Stellar IP

User's Manual

4DSP LLC, 10713 Ranch Rd 620N STE 522, Austin, TX 78726, USA

4DSP BV, Ondernemingsweg 66f, 2404 HN, Alphen aan den Rijn, Netherlands

Email: support@4dsp.com

Table of Contents

1 Definitions and scope 4

1.1 Definitions 4

2 Introduction 4

3 How to create a star 5

3.1 Wormholes 5

3.2 Command distribution 7

3.3 Clock and reset 9

3.4 Common directory structure 11

3.4.1 Stellar IP specific files 12

3.5 Creating a star library 14

3.5.1 Wormhole XML definition 15

3.5.2 Star XML definition 16

3.5.3 Star memmap XML definition 17

4 How to create a constellation 18

4.1 SDF file 20

4.1.1 Target Hardware 20

4.1.2 Project information 21

4.1.3 Memory Information 22

4.1.4 Star declaration 22

4.1.5 Star wormhole connections 22

4.1.6 Example SDF file 23

4.2 UCF file 27

4.3 TRC file 27

4.4 SET file 27
4.5 PAR file 27
4.6 NGN file 27
4.7 NGB file 27
4.8 MAP file 27
4.9 DOX file 27
4.10 BIG file 27
4.11 Constellation identification 28
5 Running stellar IP 30

1 Definitions and scope

1.1 Definitions

- **Intellectual Property core** (or IP core/block): block with a specific function, written in a format that can be compiled by an FPGA synthesis tool.
- **Star**: Intellectual Property core with a generic pin-out.
- **Constellation**: group of stars connected to one another via worm holes.
- **Worm hole**: connection between 2 stars. A worm hole can also be called a channel.

2 Introduction

4DSP has developed a software tool, called Stellar IP, to simplify firmware design on an FPGA. The goal of Stellar IP is to force users to split their algorithm into blocks, called *Stars*, to make sure that each *Star* is working the way it should. Once a library of *Stars* has been created the user will create a design that is composed of several *stars*, called a *Constellation*. The *constellation* is described in the Stellar IP description file (SDF). The SDF is used by Stellar IP to create a top level design file in VHDL that instantiates all the stars, automatically assigns address ranges for *Stars* that require settings, and it makes sure the connections between the *stars* (*Worm holes*) are correct.

Furthermore, Stellar IP will setup synthesis scripts that are required to create the configuration file for the FPGA device. A test bench skeleton is created in combination with the compilation scripts to set-up the modelsim simulation. Simulation of the constellation is further simplified when the Stars are designed with the 4DSP command interface. This will allow the user to send and receive data and commands from a script file without the need to code a single line of VHDL. The following image depicts an example constellation that implements 4 stars that are interconnected using wormholes.

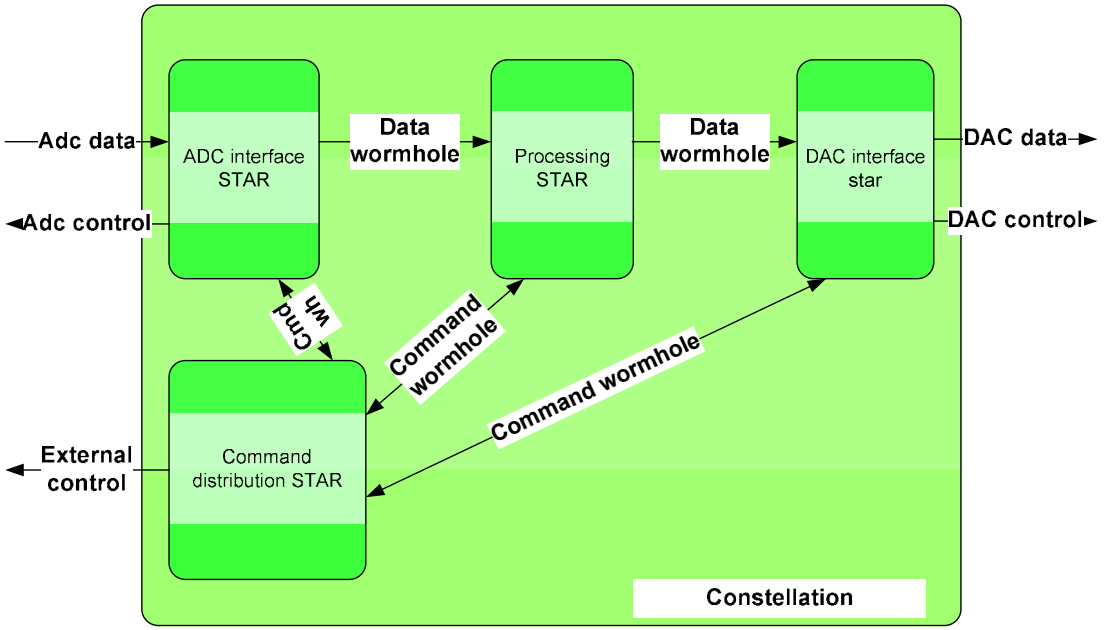

Figure 1: Example constellation

3 How to create a star

When designing a new star it is important to keep in mind a few basic rules in order to take advantage of the Stellar IP tool.

3.1 Wormholes

It is important to use the same interfaces as much as possible. Choose a common data bus width and data transfer protocol between the Stars. The interface will be defined by the Wormhole library and can be used inside Stellar IP to interconnect Stars. Two of the following wormholes would be required when you want to define a Wormhole with a 64-bit data bus, a "data valid" signal, and a stop signal.

input wormhole: wh_in

Port name	Direction	Width
data_in	Input	64
data_valid	Input	1
data_stop	Output	1

output wormhole: wh_out

Port name	Direction	Width
data_in	Output	64
data_valid	Output	1
data_stop	Input	1

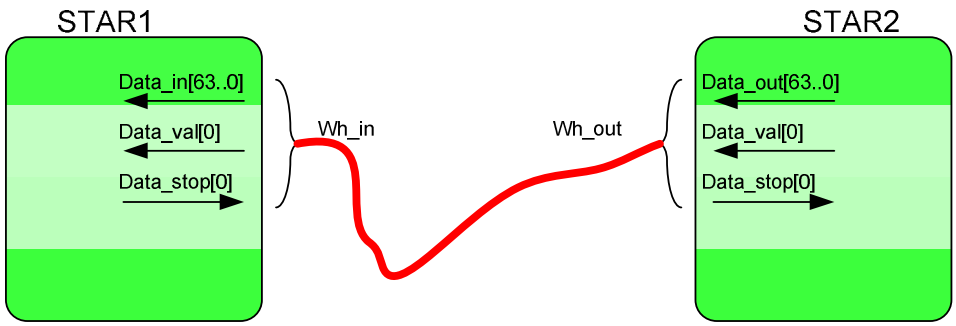

Figure 2: Example of a wormhole.

3.2 Command distribution

It is important to offer a means of providing settings to the Star. 4DSP has chosen to implement a command bus structure as depicted in the following figure.

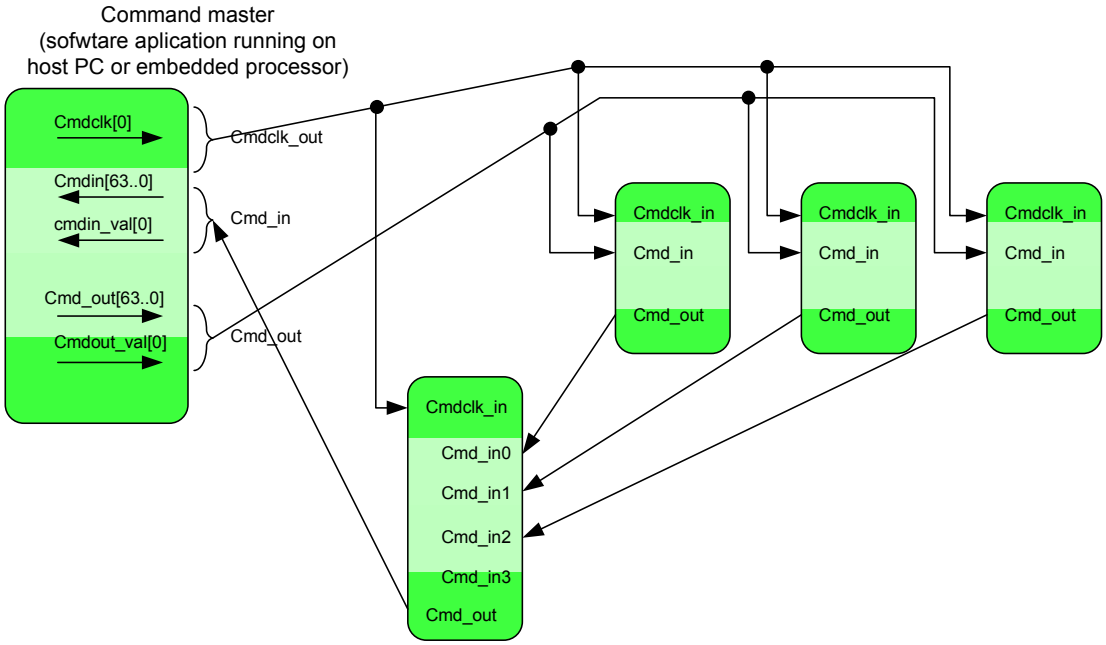

Figure 3: Command distribution

One star will be the command master of the constellation and will have three wormholes to facilitate the command distribution. There will be the cmdclk_out, the cmd_in and the cmd_out wormhole. The other stars will be command slaves and will all have a cmdclk_in, cmd_in and a cmd_out wormhole. The command outputs from all the command slaves will be routed through a multiplexer star and then to the command master cmd_in wormhole.

Commands will be distributed from the command master to all the command slaves. The format of the commands will be a packet of 64 bits. Each packet consists of a data field, address field, and a command field.

Bit nr.	Bit 63	Bit 60	Bit 59	Bit 32	Bit 31	Bit 0
Name	Cmd[3..0]		Address[27..0]		Data[31..0]	

Table 1: Command packet field definition

Each star will be assigned to a specific address range. As soon as the “data valid” signal on the command bus is asserted a Star will analyze the address and command fields. When the command

has a valid command and the address matches the Star's address range it will execute the command. So far we have defined the following commands.

Field		Description
CMD		Command
	0000	Reserved
	0001	**Write command**. The data from the data field is written to the register selected by the address field.
	0010	**Read command:** The data from the data field is ignored. The register contents of the selected register are copied in to data field. This packet will result in a response packet being generated.
	0011	reserved
	0100	**Read Acknowledge**: This command is used to acknowledge a read request. The data field holds the requested data and the address field points to the address that was requested.
	0101-1111	Reserved

Table 2: Command field description

There are two commands that can be used to read and write the registers of a star. For example, writing a value to register 1 will require the following steps from the command master:

- Prepare the 64-bit value by assigning the following values to the fields:
  - CMD = 0x1 (write command)
  - ADDR = 0x0000001 (indirect register 0x1)
  - DATA = xxxxxxxx (value to be written to the indirect register)
- Write the value to the "command out" bus.

The command master will not know if the command has been accepted or not. It might be possible to extend the commands with a write command that requires an acknowledge signal.

For example, reading a value from register 1 will require the following steps in the host software:

- Prepare the 64-bit value by assigning the following values to the fields:
    - CMD = 0x2 (read command)
    - ADDR = 0x0000001 (indirect register 0x1)
    - DATA = xxxxxxxx (the data field will contain the indirect register value after the command finishes)
- Write the value to the “command out” bus.
- Wait for the response packet. The packet will be written on the “cmd” in bus by the star that has a register range that covers the requested address.

3.3 Clock and reset

All stars typically require one or more clock and reset signals. A solution to efficiently implement the clock and reset structure of a constellation is to create a dedicated star that is in charge of the clock and reset. The star will connect to the external clock and reset signals to create the derived clocks and reset that the system requires.

Typical clocks could be the clocks required by a DDR2 or QDR controller. DDR2 and QDR memory require a sample clock and a shifted clock to create the required shift between data and clock signals.

Often an FPGA design requires several reset stages. First, the external devices are reset. After a specific time, these devices can be accessed for initialization and calibration. Once all clocks and devices are initialized, the remaining resets to the Stars can be released.

As a result of a central star for clock and reset distribution it is now possible to declare one or two wormholes that carry all clock and reset signals. Now each star will only grab the required clock and reset signals while ignore the remaining signals.

A small example is shown in the image below. The “clock reset master” Star has 3 external inputs and two wormhole outputs. The clk_out wormhole carries all the clock signals and the rst_out wormhole carries all the reset signals. Each of the other stars in the constellation have a rst_in and a clk_in wormhole. Inside each star only the required signals are used. Each star should note in the documentation which clock signals are used for each of the input and output wormholes to other stars.

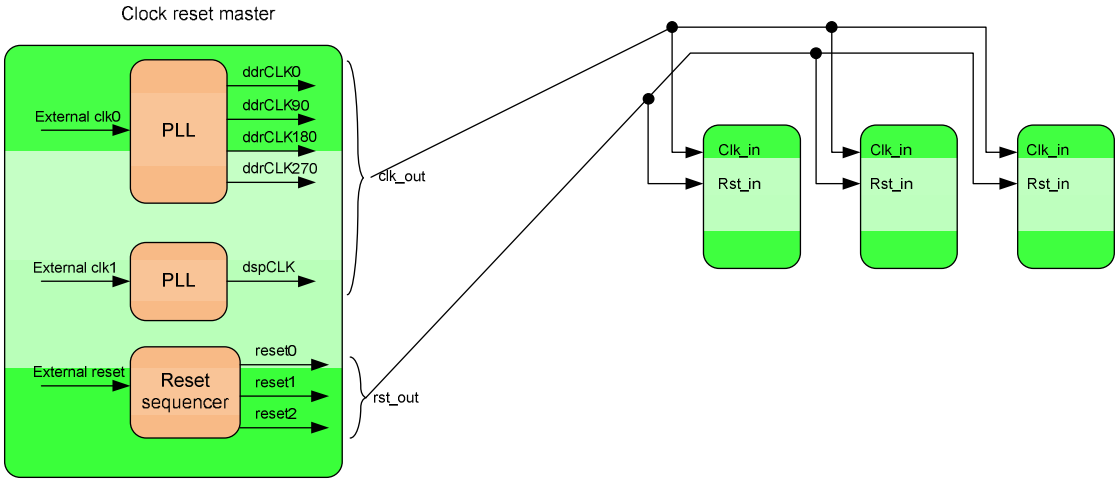

Figure 4: clock reset distribution example.

3.4 Common directory structure

To make it easier to reuse Stars and allow sharing the stars amongst other people, it is important that a clear and consistent directory structure is declared. 4DSP uses the following hierarchy and encourages users to use a similar architecture.

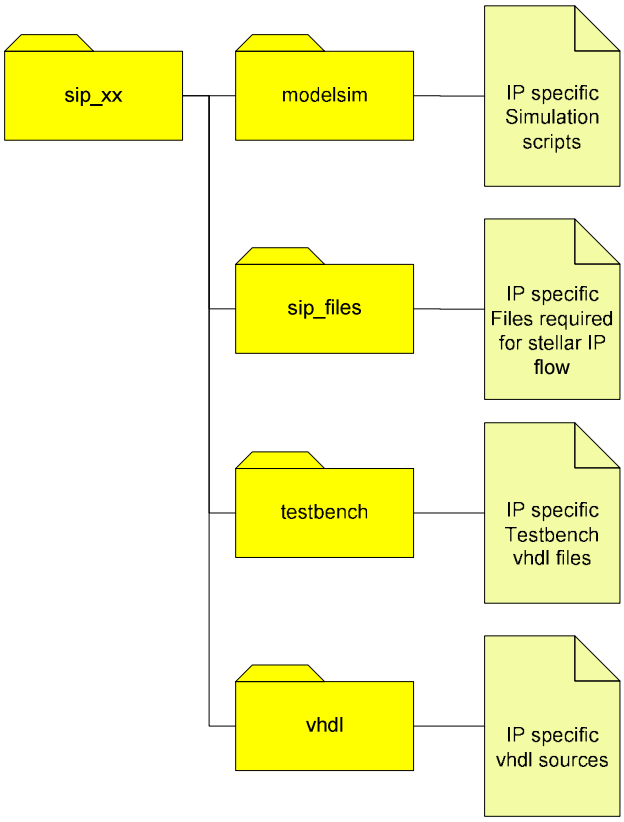

Figure 5: Star Directory structure

As depicted, each star will provide a directory with the IP specific VHDL sources. These are all the sources that are required to fully implement the star. Then there is a “testbench” directory that holds all the source files to simulate the design using a logic simulator. 4DSP uses modelsim for this purpose. The modelsim directory holds one or more scripts to setup the simulation. The last directory is the sip_files. In there you find the files that are required by Stellar IP to implement the star. Please refer to section 3.4.1 for a description of the specific stellar IP files.

3.4.1 Stellar IP specific files

Stellar IP requires specific information from each star. This is information about the location of the files that are required for the synthesis and some additional files.

3.4.1.1 LST file (required)

A text file with an .lst extension is required. The file points to the location of the following files:

- VHDL source files
- IP net list files
- A top-level wrapper file that is compatible with the definition of the star in the Stellar IP star library. This is only required if the default top level of the star uses other names and busses, for instance.

An LST file could look like this:

```
sip_cid.vhd

..\vhdl\cid.vhd

..\vhdl\cid_package.vhd

..\vhdl\cid_stellar_cmd.vhd

..\vhdl\cid_regs.vhd
```

3.4.1.2 NFO file (required)

The .nfo file is required and should have two lines with a 4 digit hexadecimal number on each line. The first line is the Star ID and the other line is the star version. The star version is formatted as follows:

```
Version_high[7..0] + version_low[7..0].
```

The .nfo file for a star with star ID 1 and revision 1.0 would need the following two lines:

```
0001

0100
```

4DSP reserves the range of 0x0000 to 0x1000 for its own Star IDs. Customers using the Stellar IP tool are advised to give their stars an ID larger then 0x1000.

3.4.1.3 Constraint files (optional)

The constraint file will be in the format that is compatible with the chosen Synthesis tool. Currently only the Xilinx tools are supported. One constraint file is required per supported board and location on the FPGA. One FPGA can be connected to more than one external memory device, for instance. The same star could be used on different products that have the same external interface mapped to other I/O pins. The Stellar IP software automatically searches for the right constraint file in the same directory where the .lst file is located. The constraint file name should end with two specific identifiers:

- Board identifier that it is made for
- Location identifier on the FPGA (StarID)

These two identifiers need to be separated by an underscore ("_"). The board identifier should match the board type that is used inside the Stellar IP description file. In case the stellar description file indicates that the target is "FM489" and this target has three identical external interfaces that can be used with this star the sip_file directory should hold the following three files:

- `<STAR_NAME>_FM489_0.ucf`
- `<STAR_NAME>_FM489_1.ucf`
- `<STAR_NAME>_FM489_2.ucf`

3.4.1.4 Wrapper file (optional)

The wrapper file is required when the star top-level is not compatible with the wormhole naming in the library. The star needs to functionally comply with the wormhole protocols, but the names could be different. In this case the wrapper just maps the wormholes to the specific ports on the star.

3.5 Creating a star library

All stars have to be added to a library file that can be used by Stellar IP. The following directory structure is used by 4DSP to organize a Star library.

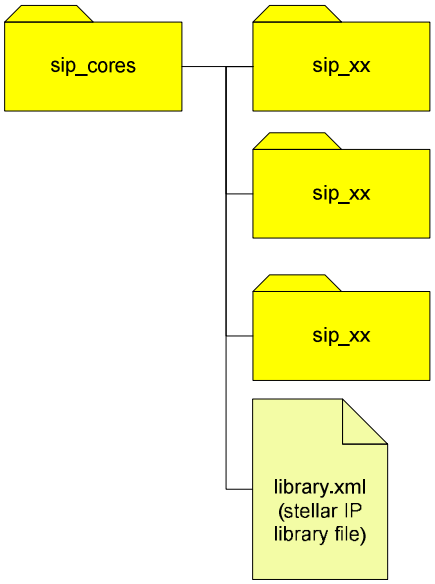

Figure 6: Star library directory structure

The library file is in XML format and holds information about:

- The available worm holes and their definition
- The available stars and their definition
- The address space requirements of each star

The format of the library file will be as depicted below:

```
<?xml version="1.0" encoding="UTF-8"?>
<library>
  <wormholes>
      Definition of all the available wormholes goes here
  </wormholes>
  <stars>
      Definition of all the available Stars goes here
  </stars>
  <memmap>
      Definition of all the available star address range requirements
  </memmap>
</library>
```

3.5.1 Wormhole XML definition

A wormhole definition is started with a tag that has the name of the worm hole. After the tag we will list all the ports of the wormhole. Each port is built using three fields that are separated using a colon (":") and each port is separated using a pipe ("|"). The definition of a port requires three mandatory fields:

- Port name
- Port width
- Port direction (in, out or inout)

Below you will find the definitions of the wormholes that are used in section ... and ...

```
<cmdclk_out>cmdclk:1:out</cmdclk_out>

<cmdclk_in>cmdclk:1:in</cmdclk_in>

<cmd_out>cmdout:64:out|cmdout_val:1:out</cmd_out>

<cmd_in>cmdin:64:in|cmd_val:1:in</cmd_in>

<wh_in>in_stop:1:out|in_dval:1:in|in_data:64:in</wh_in>

<wh_out>out_stop:1:in|out_dval:1:out|out_data:64:out</wh_out>
```

There is one reserved combination for the identification of a wormhole where the ports connect to the I/O pins of the FPGA. A wormhole with external connections should start with the prefix “ext_”. That means a wormhole without external connections should never start with the prefix “ext_”. Below is an example of a wormhole to connect to the external clock inputs of the FPGA:

```
<ext_clkin>clk125_B0n:1:in|clk125_B0p:1:in|clk_synth_Bn:1:in|clk_synth_BP:1:in</ext_clkin>
```

3.5.2 Star XML definition

Like the wormhole, the star definition is built by a tag that has the name of the star. Inside the tags there will be the definition of all the wormholes that are used in the star. Each wormhole is assigned a name and then a colon (“:”) followed by the wormhole type. Each wormhole definition is separated using a pipe (“|”). Below you can see an example definition of a star and a schematic representation of the star.

```
<sipSTAR>cmdclk_in:cmdclk_in|cmd_in:cmd_in|cmd_out:cmd_out|ext_cl_x:ext_cl_x|clk:clkin|rst:rst_in|out0:wh_out|generic:generic_def</sipSTAR>
```

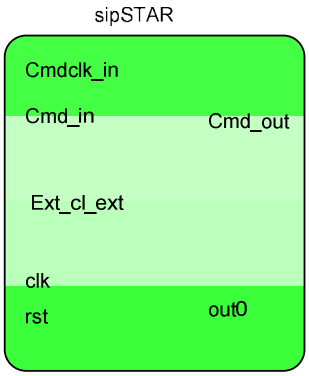

Figure 7: Example star schematic

As you can see there is also a wormhole in the star definition with the name "generic" with the type generic_def. This is a specific wormhole that is used to add a generic to the entity of the star. This generic wormhole is typically defined as followed.

```
<generic_def>global_start_addr_gen:28:in|global_stop_addr_gen:28:in|private_start_
addr_gen:28:in|private_stop_addr_gen:28:in</generic_def>
```

This is used to automatically assign the star address ranges during the constellation creation.

The Generic_def wormhole should always be the last wormhole in the star definition.

3.5.3 Star memmap XML definition

The library should hold the definition of the amount of registers that each star uses. In case the number of registers is not zero the star definition should have the generic port as described in the previous section.

The memmap definition starts with a tag that has the name of the star. Then it has the fixed identifier "nbr_registers" followed by a colon (":") and the number of registers that the Star requires.

```
<sipcl_base>nbr_registers:5</sipcl_base>
```

4 How to create a constellation

Creation of a constellation is possible once a complete Stellar IP library has been created. A constellation is defined as a group of stars that are interconnected using wormholes. Typically the constellation is described in a text file with the extention .sdf (stellar IP description file). It is advised to follow a fixed directory structure for each constellation. Below is the complete directory structure that 4DSP uses for its constellation projects.

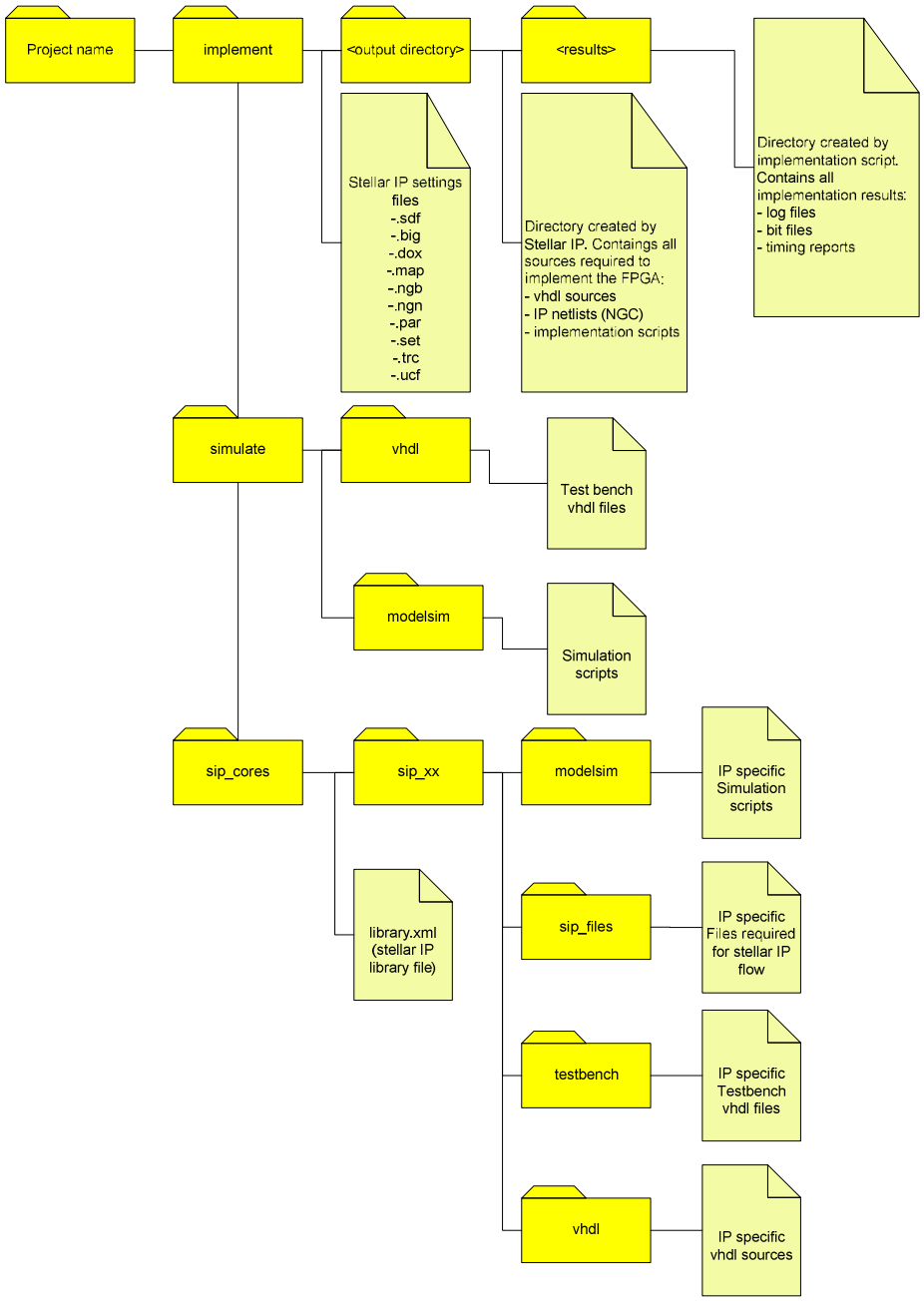

Figure 8: Stellar IP constellation directory structure

In the root directory of the project there will be three folders: implement, simulate and sip_cores. The sip_cores directory will hold a stellar IP compatible library of stars. The simulate directory will be comprised of a vhdl directory with all the files that are required to simulate the constellation. The

implement directory will hold all the files that are required by Stellar IP to create the constellation top level and the files to implement the design using the Xilinx tools.

4.1 SDF file

The SDF file is used as the main input file for the Stellar IP software. This file holds all the information that is required to generate and to implement the constellation. The SDF file should consist of the following sections:

- Target hardware
- Project information
- Memory map information
- Star declaration
- Star connections

4.1.1 Target Hardware

In the target hardware section we define what hardware and which FPGA will be used. This information is required by Stellar IP in order to choose the right ucf files for Stars with external connections. It also requires this information to properly setup the Xilinx tools.

The section needs to have 3 lines

```
TRGT_BOARD=<board name>;

TRGT_FPGA=<FPGA identifier>;

TRGT_FPGA_TYPE=<fpga part number>
```

Currently, stellar IP supports the following target boards from 4DSP:

- FM489
- FM482

The target FPGA can be:

- B

The target device type should match the actual device that is mounted on the target board:

- FM489
    - XC5VSX95T-1FF1136
    - XC5VLX110T-1FF1136
- FM482
    - XC4VLX80- 10ff1148
    - XC4VSX55- 10ff1148
    - XC4VLX40- 10ff1148

Below is an example that can be used on an FM489 with an SX95T fpga.

```
TRGT_BOARD=FM489;

TRGT_FPGA=B;

TRGT_FPGA_TYPE=XC5VSX95T-1FF1136;
```

4.1.2 Project information

The project information section holds specific information to identify the project. The following lines should be part of the project information section:

```
PROJ_NAME=<name to identify the project, no spaces allowed>;

PROJ_DESC=<brief description of the project>;

PROJ_NOTE=<notes related to the project>;

PROJ_TYPE=<project type>;

PROJ_REV=<two integers separated by a dot x.x);

PROJ_AUTHOR=<name of author>;
```

4.1.3 Memory Information

This section defines the size of the global memory range that will be reserved for all the stars. Unless required we advise to use the following settings:

```
MEM_GLOBAL_START=0x0000000;

MEM_GLOBAL_STOP=0x0001FFF;
```

4.1.4 Star declaration

This section is used to declare all the stars that will be used in the constellation. Each star is declared using the following format:

```
STAR <star_name>, ID=<star_id>, FILE=<star_lst_path>;
```

Where:

- star_name = any star present in stellar IP
- star_id = index of the star (in case there are several instances of the same star in the design)
- star_lst_path = path containing the .lst file

4.1.5 Star wormhole connections

In this section we will connect all the input wormholes of each star to an output wormhole on another star. The order in which the specific wormholes are called does not matter, but it is advised to group all the input wormholes from one star together. The wormhole connections are made using the following syntax:

```
<star_name>.<star_id> ( <port> )  <=  <star_name>.<star_id> ( <port> );
```

Where:

- star_name = any star present in stellar ip
- star_id = index of the star (in case there are several instances of the same star in the design)
- Port = port required to be connected on the star2

4.1.6 Example SDF file

As an example, this chapter will show the SDF file that belongs to the constellation shown in Figure 9. This constellation allows access to three external QDR devices through the interFPGA bus. The QDR interface is built using two stars, the sip_qdr_fifo star and the sip_qdr star. Access between the inter fpga star (sip_ifpga_sl) and the three QDR FIFOs are managed using two router stars. One routes the incoming data to one of three possible outputs that each connect to a QDR FIFO. The other router star is responsible for routing one of three possible inputs (which is connected to the QDR FIFO outputs) to one output.

Furthermore, there is the helper star (sip_clkrst_fm489) that is responsible for the clock and reset distribution.

The interfpga slave star functions as the command master in this example. Please refer to section 0 for an explanation. The returning command busses are multiplexed using command multiplexing star.

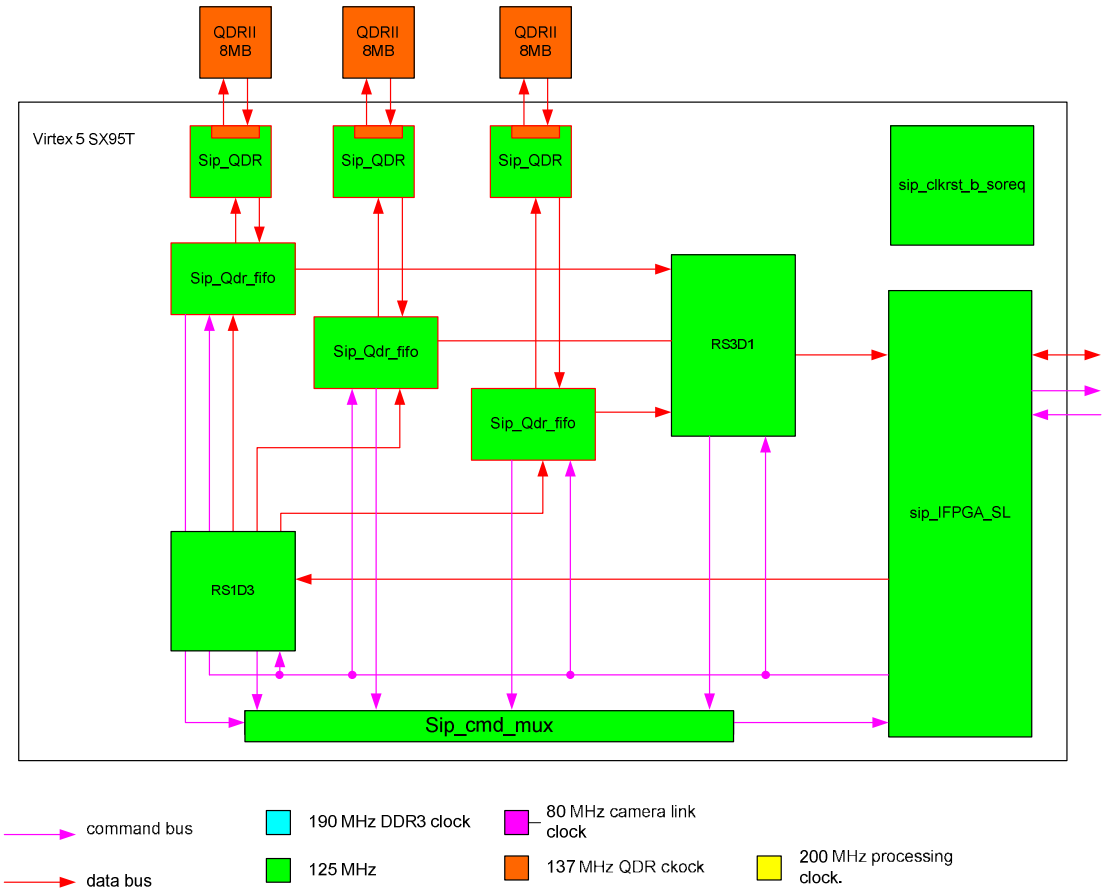

Figure 9: Example constellation

The SDF file to describe the constellation will look like this:

```
----------------------------------------
-- Declare the board type and the FPGA
----------------------------------------
TRGT_BOARD=FM489;
TRGT_FPGA=B;
TRGT_FPGA_TYPE=XC5VSX95T-1FF1136;

----------------------------------------
-- Project Information
----------------------------------------
PROJ_NAME=fm489_qqq_test_sx95t;
PROJ_DESC=BLAST test for the FM489 with 3 QDR BLASTs;
PROJ_NOTE=No special notes;
PROJ_TYPE=Reference StellarIP Design;
PROJ_REV=0.1;
PROJ_AUTHOR=Erik Barhorst ;

------------------------------------------------------------
-- Optional path pointing to some input settings files
------------------------------------------------------------
--PROJ_CONSTRAINTS=cds_2d_fft.ucf;
--PROJ_XSTSETTINGS=cds_2d_fft.set;
--PROJ_DOXYSETTINGS=cds_2d_fft.dox;
```

```
--PROJ_MAPSETTINGS=cds_2d_fft.map;

-------------------------------------------------------------------
-- Memory Information (area of the broadcast memory area)
-------------------------------------------------------------------
MEM_GLOBAL_START=0x0000000;
MEM_GLOBAL_STOP=0x0001FFF;

----------------------
-- Declare the stars.
----------------------

-- STAR <star_name>, ID=<star_id>, FILE=<star_lst_path>;
-- star_name     : anY star present in stellar ip
-- star_id       : index of the star ( we need that if we've got several instances of the same
--               : star in the design
-- star_lst_path : path containing the .lst file
--
STAR sipQDR            ,ID=0 ,FILE=sip_QDR\sip_files\sipQDR_fm489.lst;
STAR sipQDR            ,ID=1 ,FILE=sip_QDR\sip_files\sipQDR_fm489.lst;
STAR sipQDR            ,ID=2 ,FILE=sip_QDR\sip_files\sipQDR_fm489.lst;
STAR sipclkrst_fm489    ,ID=0 ,FILE=sip_clkrst_fm489\sip_files\sipclkrst_fm489.lst;
STAR sipIFPGA_SL        ,ID=0   ,FILE=sip_IFPGA_SL\sip_files\sipIFPGA_SL.lst;
STAR sipqdr_fifo        ,ID=0   ,FILE=sip_qdr_fifo\sip_files\sipqdr_fifo.lst;
STAR sipqdr_fifo        ,ID=1   ,FILE=sip_qdr_fifo\sip_files\sipqdr_fifo.lst;
STAR sipqdr_fifo        ,ID=2   ,FILE=sip_qdr_fifo\sip_files\sipqdr_fifo.lst;
STAR siprouter_s1d3     ,ID=0 ,FILE=sip_router_s1d3\sip_files\siprouter_s1d3.lst;
STAR siprouter_s3d1     ,ID=0 ,FILE=sip_router_s3d1\sip_files\siprouter_s3d1.lst;

STAR sipcmd8_mux        ,ID=0 ,FILE=sip_cm8_mux\sip_files\sipcmd8_mux.lst;

-------------------------
-- Cores connections
-------------------------
-- <star_name>.<star_id> ( <port> )  <=  <star_name>.<star_id> ( <port> );
-- star_name     : any star present in stellar ip
-- star_id       : index of the star ( we need that if we've got several instances of the same
--               : star in the design
-- port          : port required to be connected on the star2
--

--connect sip QDR 0 inputs
sipQDR.0(cmd_in)        <= sipIFPGA_SL.0(cmd_out);
sipQDR.0(cmdclk_in)     <= sipIFPGA_SL.0(cmdclk_out);
sipQDR.0(qdr_clk)       <= sipclkrst_fm489.0(clkout);
sipQDR.0(rst)           <= sipclkrst_fm489.0(rst_out);
sipQDR.0(wr_addr)       <= sipqdr_fifo.0(wr_addr);
sipQDR.0(wr_data)       <= sipqdr_fifo.0(wr_data);
sipQDR.0(rd_addr)       <= sipqdr_fifo.0(rd_addr);

--connect sip QDR 1 inputs
sipQDR.1(cmdclk_in)     <= sipIFPGA_SL.0(cmdclk_out);
sipQDR.1(cmd_in)        <= sipIFPGA_SL.0(cmd_out);
sipQDR.1(qdr_clk)       <= sipclkrst_fm489.0(clkout);
sipQDR.1(rst)           <= sipclkrst_fm489.0(rst_out);
sipQDR.1(wr_addr)       <= sipqdr_fifo.1(wr_addr);
sipQDR.1(wr_data)       <= sipqdr_fifo.1(wr_data);
sipQDR.1(rd_addr)       <= sipqdr_fifo.1(rd_addr);

--connect sip QDR 2 inputs
sipQDR.2(cmdclk_in)     <= sipIFPGA_SL.0(cmdclk_out);
sipQDR.2(cmd_in)        <= sipIFPGA_SL.0(cmd_out);
sipQDR.2(qdr_clk)       <= sipclkrst_fm489.0(clkout);
sipQDR.2(rst)           <= sipclkrst_fm489.0(rst_out);
sipQDR.2(wr_addr)       <= sipqdr_fifo.2(wr_addr);
sipQDR.2(wr_data)       <= sipqdr_fifo.2(wr_data);
sipQDR.2(rd_addr)       <= sipqdr_fifo.2(rd_addr);
```

```
--connect sip QDR fifo 0 inputs
sipqdr_fifo.0(cmdclk_in)<= sipIFPGA_SL.0(cmdclk_out);
sipqdr_fifo.0(cmd_in)   <= sipIFPGA_SL.0(cmd_out);
sipqdr_fifo.0(rd_data)  <= sipQDR.0(rd_data);
sipqdr_fifo.0(clk)      <= sipclkrst_fm489.0(clkout);
sipqdr_fifo.0(rst)      <= sipclkrst_fm489.0(rst_out);
sipqdr_fifo.0(fifo_in)  <= siprouter_s1d3.0(out0);

--connect sip QDR fifo 1 inputs
sipqdr_fifo.1(cmdclk_in)<= sipIFPGA_SL.0(cmdclk_out);
sipqdr_fifo.1(cmd_in)   <= sipIFPGA_SL.0(cmd_out);
sipqdr_fifo.1(rd_data)  <= sipQDR.1(rd_data);
sipqdr_fifo.1(clk)      <= sipclkrst_fm489.0(clkout);
sipqdr_fifo.1(rst)      <= sipclkrst_fm489.0(rst_out);
sipqdr_fifo.1(fifo_in)  <= siprouter_s1d3.0(out1);

--connect sip QDR fifo 2 inputs
sipqdr_fifo.2(cmdclk_in)<= sipIFPGA_SL.0(cmdclk_out);
sipqdr_fifo.2(cmd_in)   <= sipIFPGA_SL.0(cmd_out);
sipqdr_fifo.2(rd_data)  <= sipQDR.2(rd_data);
sipqdr_fifo.2(clk)      <= sipclkrst_fm489.0(clkout);
sipqdr_fifo.2(rst)      <= sipclkrst_fm489.0(rst_out);
sipqdr_fifo.2(fifo_in)  <= siprouter_s1d3.0(out2);

--connect siprouter_s3d1 inputs
siprouter_s3d1.0(cmdclk_in)   <= sipIFPGA_SL.0(cmdclk_out);
siprouter_s3d1.0(cmd_in)      <= sipIFPGA_SL.0(cmd_out);
siprouter_s3d1.0(clk)         <= sipclkrst_fm489.0(clkout);
siprouter_s3d1.0(rst)         <= sipclkrst_fm489.0(rst_out);
siprouter_s3d1.0(in0)         <= sipqdr_fifo.0(fifo_out);
siprouter_s3d1.0(in1)         <= sipqdr_fifo.1(fifo_out);
siprouter_s3d1.0(in2)         <= sipqdr_fifo.2(fifo_out);

--connect siprouter_s1d3 inputs

siprouter_s1d3.0(cmdclk_in)   <= sipIFPGA_SL.0(cmdclk_out);
siprouter_s1d3.0(cmd_in)      <= sipIFPGA_SL.0(cmd_out);
siprouter_s1d3.0(clk)         <= sipclkrst_fm489.0(clkout);
siprouter_s1d3.0(rst)         <= sipclkrst_fm489.0(rst_out);
siprouter_s1d3.0(in0)         <= sipIFPGA_SL.0(out_data);

--connect sipIFPGA_SL inputs
sipIFPGA_SL.0(cmd_in)   <= sipcmd8_mux.0(cmd_out);
sipIFPGA_SL.0(clkin)    <= sipclkrst_fm489.0(clkout);
sipIFPGA_SL.0(rst_in)   <= sipclkrst_fm489.0(rst_out);
sipIFPGA_SL.0(in_data)  <= siprouter_s3d1.0(out0);

--reset from if fpga slave to the clk reset
sipclkrst_fm489.0(cmdclk_in)     <= sipIFPGA_SL.0(cmdclk_out);
sipclkrst_fm489.0(cmd_in)        <= sipIFPGA_SL.0(cmd_out);
sipclkrst_fm489.0(ifpga_rst_in)  <=  sipIFPGA_SL.0(ifpga_rst_out);


--Connect the command mux wormholes
sipcmd8_mux.0(cmdclk_in)         <= sipIFPGA_SL.0(cmdclk_out);
sipcmd8_mux.0(cmd0_in)           <= sipQDR.0(cmd_out);
sipcmd8_mux.0(cmd1_in)           <= sipQDR.1(cmd_out);
sipcmd8_mux.0(cmd2_in)           <= sipQDR.2(cmd_out);
sipcmd8_mux.0(cmd3_in)           <= sipclkrst_fm489.0(cmd_out);
sipcmd8_mux.0(cmd4_in)           <= sipqdr_fifo.0(cmd_out);
sipcmd8_mux.0(cmd5_in)           <= sipqdr_fifo.1(cmd_out);
sipcmd8_mux.0(cmd6_in)           <= sipqdr_fifo.2(cmd_out);
sipcmd8_mux.0(cmd7_in)           <= siprouter_s1d3.0(cmd_out);
sipcmd8_mux.0(cmd8_in)           <= siprouter_s3d1.0(cmd_out);
```

4.2 UCF file

This file can hold some global constraints to be used during the implementation of the constellation using the Xilinx tools. This file should be available, but can be left empty.

4.3 TRC file

This file can hold some command line arguments that will be used by the TRC tool. This file should be available, but can be left empty.

4.4 SET file

This file can hold some global constraints to be used during the implementation of the constellation using the Xilinx tools. This file should be available, but can be left empty.

4.5 PAR file

This file can hold some command line arguments that will be passed to the PAR tool when called by the implementation script. This file should be available, but can be left empty.

4.6 NGN file

This file can hold some command line arguments that will be passed to the NETGEN tool when called by the implementation script. This file should be available, but can be left empty.

4.7 NGB file

This file can hold some command line arguments that will be passed to the NGBUILD tool when called by the implementation script. This file should be available, but can be left empty.

4.8 MAP file

This file can hold some command line arguments that will be passed to the MAP tool when called by the implementation script. This file should be available, but can be left empty.

4.9 DOX file

This file should be available, but can be left empty.

4.10 BIG file

This file can hold some command line arguments that will be passed to the BITGEN tool when called by the implementation script. This file should be available, but can be left empty.

4.11 Constellation identification

In general, the host interface is able to identify the constellation. 4DSP has defined that each constellation should hold a Constellation Identification (CID) Star. This star holds specific information about the constellation and is accessed through the command interface. A top-level diagram of the sip_CID star is depicted in the following figure.

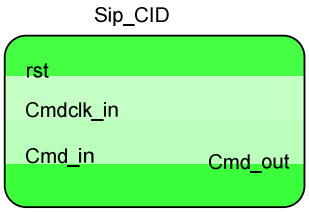

Figure 10: sip_CID top level diagram

The first three 32 bits registers will hold the information about the constellation and is formatted as follows:

offset	Bit 31		Bit 16	Bit 15	Bit 8	Bit 7	Bit 0
0x00	CID			NB_STAR			
0x01	SW_BUILD						
0x02	FW_BUILD						
0x03				CID_VER_HI		CID_VER_LO	
0x04	RSVD						
0x05	RSVD						
0x06	RSVD						
0x07	RSVD						
0x08	STAR0 BAR						
0x09	STAR0 EAR						
0x0A	STAR0_ID			STAR0_version			
0x0B	STAR1_BAR						
0x0C	Etc..						
0x0D	Etc..						

Table 3: CID register map

The CID is a unique 16 bits value that identifies the constellation.

The NB_STAR field identifies how many stars the constellation has. This field can be used to determine how many register the CID Star has in total because there will be three 32 bits registers per Star starting from the register at offset 8.

SW_BUILD is a number that is a date code to uniquely identify when the Stellar IP software created the constellation.

FW_BUILD is a number that has to be incremented each time the ISE tools implement the design

The CID_VER_HI and CID_VER_LO identify the revision of the constellation.

STARx_BAR is the base address at which the star with index x register map begins. If this value is zero then the star has no registers.

STARx_EAR is the end address at which the star with index x register map ends. If this value is zero then the star has no registers.

The sip_CID star is configured using a VHDL package file that assigns the specific values for each of the registers. This package file is generated on the fly by the stellar IP software tool.

5 Running stellar IP

Once a valid star library is created and a constellation description is made we can let Stellar IP generate:

- VHDL top-level file for the constellation
- ISE project files
    - UCF
    - Implementation batch file
    - Copy all required sources to the correct location
    - Create an ISE project file
- Create a software header file with all the star register base addresses and star register offsets.
- Create a modelsim compilation script to compile the FPGA.
- Call the Xilinx tool chain to implement the design.