereas, the second parameter 'FIRFiltBlock1coeff' has multiple data units and hence, only hexadecimal values are provided.

Size: Size in Bytes

Data: Data in Hexadecimal (Most significant byte comes first)

4 Packetizing Data

The data received while exporting the code and parameters has to be packetized properly before sending it to the SHARC Target board.

4.1 Communication Protocol

All the communication between the SigmaStudio Host PC and the SHARC Target use the following protocol:

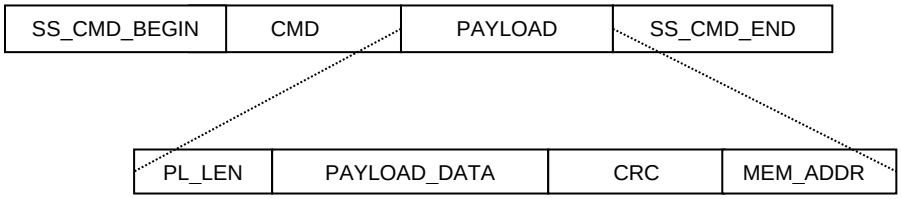

Figure 5: Communication Protocol

Each packet contains a begin command, size of the packet, payload data, CRC value, SHARC Target memory type and an end command. The individual contents of the packet are listed below.

4.1.1 SS_CMD_BEGIN: (32-bit word)

This field indicates beginning of the packet. Refer to Table 2 for 32-bit hexadecimal value of this field. Bits 12-15 of “SS_CMD_BEGIN” is used for indicating the “Core Id” and “Instance Id” in SigmaStudio for SHARC (ADSP-SC5xx). The details of those bit fields are described in Figure 6.

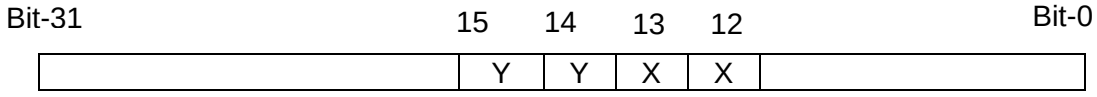

Figure 6: Core ID and Instance ID in SS_CMD_BEGIN

The bit fields 15-14 (Y) is used to indicate the Core ID i.e., 00 – ARM Core0 (not used for Host communication), 01 – SHARC Core1 and 10 – SHARC Core2.

The bit fields 13-12 (X) is used to indicate the Instance ID. For a single instance schematic, this fields shall be always 00.

4.1.2 CMD: (32-bit word)

The CMD field indicates the nature of payload in the packet. The value of this field can vary from packet to packet. Refer to Table 2 for possible 32-bit hexadecimal values of this field.

4.1.3 Payload

This contains the data to be transferred from the SigmaStudio Host to the SHARC Target. The contents of payload are given below.

4.1.4 PL_LEN: (32-bit word)

This field indicates the total length of the payload, including CRC. The maximum payload length should be 1024 words (4096 Bytes) or less.

4.1.4.1 PAYLOAD_DATA:

Actual payload in the packet appears in this portion. The payload can be data/code/parameter.

4.1.4.2 CRC: (32-bit word)

The CRC field is used for protection. The current version uses simple checksum. The code used to compute CRC data is given below. Here `pData[]` is the buffer of type `unsigned int`.

```
for (i = 0; i < nSizeBytes/4; i++)
{
    nSum += pData[i];
}
nCRC = ~nSum + 1;
```

The SHARC Target can compute the CRC of the received packet either only on the payload or on the entire packet based on the configuration parameter. The above example shows CRC checksum computed only on the payload. Alternatively, the CRC checksum can also be computed on the entire packet.

4.1.4.3 MEM_ADDR: (32-bit word)

This field indicates the memory address to which the code/parameter should be loaded. This field is used only when the command is either `SS_CMD_BLOCK_SAFE` or `SS_CMD_PARAMETER_NO_SAFE`.

4.1.5 SS_CMD_END: (32-bit word)

This 32-bit word indicates the end of a communication packet. Refer to Table 2 for 32-bit hexadecimal value of the field.

4.2 Commands for Communication

All the commands and fields indicating begin/end with their 32-bit hexadecimal values are listed in Table 2.

Command	Description	Value	Payload
SS_CMD_BEGIN	Signifies the beginning of packet	0xf4190be6	None

SS_CMD_PROGRAM_SSN	Indicates that the packet contains code	0xfaad0552	SigmaStudio generated code
SS_CMD_PARAMETER_SAFE	Indicates that the packet contains safe-load parameters.	0xa5015afe	Up to 5 sets of safe-load parameters and respective addresses
SS_CMD_PARAMETER_NO_SAFE	Indicates that the packet contains a block of parameters that can be directly loaded on to the parameter memory. Used for initial parameter set and block parameter update during Tuning.	0xffa1f05e	Block of parameters
SS_CMD_BLOCK_SAFE	Indicates that the packet contains a block of parameters to be safe-loaded. Used for block parameter update during Tuning.	0x4ea5b15a	Block parameters
SS_CMD_CMD1	Predefined command 1. Used for sending Reset.	0xf3f20c0d	Word and Block DMA receive mode flag
SS_CMD_CMD2	Predefined command 2. Used for sending version information.	0xf3e20c1d	Version information
SS_CMD_CMD3	Predefined command 3. Used for sending read request.	0xf3d20c2d	Read request
SS_CMD_CMD4	Predefined command 4 that calls a call-back function with the payload as arguments.	0xf3c20c3d	User defined data that is interpreted by the call-back function
SS_CMD_CMD5	Payload containing custom command that can be obtained by calling adi_ss_comm_GetProperties() function	0xf3b20c4d	User defined command
SS_CMD_CMD6	Predefined command 6. Used for sending Schematic bypass flag.	0xf3a20c5d	Bypass flag
SS_CMD_CMD7	Indicates that the packet contains SMAP data. This is applicable or used only when the IC is ADSP-SC5xx.	0xf3920c6d	SMAP
SS_CMD_PARTIAL_END	Partial end of current packet	0xe1d21e2d	None
SS_CMD_END	End of current packet	0xf1d20e2d	None

Table 2: Communication Commands

4.3 Read Request Payload

PAYLOAD_DATA inside read request packets are of length 3 or 4 words depending on the read request type and is described below.

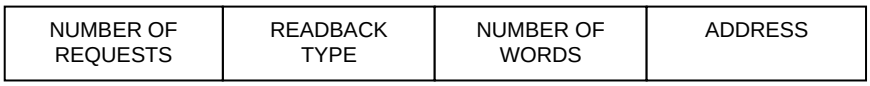

Figure 7: Read Request Payload

4.3.1 NUMBER_OF_REQUESTS: (32-bit word)

The field indicates the number of read requests in the current read request packet. This should always be set to 1.

4.3.2 READBACK_TYPE: (32-bit word)

The field indicates the type of read request. Refer to the table below for more details.

Command	Description	Value
SS_CMD_BK_MIPS_VALUE	Indicates read request for MIPS	0x00CE6319U
SS_CMD_BK_DC_MIPS_VALUE	Indicates read request for Dual-Core MIPS	0x00DC6239U
SS_CMD_BK_VERSION_INFO	Indicates read request for Version	0x003EDC12U
SS_CMD_BK_READ_VALUE	Indicates read request from Parameter memory. Read back, Level Detector etc. Modules use this command to read value from memory.	0x00BC543AU
SS_CMD_BK_ERROR_CODE	Indicates read request for code download status.	0x00F7E808U
SS_CMD_BK_ACKNOWLEDGE	Indicates acknowledgement for successful read.	0x00000000

Table 3: Read-back Type

4.3.3 NUMBER_OF_WORDS: (32-bit word)

This field indicates the number of values to be read from the SHARC Target. This field is valid only when the read type is SS_CMD_BK_READ_VALUE. This field should be 0 for other read types. For memory read, the first 2 nibbles of the NUMBER_OF_WORDS field corresponding to read request payload must always be 0x30. Also, the last nibble must always be 0x0C. For parameter read, the first 2 nibbles of the NUMBER_OF_WORDS field corresponding to read request payload must be 0x00. The last 24-bits (remaining 6 nibbles) of the 'NUMBER_OF_WORDS' field indicate the number of words to be read from the parameter offset specified as part of the 'ADDRESS' field of the read request payload.

4.3.4 ADDRESS: (32-bit word)

This field indicates the offset, from the start of parameter memory to the memory location from where the value has to be read. This field is included in the payload only when the read type is SS_CMD_BK_READ_VALUE. Therefore, the payload size is 3 for other read types.

5 Payload Types

5.1 Code

'Code' refers to the compiled binary code of the entire Schematic. SS_CMD_PROGRAM_SSN command indicates that the packet contains code payload. MEM_ADDR field is not used.

5.2 Initial Parameter

'Initial Parameter' refers to the entire parameter set of the Schematic. SS_CMD_PARAMETER_NO_SAFE command indicates that the packet contains complete block parameter payload. MEM_ADDR field is not used.

5.3 Block Parameter

'Block Parameter' refers to a block of parameters sent to the SHARC Target as part of Tuning the Schematic. The SS_CMD_PARAMETER_NO_SAFE command indicates that the packet contains block parameter payload. The MEM_ADDR field indicates the offset of the block from the start of the parameter buffer. When the block parameter is received on the SHARC Target, it is immediately copied to the parameter buffer.

5.4 Safeload Parameter

'Safeload Parameter' refers to payload of up to 5 sets of parameter and address combination sent to the SHARC Target as part of Tuning the Schematic. The SS_CMD_PARAMETER_SAFE command indicates that the packet contains safeload parameter payload. MEM_ADDR field is not used. When safeload parameters are received on the SHARC Target, it is not immediately copied to the parameter buffer. Parameters are updated just before the execution of the Schematic code. This ensures that the parameter updates are not performed during the execution of the Algorithm.

5.5 Block Safeload Parameter

'Block Safeload Parameter' refers to a block of parameters sent to the SHARC Target as part of Tuning the Schematic. SS_CMD_BLOCK_SAFE command indicates that the packet contains block safeload parameter payload. MEM_ADDR field indicates the offset of the block from the start of the parameter buffer. When block safeload parameter is received on the SHARC Target, it is not immediately copied to the parameter buffer. Parameters are updated only after ensuring that the SigmaStudio Schematic code is not being executed. If the code is being executed, the update waits until the code execution completes.

6 Using Exported Data

6.1 Deriving Payload Contents

The exported files ***NumBytes_IC_y.dat*** and ***TxBuffer_IC_y.dat*** need to be modified before directly including in the Application. A variable can be declared in such a manner that it contains all values in ***TxBuffer_IC_y.dat***. Thus, the content of ***TxBuffer_IC_y.dat*** can be modified as given below

```
unsigned char aCodeParam[] = {
0x00, 0x00,                   /* (0) Reset */
0x00, 0x00, 0x00, 0x00,
0x00, 0x00,                   /* (1) Version Info */
0x20, 0x00, 0x00, 0x00,
0x00, 0x00,                   /* (2) Program Data */
0xFF, 0xFF, 0xFF, 0xFA,
----------------------
----------------------
};
```

Similarly a variable of data type `int` can be used to modify ***NumBytes_IC_y.dat***. After the modifications are done, the files can be directly included in the Application source code.

The "***.h***" files generated during export can be directly included in the Application. In addition, these header files include ***SigmaStudioFW.h***. This has to be taken from the installation folder for SigmaStudio (E.g. "***C:\Program Files\Analog Devices\SigmaStudio 4.6***")

Note that every parameter in ***TxBuffer_IC_y.dat*** has two preceding bytes with zero value. These are not part of the parameter value and should be skipped before using data for communication. In addition, the file ***xxx_IC_y.h*** contains code, reset version etc. in header file format.

Alternately, the file "***TxMetaBuffer_IC_y.dat***" can be used to derive the start address of the parameter required by the SigmaStudio Host for tuning the parameter based on its name. The parameter name can be obtained from the file "***TxMetaBuffer_IC_y.dat***". The parameter name is a combination of "Cell Name" and "Algorithm Index" which are present in the file "***xxx.params***". The parameter name is nothing but "Cell Name" + "_" + "Algorithm Index" capitalized.

6.1.1 SMAP

Command used: SS_CMD_CMD7

This is applicable only when the IC used is ADSP-SC5xx.

The entire set of SMAP data is present in the file ***xxx_IC_y.h*** in hexadecimal `unsigned char` format under the variable name `SMAP_Data[]`. The size of the SMAP structure can be taken from the macro `SMAP_SIZE`. Note that there are no dummy bytes when the data is taken from this header file.

Alternatively, the parameter data can be taken from ***TX_Buffer_IC_y.dat*** using the offset and size from ***NumBytes_IC_y.dat*** file. Care should be taken to skip the initial two bytes in this case.

6.1.2 Reset

Command used: SS_CMD_CMD1

The reset parameter consists of a 32-bit hexadecimal word with value 0 or 1. Sending this parameter with the associated command to the SHARC Target prepares the SHARC Target to receive new SSn code and parameter data. When the parameter is 0, the SHARC Target receives the data sent by SPI as single words and processes it. On sending a value of 1, the SHARC Target receives code and parameter as a block and processes the CRC check on the next call of `adi_ss_schematic_process()` function.

6.1.3 Version Information

Command used: SS_CMD_CMD2

The version information is also a 32-bit hexadecimal value whose first 2 bytes give the version of the process API used in the Application. The version of the process API used in the Application for this release is 2.1.0. This data is present in ***TX_Buffer_IC_y.dat*** with the most significant byte coming first. The offset to access the version information and its size can be obtained from the ***NumBytes_IC_y.dat*** file. Although the size of the version information is given as 6 bytes, the first two bytes are dummy values and have to be omitted to obtain the real version information value.

6.1.4 Schematic Code

Command used: SS_CMD_PROGRAM_SSN

The code for Schematic is present in ***TX_Buffer_IC_y.dat*** as the third parameter. Its size and offset to access it can be obtained from ***NumBytes_IC_y.dat*** file. The Schematic code also has 2 dummy bytes in the beginning and these have to be skipped to access the code. The size of the code varies for each Schematic and should be updated when a Schematic is modified. The data present is such that the most significant byte comes first.

6.1.5 Parameters

Command used: SS_CMD_PARAMETER_NO_SAFE / SS_CMD_PARAMETER_SAFE / SS_CMD_BLOCK_SAFE

The entire set of parameters is present in the file ***xxx_IC_y.h*** in hexadecimal `unsigned char` format under the variable name `Param_Data[]`. The size of the parameter table can be taken from the macro `PARAM_SIZE`. Note that there are no dummy bytes when the data is taken from this header file.

Alternatively the parameter data can be taken from ***TX_Buffer_IC_y.dat*** using the offset and size from ***NumBytes_IC_y.dat*** file. Care should be taken to skip the initial two bytes in this case.

6.2 Sending Schematic Code

The Schematic code has to be sent to the SHARC Target when a new Schematic is generated or an existing Schematic is modified. In order to successfully send the Schematic code to the SHARC Target, the following steps have to be executed.

1. Send SMAP for the schematic (applicable only for ADSP-SC5xx). This reserves the necessary memory blocks in the Target.
2. Send Reset parameter: This prepares the SHARC Target for receiving the code
3. Send Version Information: This ensures compliance between the SHARC Target and the SigmaStudio Host.
4. Send Schematic Code: Algorithm code for the SHARC Target
5. Send Parameters: Parameters for the code to work with.

Once the entire data is sent to the SHARC Target running SigmaStudio, the SHARC Target can start processing the data.

Note: There should be some delay required between SMAP, Code and Parameter data when sending data from Microcontroller host. The SigmaStudio host use 100 msec between SMAP, Code and Parameter data which is required for target framework initializations.

6.3 Parameter Reloading

If the Schematic code and parameters are successfully sent to the SHARC Target then the block of parameters for the particular Schematic alone can be updated. A step-by-step procedure is given below.

1. Send Reset parameter: This prepares the SHARC Target for receiving the code
2. Send Version Information: This ensures compliance between the SHARC Target and the SigmaStudio Host.
3. Send Parameters: Parameters for the code to work with.

Once the entire data is sent to the SHARC Target running SigmaStudio, the SHARC Target can start processing the data.

6.4 Sending Parameter for Tuning

To send a parameter for Tuning, both the parameter and its address offset need to be sent to the SHARC Target. The parameter data sent should be the 32-bit hexadecimal equivalent of floating point or fixed point value, followed by the offset address location of the parameter.

Every single value to be written to the parameter memory during Tuning should be succeeded by the address location to which it is to be written. The values should then be packetized and sent to the SHARC Target. Hence the data to be packetized is of the form

1. Parameter value in 32-bit hexadecimal
2. Offset address location in 32-bit hexadecimal.

Note that Reset and Version packets are not required during Tuning.

6.4.1 Flashing Data and Accessing it for Tuning

The file ***TxMetaBuffer_IC_y.dat*** has a tune table which contains parameter data, its address and a string to uniquely identify the Algorithm. The file can be directly flashed to the memory of a micro-controller and can be directly accessed from there. This approach has the advantage that once the Schematic is frozen and only the Tuning has to be performed, a Tuning engineer can do so without modifying the micro-controller code.

6.4.2 Using Header Files

The parameter data and its offset location can be found as a macro in the file ***xxx_IC_y_PARAM.h***. The file ***xxx_IC_y.h*** contains a parameter data buffer named `Param_Data_IC_<y> []` and a macro defining the size of the parameter data termed `PARAM_SIZE`. Since the data in the buffer `Param_Data_IC_<y> []` is given in bytes, the offset address can be multiplied with 4 to derive the address of the parameter data. An example use case is given below.

Let the offset address of the Mute Algorithm in a Schematic be given by the macro `MOD_MUTE1_MUTE_ADDR`, the value of parameter can be derived as

```
aParamMute[0] = Param_Data[MOD_MUTE1_MUTE_ADDR*4]<< 24;
aParamMute[0] += Param_Data[MOD_MUTE1_MUTE_ADDR*4 + 1]<< 16;
aParamMute[0] += Param_Data[MOD_MUTE1_MUTE_ADDR*4 + 2]<< 8;
aParamMute[0] += Param_Data[MOD_MUTE1_MUTE_ADDR*4 + 3];
```

7 Parameter Address Calculation

7.1 IIR Filters

Coefficient arrangement in SHARC memory of IIR fitters is different for Type 1 and Type 2 filters.

Type1 Cascaded 3 stage filter

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \cdot \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \cdot \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \cdot$$

Equation 1: Type 1 Cascaded 3 Stage IIR Filter

Type 1 Filter, n[th] stage (where n is 1, 2 or 3 for 3 stage cascaded filter)

b2 → GenSecOrderFiltBlk12Bn → c[n-1][0]

b1 → GenSecOrderFiltBlk11Bn → c[n-1][1]

b0 → GenSecOrderFiltBlk10Bn → c[n-1][2]

a2 → GenSecOrderFiltBlk12An → c[n-1][3]

a1 → GenSecOrderFiltBlk11An → c[n-1][4]

Type2 Cascaded 3 stage filter

$$H(z) = b_0' \frac{1 + b_1' z^{-1} + b_2' z^{-2}}{1 + a_1' z^{-1} + a_2' z^{-2}} \cdot \frac{1 + b_1' z^{-1} + b_2' z^{-2}}{1 + a_1' z^{-1} + a_2' z^{-2}} \cdot \frac{1 + b_1' z^{-1} + b_2' z^{-2}}{1 + a_1' z^{-1} + a_2' z^{-2}}$$

$$b_0' = b_0 . b_1 . b_2$$

Equation 2: Type 2 Cascaded 3 Stage IIR Filter

Type 2 Filter, n[th] stage (where n is 1, 2 or 3 for 3 stage cascaded filter)

b'0 → GenSecOrderFiltT2Blk10Bn → c[4*number of stages]

b'1 → GenSecOrderFiltT2Blk11Bn → c[n-1][3]

b'2 → GenSecOrderFiltT2Blk12Bn → c[n-1][2]

a'1 → GenSecOrderFiltT2Blk11An → c[n-1][1]

a'2 → GenSecOrderFiltT2Blk12An → c[n-1][0]

For Type2 IIR, b0 coefficient is same for all the stages. The offset values for b0 is the same for all the stages.

Implementation	0B	1B	2B	1A	2A
Type 1	b0	b1	b2	-a1	-a2
Type 2	Product of b0 of all cascaded stages	b1' = b1/b0	b2' = b2/b0	a1' = a1	a2' = a2

Table 4: Filter Coefficient Computation

Parameter arrangement of Type1 cascaded filters in memory

	c[0][0]	c[0][1]	c[0][2]	c[0][3]	c[0][4]	c[1][0]	c[1][1]	c[1][2]	c[1][3]	c[1][4]	c[2][0]	c[2][1]	c[2][2]	c[2][3]	c[2][4]
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Growth	1					2					3				
	ALG0														

Parameter arrangement of Type2 cascaded filters in memory

	c[0][0]	c[0][1]	c[0][2]	c[0][3]	c[1][0]	c[1][1]	c[1][2]	c[1][3]	c[2][0]	c[2][1]	c[2][2]	c[2][3]	c[12]
Offset	0	1	2	3	4	5	6	7	8	9	10	11	12
Growth	1				2				3				
Add Count	ALG1												

Figure 8: Parameter Arrangement

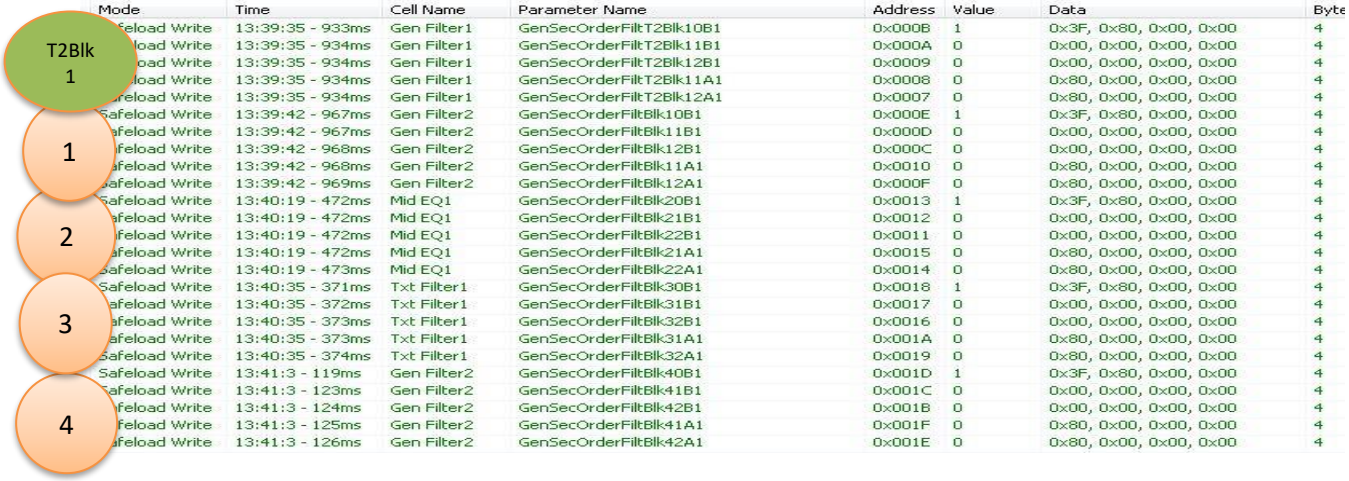

Figure 9: Export Example Capture Window

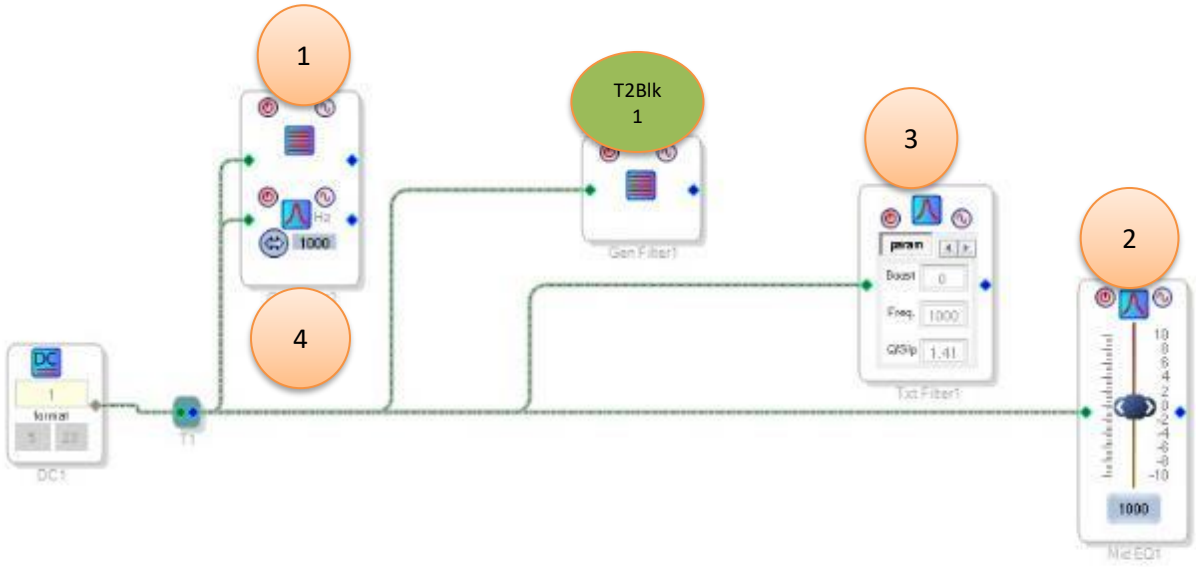

Figure 10: Export Example Schematic

8 Sending Custom Commands

Custom commands can be sent from the SigmaStudio Host to the SHARC Target in 2 different ways; using `SS_CMD_CMD4` and `SS_CMD_CMD5`.

8.1 Call-back Method

A packet sent to the SHARC Target with `SS_CMD_CMD4` as the command executes a call-back function with the payload information as the function arguments. The call-back function to be executed can be assigned through the configuration parameter `pfCommCallBack` in the communication configuration structure `ADI_SS_COMM_CONFIG` passed to `adi_ss_comm_init()`.

```
/* Parameters for SigmaStudio communication instance initialization */
oCommConfig.baudRateTx = 0x100;       /* currently not used */
oCommConfig.nSelectSPI = SELECT_SPI0; /* SELECT_SPI0 : Primary SPI */
oCommConfig.bCRCBypass = 0;           /* Bypass CRC check */
oCommConfig.pfCommCallBack = app_ss_comm_callback_cmd4;

/* SigmaStudio for SHARC communication instance initialization */
eCommResult = adi_ss_comm_init(hSSComm, &oCommConfig);
```

Prototype of the call-back function is shown below.

```
void app_ss_comm_callback_cmd4(int               *pCommPayloadBuff,
                               int                nPayloadCount,
                               ADI_SS_SSN_HANDLE  hSSn)
{
      ..................
      ..................
      ..................
}
```

Parameters

Name: `pCommPayloadBuff`

Type: `Int *`

Direction: Input

Description: Payload buffer pointer.

Name: `nPayloadCount`

Type: `int`

Direction: Input

Description: Size of the payload.

Name: `hSSn`

Type: `adi_ss_sample_t *`

Direction: Input

Description: Handle to the SigmaStudio module instance which received the command.

8.2 Get Properties Method

A packet sent to the SHARC Target with `SS_CMD_CMD5` as the command is received by the SigmaStudio instance on the SHARC Target. The received data/command is extracted by the Application from the SigmaStudio instance using the `adi_ss_comm_GetProperties()` function. Data is available in `oGetProperties.pSSnBuf`, where `oGetProperties` is the properties structure of type `ADI_SS_COMM_PROPERTIES` passed to `adi_ss_comm_GetProperties()`.

```
adi_ss_comm_GetProperties(hCommHandle, &oGetSSCommProp);;

if(oGetSSCommProp.bCustomCmdRcvd)
{
    /* data is available in oGetSSCommProp.pSSnBuf */
       oGetSSCommProp.bCustomCmdRcvd = false;
}
```

8.3 Custom Command Example

8.3.1 Reset Communication Instance from SigmaStudio Host

The custom command `SS_CMD_CMD4` can be used to reset the communication instance from an error state. This is required to continue processing after encountering a communication error. The below code snippet shows how to trigger a communication reset from the SigmaStudio Host with the help of a call-back function.

```
ADI_SS_COMM_HANDLE hSSComm = 0;
ADI_SS_COMM_CONFIG oCommConfig;

void app_ss_comm_callback_cmd4(int               *pCommPayloadBuff,
                               int                nPayloadCount,
                               ADI_SS_SSN_HANDLE  hSSn)
{
      int payLoadHdr = pCommPayloadBuff[0];

       switch(payLoadHdr)
       {
          /* SigmaStudio Host should send cmd4 with payload as 2 to reset the comm */
          case 2:
               if(hSSComm != NULL)
               {
                     adi_ss_comm_Reset (hSSComm, &oCommConfig);
               }
               break;
       }
}
```

A. Read-Back Communication

The source code to packetize and send the read-back information is available with the released Application. Refer to the files ***backchannel.c*** and ***backchannel.h*** for source code and further details (ADSP-213xx and ADSP-214xx processor series).

A.1 Back Channel Protocol (SHARC Target to SigmaStudio Host)

All communications from the SHARC Target to the SigmaStudio Host PC application use a byte based protocol as shown below.

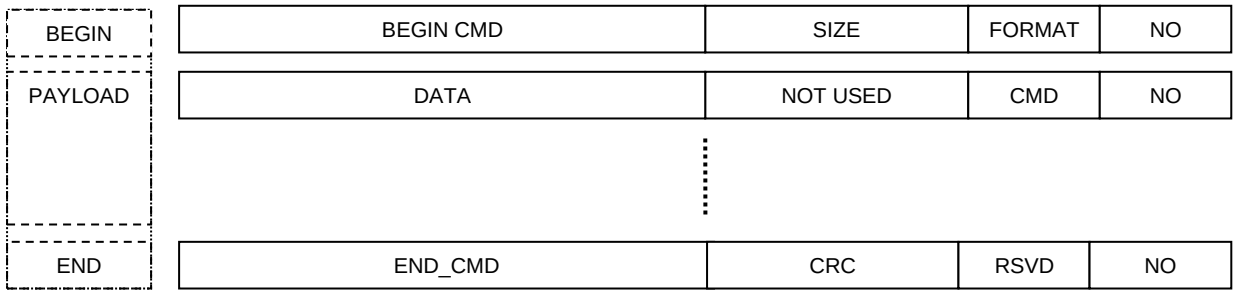

Figure 11: Back Channel Communication SPI Packet Format

The lower 4 bits of each word enumerate each word. The words in the packet are enumerated in ascending order. The figure given above is used as a reference for listing individual content of packet.

A.1.1 BEGIN: (32-bit)

The BEGIN word of size 32-bit indicates the beginning of packet. This contains a `BEGIN_CMD` command, the size of sub packets in bytes, format of sub packet and word number.

A.1.1.1 BEGIN_CMD: (16-bit)

This 16-bit word indicates the beginning of a communication packet.

A.1.1.2 SIZE: (8-bit)

This indicates the total length of data in payload in bytes.

A.1.1.3 **FORMAT: (4-bit)**

This field designates the format of the packet contents. The packet contents can be in 8-bit format or 24-bit format. When the content of the packet is in 8-bit format, the MSB of the field is 1, else 0.

A.1.1.4 **NO: (4-bit)**

This field indicates the serial number of each 32-bit word.

A.1.2 PAYLOAD: (32-bit)

Data to be transferred from the SHARC Target to the SigmaStudio Host is available in the payload. Each word contains 16-bit data, 4-bit command and 4-bit word number.

A.1.2.1 **DATA: (16-bit)**

The 16-bits in MSB of every payload contain actual payload data. Float values are split into two 32-bit payload words with each payload word containing 16-bits of data. When a 32-bit word is split into two 32-bit payload words, the LSB bits of the float word are put inside the first payload word. There is only one payload word in case of Version read and the 16 MSB of the version is inserted in the data field. `nData` element of the back channel information structure (`pBkChnlInfo->nData`) contains the Version and Code download status information. MIPS is computed by the Application. Other Read-back values are extracted from the memory using the offset and size.

Command	Number of 16-bit Data Words in the Response	Response data			
		Word 1	Word 2	Word 3	Word 4
SS_CMD_BK_MIPS_VALUE	4	16 LSB bits of MIPS	16 MSB bits of MIPS	16 LSB bits of Peak MIPS	16 MSB bits of Peak MIPS
SS_CMD_BK_VERSION_INFO	2	16 LSB bits of 32 bit Version Value	16 MSB bits of 32 bit Version Value		
SS_CMD_BK_READ_VALUE	2 * Number of Read-back Values	16 LSB bits of first value	16 MSB bits of first value	16 LSB bits of second value	16 MSB bits of second value
SS_CMD_BK_ERROR_CODE	2	16 LSB bits of error code	16 MSB bits of error code		
SS_CMD_BK_ACKNOWLEDGE	No return for acknowledgement packet				

Table 5: Read-back Payload Data Field

A.1.2.2 CMD: (4-bit)

This field identifies the type of read back data in the packet. This is the 4 LSB bits of the read back type given in Table 3 associated with current packet.

A.1.2.3 NO: (4-bit)

This field indicates the serial number of each 32-bit word.

A.1.3 END: (32-bit)

The END word consists of the `END_CMD` command, which indicates the end of packet, the CRC checksum value for the packet and word number.

A.1.3.1 END_CMD: (16-bit)

This 16-bit word indicates the end of a communication packet.

A.1.3.2 CRC: (8-bit)

CRC field is used for protection. The current version uses a simple checksum of individual bytes of payload contents.

A.1.3.3 RSVD: (4-bit)

This field is reserved.

A.1.3.4 NO: (4-bit)

This field indicates the serial number of each 32-bit word.

Terminology

Table 6: Terminology

Term	Description
ADI	Analog Devices Inc.
API	Application Program Interface
OBJ	Object file format in CrossCore Embedded Studio environment
SPI	Serial Peripheral Interface
SSn	SHARC machine code corresponds to SigmaStudio Schematic
USB	Universal Serial Bus
XML	Extensible Markup Language

References

Table 7: References

Reference No.	Description
[1]	AE_42_SS4G_QuickStartGuide.docx
[2]	AE_42_SS4G_ReleaseNotes.docx