psi_tb

Documentation

Content

Table of Contents

Figures

1 Introduction

The purpose of this library is to provide HDL packages with procedures and functions that help writing test benches. This document serves as description or as a list of the different functionality that can be found in psi_tb.

1.1 Working Copy Structure

If you just want to use some components out of the *psi_common* library, no special structure is required and the repository can be used standalone.

If you want to also run simulations and/or modify the library, additional repositories are required (available from the same source as *psi_common*) and they must be checked out into the folder structure shown in the figure below since the repositories reference each-other relatively.

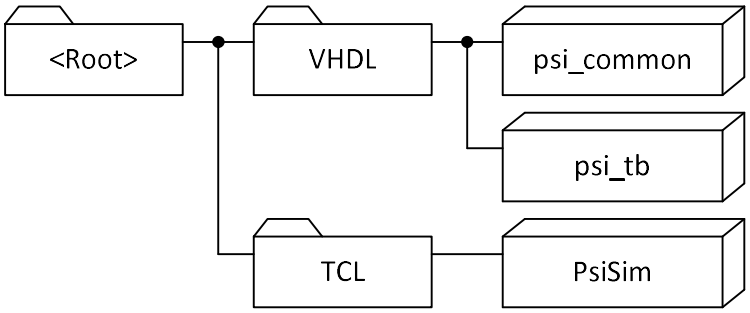

Figure 1: Working copy structure

It is not necessary but recommended to use the name *psi_lib* as name for the *<Root>* folder.

1.2 VHDL Libraries

The PSI VHDL libraries (including *psi_common*) require all files to be compiled into the same VHDL library.

There are two common ways of using VHDL libraries when using PSI VHDL libraries:

a) All files of the project (including project specific sources and PSI VHDL library sources) are compiled into the same library that may have any name.
In this case PSI library entities and packages are referenced by *work.psi_<library>_<xxx> (e.g. work.psi_common_pl_stage* or *work.psi_common_array_pkg.all*).

b) All code from PSI VHDL libraries is compiled into a separate VHDL library. It is recommended to use the name *psi_lib*.
In this case PSI library entities and packages are referenced by *psi_lib.psi_<lib>_<xxx> (e.g. psi_lib.psi_common_pl_stage* or *psi_lib.psi_common_array_pkg.all*).

1.3 Running Simulations

1.3.1 Regression Test

1.3.1.1 Modelsim

To run the regression test, follow the steps below:

- Open Modelsim
- The TCL console, navigate to *<Root>/VHDL/psi_common/sim*
- Execute the command "*source ./run.tcl*"

All test benches are executed automatically and at the end of the regression test, the result is reported.

1.3.1.2 GHDL

In order to run the regression tests using GHDL, GHDL must be installed and added to the path variable. Additionally a TCL interpreter must be installed.

To run the regression tests using GHDL, follow the steps below:

- Open the TCL interpreter (usually by running *tclsh*)
- The TCL console, navigate to *<Root>/VHDL/psi_common/sim*
- Execute the command "*source ./runGhdl.tcl*"

All test benches are executed automatically and at the end of the regression test, the result is reported

1.3.2 Working Interactively

During work on library components, it is important to be able to control simulations interactively. To do so, it is suggested to follow the following flow:

- Open Modelsim
- The TCL console, navigate to *<Root>/VHDL/psi_common/sim*
- Execute the command "*source ./interactive.tcl"*
    - This will compile all files and initialize the PSI TCL framework
    - From this point on, all the commands from the PSI TCL framework are available, see documentation of *PsiSim*
- Most useful commands to recompile and simulate entities selectively are
    - *compile_files –contains <string>*
    - *run_tb –contains <string>*

The steps vor GHDL are the same, just in the TCL interpreter shall instead of the Modelsim TCL console.

1.4 Contribute to PSI VHDL Libraries

To contribute to the PSI VHDL libraries, a few rules must be followed:

- Good Code Quality
  - There are not hard guidelines. However, your code shall be readable, understandable, correct and save. In other words: Only good code quality will be accepted.
- Configurability
  - If there are parameters that other users may have to modify at compile-time, provide generics. Only code that is written in a generic way and can easily be reused will be accepted.
- Self checking Test-benches
  - It is mandatory to provide a self-checking test-bench with your code.
  - The test-bench shall cover all features of your code
  - The test-bench shall automatically stop after it is completed (all processes halted, clock-generation stopped). See existing test-benches provided with the library for examples.
  - The test-bench shall only do reports of severity *error*, *failure* or even *fatal* if there is a real problem.
  - If an error occurs, the message reported shall start with "###ERROR###:". This is required since the regression test script searches for this string in reports.
- Documentation
  - Extend this document with proper documentation of your code.
  - Highlight all documentation changes in feature branches in yellow so they can be found easily when merging back to master.
- New test-benches must be added to the regression test-script
  - Change */sim/config.tcl* accordingly
  - Test if the regression test really runs the new test-bench and exits without errors before doing any merge requests.

1.5 Handshaking Signals

1.5.1 General Information

The PSI library uses the AXI4-Stream handshaking protocol (herein after called AXI-S). Not all entities may implement all optional features of the AXI-S standard (e.g. backpressure may be omitted) but the features available are implemented according to AXI-S standard and follow these rules.

The full AXI-S specification can be downloaded from the ARM homepage:
https://developer.arm.com/docs/ihi0051/a

The most important points of the specification are outlined below.

1.5.2 Excerpt of the AXI-S Standard

A data transfer takes place during a clock cycle where TVALID and TREADY (if available) are high. The order in which they are asserted does not play any role.

- A master is not permitted to wait until TREADY is asserted before asserting TVALID.
- Once TVALID is asserted it must remain asserted until the handshake occurs.
- A slave is permitted to wait for TVALID to be asserted before asserting the corresponding TREADY.
- If a slave asserts TREADY, it is permitted to de-assert TREADY before TVALID is asserted.

An example an AXI handshaking waveform is given below. All the points where data is actually transferred are marked with dashed lines.

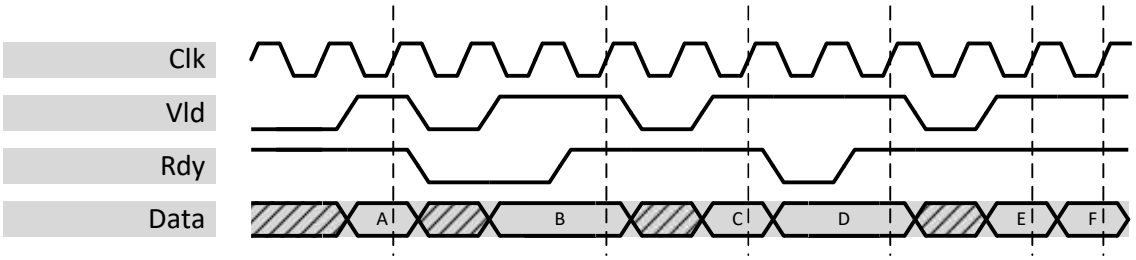

Figure 2: Handshaking signals

1.5.3 Naming

The naming conventions of the AXI-S standard are not followed strictly. The most common synonyms that can be found within the PSI VHDL libraries are described below:

TDATA InData, OutData, Data, Sig, Signal, <application specific names>

TVALID Vld, InVld, OutVld, Valid, str, str_i

TREADY Rdy, InRdy, OutRdy

Note that instead of one TDATA signal (as specified by AXI-S) the PSI VHDL Library sometimes has multiple data signals that are all related to the same set of handshaking signals. This helps with readability since different data can is represented by different signals instead of just one large vector.

2 Packages

2.1 psi_tb_activity_pkg

2.1.1 Description

The activity package allows generating simple signals and checking activities or value in test benches. It is able for instance generating pulse and valid/strobe signals to feed in a design under test by specifying frequency. Other procedures allows for example verifying if a value expected is arrived within a specific period of time. The $2.1.3 gives the full list of procedure present in this package.

2.1.2 Dependencies

- psi_tb.txt_util
- psi_tb_compare_pkg

2.1.3 List

Procedure	Description
CheckNoActivity	Wait for a given time and check if signal is idle, expected input type is std_logic
CheckNoActiviyStlv	Wait for a given time and check if signal is idle, expected input type is std_logic_vector
CheckLastActivity	Check when a signal had its last activity (without waiting) expected input type is std_logic
CheckLastActivityStlv	Check when a signal had its last activity (without waiting), expected input type is std_logic_vector
PulseSig	pulse a signal
ClockedWaitFor	Clocked wait for a signal
WaitClockCycles	Wait for a number of clock cycles
ClockedWaitTime	Wait for a time and quit on rising edge
GenerateStrobe	Generate a valid/strobe signal, expected parameters clock frequency and valid/strobe frequency in Hz (type is real)
WaitForValueStdlv	check if value is arrived within a defined period of time, expected input type is std_logic_vector
WaitForValueStdl	check if value is arrived within a defined period of time, expected input type is std_logic

2.2 psi_tb_compare_pkg

2.2.1 Description

This package allows doing comparison between two values, an expected value and an output signal with different types. Over more it is possible to specify a tolerance of uncertainty and if a mismatch occurs an error message will be thrown as well as an assertion will be raised.

2.2.2 Dependencies

- psi_tb.txt_util

2.2.3 List

Procedure	**Description**
StdlvCompareInt	std_logic_vector compare to integer
StdlvCompareStdlv	std_logic_vector compare to std_logic_vector
StdlCompare	std_logic compare std_logic
IntCompare	integer compare to integer
RealCompare	real compare to real
SignCompare	signed compare to signed
UsignCompare	unsigned compare to unsigned
SignCompareInt	signed compare to integer
UsignCompareInt	unsigned compare to integer
Function	
IndexString	returns an index string in the form "[3]"

2.3 psi_tb_textfile_pkg

2.3.1 Description

The package contains three procedures that helps manipulating text file. Reading and applying data from text file to a design under test (DUT), comparing text file content and signals and writing to a text file DUT's output.

2.3.2 Dependencies

- psi_tb.txt_util

2.3.3 List

Procedure	Description
ApplyTextfileContent	Read a text file and apply it to signals column by column, type of data output is integer
CheckTextfileContent	Read a text file and compare it column by column to signals, type of data is integer
WriteTextfile	Write a text file with header line 1 name of data & second line data, type of data to write is integer

2.3.4 Example

```
stim_data_inp_sti <= std_logic_vector(to_signed(stim_data_inp_sti_int,stim_data_inp_sti'length));
stim_data_qua_sti <= std_logic_vector(to_signed(stim_data_qua_sti_int,stim_data_inp_sti'length));
proc_stim : process
begin
      wait until rst_sti = '0';
      --TAG Apply Stimuli
      ApplyTextfileContent( Clk         => clk_sti,
                            Rdy         => PsiTextfile_SigOne,
                            Vld         => str_sti,
                            Data(0)     => stim_data_inp_sti_int,
                            Data(1)     => stim_data_qua_sti_int,
                            Filepath    => FileStimFolder_g & "/input.txt",
                            ClkPerSpl   => 1,
                            MaxLines    => -1,
                            IgnoreLines => 1);
      --
      process_done_s(0) <= '1';
      wait
end process;
```

The example shows the procedure to read data from a file which contains two columns, as one can observe the number of clock per sample is set to one (full speed data output) and data are converted from integer to standard logic vector above. The maximum of lines -1 defines it as infinite.

2.4 psi_tb_txt_util

2.5 psi_tb_axi_pkg

2.6 psi_tb_axi_conv_pkg

2.7 psi_tb_i2c_pkg