4Links SpaceWire CoDec IP: Hello\_World FPGA Example Design (Xilinx)

# Overview

This document is a guide to the 4Links SpaceWire CoDeC IP Hello\_World example design. The example design outlined in this document is for use with the Xilinx Vivado FPGA toolchain (2023.1). Use of newer/older version of Vivado is not recommended, however may still be supported.

For guidance on using the example design with other vendors toolchains, please see the relevant directory in this documents folder structure. Enquiries should be sent to [support@4links.space](mailto:support@4links.space).

# Building the Example Design

Make sure that Vivado 2023.1 toolchain is installed on the system. For Linux users, make sure that the intended root directory is currently targeted. This guide is aimed at users on Windows 10/11 however similar steps can be used to build the project in Linux (Ubuntu 20+).

Move the example design folders to the intended project location.

Open Vivado.

When the Vivado GUI appears, go to Tools -> Run TCL.

A dialog box will appear, select the project TCL file to run.

Vivado should now begin building the project, this make take a few minutes.

Make sure to check the TCL Console window for any errors that may have occurred during building.

If successful, you should see the following project window.

Make sure to select the follow items as Top-Level under Design and Simulation sources, respectively.

# Design Architecture

## Top Level Diagram

A diagram of a computer

Description automatically generated with low confidence

Figure : Example Top-Level Architecture of SpW\_hello\_world\_logic.vhd

## SpaceWire Data Controller

The data controller is responsible for managing the Tx\_data and Rx\_data signal groups on the 4Links SpaceWire IP. Separate processes are used to control each data channel. The Tx\_data channel employs a full state-machine, whereas the Rx\_data channel uses a simple handshake to push data onto the relevant controller interface.

A screenshot of a computer program

Description automatically generated with low confidence

Figure : Elaborated Data Controller

The controller reads the contents of a RAM where the ASCII character values for “HELLO\_WORLD” are stored. Each character is read individually with the data presented on the *spw\_Tx\_data* interface. Once the full text has been read (11 characters), an EOP is generated and sent on the *spw\_Tx\_data* interface with the *spw\_Tx\_Con* bit asserted. This behaviour is performed in a looping state machine which will execute indefinitely until *rst\_in* is applied.

For receiving data, a synchronous process is used to perform the Rx\_data handshake and analyse the received data. If the *spw\_Rx\_data* channel contains a control character then the received data is pushed to the *rx\_cmd* interface, else it is pushed to the *rx\_data* interface.Both interfaces use a ready/valid handshake for reading data. Either *rx\_cmd\_ready* or *rx\_data\_ready* must be asserted for the SpW Rx\_data handshake to occur. The *rx\_cmd\_valid* and *rx\_data\_valid* signals are only de-asserted once the respective ready/valid handshake has occurred.

## SpaceWire Tx Data RAM

A single-port RAM is used to store the ascii character data. The RAM is configured on instantiation using generics. The RAM follows instantiation guidelines as laid out by Xilinx. Functions inside the RAMs architecture declaration are used to convert VHDL strings into ASCII character values which are stored in the memory elements. These functions can be used and adapted as required.

An example of changing the ASCII value stored in memory is included later in this guide. This may help to debug custom designs if required.

# Behavioural Simulation

Before running the behavioural simulation, make sure that the correct testbench is set as the top-level simulation source. This should be *spw\_hw\_tb(bench)*. Right click on the simulation source and click “Set as Top” if required. The option will be greyed out (Figure 3) if the source is already set to top level.

A screenshot of a computer

Description automatically generated

Figure 3: Set Simulation Source as Top Level

## Behavioural Testbench

The behavioural testbench is comprised of several processes used to stimulate and debug the behaviour of the example design. The architecture of the behavioural testbench can be seen in Figure 3 below.

A screenshot of a computer

Description automatically generated with medium confidence

Figure : Behavioural Testbench Architecture

The testbench instantiates two entities of the example design. These are named *u\_spw\_hw\_tx* and *u\_spw\_hw\_rx*. The suffix, tx/rx is assigned based on the connection of the debug processes. Both entities perform the same task, so for the sake of simplicity, we are assigning one as our transmitter and one as our receiver. The dark-grey blocks above are our monitoring processes used to debug the simulation and provide stimulus.

On start-up, *rst\_in* is asserted until the mmcm in both designs has successfully locked output frequency. The simulation will automatically stop once an EOP is detected on the *rx\_rx\_command\_out* signal.

## Waveform Output

The waveform output should show that the characters for “HELLO\_WORLD” are loaded into a data buffer *rx\_rx\_ascii\_buf*. If, for some reason, the signals do not correctly display in the simulator window. Go to the scope window, right click the top-level entity and select “add wave to window”.

A screenshot of a computer

Description automatically generated

Figure 5: add entity waveform to simulation window

Once new waves are added, you may need to re-simulate the design to update the waveform GUI. When doing this, make to save the waveform changes when prompted. You can expand the simulation entities in scope and add lower-level signals if desired. Again, adding these will require the design to be re-simulated for the waveform GUI to update.

To change the radii of simulation waveforms, right click the desired signal, select “Radix” and select the desired format. By selecting ASCII, we can verify that the text “HELLO\_WORLD” was successfully sent across the SpaceWire link.

A screenshot of a computer

Description automatically generated

Figure 6: Change data Radix to ASCII in simulation

## Editing the ASCII Transmission

The transmitted ASCII string can be customized by editing the constant(s) found on lines 92 & 93 on the behavioural testbench. The testbench uses signal attributes to automatically configure to a new string length if required.

A picture containing text, multimedia software, software, font

Description automatically generated

Figure 7: ASCII String Constants in Testbench

For example, changing line 92, in Figure 7, to “HELLO\_4LINKS” yields this simulation response:

A screenshot of a computer

Description automatically generated with medium confidence

Figure 8: Modified ASCII String Transmission

The “stim\_gen” process is set to automatically stop the simulation ~12us after an EOP has been read on the *u\_spw\_hw\_rx* entity *Rx\_data* port.

# Single-Ended Operation

Although the 4Links SpaceWire Codec is intended to operate using LVDS IO. A wrapper is provided which configures the IP core for single-ended operation. This may be useful if your target FPGA device does not support LVDS IO buffers. Single-ended operation of the core is included in this design example. To simulate a single-ended design, modify the testbench constant *c\_mode* to “single”. This will select Single-Ended mode for core operation.

A computer screen shot of a computer

Description automatically generated

Figure : Single-Ended Operation of IP Core

# Design Constraints

The design constraints file applies to the top-level wrapper for the example design. The top-level wrapper entity includes I/O ports which correspond to the uncommented lines in the provided constraints file. The Constraints file is valid for the KCU116 Development Board for the Xilinx Kintex UltraScale+ FPGA part. The constraints file included in the example design was provided by Xilinx and modified by 4Links. 4Links take no responsibility for part damage which occurs from using the constraints file supplied as is in this example design. Always check with the manufacturer for the most up-to-date constraints file(s) for your components.

The *spw\_wrapper\_xilinx* entity contains instantiations for the appropriate IO buffers and DDR registers for this design. The voltage levels set by the pin constraints files comply with the limits set by these primitives.

The primitive instantiation is done to ensure correct design functionality when porting between different hardware within the same device family. Under certain circumstances it can be viable to leave primitives un-instantiated and allow for the toolchain synthesizer to automatically include relevant IO buffers and hardware as required by the synthesized design.

Always check for guidance when instantiating device primitives to ensure strict operating rules are adhered to.

# Synthesizing the Design

Running design synthesis is recommended only after a behavioural simulation has been successfully performed. If all steps have been correctly followed the design should synthesize correctly. Do be aware that a synthesis timing report may warn of inadequate holding slack and subsequently throw timing violations.

This is normal for some designs, especially with hold-timing violations. When implementation is run the functional netlist will be routed, and timing corrections will be applied. Whilst some information can be gleaned from post-synthesis timing reports they should never be relied upon or taken as absolute.

# Implementing the Design

Upon implementing the design, you should see that any timing errors which appeared during synthesis have been resolved. Any failures to implement the design should be addressed and implementation re-tried.

## Post-implementation simulation

For VHDL based designs, Vivado supports “Functional Simulations” natively and “Timing Simulations” using a Verilog-based netlist. If the project built correctly, your simulator language should be set to “mixed”. This will allow for both Timing and Functional simulations to be performed. This example will cover “Timing Simulations” as they are usually preferred for verification.

Set the *spw\_hw\_tb(implement)* simulation source file as “Top Level”. This will ensure that the correct timing netlist for the implemented design is instantiated into the test bench.

By default, the testbench contains only high-level signals. You can add lower-level signals to the waveform viewer using the steps provided in previous sections. Like before, you may need to re-simulate the design in order to update the waveform GUI.

# Example Summary

By now you should be familiar with how the 4Links SpaceWire IP works and how you might implement a design to send data over a SpaceWire link using the 4Links IP. In the next example we will look at using the 4Links SpaceWire IP with the 4Links RMAP Client and RMAP Target IP to communicate with multiple devices over a SpaceWire link.