Configuring a freeDSP-Infinitas Board for Multi-Channel Audio Applications

Friedrich Beyer and Tom Wuehle, Chair for Acoustics and Haptics, TU Dresden, Germany
6th January 2021

1 Introduction

The *freeDSP-Infinitas* is a very versatile DSP and digital IO board. It features an *Analog Devices Sigma DSP ADAU 1452* processor, an *XMOS XE216-512-TQ128* chip providing a USB Audio Class 2.0 interface and a *Lattice Semiconductor LCMXO2-1200HC* FPGA-chip, that can be used for flexible on-board signal routing.

This application note will guide you through setting up a *freeDSP-Infinitas* for multi-channel audio use. The considered setup comprises 32 channel USB input, 32 channel direct I2S output, 32 channel USB return with DSP processing, 32 channel I2S output with DSP processing and S/PDIF IO. It will also show how to access the signal inputs and outputs in an example DSP program. This program contains a very simple throughput algorithm that can serve as basis for more sophisticated processing algorithms in your future projects.

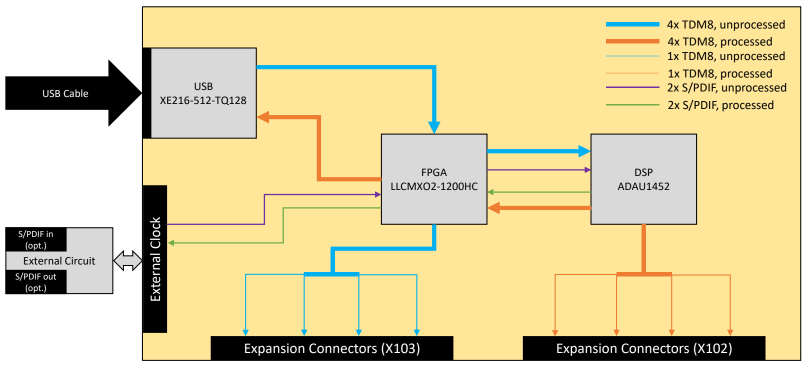


Figure 1: Signal flow chart of the freeDSP-Infinitas configuration in this application note.

The setup discussed in this application note is based on and extents the example project included in the repository and described in the Getting Started guide, which needs to be set up before. Since the example project in the repository does not provide DSP functionality, to integrate the DSP is the central aim of this application note.

To create a multi-channel audio DSP board with 34 input audio channels and 34 output audio channels, it is necessary to apply some changes in the FPGA routing. Furthermore, the DSP needs to be configured and programmed. The XMOS firmware is kept untouched in the setup discussed in this application note. However, depending on the hardware configuration of the board used, changes to the XMOS firmware can be necessary if the DSP is to be programmed through the XMOS-chip instead of through USBi (refer to the Getting Started section about DSP). Here, the hardware configuration of the *freeDSP-Infinitas* allows programming through USBi.

In the considered multi-channel setup, 32 input audio channels are provided by the USB Audio interface via the *XMOS* chip. These input channels are routed to the input of the DSP, as well as to the *Standard freeDSP Expansion Connectors* on header X103, using I2S interfaces with TDM-8 streams. Two additional inputs, as well as two outputs are realised through an S/PDIF interface that is integrated in the External Clock header X104. To access the S/PDIF interface an external circuit was built which converts an optical S/PDIF signal into an electrical one and vice versa, as well as provides the necessary *Micro-MaTch* connector that can be connected to header X104. The S/PDIF input and output signals are routed from the External Clock header to the DSP.

All 34 input signals can be processed in the DSP, which provides a huge variety of signal processing possibilities. Since the ADAU1452 has two serial ports that support TDM-16 and two that support TDM-8 for input and output, respectively, it would be possible to output a total of 48 channels on I2S interfaces. In this application note, however, only 4 x 8 output I2S channels will be used. The four serial I2S output ports of the DSP are routed to four *Standard freeDSP Expansion Connectors* on header X102 and to the XMOS chip in order to return the processed signals via USB.

2 Editing the FPGA Routing

2.1 Changes to the VHDL-file

In order to apply a new FPGA routing that complies with the above mentioned specifications, it is necessary to edit the VHDL-file in the Getting Started example project. This can be done either in *Lattice Diamond* or in the source file directly, using an arbitrary text editor. The following changes have to be made.

The *entity* definition can be kept as is. In this part all necessary ports of the FPGA chip have been already declared and their functions have been defined.[1] In the *architecture* block, however, some changes have to be made. First of all, since *Analog Digital Converters* (*ADCs*) used in the Getting Started example project are not used in the current configuration, all corresponding signal definitions can be deleted. Therefore, only the following signals are kept:

```
architecture Behavioral of infinitas is
 signal bclk : std_logic := '0';
 signal lrck : std_logic := '0';
```

After signal definition, the behaviour programming begins. At first, the different clock signals need to be distributed. For this purpose, code from the Getting Started example project can be reused. In doing so the master clock (on FPGA-port CLKIN), generated by an external oscillator on the PCB, is linked to the master clock of the XMOS chip and the DSP, as well as, to all master clock pins of the eight *Standard freeDSP Expansion Connectors*. Subsequently, the *Bit-Clock* (BCLK) and the *Left-Right-Clock* (LRCK) signals of the XMOS chip are assigned to the signal variables defined above. This is useful since the XMOS chip is defined as the I2S master device.

```
xMCLK    <= CLKIN;
dspMCLK  <= CLKIN;
expMCLK1 <= CLKIN;
expMCLK2 <= CLKIN;
expMCLK3 <= CLKIN;
expMCLK4 <= CLKIN;
expMCLK5 <= CLKIN;
expMCLK6 <= CLKIN;
expMCLK7 <= CLKIN;
expMCLK8 <= CLKIN;

bclk <= xBCLK;  --XMOS is master
lrck <= xLRCK;
```

Now the *bclk* and the *lrck* signals need to be assigned to the corresponding slave devices. This configuration differs from the Getting Started example project, where the DSP is not used and is hence not provided with this signals at all. So, next to supplying the *bclk* and *lrclk* signals to the BCLK and LRCK ports of the *Standard freeDSP Expansion Connectors* on headers X102 and X103, the two signals have to be connected to the BCLK and LRCK pins of all seral input and output ports of the DSP. The DSP consist of different clock domains for every serial port, which are all defined as slave devices of the XMOS.
Note that the naming conventions of the DSP ports (starting at 0) and the expansion headers (starting at 1) might be confusing. This was taken from the Getting Started example project but could be changed to a clearer naming in the future. (Keep in mind, VHDL-comments start with "--".)

Code from current setup	**Code from Getting Started example project**

```
-- bclk/lrck to expansion header
-- slot 1
 expBCLK1      <= bclk;
 expLRCK1      <= lrck;
-- bclk/lrck to DSP serial input
-- port 0
 dspBCLKIN0    <= bclk;
```
	```
	-- bclk/lrck to expansion header
	-- slot 1
	 expBCLK1 <= bclk;
	 expLRCK1 <= lrck;
	-- bclk/lrck to expansion header
	-- slot 2
	```

[1] Changes need to be applied in this section e.g. if the roles of I2S devices are swapped. Here, the serial input ports, as well as, the serial output ports of the DSP are I2S slave devices of the XMOS-chip. In case the DSP becomes I2S master on serial outputs, the corresponding ports of the FPGA must be adjusted from *out*-ports to *in*-ports.

` dspLRCKIN0    <= lrck;`
`-- bclk/lrck to DSP serial output`
`-- port 0`
` dspBCLKOUT0   <= bclk;`
` dspLRCKOUT0   <= lrck;`
`-- bclk/lrck to expansion header`
`-- slot 2`
` expBCLK2      <= bclk;`
` expLRCK2      <= lrck;`
`...`
	` expBCLK2 <= bclk;`
	` expLRCK2 <= lrck;`
	`...`

The remaining two lines of clock distribution can be taken over from the Getting Started example project.

```
 pllCLKREF  <= CLKIN;
 pllWCLKREF <= WCLKIN;
```

After all the clock signals have been connected to the correct pins the signal lines have to be linked. At first, the S/PDIF input and output from the External Clock header need to be connected to the S/PDIF input and output of the DSP.

```
 dspSPDIFIN <= SPDIFIN;
 SPDIFOUT   <= dspSPDIFOUT;
```

Afterwards, the TDM streams of the different devices and expansion headers have to be connected to each other. *Note: currently each expansion header can exclusively only be used as I2S input or output. In this application note all expansion headers are configured as outputs.*

```
-- the TDM streams from XMOS are connected to expansion headers 2, 4, 6, 8
-- and to the serial input ports of the DSP (0, 1, 2, 3)
-- the TDM streams from the serial output ports of the DSP are linked to the
-- expansion headers 1, 3, 5, 7 and to the input of the XMOS
-- XMOS is also for DSP output I2S master device
 expMDO2      <= xTDMOUT1;
 dspTDMIN0    <= xTDMOUT1;
 expMDO1      <= dspTDMOUT0;
 xTDMIN1      <= dspTDMOUT0;

 expMDO4      <= xTDMOUT2;
 dspTDMIN1    <= xTDMOUT2;
 expMDO3      <= dspTDMOUT1;
 xTDMIN2      <= dspTDMOUT1;

 expMDO6      <= xTDMOUT3;
 dspTDMIN2    <= xTDMOUT3;
 expMDO5      <= dspTDMOUT2;
 xTDMIN3      <= dspTDMOUT2;

 expMDO8      <= xTDMOUT4;
 dspTDMIN3    <= xTDMOUT4;
 expMDO7      <= dspTDMOUT3;
 xTDMIN4      <= dspTDMOUT3;
```

In the code of the Getting Started example project, further *process*-blocks can be found that are used to connect to specific ADC/DAC- boards. These parts of the code can be deleted in the setup of this application note.

2.2 Changes to the LPF-file

In the setup of this application note, some connections from the FPGA to the DSP are added. The corresponding ports on the FPGA have already been defined in the VHDL-file. However, the compiler needs to know which VHDL-signal corresponds to which physical pin of the FPGA and what is its physical behaviour. Therefore, some editing needs to be done in the LPF-file to constrain each signal to a certain pin. This file can be found in the File-List tab within *Lattice Diamond* (but can be edited in an arbitrary text editor, too). Most pins are defined in the Getting Started example LPF-file already, but there are a couple of ports missing, that need to be defined now.

In the LPF-file a pin is specified by typing `LOCATE COMP “`*`signal_name`*`” SITE “`*`pin_name/number`*`”;`, where *signal_name* represents the signal name in the VHDL-file and *pin_name/number* the pin name of the physical device. Next to the pin definition, an input buffer type must be defined for each pin, which is done with the line `IOBUF PORT "signal_name" IO_TYPE=LVCMOS33 PULLMODE=NONE;`.

The missing pin constraints are the *SPDIFIN*, *dspSPDIFIN* and *dspSPDIFOUT* signals and the serial outputs of the DSP *dspTDMOUTx* . Hence it is necessary to add the following lines:

```
IOBUF PORT "SPDIFIN" IO_TYPE=LVCMOS33 PULLMODE=NONE ;
IOBUF PORT "dspSPDIFOUT" IO_TYPE=LVCMOS33 PULLMODE=NONE ;
IOBUF PORT "dspTDMOUT0" IO_TYPE=LVCMOS33 PULLMODE=NONE ;
IOBUF PORT "dspTDMOUT1" IO_TYPE=LVCMOS33 PULLMODE=NONE ;
IOBUF PORT "dspTDMOUT2" IO_TYPE=LVCMOS33 PULLMODE=NONE ;
IOBUF PORT "dspTDMOUT3" IO_TYPE=LVCMOS33 PULLMODE=NONE ;
LOCATE COMP "SPDIFIN" SITE "48" ;
LOCATE COMP "dspSPDIFOUT" SITE "115" ;
LOCATE COMP "dspTDMOUT0" SITE "12" ;
LOCATE COMP "dspTDMOUT1" SITE "11" ;
LOCATE COMP "dspTDMOUT2" SITE "143" ;
LOCATE COMP "dspTDMOUT3" SITE "142" ;
```

2.3 Flashing the FPGA

The flashing procedure is performed according to the Getting Started guide. However, a new JED-file needs to be generated before flashing. This can be done by switching to the Process tab in *Lattice Diamond* (left side of the window) and double-click *JEDEC File* in the Export Files group. This will start a new compilation process. After the compilation finishes successfully, the programmer needs to be opened as described in the Getting Started guide. It might be necessary to update the *File Name* parameter to the recently compiled JED-file. After that, the FPGA can be flashed with the new routing, which should exit with no errors.

3 DSP Programming

After the routing described above is applied to the FPGA, the serial audio streams (I2S, S/PDIF) can be accessed by the DSP on the serial input ports or the S/PDIF port, respectively. This audio streams can be processed on the DSP core afterwards. The processed audio can be output via serial TDM signals using I2S interfaces (up to 48 channels, here only 32) or via the S/PDIF interface (two channels).

The DSP provides a big variety of possible processing blocks (for further information see Sigma Studio references or e.g. Getting Started with the freeDSP CLASSIC). In the setup of this application note, a simple throughput will be realized as a starting point. On this basis, more sophisticated signal processing can be applied to the individual signal paths by adding certain signal processing blocks.

3.1 Hardware Configuration

In order to make sure the audio input and output works as intended, some hardware configurations need to be done. After creating a new *Sigma Studio* project (see Getting Started with the freeDSP CLASSIC) three hardware components will be added to the *Config* sub-tab in the *Hardware Configuration* tab within *Sigma Studio*. It is recommended to add the ADAU1452 IC before the E2Prom IC. In this way the DSP core becomes IC1 and the E2Prom IC2, which might be handy in case programming will be done through the XMOS chip of the freeDSP-Infinitas (see also Getting Started with freeDSP-Infinitas).

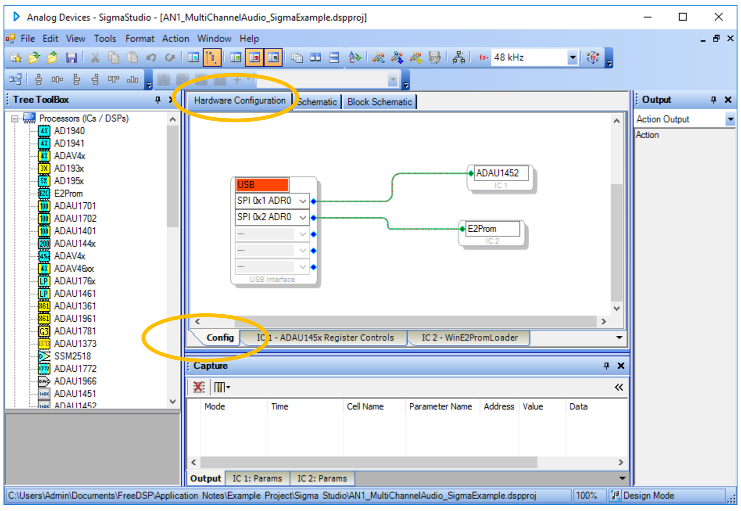
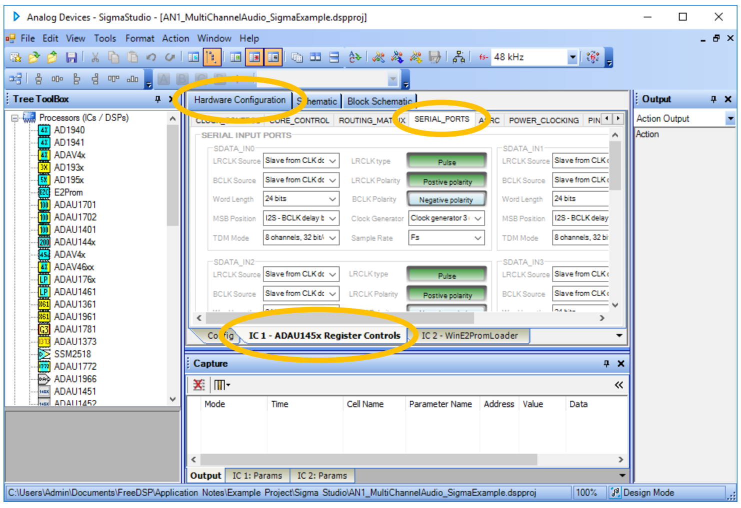


Figure 2: Sigma Studio Hardware Configuration

3.1.1 Serial Input Configuration of ADAU1452

After setting up the basic hardware configuration, some register configurations need to be done on the *ADAU1452* IC. The serial ports can be configured in the sub tab *IC 1 – ADAU145x Register Controls*.


Analog Devices - SigmaStudio - [AN1_MultiChannelAudio_SigmaExample.dspproj]
File Edit View Tools Format Action Window Help
48 kHz
Tree ToolBox
Hardware Configuration
Schematic
Block Schematic
CORE_CONTROL
ROUTING_MATRIX
SERIAL_PORTS
POWER_CLOCKING
Output
Action Output
Action
Config
IC 1 - ADAU145x Register Controls
IC 2 - WinE2PromLoader
Capture
Mode
Time
Cell Name
Parameter Name
Address
Value
Data
Output
IC 1: Params
IC 2: Params
C:\Users\Admin\Documents\FreeDSP\Application Notes\Example Project\Sigma Studio\AN1_MultiChannelAudio_SigmaExample.dspproj
100%
Design Mode

SERIAL INPUT PORTS

SDATA_IN0

LRCLK Source	Slave from CLK dc	LRCLK type	Pulse
BCLK Source	Slave from CLK dc	LRCLK Polarity	Postive polarity
Word Length	24 bits	BCLK Polarity	Negative polarity
MSB Position	I2S - BCLK delay b	Clock Generator	Clock generator 3
TDM Mode	8 channels, 32 bit/	Sample Rate	Fs

SDATA_IN1

LRCLK Source	Slave from CLK
BCLK Source	Slave from CLK
Word Length	24 bits
MSB Position	I2S - BCLK delay
TDM Mode	8 channels, 32 bi

SDATA_IN2

LRCLK Source	Slave from CLK dc	LRCLK type	Pulse
BCLK Source	Slave from CLK dc	LRCLK Polarity	Postive polarity

SDATA_IN3

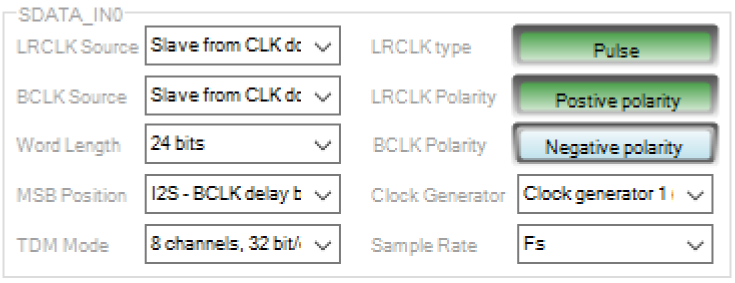
LRCLK Source	Slave from CLK
BCLK Source	Slave from CLK

Figure 3: Serial Ports Configuration in Sigma Studio

Since the DSP is an I2S slave device of the XMOS, the configurations need to be adjusted according to the definitions that were done in the XMOS programming. The following setup works with the XMOS code of the Getting Started example project.

Since the XMOS chip is the I2S master, the *LRCLK Source* and the *BCLK Source* must be set to *Slave from CLK domain x* (where x is 0...3). Because the LRCLK and the BCLK of the XMOS were distributed to all serial ports equally by the FPGA, it is possible to select any of the four sources for each of the serial port. That is, e.g. *Slave from CLK domain 0* can be chosen for all serial inputs.

Figure 4 depicts the settings that can be used for all serial input ports.


SDATA_IN0

LRCLK Source	Slave from CLK dc	LRCLK type	Pulse
BCLK Source	Slave from CLK dc	LRCLK Polarity	Postive polarity
Word Length	24 bits	BCLK Polarity	Negative polarity
MSB Position	I2S - BCLK delay b	Clock Generator	Clock generator 1 (
TDM Mode	8 channels, 32 bit/	Sample Rate	Fs

Figure 4: Serial port configuration in Sigma Studio of port 0

In order to power the serial input ports, they have to be enabled in the POWER_CLOCKING section. Therefore, clicking *Enable serial input power* (must be green) enables power for the corresponding serial input port.

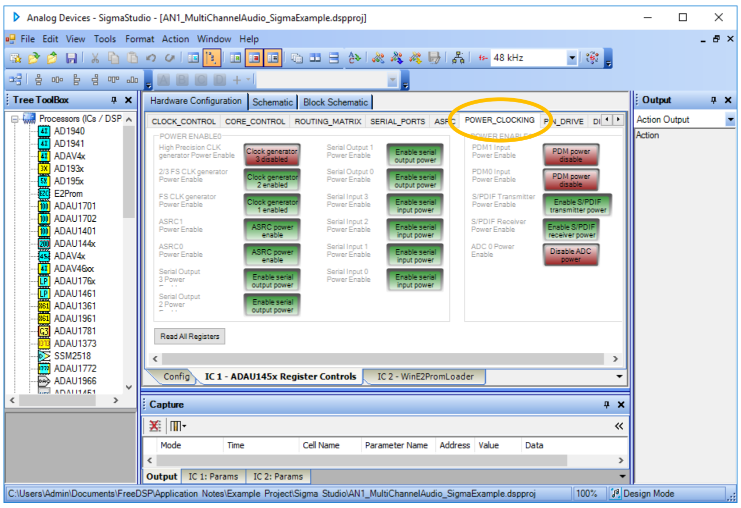

Analog Devices - SigmaStudio - [AN1_MultiChannelAudio_SigmaExample.dspproj]
Hardware Configuration
Schematic
Block Schematic
CLOCK_CONTROL
CORE_CONTROL
ROUTING_MATRIX
SERIAL_PORTS
POWER_CLOCKING
Enable serial input power
Enable serial output power
Read All Registers
IC 1 - ADAU145x Register Controls
Capture
Output

Figure 5: Power configuration in Sigma Studio

3.1.2 Serial Output Configuration of ADAU1452

The serial output ports can be configured in the SERIAL_PORTS section, too. In the setup of this application note, the serial output ports of the DSP are considered the I2S-slave devices of the XMOS, which is why the *LRCLK Source* and the *BCLK Source* need to be set to *Slave from CLK domain x* (where x is 0...3) as well (see 3.1.1).

Also the remaining properties are equal to the serial input ports. The additional option, only available in serial output ports options *Drive every output channel*[2] might depend on the devices attached to the ports and can be set active in this application note.

In case the external devices attached to the PCB via *Standard freeDSP Expansion Connectors* understand TDM16 streams, serial outputs 1 and 2 can be used in TDM16 mode. In that case, in order to use all 16 channels this has to be selected *TDM Mode* Pull-Down menu. If a lower number of channels shall be used, this can be adjusted in this menu, too. In this case, the return via USB should be avoided.

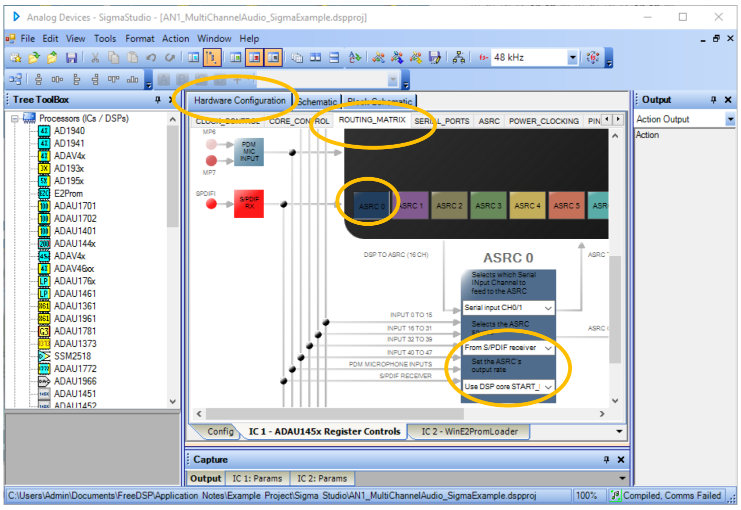
Also the serial outputs have to be switched on in the POWER_CLOCKING section. Therefore, click *Enable serial output power* (must be green) for any serial output port that shall be used.

3.1.3 S/PDIF Input Configuration of ADAU1452

According to the ADAU1452 reference, it is necessary to use one of the eight *Asynchronous Sample Rate Converters* (*ASRCs*) to access the S/PDIF input data. Hence one ASRC must be configured in the

[2] If not active, "[…] the serial data pin is high impedance during unused output channels" (ADAU1452 reference, p. 108).

ROUTING_MATRIX section. Here, ASRC 0 will be used. The ASRC preferences can be opened by clicking on the intended ASRC symbol. Then the ASRC source must be set to *From S/PDIF receiver* and the ASRC output rate to *Use DSP core START_PULSE rate*. Using the S/PDIF receiver as source, the Serial input channel selection has no effect and might be arbitrary.



ASRC 0
Selects which Serial INput Channel to feed to the ASRC: Serial input CH0/1
Selects the ASRC source: From S/PDIF receiver
Set the ASRC's output rate: Use DSP core START_

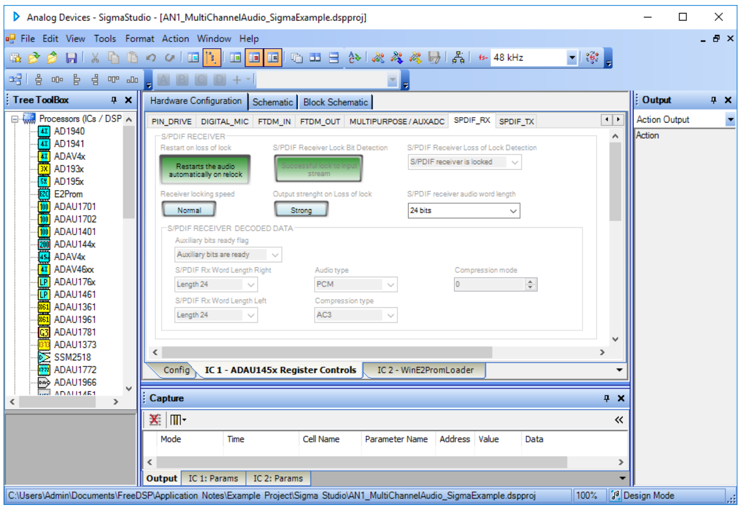
Figure 6: ASRC configuration in Sigma Studio

In the ASRC section further preferences can be made. Especially, it is possible to Mute/Unmute the ASRCs individually. For now, it is not necessary to make changes in this section.

Power must be enabled for the ASRCs, too. There are two power groups of ASRCs. According to the ADAU1452 reference ASRC0 corresponds to ASRCs 0-3 (confusing naming in Sigma Studio) in the POWER_CLOCKING section. Accordingly, ASRC1 refers to ASRCs 4-7.

To power up also the S/PDIF receiver, in the POWER_CLOCKING section *Enable S/PDIF receiver power* must be enabled (green).

Further preferences can be made in the SPDIF_RX section. Enable *Restart the audio automatically on relock* and select the correct *S/PDIF receiver audio word length*, that corresponds to the expected S/PDIF format.



S/PDIF RECEIVER
Restart on loss of lock: Restarts the audio automatically on relock
S/PDIF Receiver Lock Bit Detection: Successful lock to input stream
S/PDIF Receiver Loss of Lock Detection: S/PDIF receiver is locked
Receiver locking speed: Normal
Output strenght on Loss of lock: Strong
S/PDIF receiver audio word length: 24 bits

S/PDIF RECEIVER DECODED DATA
Auxiliary bits ready flag: Auxiliary bits are ready
S/PDIF Rx Word Length Right: Length 24
Audio type: PCM
Compression mode: 0
S/PDIF Rx Word Length Left: Length 24
Compression type: AC3

Analog Devices - SigmaStudio - [AN1_MultiChannelAudio_SigmaExample.dspproj]
File Edit View Tools Format Action Window Help
48 kHz
Tree ToolBox
Hardware Configuration
Schematic
Block Schematic
PIN_DRIVE
DIGITAL_MIC
FTDM_IN
FTDM_OUT
MULTIPURPOSE / AUXADC
SPDIF_RX
SPDIF_TX
Config
IC 1 - ADAU145x Register Controls
IC 2 - WinE2PromLoader
Capture
Mode
Time
Cell Name
Parameter Name
Address
Value
Data
Output
C:\Users\Admin\Documents\FreeDSP\Application Notes\Example Project\Sigma Studio\AN1_MultiChannelAudio_SigmaExample.dspproj
100%
Design Mode

Figure 7: S/PDIF Receiver preferences in Sigma Studio.

3.1.4 S/PDIF Output Configuration of ADAU1452

To use the S/PDIF transmitter, first in the ROUTING_MATRIX section click on S/PDIF TX and select *From DSP core program*.

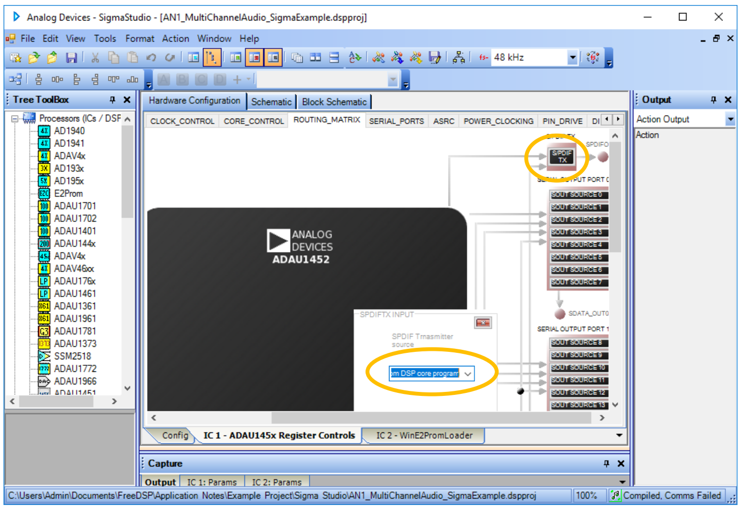
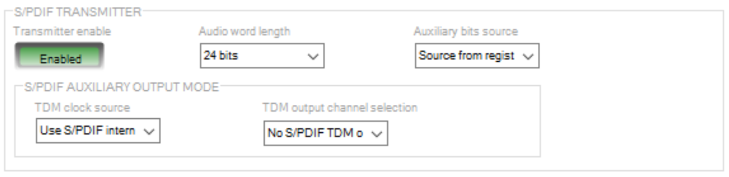


Figure 8: S/PDIF TX preferences in ROUTING_MATRIX section in Sigma Studio

Also for the S/PDIF output the power supply must be enabled in the POWER_CLOCKING section (*Enable S/PDIF transmitter power*).

In the SPDIF_TX section the transceiver must be *Enabled*. Also the *Audio word length* must be set to the expected word length. Further options, corresponding to S/PDIF Auxiliary bits can be made as well in this section but will be left untouched in this application note.


S/PDIF TRANSMITTER

Transmitter enable: Enabled
Audio word length: 24 bits
Auxiliary bits source: Source from regist

S/PDIF AUXILIARY OUTPUT MODE

TDM clock source: Use S/PDIF intern
TDM output channel selection: No S/PDIF TDM o

Figure 9: Preferences in SPDIF_TX section in Hardware Configuration in Sigma Studio.

3.2 DSP Programming

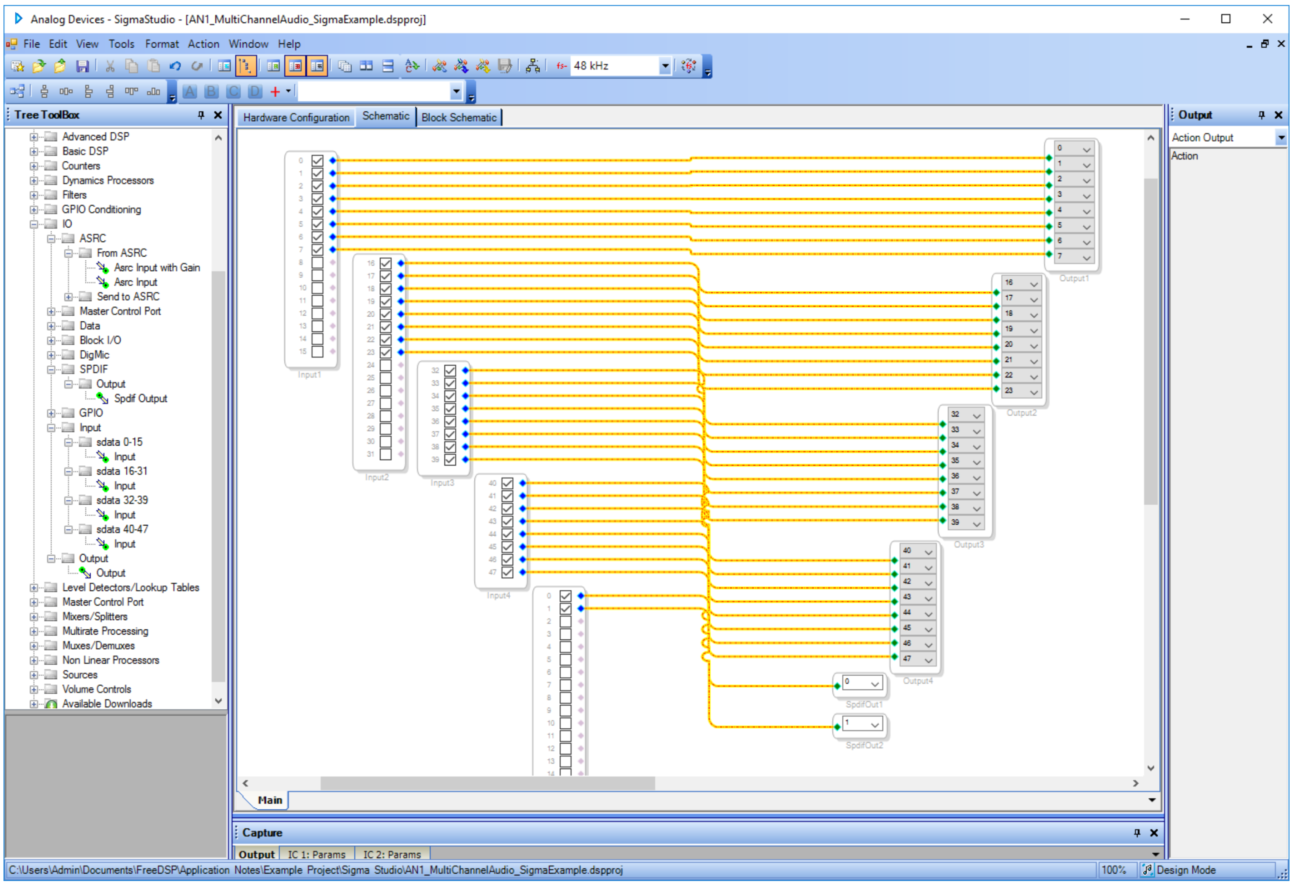


Figure 10: Schematic of a Sigma Studio patch. In this case all inputs all let through to one output each.

In the setup of this application note, all available inputs are just put through to one output each. There are different blocks which represent different types of ports. First, to create serial (I2S) **Inputs** navigate to ADAU1452/IO/Input/sdata x-x/Input in the *Tree ToolBox* of *Sigma Studio* and pull it into the schematic. Here, every sdata x-x container represents one serial input port. By pulling it into the schematic the block that is created consists of as many input channels that are possible on the corresponding serial port at most. That is 16 for the serial input ports 0 and 1 and eight for the other two. By ticking the channels in the schematic they can be activated. The higher eight channels of port 0 and 1 will not work if TDM16 was not selected in the *Hardware Configuration*. *Note that the channel numbers are fix, independently from the TDM mode. So, serial input 0 reaches from channel 0 to 15, input 1 from 16 to 31, input 2 from 32 to 39 and input 3 from channel 40 to 47.*

In order to access the incoming S/PDIF signal, an **Asrc Input** object needs to be created (ADAU1452/IO/ASRC/From ASRC/ Asrc Input). *Note, there is no S/PDIF Input object available*. This object provides 16 channels, two for each ASRC. Depending on which ASRC was chosen in the *Hardware Configuration* (in this example ASRC 0) the corresponding channels in the block in the schematic need to be activated. Hence, here channels 0 and 1 need to be selected.

Serial outputs can be generated with the **Output** block (ADAU1452/IO/Output/Output). By default, this creates a block with one channel. To increase the number of channels in an output block, the algorithm can be grown by right-clicking the object and choosing *Grow Algorithm* and then the number of channels by which the algorithm is to be increased. *Note, also for serial outputs the channel numbering is fixed to the maximum possible number of channels. Hence, in TDM8 mode (for Output 0 and 1) channels 8 to 15 and 24 to 31 do not work.*

Finally, **Spdif Output** object must be created (ADAU1452/IO/SPDIF/Output/Spdif Output) to output S/PDIF signals. This algorithm does not support to be grown to more channels. So, a separate object needs to be created for every S/PDIF output channel.

3.3 Flashing the DSP

In the setup of this application note a *freeDSP-Infinitas* was used, whose hardware was configured such that the DSP can be programmed via the USBi interface. Downloading and storing the program will be done according to the Getting Started guide . For *freeDSP-Infinitas* boards configured such that the DSP only can be programed via the XMOS-chip and I2C, refer to the Getting Started guide likewise.

4 Summary

In this application note, the procedure was shown to get up running an audio multi-channel DSP using a *freeDSP-Infinitas* board. Therefore, starting from a board that was taken into action according to the Getting Started guide, it was shown how to setup a new FPGA routing and how to configure the DSP. Finally, a minimal DSP programming example was shown, which can be used as a starting point to realize more sophisticated signal processing based on the great variety of possibilities provided by the ADAU1452 DSP.

5 References

[1] Auverdion, “Getting Started with Infinitas,” September 2018. [Online]. Available: https://github.com/freeDSP/freeDSP-INFINITAS/blob/master/DOCUMENTATION/GettingStarted.pdf. [Accessed August 2020].

[2] Analog Devices, “SigmaDSP Digital Audio Processor,” 2014. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADAU1452.pdf. [Accessed August 2020].

[3] freeDSP, “freeDSP Guidlines,” 2016. [Online]. Available: https://github.com/freeDSP/WIKI-AND-GENERAL-TOPICS/wiki/freeDSP-Guidelines. [Accessed Dezember 2020].

[4] freeDSP, “freeDSP Infinitas (Github),” 2018. [Online]. Available: https://github.com/freeDSP/freeDSP-INFINITAS. [Accessed August 2020].

[5] freeDSP, “Getting Started with the freeDSP,” 2016. [Online]. Available: https://docs.google.com/document/d/1gQ5PzCN1hFmIgi31e2L5NF1VgZXVvlqLSrGzaxm_bFE/edit. [Accessed Dezember 2020].

[6] Analog Devices, “SigmaStudio and SigmaDSP Documentation [Analog Devices Wiki],” 29 Juli 2020. [Online]. Available: https://wiki.analog.com/resources/tools-software/sigmastudio. [Accessed 4 Januar 2021].