# 4Links SpaceWire CoDec IP Core: Product User Guide (Xilinx)

Contents

[4Links SpaceWire CoDec IP Core: Product User Guide (Xilinx) 1](#_Toc139527696)

[Table of Figures 2](#_Toc139527697)

[Overview 3](#_Toc139527698)

[Core Architecture 3](#_Toc139527699)

[Core Configuration 4](#_Toc139527700)

[I/O Ports 4](#_Toc139527701)

[Clock & Reset Ports 4](#_Toc139527702)

[Data Ports 4](#_Toc139527703)

[Rx Info Ports 5](#_Toc139527704)

[Time Code Ports 5](#_Toc139527705)

[Control Ports 5](#_Toc139527706)

[SpW Bypass IO Ports (custom mode only) 5](#_Toc139527707)

[SpW IO Ports (diff & single modes only) 5](#_Toc139527708)

[Using the Core 6](#_Toc139527709)

[Instantiating the Core 6](#_Toc139527710)

[Selecting IO Mode 6](#_Toc139527711)

[Modifying the Core 6](#_Toc139527712)

[Connecting the Core 7](#_Toc139527713)

[Clock & Reset Signals 7](#_Toc139527714)

[Core Bring-Up 8](#_Toc139527715)

[Data Channels 8](#_Toc139527716)

[Time Code Channels 9](#_Toc139527717)

[Sending Control Characters 9](#_Toc139527718)

[IP Example Design (Xilinx Vivado 2023.1) 9](#_Toc139527719)

[Testbench I/O & Signals 10](#_Toc139527720)

[Clock Generation 13](#_Toc139527721)

[Stimulus Generation 14](#_Toc139527722)

# Table of Figures

[Figure 1: Top-Level Core Architecture (spw.vhd). 3](#_Toc139527682)

[Figure 3: IP Core Start-up Synchronization 7](#_Toc139527683)

[Figure 4: Core Clock(s) Example Waveform 7](#_Toc139527684)

[Figure 5: Tx\_data **IR**/**OR** Handshake Example 8](#_Toc139527685)

[Figure 6: Rx\_data **IR/OR** handshake Example 8](#_Toc139527686)

[Figure 7:Tx\_Time IR/OR Handshake 9](#_Toc139527687)

[Figure 8: Rx\_Time IR/OR Handshake 9](#_Toc139527688)

[Figure 9: Send EOP Waveform Example 9](#_Toc139527689)

[Figure 10: I/O Signal Declarations for IP Example Design 10](#_Toc139527690)

[Figure 11: SpW\_ip\_tb signal list 13](#_Toc139527691)

[Figure 12: Testbench Clock Generation Procedures 13](#_Toc139527692)

[Figure 13: Testbench Configuration constants 13](#_Toc139527693)

[Figure 14: Testbench Clock initialization 13](#_Toc139527694)

[Figure 15: Testbench Basic Architecture 14](#_Toc139527695)

# Overview

This document serves as a product user guide for the 4Links SpaceWire (SpW) CoDec IP Core. In this document you will find information to help you integrate the 4Links SpW CoDec into your designs. A walkthrough for the IP Example design is included at the end of the document. The example design is used as a demonstration platform for the core. The example design can be modified as required. A powerful set of simulation procedures are included in the example design package ***SpW\_Sim\_lib.vhd***. These can be used to debug and verify your SpaceWire designs before implementation.

The target HDL for the SpW CoDec and testbench environment is VHDL.

# Core Architecture

The 4Links SpW CoDec IP is comprised of several VHDL entities. The architecture can be split into three distinct sections; Transmit, Receive and Control. A top-level diagram of the core is included in Figure 1. For instantiating the IP,

A picture containing text, diagram, screenshot, font

Description automatically generated

Figure 1: Top-Level Core Architecture (spw.vhd).

# Core Configuration

The top-level design ***spw\_wrap.vhd*** is configured using VHDL generics. There are three generics to be used.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **NAME** | **TYPE** | **RANGE(LOW)** | **RANGE (HIGH)** | **UNITS** |
| g\_clock\_frequency | REAL | 2\_000\_000 | N/A | Hz |
| g\_rx\_fifo\_size | INTEGER | 9 | 56 | N/A |
| g\_tx\_fifo\_size | INTEGER | 9 | 56 | N/A |
| g\_mode | STRING | -- | -- | -- |

1. **g\_clock\_frequency**

The clock frequency of the clock input to the IP Core (Fixed value) in Hz. For example, if using a 125MHz input clock, the value would be *125\_000\_000.0.* As this value is of type REAL, it must contain a decimal point to be valid. Otherwise, your toolchain may throw errors.

1. **g\_rx\_fifo\_size**

Depth of the RX FIFO used to store received SpW data. Recommended value is 16

1. **g\_tx\_fifo\_size**

Depth of TX FIFO used to send SpW data. Recommended value is 16.

1. **g\_mode**

core IO mode, choice between “diff”, “single” and “custom”. For differential, single-ended and custom IO signalling parameters.

# I/O Ports

This section covers the numerous I/O ports on the 4Links SpW CoDec. Details on the function of each I/O can be found in the relevant section below.

## Clock & Reset Ports

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **STATUS** | **BRIEF** |
| clock | std\_logic | rising\_edge | Positive clock source |
| clock\_b | std\_logic | rising\_edge | Negative clock source (clock θ \* -180°) |
| reset | std\_logic | active high (‘1’) | IP reset input (asynchronous) |

## Data Ports

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **STATUS** | **BRIEF** |
| TX\_data | std\_logic\_vector (8 downto 0) | Output data | Output data from SpW |
| Tx\_OR | boolean | active high (‘1’) | Assert to load data |
| Tx\_IR | boolean | active high (‘1’) | Asserted when ready for Tx data |
| RX\_data | std\_logic\_vector (8 downto 0) | Input data | Input data to SpW |
| Rx\_OR | boolean | active high (‘1’) | Asserted when valid Rx data |
| Rx\_IR | boolean | active high (‘1’) | Assert to read Rx data |

## Rx Info Ports

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **STATUS** | **BRIEF** |
| RX\_ESC\_ESC | boolean | active high (true) | ESC-ESC Detected | |
| RX\_ESC\_EOP | boolean | active high (true) | ESC-EOP Detected | |
| RX\_ESC\_EEP | boolean | active high (true) | ESC-EEP Detected | |
| RX\_Parity\_error | boolean | active high (true) | Parity Error Detected | |
| RX\_bits | integer (0 to 2) | N/A | Bits read on last clock | |
| RX\_rate | std\_logic\_vector (15 downto 0) | Output Info | Rx Channel bitrate | |

## Time Code Ports

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **STATUS** | **BRIEF** |
| Rx\_Time | std\_logic\_vector(7 downto 0) | Output Timecode | Received TC data |
| Rx\_Time\_OR | boolean | active high (true) | High when received TC data |
| Rx\_Time\_IR | boolean | active high (true) | Assert to read TC data |
| Tx\_Time | std\_logic\_vector(7 downto 0) | Input Timecode | TC data to send |
| Tx\_Time\_OR | boolean | active high (true) | Assert to load TC data |
| Tx\_Time\_IR | boolean | active high (true) | High when ready for TC data |

## Control Ports

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **STATUS** | **BRIEF** |
| Disable | boolean | active high (true) | Assert to disable SpW channel |
| Legacy | boolean | active high (true) | Assert to use legacy spec. |
| Connected | boolean | active high (true) | High when SpW connected |
| Error\_select | std\_logic\_vector(3 downto 0) | Select error | Select error to inject on Rx |
| Error\_inject | boolean | active high (true) | Inject error code on SpW Rx |
| Out\_Stalled | boolean | active high (true) | High when SpW has stalled |
| Tx\_PSC | std\_logic\_vector(7 downto 0) |  | Tx-side line rate prescalar |
| Tx\_PSC\_valid | Std\_logic | Active high |  |
| Tx\_PSC\_ready | Std\_logic | Active high | Asserted when valid is high |

## SpW Bypass IO Ports (custom mode only)

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **STATUS** | **BRIEF** |
| DDR\_din\_r | std\_logic | Rx Data (clock) | Sampled on rising edge of clock |
| DDR\_din\_f | std\_logic | Rx Data (clock\_b) | Sampled on rising edge of clock\_b |
| DDR\_sin\_r | std\_logic | Rx Strobe (clock) | Sampled on rising edge of clock |
| DDR\_sin\_f | std\_logic | Rx Strobe (clock\_b) | Sampled on rising edge of clock\_b |
| SDR\_Dout | std\_logic | Tx Data | SpW Tx Data Output (SDR) |
| SDR\_Sout | std\_logic | Tx Strobe | SpW Tx Strobe Output (SDR) |

## SpW IO Ports (diff & single modes only)

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **STATUS** | **BRIEF** |
| Din\_p | std\_logic | LVDS Din Positive | Used as Single Ended Input when “single” |
| Din\_n | std\_logic | LVDS Din Negative |  |
| Sin\_p | std\_logic | LVDS Sin Positive | Used as Single Ended Input when “single” |
| Sin\_n | std\_logic | LVDS Sin Negative |  |
| Dout\_p | std\_logic | LVDS Dout Positive | Used as Single Ended output when “single” |
| Dout\_n | std\_logic | LVDS Dout Negative |  |
| Sout\_p | std\_logic | LVDS Sout Positive | Used as Single Ended output when “single” |
| Sout\_n | std\_logic | LVDS Sout Negative |  |

# Using the Core

## Instantiating the Core

### Selecting IO Mode

The IP core has three different IO modes, “diff”, “single” and “custom”. These are configured through the ***g\_mode*** generic parameter.

* Differential (LVDS) Operation “diff”.

This is the default operating mode of the core. This mode requires that the “spw\_wrap\_top\_level” entity be edited to include device primitives for DDR registers and LVDS input buffers.

* Single Ended Operation “single”.

Single ended mode removes the need to use differential IO. All IO transactions on Data & Strobe IO will use the positive (\_p) IO only. The wrapper should be modified to include device primitives for DDR registers on D/S Inputs. Outputs are SDR. This is useful if using IO-limited devices with no support for LVDS IO buffers.

* Custom IO Operation “custom”.

Custom mode bypasses the normal Din/Sin and Dout/Sout IO. Instead, this exposes the cores internal DDR register inputs and SDR outputs. These can then be connected directly to DDR registers, without the need to modify the “spw\_wrap\_top\_level” entity. This mode is useful when designing with block GUIs or creating one-off designs on new target devices.

For small designs, the “custom” option me be preferred. However, modifying the core wrapper for your target device is encouraged. This will greatly streamline future developments with the 4Links SpaceWire IP Core. Note that for verification, the differential and single-ended modes are modelled using processes in the “spw\_wrap\_top\_level”.

Whilst these processes accurately describe the behaviour of DDR registers and differential IO buffers, there is no guarantee that your toolchain will be able to match these processes to your target device resources. It is for this reason we insist that device primitive instantiation is used, as outlined above.

For an example of DDR registers/LVDS buffers primitive instantiation, for a Xilinx Kintex US+ FPGA, please see “spw\_wrap\_top\_level\_xilinx.vhd”.

## Modifying the Core

To modify the core with your own primitives, see the relevant “generate” section of the core. For the default top-level wrapper file, these sections are:

* “diff”: lines 267 to 317.
* “single”: lines 318 to 359.
* “custom” 362 to 371.

Use the “spw\_wrap\_top\_level\_xilinx” entity as an example of adding primitives to the core generation sections. Those using Kintex UltraScale+ FPGAs may use the “\_xilinx” wrapper file as is.

## Connecting the Core

When using the IP Core, you must make sure that both Rx and Tx channels are connected to the target SpaceWire device. Connecting only one channel will cause the SpaceWire Link to timeout.

If the SpaceWire Link is lost, the core will automatically attempt re-connection to the SpaceWire target.

To establish a SpaceWire link, the IP core will first enter a synchronization state. Here NULL characters are sent at a low frequency (~2Mb). The end of this synchronization period is marked by the transmission of successive FCTs. Once synchronization has been performed, the IP core will begin transmitting data at the full specified rate, as defined by the clock frequency generic.

A screenshot of a video game

Description automatically generated with medium confidence

Figure 3: IP Core Start-up Synchronization

Should synchronization fail, the core will periodically attempt to re-connect to the target device until stopped by the user application.

## Clock & Reset Signals

The core requires two clock signals and a single reset.

The clock signals should be generated using a PLL/MMCM/DCM (depending on required frequency and technology). ***Clock*** and ***clock\_b*** should be configured so that ***clock­\_b*** is 180° out of phase with ***clock***. Both clocks should otherwise be identical. See Figure 7 for an example.

A screenshot of a computer screen

Description automatically generated with low confidence

Figure 4: Core Clock(s) Example Waveform

The ***reset*** signal is asynchronous to either ***clock*** or ***clock\_b***.

Note that ***clock\_b*** is only used for sampling and sending data via DDR registers. All other IP functions are aligned to the rising edge of ***clock***. Therefore, when operating the core, all I/O signals should be timed with respect to the **rising edge** of ***clock***.

## Core Bring-Up

To safely bring up the core, it is recommended to assert the ***reset*** input ***high*** for several clock cycles. This will ensure that any state-machine registers are in a safe state. If using a PLL/MMCM/DCM. Keep reset high for several clock cycles ***after*** the core has locked the output frequency for ***clock*** and ***clock\_b***. Again, this ensures that the core starts safely. A guideline of ***4*** clock cycles is advised.

Once ready, de-assert the ***reset*** signal and wait for ***Connected*** to go ***true***. When ***connected*** is ***true*** the core is ready to send and receive data over SpW. If ***connected*** remains **false***,* check the **RxInfo** ports for debug information.

## Data Channels

The Data channels use an ***Input\_Ready (IR), Output\_Ready (OR)*** handshake to read/write data. This is analogous to the AXI-handshake process between Master & Slave devices. A data transaction is only valid when both ***IR*** and ***OR*** ports are ***true*** on the rising edge of ***clock***. Figure 8 shows the ***IR/OR*** handshake for the ***Tx\_data*** input on the core.

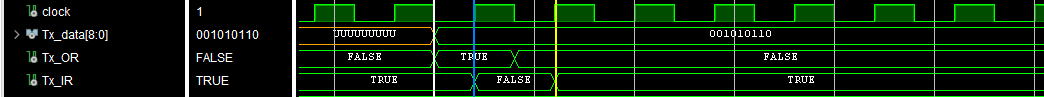


Figure 5: Tx\_data **IR**/**OR** Handshake Example

***Tx\_IR*** is controlled by the IP core and ***Tx\_OR*** is asserted by user logic. To load data into the Tx FiFo, connecting logic should:

* Wait for ***Tx\_IR*** to be ***true***.
* Assert ***Tx\_data*** and ***Tx\_OR***.
* De-assert ***Tx\_OR*** when ***Tx\_IR*** is ***false***.

The example in Figure 8 is from a testbench, where ***Tx\_OR*** changes on the falling\_edge of ***clock.*** This is not required for operation; it is simply a characteristic of simulating a delta-cycle in a testbench environment. In operation, it is expected that ***Tx\_OR*** will be aligned to the rising edge of ***clock***.

Figure 9 shows the waveform of the ***IR/OR*** handshake for reading ***Rx\_data*** from the core.

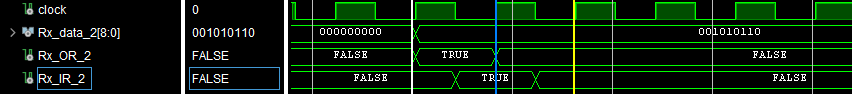


Figure 6: Rx\_data **IR/OR** handshake Example

Note that for the cores **Rx channels**, ***OR*** is set by the core, and ***IR*** is controlled by user logic. Unlike the **Tx channels** on the core, where the opposite is true.

The data channels are 9 bits wide, where the MSB (8) is the control/data select bit and bits (7 downto 0) are the payload data bits.

## Time Code Channels

The Time Code (TC) channels follow a similar pattern to the Data Channels on the core. Note that the Data channels are 9bits wide, whereas the TC channels are 8bits wide.

A screenshot of a video game

Description automatically generated with medium confidence

Figure 7:Tx\_Time IR/OR Handshake

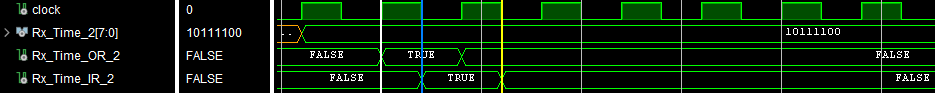


Figure 8: Rx\_Time IR/OR Handshake

## Sending Control Characters

Control characters, such as FCT, EOP, EEP, and ESC can be inserted on the SpW Tx channel using the ***Tx\_data***port. Here the MSB of the port (8) represents the data selection bit used in a SpW packet. When the MSB is set to ‘1’, the data input is automatically configured to read the 2 LSB on the port. As to conform with the SpaceWire standard on control characters. Parity is automatically calculated and sent by the IP.

A screenshot of a video game

Description automatically generated with medium confidence

Figure 9: Send EOP Waveform Example

EEPs and EOPs will only be pushed to the IP Cores Rx\_data interface if some data has been transmitted beforehand. If an EEP is sent directly after an EOP, with no data sent in between, then the core will ignore the EEP, and it will not be pushed to the Rx interface. The same is true if EEP precedes the EOP.

# IP Example Design (Xilinx Vivado 2023.1)

In the IP example design, two “spw\_wrap\_top\_level” entities are connected. One core is the “Tx” side, and the other is the “Rx” side. The signal list for the Rx side has “\_2” appended to unique signals.

## Testbench Core Configuration

The cores can be configured to operate in Single-Ended or Differential IO mode. By default, the cores will operate in Differential IO mode. To change IO modes, use the constant “c\_mode” on line 54 of the testbench. Valid options are “diff”, “single” and “custom”, as defined in section *Selecting IO Mode.* Custom mode cannot be used without first modifying the simulation.

The single “c\_mode” constant is passed to both Instantiated cores. Updating this single constant will update the IO mode of both cores automatically.

## Testbench I/O & Signals

The two ***spw\_wrap*** entities instantiated in the testbench are names ***SpW\_DUT\_tx*** and ***SpW\_DUT\_rx***. I/O signals from the ***SpW\_DUT\_rx***entity use a ***\_2*** suffix, whilst those from the ***SpW\_DUT\_tx*** entity have no suffix.

A screen shot of a computer

Description automatically generated with medium confidence

Figure 10: I/O Signal Declarations for IP Example Design

Ports which are shared, such as ***clock, clock\_b, reset*** and the ***SpW\_Tx/Rx*** channels use no suffix as only one signal declaration is required Figure 14 contains a full description of the testbench signals used in the example design.

|  |  |  |
| --- | --- | --- |
| **NAME** | **TYPE** | **DESCRIPTION** |
| **Global IO Signals** |  |  |
| clock | std\_logic | positive clock input |
| clock\_b | std\_logic | negative clock input |
| reset | std\_logic | reset input (active high) |
| **SpW\_DUT\_tx IO Signals** |  |  |
|  |
| Tx\_data | std\_logic\_vector(8 downto 0) | Data to Transmit over SpW Tx Channel |  |
| Tx\_OR | boolean | Output Ready Signal |  |
| Tx\_IR | boolean | Input Ready Signal |  |
|  |  |  |  |
| Rx\_data | std\_logic\_vector(8 downto 0) | Data Receiver over SpW Rx Channel |  |
| Rx\_OR | boolean | Output Ready Signal |  |
| Rx\_IR | boolean | Input Ready Signal |  |
|  |  |  |  |
| Rx\_ESC\_ESC | boolean | Asserted when ESC\_ESC detected on Rx |  |
| Rx\_ESC\_EOP | boolean | Asserted when ESC\_EOP detected on Rx |  |
| Rx\_ESC\_EEP | boolean | Asserted when ESC\_EEP detected on Rx |  |
| Rx\_Parity\_error | boolean | Asserted when Parity\_error detected on Rx |  |
| Rx\_bits | integer 0 to 2 | Number of valid SpW Rx bits on sample cycle |  |
| Rx\_rate | std\_logic\_vector(15 downto 0) | Bitrate information for SpW Rx channel |  |
|  |  |  |  |
| Rx\_Time | std\_logic\_vector(7 downto 0) | Timecode Received over SpW Rx channel |  |
| Rx\_Time\_OR | boolean | Output Ready Signal |  |
| Rx\_Time\_IR | boolean | Input Ready Signal |  |
|  |  |  |  |
| Tx\_Time | std\_logic\_vector(7 downto 0) | Timecode to send over SpW Tx Channel |  |
| Tx\_Time\_OR | boolean | Output Ready Signal |  |
| Tx\_Time\_IR | boolean | Input Ready Signal |  |
|  |  |  |  |
| Disable | boolean | Assert to disable SpW Rx/Tx channel |  |
| Connected | boolean | Asserted when SpW UpLink is active |  |
| Error\_select | boolean | Error code to select |  |
| Error\_inject | std\_logic\_vector(3 downto 0) | Assert to inject slected Error code |  |
| **SpW\_DUT\_rx IO signals** |  |  |  |
|  |
| Tx\_data\_2 | std\_logic\_vector(8 downto 0) | Data to Transmit over SpW Tx Channel |  |
| Tx\_OR\_2 | boolean | Output Ready Signal |  |
| Tx\_IR\_2 | boolean | Input Ready Signal |  |
|  |  |  |  |
| Rx\_data\_2 | std\_logic\_vector(8 downto 0) | Data Receiver over SpW Rx Channel |  |
| Rx\_OR\_2 | boolean | Output Ready Signal |  |
| Rx\_IR\_2 | boolean | Input Ready Signal |  |
|  |  |  |  |
| Rx\_ESC\_ESC\_2 | boolean | Asserted when ESC\_ESC detected on Rx |  |
| Rx\_ESC\_EOP\_2 | boolean | Asserted when ESC\_EOP detected on Rx |  |
| Rx\_ESC\_EEP\_2 | boolean | Asserted when ESC\_EEP detected on Rx |  |
| Rx\_Parity\_error\_2 | boolean | Asserted when Parity\_error detected on Rx |  |
| Rx\_bits\_2 | integer 0 to 2 | Number of valid SpW Rx bits on sample cycle |  |
| Rx\_rate\_2 | std\_logic\_vector(15 downto 0) | Bitrate information for SpW Rx channel |  |
|  |  |  |  |
| Rx\_Time\_2 | std\_logic\_vector(7 downto 0) | Timecode Received over SpW Rx channel |  |
| Rx\_Time\_OR\_2 | boolean | Output Ready Signal |  |
| Rx\_Time\_IR\_2 | boolean | Input Ready Signal |  |
|  |  |  |  |
| Tx\_Time\_2 | std\_logic\_vector(7 downto 0) | Timecode to send over SpW Tx Channel |  |
| Tx\_Time\_OR\_2 | boolean | Output Ready Signal |  |
| Tx\_Time\_IR\_2 | boolean | Input Ready Signal |  |
|  |  |  |  |
| Disable\_2 | boolean | Assert to disable SpW Rx/Tx channel |  |
| Connected\_2 | boolean | Asserted when SpW UpLink is active |  |
| Error\_select\_2 | boolean | Error code to select |  |
| Error\_inject\_2 | std\_logic\_vector(3 downto 0) | Assert to inject slected Error code |  |
| **SpW Tx/Rx Channel LVDS Signals** |  |  |  |
|  |
| Din\_p | std\_logic | SpW Rx\_Channel\_DATA (positive) |  |
| Din\_n | std\_logic | SpW Rx\_Channel\_DATA (negative) |  |
| Sin\_p | std\_logic | SpW Rx\_Channel\_STROBE (positive) |  |
| Sin\_n | std\_logic | SpW Rx\_Channel\_STROBE (negative) |  |
| Dout\_p | std\_logic | SpW Tx\_Channel\_DATA (positive) |  |
| Dout\_n | std\_logic | SpW Tx\_Channel\_DATA (negative) |  |
| Sout\_p | std\_logic | SpW Tx\_Channel\_STROBE (positive) |  |
| Sout\_n | std\_logic | SpW Tx\_Channel\_STROBE (negative) |  |
| **SpW Tx Channel Debug signals** |  |  |  |
|  |
| spw\_debug\_tx | std\_logic\_vector(8 downto 0) | SpW Tx Debug output data |  |
| spw\_debug\_raw | std\_logic\_vector(13 downto 0) | SpW Tx Debug raw output data |  |
| spw\_debug\_parity | std\_logic | SpW Tx debug parity bit |  |
| spw\_debug\_cmd | string | SpW Tx debug command |  |
| spw\_debug\_time | std\_logic\_vector(7 downto 0) | SpW Tx debug timecode output |  |
| **SpW Tx/Rx Channel Recovered Data Clock Signals** |  |  |  |
|  |
| Tx\_Rec\_Clock | std\_logic | SpW Tx Recovered Data Clock |  |
| Rx\_Rec\_Clock | std\_logic | SpW Rx Recovered Data Clock |  |

Figure 11: SpW\_ip\_tb signal list

## Clock Generation

The example design uses two procedures to generate ***clock*** and ***clock\_b***.

A screen shot of a computer program

Description automatically generated with low confidence

Figure 12: Testbench Clock Generation Procedures

The procedures work by inverting the clock signal after half of the specified clock period. The clock period is calculated automatically from the ***CLOCK\_FREQUENCY*** constant used in the testbench.

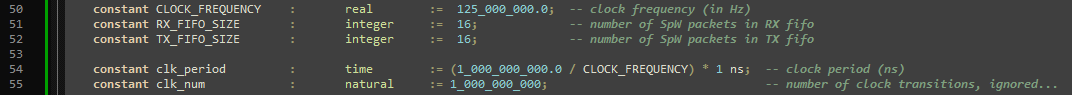


Figure 13: Testbench Configuration constants

For the correct clocking behaviour, the clock signals are initialized using opposite values, as seen in Figure 17.



Figure 14: Testbench Clock initialization

## Stimulus Generation

A diagram of a computer

Description automatically generated with low confidence

Figure 15: Testbench Basic Architecture

The ***stim\_gen*** process is used to provide I/O stimulus for the two SpW CoDecs. Clock generation is accomplished using two external processes, one for each clock signal. For clarity Figure 18 does not show detailed clock connections between the IP and testbench.

The default test regime performed by ***stim\_gen*** is as follows:

* Apply reset for ~67 us.
* Check valid SpW uplink was achieved
* Send SpW Tx data “001010110” from SpW\_DUT\_tx.
* Read SpW Rx data on SpW\_DUT\_rx.
* Send SpW Tx Timecode “1011100” from SpW\_DUT\_tx.
* Read SpW Rx Timecode on SpW\_DUT\_rx.
* Send SpW EOP character from SpW\_DUT\_tx.
* Read SpW EOP character on SpW\_DUT\_rx.
* Send SpW Tx data “001011110” from SpW\_DUT\_tx.
* Read SpW Rx data from SpW\_DUT\_rx.
* Send SpW EEP from SpW\_DUT\_tx.
* Read SpW EEP from SpW\_DUT\_rx.
* Send 8 Bytes of SpW Tx data from SpW\_DUT\_tx.
* Read 8 bytes of SpW Rx data from SpW\_DUT\_rx.
* Check the 8 bytes received and report errors.
* Assert Disconnect on Rx Core.
* De-assert Disconnect on Rx Core, Wait for Uplink to re-establish.

Once these procedures have completed, the process will report the stimulus has finished and the current simulation time. The report uses a severity of failure to stop the simulation immediately. The TCL console can be used to monitor the simulation status.