

Getting Started SpaceWire Codec

Date: 26 April 2019

Key Words: 4Links, SpaceWire Codec, FPGA, ASIC, IP

This document gives an overview of the features and capabilities for the SpaceWire Codec. It details the Out of Box test to prove the functionality of the IP delivered and an integration example to build the SpaceWire Codec design for operation in the PXle_S6 board from 4Links Ltd. Additional information is included covering the integration aspects of taking the IP and putting it into the users design and what files need to be modified to support the target FPGA/ASIC technology.

Copyright (c) 2019, 4Links Ltd All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement: This product includes software developed by 4Links Ltd.
4. Neither the name of 4Links Ltd nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY 4LINKS LTD "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL 4LINKS LTD BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

1 CONTENTS

2	Introduction	5
2.1	Features	5
3	Out Of Box testing.....	5
3.1	Required Software tools	5
3.2	Required Hardware	5
3.3	Directory Structure	5
3.3.1	spw_codec.....	6
3.3.2	common	7
4	Getting Started.....	7
4.1	Installing the IP.....	7
4.2	Integration test	7
4.2.1	Building the FPGA Image.....	7
4.2.2	Programming the PXle_S6	7
4.2.3	Running a simulation under Modelsim/Questa.....	8
4.2.4	Running a simulation under ISE	8
4.3	Out of Box test	9
4.3.1	Running a simulation under Modelsim/Questa.....	9
5	Architectural Overview	10
5.1	Receive link	10
5.1.1	spw_rx_to_2b.vhd	10
5.1.2	spw_rx_sync.vhd.....	11
5.1.3	spw_rx_to_data.vhd	11
5.1.4	spw_rx_fifo_2c.vhd.....	11
5.1.5	spw_rx_add_epp.vhd.....	11
5.1.6	spw_rx_filter_error.vhd	11
5.1.7	spw_timeout_det.vhd.....	11
5.1.8	spw_rx_bit_rate	11
5.2	Control	12
5.2.1	spw_ctrl.vhd.....	12
5.2.2	spw_rx_flowcredit_x.vhd.....	12
5.3	Transmit Link.....	12
5.3.1	spw_tx_discard.vhd	12
5.3.2	spw_tx_flowcontrol.vhd	12
5.3.3	spw_tx_ds.vhd	12
6	Signal Description.....	13

6.1	Clock and Reset	13
6.2	SpaceWire	13
6.3	Data Flow Channels.....	13
6.3.1	Received Data Flow Channel.....	13
6.3.2	Received Time Data Flow Channel.....	13
6.3.3	Transmit Data Flow Channel	13
6.3.4	Transmit Time Data Flow Channel	14
6.4	Control and Status	14
6.5	Generics	14
7	Signal Timing	15
7.1	Clock and Reset	15
7.2	SpaceWire	15
7.2.1	Received Data	15
7.2.2	Transmit Data	15
7.3	Data Flow Channel	16
7.4	Control and Status	16
8	Integration	17
8.1	Reset synchronisation	17
8.2	Clock generation	17
8.3	IO Pads	18
8.3.1	Input	18
8.3.2	Output	18
8.3.3	UCF constraints	18
8.4	Timing Constraints	19
8.4.1	Clock and Reset	19
8.4.2	SpaceWire interface.....	19
8.4.3	Data Flow Channel	19
8.4.4	Control and Status.....	19
9	Test Bench.....	20
9.1	Stimulus & Log File Format	20
9.2	Out of Box	20
9.3	Integration	21

2 INTRODUCTION

The SpaceWire Codec IP is designed and developed by 4Links Ltd. It enables the user to easily and efficiently implement a Codec solution into their design. It is fully compliant with ECSS-50-12A, implemented as a fully synchronous design (using 4Links patented technology) and requiring very little constraint management of the IP for integration into FPGAs or ASICs.

2.1 FEATURES

The table lists the main features of the Codec.

Feature	Performance	Notes
Clock Frequency	>200MHz	Implementation in Spartan6-3C device
Design Size	~423 LUTs, 1 BRAM	Implementation in Spartan6-3C device
Data Rate	>200Mbps	Based on the DDR oversampling design
Receive Buffer	8-56 Bytes	Configurable at synthesis
Transmit Buffer	None	

The top level wrapper (spw_wrap.vhd) contains all the target technology specific IP. This is the only file that needs changing to customise to the user's specific FPGA family. All Xilinx, Altera, MicroSemi FPGAs which support input DDR FlipFlops in the IO pads are supported. The implementation of LVDS IO is advisable for signal integrity and performance issues, but not mandatory to implement the IP.

3 OUT OF BOX TESTING

3.1 REQUIRED SOFTWARE TOOLS

To rebuild the FPGA design, you will need to install Xilinx™ ISE WebPack version 14.7 on to the host computer and ensure that the executables are in the search path.

To upload the design to the 4Links PXle_S6 board you will also need to install Java™ 7/8 runtime environment from Oracle™ and ensure that is in the search path.

The hidio.jar application is in the root directory of the IP together with the build and programming batch files. A newer version of hidio.jar maybe available from the 4Links CD and can be used in place of the version supplied.

If you want to perform simulation, then either the Xilinx ISE simulator or Modelsim (Questa) can be used. Scripts for Modelsim are in the sim\Modelsim directory. Both standalone IP and FPGA level testbenches are included in the deliverables.

3.2 REQUIRED HARDWARE

The hardware platform used for the example implementation is the 4Links PXle_S6 board, which has four SpaceWire interfaces, a mini-USB configuration connector and a colour OLED display on the front panel. The board contains a Xilinx Spartan6 FPGA (XC6SLX45T-3CSG324C) and ISSI asynchronous SRAM (IS61WV51216BLL-10TLI), implemented as 512kx16bits (1MB).

3.3 DIRECTORY STRUCTURE

The directory structure used by 4Links for their deliverables is split into two main areas;

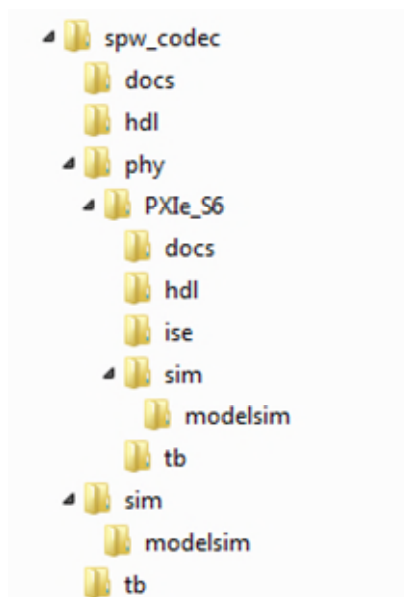
- ip_4l directory contains the 4Links IP that is incorporated in the design.
- ip_ext contains the 3rd party IP that is incorporated in the design.

These two directories are under the 4Links deliverable directory <4Links_root>.

The spw_codec IP does not include any 3rd party IP (except for wrappers to the FPGA technology) so ip_ext is not present in this delivery.

3.3.1 spw_codec

The directory spw_codec is under the ip_4l directory and contains all the IP and implementations specific to the IP deliverable, including scripts to run the simulations of the Codec and also compile the example design to the FPGA on the PXIe_S6 board from 4Links.

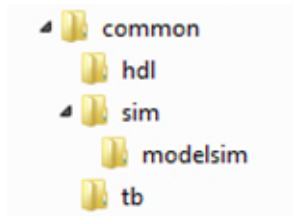


The sub directories are shown below with brief explanations about their function.

- docs: the documents directory containing this and other relevant documents.
- hdl: directory containing the IP VHDL files with a generic wrapper for the user to integrate into their design.
- phy: is the directory with all the example (OoB) physical implementations and associated scripts and wrappers
 - PXIe_S6 is the example target board for the OoB FPGA design.
 - docs: any docs specific to the physical implementation
 - hdl: FPGA top level rtl and wrapper rtl for the IP to implement the OoB design in the target board.
 - ise: directory containing the tcl scripts, ise project file and constraints files used to run ISE and generate the FPGA image. The logs and output files from the run are stored here too.
 - sim: directory for board level simulations to be run
 - modelsim: the Modelsim/Quarta directory with scripts and configuration files for running the board level simulation.
 - tb: behavioural IP for use in the board level simulation.
 - *Other target boards or FPGA technologies can go here!*
- sim: directory for IP level simulations to be run
 - modelsim: the Modelsim/Quarta directory with scripts and configuration files for running the IP level simulation.
- tb: behavioural IP for use in the IP level simulation.

3.3.2 common

The directory common is under the ip_4l directory and contains all the common IP shared between the 4Links deliverables, including header files for common SpaceWire functions and Bus Functional Models for behavioural modelling of the design.



- hdl: directory containing the common VHDL for the user to integrate along with the IP into their design.
- sim: directory for testbench IP level simulations to be run
 - modelsim: the Modelsim/Questa directory with scripts and configuration files for running the testbench IP level simulation.
- tb: behavioural IP for use in the testbench IP level simulation.

4 GETTING STARTED

4.1 INSTALLING THE IP

The IP can be placed anywhere within the users directory structure, as the files are referenced by relative paths. The location of the files relative to the top-level cannot be moved. This level is referred to as <4Links_root>.

4.2 INTEGRATION TEST

The FPGA image generation for running on the PXIe_S6 board and the associated simulation are called the Integration test, as it demonstrates how the IP is integrated into an FPGA design.

4.2.1 Building the FPGA Image

Once the Xilinx ISE 14.7 WebPack software is installed on the host computer then you can execute the fpga_build.bat file under windows to build the FPGA image. This is located at the top-level of the directory structure.

```
<4Links_root>\ip_4l\spw_codec\phy\PXIe_S6\ise\fpga_build.bat
```

4.2.2 Programming the PXIe_S6

Once ISE has completed building the FPGA image (fpga_top.bit in the ise directory), the PXIe_S6 board can be programmed.

- Power on the PXIe_S6 board
- Connect a USB cable from the host computer to the front panel of the PXIe_S6.
- Run the batch file fpga_prog.bat.
 - <4Links_root>\ip_4l\spw_codec\phy\PXIe_S6\ise\fpga_prog.bat
- Once the programming has completed and verified, Power cycle the PXIe_S6 board.
- Connect the board inline between an existing link and traffic should transfer automatically
 - Use ports SpW1 and SpW2 to prove IP
- If the SpaceWire link is not functional then perform the following check
 - Use ports SpW3 and SpW4 as an electrical route through to prove the cabling and system is working.

4.2.3 Running a simulation under Modelsim/Quarta

To run the integration simulation under modelsim, change into the modelsim directory and run dosim.bat.

```
<4Links_root>\ip_4l\spw_codec\phy\PXIe_S6\sim\modelsim\dosim.bat
```

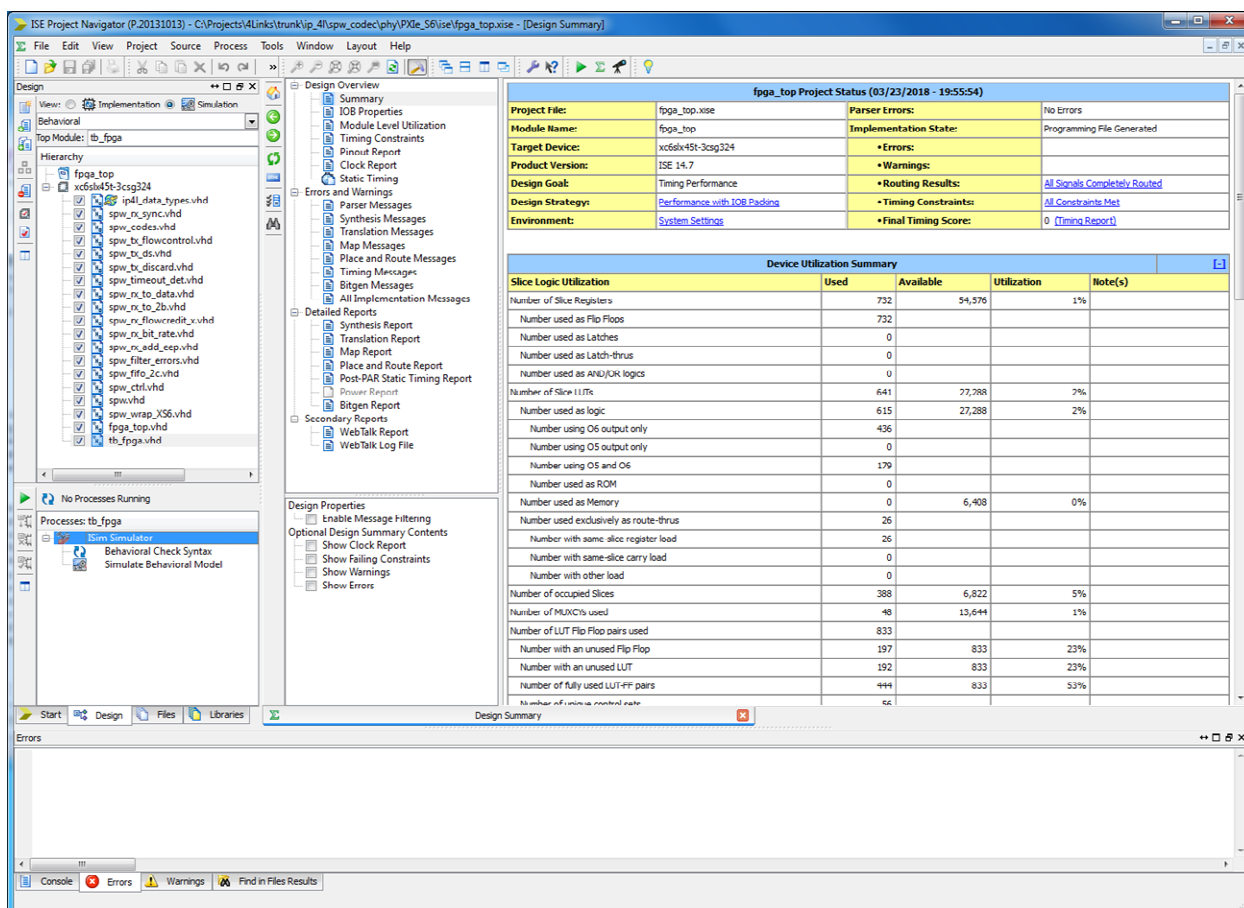
The simulation should run for 100 us and not generate any error outputs on the transcript window of the simulator.

4.2.4 Running a simulation under ISE

To run the integration simulation under ISE (ISIM), change into the ise directory and select the project fpga_top.ixse.

```
<4Links_root>\ip_4l\spw_codec\phy\PXIe_S6\ise\fpga_top.ixse
```

The ISE GUI will open, then select simulation on the view panel, tb_fpga.vhd in the Hierarchy panel and ISim Simulator in the Process panel.



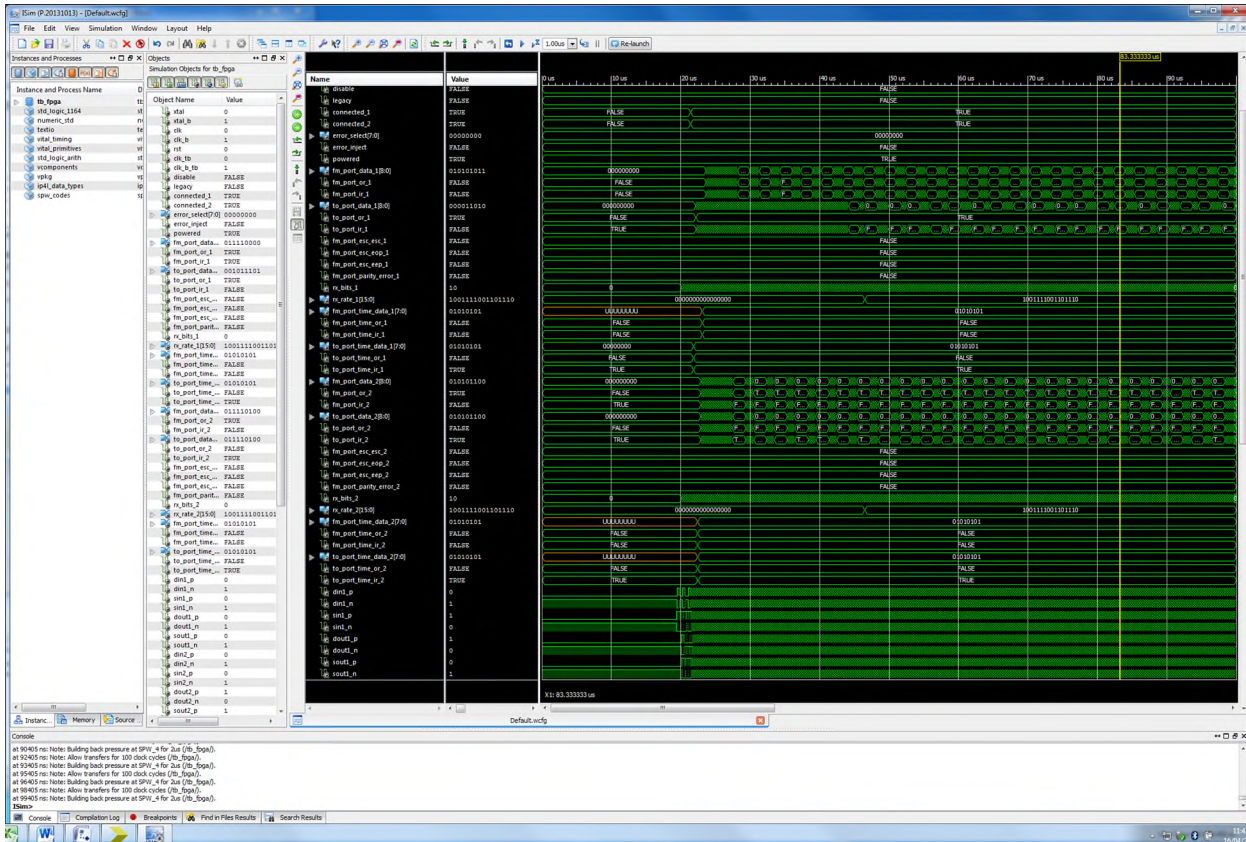
fpga_top Project Status (03/23/2018 - 19:55:54)

Project File:	fpga_top.ixse	Parser Errors:	No Errors
Module Name:	fpga_top	Implementation State:	Programming File Generated
Target Device:	xc6slx45t-3csg324	Errors:	
Product Version:	ISE 14.7	Warnings:	
Design Goal:	Timing Performance	Routing Results:	All Signals Completely Routed
Design Strategy:	Performance with IOB Padding	Timing Constraints:	All Constraints Met
Environment:	System Settings	Final Timing Score:	0 (Timing Report)

Device Utilization Summary

Slice Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Registers	732	54,576	1%	
Number used as Flo Flops	732			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	0			
Number of Slice LUTs	641	27,288	2%	
Number used as logic	615	27,288	2%	
Number using O6 output only	436			
Number using O5 output only	0			
Number using O5 and O6	179			
Number used as ROM	0			
Number used as Memory	0	6,408	0%	
Number used exclusively as route-thrus	26			
Number with same-slice register load	26			
Number with same-slice carry load	0			
Number with other load	0			
Number of occupied Slices	388	6,822	5%	
Number of MUXCt0s used	48	13,644	1%	
Number of LUT Flip Flop pairs used	833			
Number with an unused Flip Flop	197	833	23%	
Number with an unused LUT	192	833	23%	
Number of fully used LUT-FF pairs	444	833	53%	
Number of unused output pins	66			

Double Clicking on the Simulate Behavioural Model icon will start the simulation and you should see a waveform display appear to show the signals.



The simulation should run for 100 us and not generate any error outputs on the transcript window of the simulator.

4.3 OUT OF BOX TEST

This is a Codec only simulation to prove it can synchronise and transfer data between two codecs, using a bus traffic generator and a data logger. Once the simulation has completed you can compare the two files and see a match if the transfers are all correct.

4.3.1 Running a simulation under Modelsim/Questa

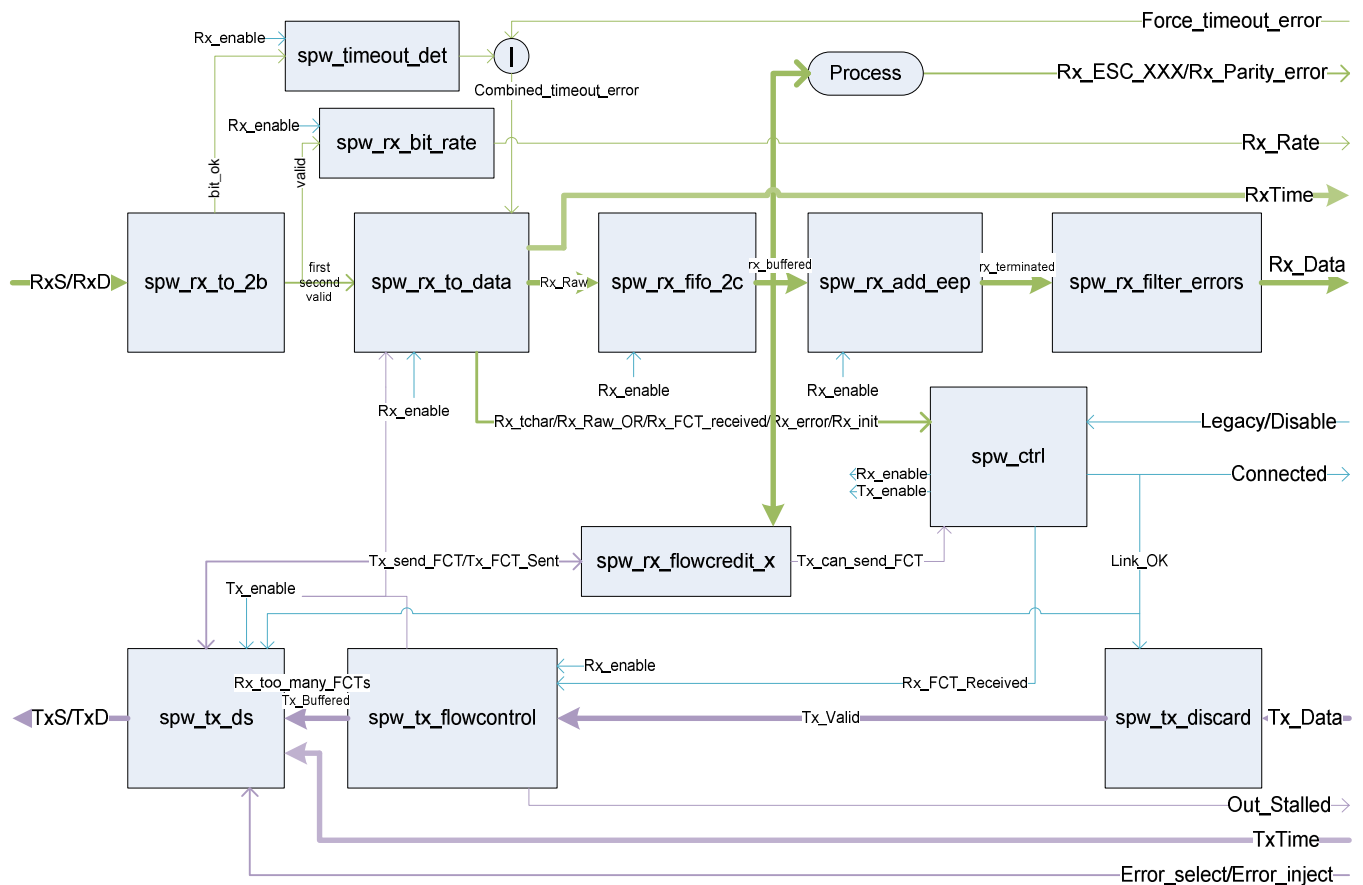
To run the integration test under Modelsim, change into the modelsim directory and run dosim.bat.

```
<4Links_root>\ip_41\spw_codec\sim\modelsim\dosim.bat
```

The simulation should run for 100 μ s and not generate any error outputs on the transcript window of the simulator.

5 ARCHITECTURAL OVERVIEW

The diagram shows the functional top-level of SpW. Some of the signal names are grouped for clarity. Please refer to the signal descriptions for more details.

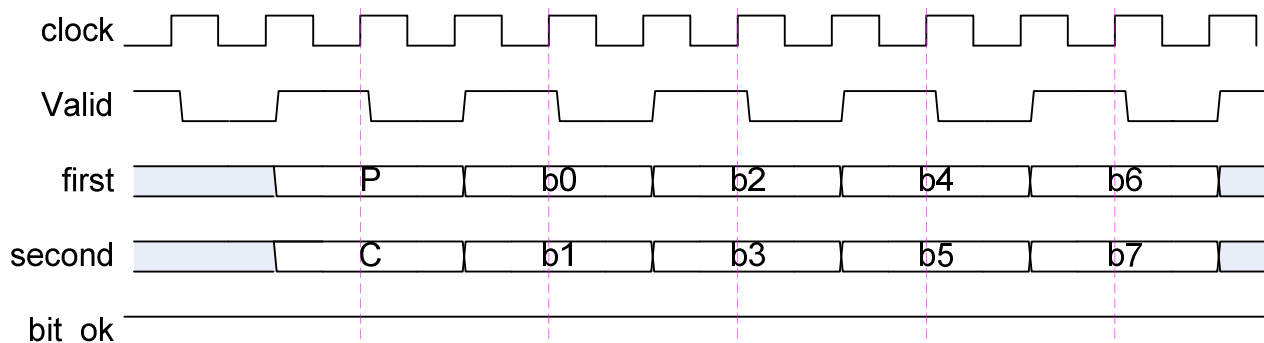


The design can be split into three sections for implementation; Receive link, transmit link and control. Each Block will be discussed below

5.1 RECEIVE LINK

5.1.1 spw_rx_to_2b.vhd

This entity receives the IO Pad DDR flop input signals and generates a two bit data stream as shown below.



The two bit data stream is valid on the rising edge that Valid and bit_ok are high. The data follows the SpaceWire specification with Parity (P) for the previous word followed by the Command/Control bit and then the data bit LSB first (b0-b7).

The signal bit_ok is active high when a valid bit has been detected on the SpaceWire link received channel. The valid signal is active high when two bits are received sequentially and ready for reading.

5.1.2 spw_rx_sync.vhd

This sits as a sub component to spw_rx_to_2b and aligns the IO Pad DDR flop input signals (rising and falling clock edges) for the received Data and Strobe signals to the rising clock edge and then registers them twice to reduce metastability due to the over sampling implementation.

5.1.3 spw_rx_to_data.vhd

The two bit data from spw_rx_to_2b is converted into the 9bit data representing a SpaceWire Data or Control character.

The Received TimeCodes are decoded and output by spw_rx_to_data

5.1.4 spw_rx_fifo_2c.vhd

A FIFO buffer is implemented to control the data flow and buffer the received data. This is configurable by the parameter 'RX_FIFO_SIZE'.

The data from spw_rx_fifo_2c is used for received flow credit control and is also the output for the ESC Control characters received on the SpaceWire link.

5.1.5 spw_rx_add_epp.vhd

Checks the data packets and adds the EPP if there is an error.

5.1.6 spw_rx_filter_error.vhd

The input 'Report_all_errors' is used to enable passing through of all data to the Received output regardless of errors.

If a mid packet error occurs and report all errors is not set then an EEP will be generated on the received data.

5.1.7 spw_timeout_det.vhd

The timeout detection is triggered on two conditions, defined in the SpaceWire specification. These are 200ns and 850ns timeouts, based on the bit_ok signal from spw_rx_to_2b. The timeouts are only generated when the enable signal is active.

The timeout periods are generated from the parameter 'CLOCK_FREQUENCY'.

The signal 'paused' goes active after the 200ns timeout period.

The signal 'timeout_error' goes active after the 850ns timeout period.

5.1.8 spw_rx_bit_rate

The function of spw_rx_bit_rate is to generate a number representing the rx_rate of the received data stream based on the valid signal from spw_rx_to_2b.

The signal bits_per_clock is derived from valid and has a value of 2, so each time a valid pair of bits is detected the count is incremented by two.

The output values for rx_rate is defined below

rx_rate[15:10]	rx_rate[9:0]	Note
100110	Count	1.00 to 9.99 Mbps. 400µs sample period
100111	Count	10.0 to 99.9 Mbps. 40µs sample period

101000	Count	100 to 999 Mbps. 4µs sample period
--------	-------	------------------------------------

5.2 CONTROL

5.2.1 spw_ctrl.vhd

The main control logic is implemented here and uses the top level inputs 'Disable' to enable the Link training and operation, 'Legacy' to select either IEEE-1355 or SpaceWire and the 'Connected' output to reflect when the Link is trained and operational.

It monitors the Rx signals from spw_rx_to_data, flow credits from spw_rx_flowcredit_x and handles all the Flow Control Tokens (FCTs) for transmitting data (spw_tx_flowcontrol).

5.2.2 spw_rx_flowcredit_x.vhd

This monitors and controls the receive data path Flow Control Tokens (FCTs), by monitoring the received data (from spw_rx_fifo_2c) and requests tokens to be sent by spw_tx_ds.

5.3 TRANSMIT LINK

5.3.1 spw_tx_discard.vhd

The transmit data is passed through if the Link_OK is set (Connected). If the link breaks the data will be discarded.

5.3.2 spw_tx_flowcontrol.vhd

If the received FCTs are coming in then spw_tx_flowcontrol will pass through the data to spw_tx_ds. If there are no FCTs it will not send the data and apply backpressure to spw_tx_discard and further down the channel.

If too many FCTs are received then it will signal 'too_many_fcts' back up to spw_rx_to_data. The signal 'out_stalled' is driven when the number of FCTs is zero but data is waiting to be sent.

5.3.3 spw_tx_ds.vhd

The data is serialised and send out to the driving flip flops and LVDS pad drivers.

If 'Tx_send_FCT' is received from spw_rx_flowcredit_x it will send an FCT over the SpaceWire interface and respond with an tx_FCT_Sent back to spw_rx_flowcredit_x.

The sideband signals Error_select and Error_inject will create error data to be sent over the SpaceWire link.

Error_select	Value (SL)	Notes
FORCE_PARITY_ERROR	0011	
FORCE_ESC_EOP	0100	
FORCE_ESC_EEP	0101	
FORCE_ESC_ESC	0110	

6 SIGNAL DESCRIPTION

The top level signals of spw.vhd are listed below with details about their operation.

6.1 CLOCK AND RESET

Signal	Direction (wrt IP)	Type	Size	Polarity	Function
clock	input	SL	1	rising	System Clock for all IP
clock_b	input	SL	1	rising	System Clock inverted for all IP
reset	input	SL	1	high	Reset for System Clock

The clock, clock_b are 180° phase shifted clocks to one another and used to drive the DDR input registers to oversample the input data and strobe signals. This gives a 2x sampling rate of the input signals.

All the clocks (clock, clock_b) in the design are the same frequency, with the clocks being phase aligned.

The reset signal is synchronous to the clock.

6.2 SPACEWIRE

Signal	Direction (wrt IP)	Type	Size	Polarity	Function
RxD_r	input	SL	1	-	Received Data captured on rising edge of clock
RxD_f	input	SL	1	-	Received Data captured on rising edge of clock_b
RxS_r	input	SL	1	-	Received Strobe captured on rising edge of clock
RxS_f	input	SL	1	-	Received Strobe captured on rising edge of clock_b
TxD	output	SL	1	-	Transmit Data output
TxS	output	SL	1	-	Transmit Strobe output

6.3 DATA FLOW CHANNELS

6.3.1 Received Data Flow Channel

Signal	Direction (wrt IP)	Type	Size	Polarity	Function
Rx_Data	output	SLV	9	-	Data received by the SpaceWire interface
Rx_Data_OR	output	Bool	1	high	Strobe to show data is ready to for reading
Rx_Data_IR	input	Bool	1	high	Strobe to show data is ready to be accepted

6.3.2 Received Time Data Flow Channel

Signal	Direction (wrt IP)	Type	Size	Polarity	Function
Rx_Time	output	SLV	9	-	Data received by the SpaceWire interface
Rx_Time_OR	output	Bool	1	high	Strobe to show data is ready to for reading
Rx_Time_IR	input	Bool	1	high	Strobe to show data is ready to be accepted

6.3.3 Transmit Data Flow Channel

Signal	Direction (wrt IP)	Type	Size	Polarity	Function
Tx_Data	input	SLV	9	-	Data to be sent by the SpaceWire interface
Tx_Data_OR	input	Bool	1	high	Strobe to show data is ready to for reading
Tx_Data_IR	output	Bool	1	high	Strobe to show data is ready to be accepted

6.3.4 Transmit Time Data Flow Channel

Signal	Direction (wrt IP)	Type	Size	Polarity	Function
Tx_Time	input	SLV	9	-	Data to be sent by the SpaceWire interface
Tx_Time_OR	input	Bool	1	high	Strobe to show data is ready to for reading
Tx_Time_IR	output	Bool	1	high	Strobe to show data is ready to be accepted

6.4 CONTROL AND STATUS

Signal	Direction (wrt IP)	Type	Size	Polarity	Function
Disable	input	Bool	1	high	Disable link
Legacy	input	Bool	1	high	Select 1355 or SpaceWire
Error_Select	input	SLV	4	-	See 5.3.3 above for details of values
Error_inject	input	Bool	1	high	Inject Error into Tx path
Fore_timeout_error	input	Bool	1	high	Force a timeout Error
Out_Stalled	output	Bool	1	high	No FCTs but data waiting to be sent
Connected	output	Bool	1	high	Link is trained and operational

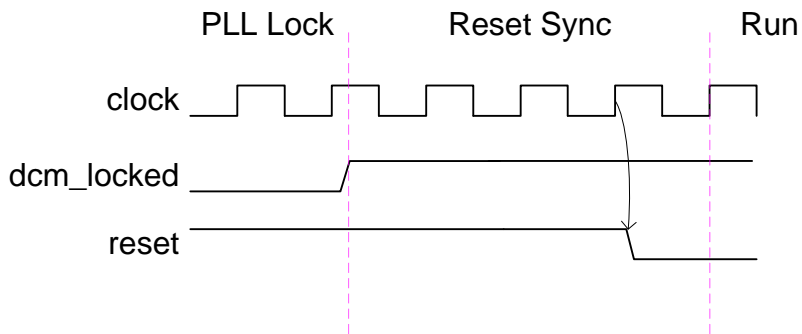
Signal	Direction (wrt IP)	Type	Size	Polarity	Function
Rx_ESC_ESC	output	Bool	1	high	Received ESC ESC
Rx_ESC_EOP	output	Bool	1	high	Received End Of Packet
Rx_ESC_EEP	output	Bool	1	high	Received Error End of Packet
Rx_Parity_error	output	Bool	1	high	Received Parity error
Rx_bits	output	int	2	high	Number of bits per clock (2)
Rx_rate	output	SLV	16	high	See 5.1.8

6.5 GENERICS

Signal	Type	Range	Polarity	Function
CLOCK_FREQUENCY	Real	-	-	Frequency in Hz that the clock is running at
RX_FIFO_SIZE	int	8 - 56	high	Depth of FIFO
TX_FIFO_SIZE	int	8 - 56	high	Depth of FIFO

7 SIGNAL TIMING

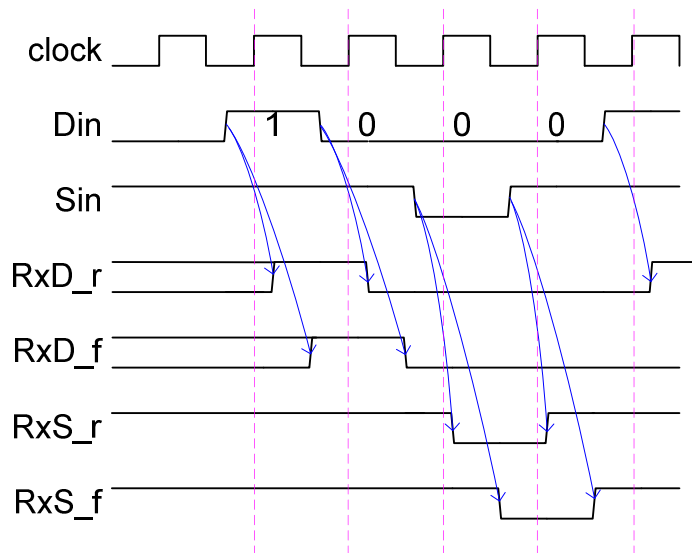
7.1 CLOCK AND RESET



7.2 SPACEWIRE

7.2.1 Received Data

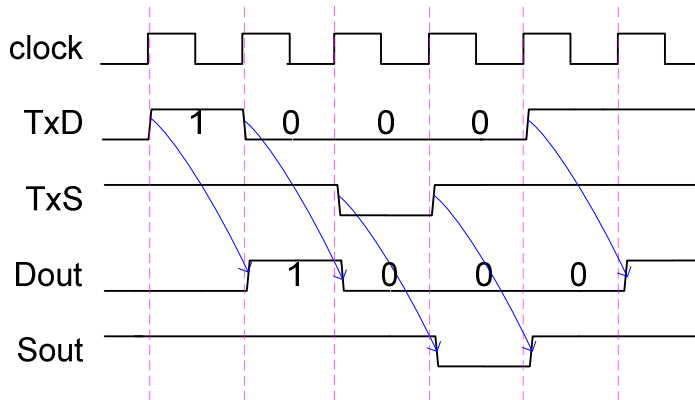
The following diagram shows the received data timing from the LVDS input pads through the DDR flip-flops and wrapper to the Codec.



The spw_rx_sync.vhd code aligns the rising and falling edge (_r/_f) signals to rising edge only outputs (clock), with a two clock cycle latency. It also reduces any metastability issues with the sampled signals.

7.2.2 Transmit Data

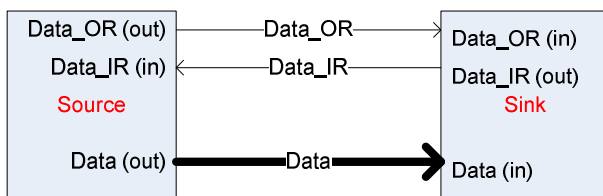
The following diagram shows the transmitted data timing from the Codec through the wrapper to drive the LVDS IO pads



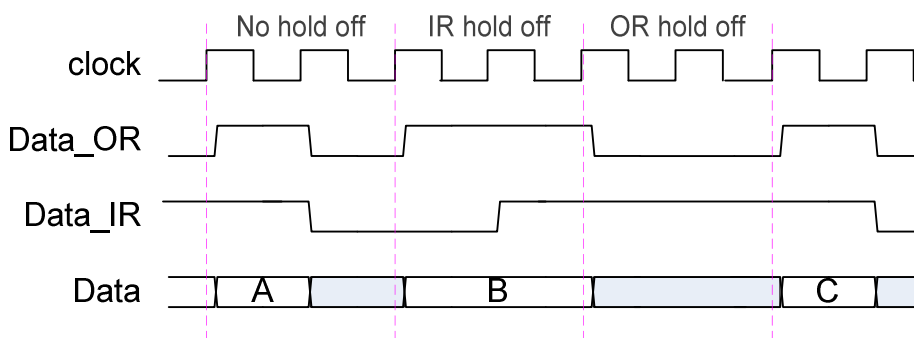
7.3 DATA FLOW CHANNEL

All IP implements the Data Flow Channels, two data flow channels in a bi-directional connection is called a Data Flow Link.

The data flow channel implements a simple handshake driven interface with an Output Ready (OR) and an Input Ready (IR) signal pair.



The OR signal always follows the data flow direction and the IR signal is always against the data flow direction.



Either the Source or Sink can hold off a transaction by keeping the respective OR or IR signals low. A data transfer is completed when both IR and OR are high on a rising clock edge. The Source will always drive OR low after a transfer, meaning that all transfers will be two cycles in length.

7.4 CONTROL AND STATUS

All control and status signals are output or sampled on the rising edge of clock.

8 INTEGRATION

The SpW Codec Wrapper is used to instantiate all the FPGA technology specific IP as well as the SpaceWire Codec.

In the example implementation the reset signal into the wrapper is synchronous with the top-level user logic above performing the synchronisation to the clock using a double buffering implementation to remove glitches and meta-stability that may occur from an asynchronous reset.

8.1 RESET SYNCHRONISATION

An example synchronisation circuit is shown below.

```
-- synchronise the reset to the DCM reset input
p_reset_gen: process( clk )
begin
    if (dcm_locked = '0')
    then
        rst_reg <= "111";
        rst <= '1';
    elsif rising_edge( clk )
    then
        rst_reg <= rst_reg(1 downto 0) & (not dcm_locked);
        rst <= rst_reg(2) or rst_reg(1);
    end if;
end process p_reset_gen;
```

8.2 CLOCK GENERATION

It is suggested that the clock and clock_b signals are derived from a clock source that can ensure low skew between them. Below is an example Spartan6 implementation using a DCM to derive the clocks.

```
clkbbuffer: IBUFGDS port map ( I => CLK_125MHz_p, IB => CLK_125MHz_n,
                                O => iclock );

clkdcn: DCM
generic map (
    CLKFX_MULTIPLY => CLOCK_MUL,
    CLKFX_DIVIDE   => CLOCK_DIV
)
port map (
    CLKIN      => iclock,
    CLKFB      => clk000,
    RST        => '0',
    DSEN       => '0',
    PSINCDEC   => '0',
    PSEN       => '0',
    PSCLK      => '0',
    CLKDV      => clkdv,
    CLK0       => clk000,
    CLK90      => open,
    CLK180     => open,
    CLK270     => open,
    CLK2X      => open,
    CLK2X180   => open,
    CLKFX      => clkfx,
    CLKFX180   => clkfx_b,
    LOCKED     => dcm_locked,
    STATUS     => open,
    PSDONE     => open
);
```

```
clk_buffer:  BUFG port map ( I => clkfx,  O => clk );
clk_b_buffer: BUFG port map ( I => clkfx_b, O => clk_b );
```

The parameters set the DCM output frequency based on the input crystal frequency.

$\text{clkfx} = (\text{CLKFX_MUL}/\text{CLKFX_DIV}) * \text{CRYSTAL_FREQUENCY}$

```
constant CRYSTAL_FREQUENCY : real      := 125.0e6; -- Crystal = 125MHz
constant CLOCK_MUL          : integer   := 8;      -- 125*4 = 500MHz
constant CLOCK_DIV          : integer   := 5;      -- 500/5 = 100MHz
```

8.3 IO PADS

Instantiating the correct IO Pads and flip-flops is key to performance and reliability of the interface. This requires both instantiation of the correct parts and configuring them for the best performance.

The constraints for the pads can be set via the HDL or in the UCF file.

8.3.1 Input

The parameters passed to the pad input buffers set the termination and IO type to be implemented. In this case LVDS with termination.

```
-- Use LVDS input buffers
u_din_buffer: IBUFDS generic map (DIFF_TERM => TRUE, IOSTANDARD => "LVDS")
    port map( I => Din_p, IB=> Din_n, O => din );
u_sin_buffer: IBUFDS generic map (DIFF_TERM => TRUE, IOSTANDARD => "LVDS")
    port map( I => Sin_p, IB=> Sin_n, O => sin );

-- Register inputs, sampling at DDR
u_din_iddr: iDDR generic map ( DDR_CLK_EDGE => "OPPOSITE_EDGE", INIT_Q1 => '0',
    INIT_Q2 => '0', SRTYPE => "ASYN")
    port map( D => din, Q1 => din_r, Q2 => din_f, C => clock, CE => '1',
        S => '0', R => '0' );
u_sin_iddr: iDDR generic map ( DDR_CLK_EDGE => "OPPOSITE_EDGE", INIT_Q1 => '0',
    INIT_Q2 => '0', SRTYPE => "ASYN")
    port map( D => sin, Q1 => sin_r, Q2 => sin_f, C => clock, CE => '1',
        S => '0', R => '0' );
```

8.3.2 Output

The parameters passed to the pad output buffers set the slew rate and IO type to be implemented. In this case LVDS with fast slew.

```
-- Use LVDS output buffers
u_dout_buffer: OBUFDS generic map (SLEW => "FAST", IOSTANDARD => "LVDS")
    port map( I => i_doutp, O => Dout_p, OB => Dout_n );
u_sout_buffer: OBUFDS generic map (SLEW => "FAST", IOSTANDARD => "LVDS")
    port map( I => i_soutp, O => Sout_p, OB => Sout_n );

-- Register outputs, SDR
u_dout_off: FDCE port map( D => dout,  Q => i_doutp, C => clock, CE => '1',
    CLR => reset );
u_sout_off: FDCE port map( D => sout,  Q => i_soutp, C => clock, CE => '1',
    CLR => reset );
```

8.3.3 UCF constraints

The following code snippet show the UCF constraints for the IO pads

```
#=====
# SpW port
```

NET Din_p	IOSTANDARD = LVDS		DIFF_TERM = "TRUE" ;
NET Din_n	IOSTANDARD = LVDS		DIFF_TERM = "TRUE" ;
NET Sin_p	IOSTANDARD = LVDS		DIFF_TERM = "TRUE" ;
NET Sin_n	IOSTANDARD = LVDS		DIFF_TERM = "TRUE" ;
NET Dout_p	IOSTANDARD = LVDS		DRIVE = 12 SLEW = FAST;
NET Dout_n	IOSTANDARD = LVDS		DRIVE = 12 SLEW = FAST;
NET Sout_p	IOSTANDARD = LVDS		DRIVE = 12 SLEW = FAST;
NET Sout_n	IOSTANDARD = LVDS		DRIVE = 12 SLEW = FAST;

8.4 TIMING CONSTRAINTS

8.4.1 Clock and Reset

The reset signal is synchronous to the clock and is an internal signal. It is expected that the wrapper will perform synchronisation to the clock and as such no timing requirements are present on the reset signal into the IP as the FPGA tools will budget the timing during synthesis and P&R.

The clock and clock_b signals should be derived from a clock generator within the FPGA or by low skew clock sources external. The clock constraints are then either derived from the external clock source and PLL/DLL structure or directly from the clock source.

Refer to the example PXIe_S6.ucf file for setting the clock timing parameter.

```
<4Links_root>\ip_41\spw_codec\phy\PXIe_S6\ise\PXIe_S6.ucf

# Clock timing constraints
NET "iclock" TNM_NET = iclock;
TIMESPEC TS_iclock = PERIOD "iclock" 8 ns HIGH 50%;
```

This is the only timing constraint required.

8.4.2 SpaceWire interface

The input signals have instantiated IO pad buffers and flip-flop's to minimise delay and skew. As such the timing is fixed by the Silicon. This minimises delay and skew between the two input signal pairs by design.

For output signals these are instantiated IO pad flip-flop's and pad buffers, as such the timing is fixed by the Silicon. This minimises delay and skew between the two output signal pairs by design.

8.4.3 Data Flow Channel

Since the Data Flow Channel is synchronous to the clock and is designed to connect to other synchronous signals the clock constraint will define the timing budget.

8.4.4 Control and Status

Since the Control and Status signals are synchronous to the clock and are designed to connect to other synchronous signals the clock constraint will define the timing budget.

9 TEST BENCH

9.1 STIMULUS & LOG FILE FORMAT

The Text files used for the stimulus and log files are the same format. This allows you do direct comparisons between them. They are a nine bit word with the MSB on the right and LSB on the left.

```
876543210 bit number
000000000
000000001
000000010
000000011
000000100
000000101
000000110
000000111
000001000
000001001
000001010
000001011
```

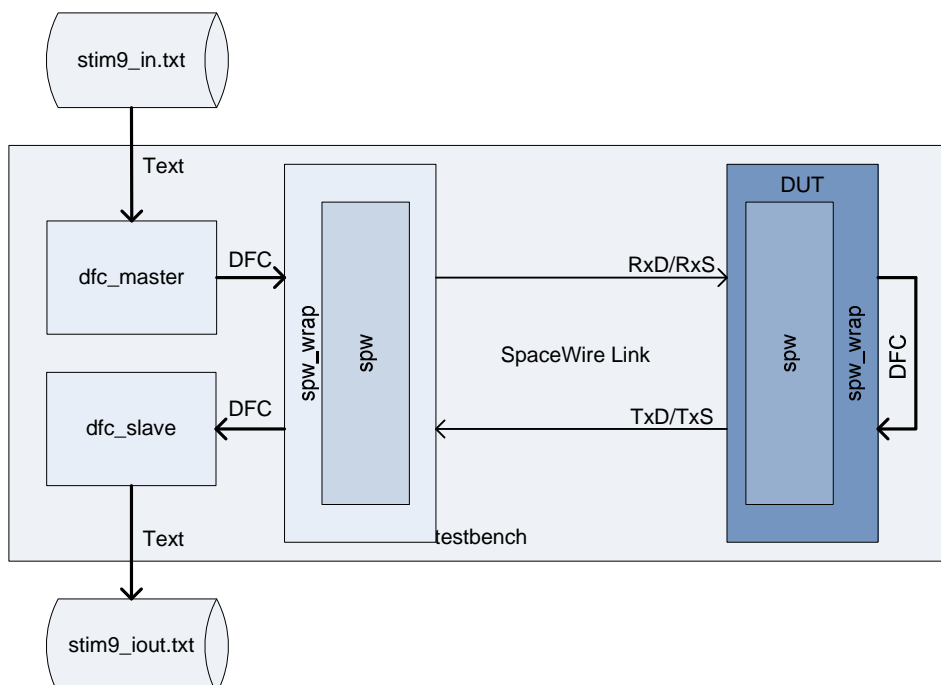
The dfc_master reads the file (stim9_in.txt) line by line and outputs the data over the Data Flow Channel. The flow control implementation of the interface means the data cannot be lost or pushed into the simulation; it is pulled by the device connected to the dfc_master. The timing of the dfc interface is defined in 7.3. The dfc_master is unable to implement any complex functions such as delays or decisions and can only output sequentially the data.

The dfc_slave writes the data line by line into the log file (stim9_out.txt). It will always accept data and write it to the log file in the 2 cycle behaviour defined in 7.3.

9.2 OUT OF BOX

The Out-of-Box testbench implements a simple design to demonstrate how the SpaceWire Codec works and the testbench IP (dfc_master and dfc_slave). This can be used as the basis for more complex test-benches to be developed.

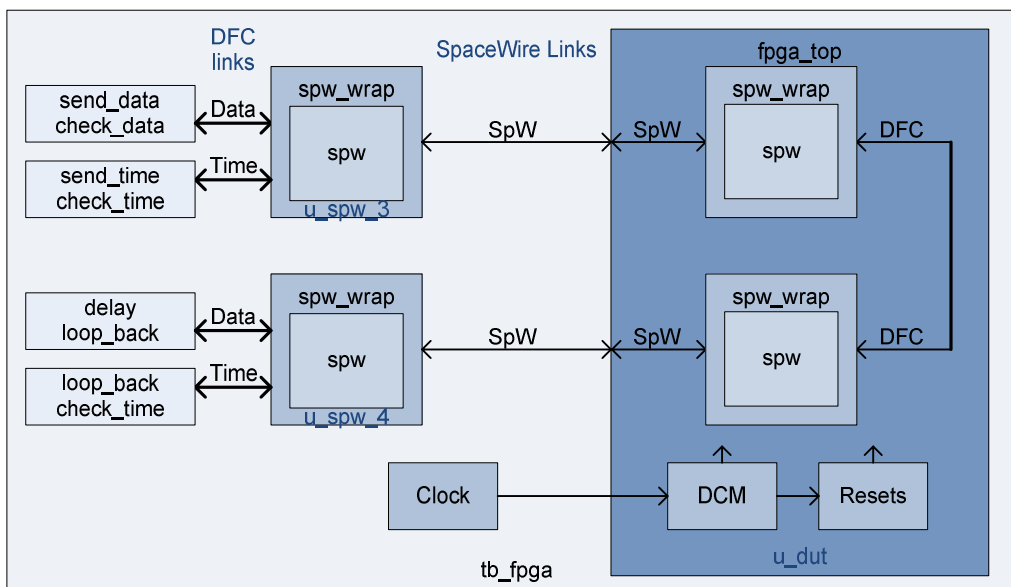
The diagram below shows the top-level structure of the test-bench.



9.3 INTEGRATION

The integration test-bench is developed to model the PXIe-S6 board developed by 4Links. It enables a simple simulation of the full FPGA design, including DCM and IO Pad components.

The test-bench implements a simple pass-through design in `fpga_top`, with data and time stamps passed in both directions between the two SpaceWire ports.



The test-bench sends the data as an incrementing number from `u_spw_3`; the sequence and delays can be seen through the system. Backpressure is added on the dfc interface of `u_spw_4` (delay and loop_back) by holding off dfc transactions, this shows the handshaking protocol works and packets are not lost as buffers fill in the design.

The timestamp is generated early in the simulation, just after training and passes a single timestamp from `u_spw_3` through `u_dut` to `u_spw_4` where it is returned to `u_spw_3` via the `u_dut`. A check is performed to show the timestamp has completed the round trip successfully.