**psi\_tb**

Documentation

**Content**

**Table of Contents**

[1 Introduction 4](#_Toc25670712)

[1.1 Working Copy Structure 4](#_Toc25670713)

[1.2 VHDL Libraries 4](#_Toc25670714)

[1.3 Running Simulations 5](#_Toc25670715)

[1.4 Contribute to PSI VHDL Libraries 6](#_Toc25670716)

[1.5 Handshaking Signals 7](#_Toc25670717)

[2 Packages 8](#_Toc25670718)

[2.1 psi\_tb\_activity\_pkg 8](#_Toc25670719)

[2.2 psi\_tb\_compare\_pkg 9](#_Toc25670720)

[2.3 psi\_tb\_textfile\_pkg 10](#_Toc25670721)

[2.4 psi\_tb\_txt\_util 12](#_Toc25670722)

[2.5 psi\_tb\_axi\_pkg 13](#_Toc25670723)

[2.6 psi\_tb\_axi\_conv\_pkg 14](#_Toc25670724)

[2.7 psi\_tb\_i2c\_pkg 15](#_Toc25670725)

**Figures**

[Figure 1: Working copy structure 4](#_Toc24121350)

[Figure 2: Handshaking signals 7](#_Toc24121351)

# 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.

## 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.

## 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:

1. 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*).
2. 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*).

## Running Simulations

### Regression Test

#### 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.

#### 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

### 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.

## 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.

## Handshaking Signals

### 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.

### 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

### 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.

# Packages

## psi\_tb\_activity\_pkg

### 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.

### Dependencies

* psi\_tb\_txt\_util
* psi\_tb\_compare\_pkg

### 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 |

## psi\_tb\_compare\_pkg

### 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.

### Dependencies

* psi\_tb\_txt\_util

### List

|  |  |
| --- | --- |
| Procedure | Description |
| StdlvCompareInt | std\_logic\_vector compare to integer |
| StdlvCompareStdlv | std\_logic\_vector compare to std\_logic\_vector |
| StdlCompare | std\_logic or integer 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]" |

## psi\_tb\_textfile\_pkg

### Description

The package contains three procedures that helps manipulating text files. 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.

### Dependencies

* psi\_tb\_txt\_util

### 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 |

### Example

#### ApplyTexfileContent

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.

#### CheckTexfileContent

check\_apa\_array\_s(0) <= to\_integer(unsigned(check\_amp\_s));

check\_apa\_array\_s(1) <= to\_integer(unsigned(check\_phi\_s));

proc\_check : **process**

**begin**

**wait** **until** rst\_sti = ***'0'***;

-- Check

CheckTextfileContent( Clk => clk2\_sti,

Rdy => PsiTextfile\_SigUnused,

Vld => check\_str\_s,

Data(0) => check\_apa\_array\_s(0),

Data(1) => check\_apa\_array\_s(1),

***Filepath*** => ***FileStimFolder\_g*** & "/output.txt"

***IgnoreLines*** => 1);

process\_done\_s(1) <= ***'1'***;

**wait**;

**end** **process**;

The example shows the procedure to check data from a DUT and a file which contains two columns, as one can observe the data enter the procedure with strobe/valid signal which gives the procedure to jump to next line, the ignore lines parameter this to start comparing at line 2 if a header is present. If values don’t match asserts will be raised.

## psi\_tb\_txt\_util

Text util package is an adaptation of a package txt\_util\_pkg.vhd that can be found in internet (opencores.org), it offers functions to help manipulating VHDL type that become more convenient to edit with.

### Dependencies

* std\_textio

### List

|  |  |
| --- | --- |
| Procedure | Description |
| print(string) | Print a message to the prompt |
| print(boolean, string) | Print the message when active |
| str\_read(TEXT, out string) | Read variable length string from input file |
| str\_write(TEXT, in string) | Write variable length string to file |
| print(TEXT, in string) | Print variable length string to a file and start a new line |
| print(TEXT, in character) | Print variable length character to a file and start a new line |
| **Function** |  |
| chr(std\_logic) | Converts std logic into a character |
| chr(integer) | Converts integer into a character |
| str(std\_logic) | Converts std logic into a string |
| str(boolean) | Converts boolean into a string |
| str(std\_logic\_vector) | Converts std logic vector into a string |
| str(integer) | Converts integer into a string |
| str(integer,base:integer) | Converts integer using specified base into a string |
| hstrt(std\_logic\_vector) | Converts std logic vector in hex format into a string |
| hstrt(unsigned) | Converts unsigned in hex format into a string |
| to\_string(integer) | Converts integer to string, VHDL2008 built-in equivalent |
| to\_string(real) | Converts integer to real, VHDL2008 built-in equivalent |
| to\_string(unsigned) | Converts integer to unsigned, VHDL2008 built-in equivalent |
| to\_string(signed) | Converts integer to signed, VHDL2008 built-in equivalent |
| to\_string(std\_logc\_vector) | Converts integer to std logic vector, VHDL2008 built-in equivalent |
| to\_upper(character) | Converts a character to upper case into a string |
| to\_lower(character) | Converts a character to lower case into a string |
| to\_upper(string) | Converts a string to upper case into a string |
| to\_lower(string) | Converts a string to lower case into a string |
| to\_std\_logic(character) | Converts a character to std logic (U,X,0,1,Z,W,L,H,-,X) |
| to\_std\_logic\_vector(string) | Converts a string to a std logic vector |

## psi\_tb\_axi\_pkg

TB AXI package allows simulating bus transactions for AXI (Bus Functional Model). It also contains type record used to map the AXI bus. (MS => Master Slave; SM => Slave master)

### Dependencies

* psi\_common\_math\_pkg
* psi\_tb\_compare\_pkg
* psi\_tb\_txt\_util

### List

|  |  |
| --- | --- |
| Procedure | Description |
| axi\_master\_init | Initialization master |
| axi\_slave\_init | Initialization slave |
| axi\_single\_write | AXI transaction to perform a single write value at specific address |
| axi\_single\_read | AXI transaction to perform a single read at specific address |
| axi\_single\_expect | Compare read value with an expected value and produces an error if different |
| axi\_apply\_aw | Partial transactions apply write address |
| axi\_apply\_ar | Partial transactions apply read address |
| axi\_apply\_wd\_single | Partial transactions apply write data single |
| axi\_apply\_wd\_burst | Partial transactions apply write data burst |
| axi\_expect\_aw | Compare write address partial transaction with an expected one and produces an error if different |
| axi\_expect\_ar | Compare read address partial transaction with an expected one and produces an error if different |
| axi\_expect\_wd\_single | Compare single write data partial transaction with an expected one and produces an error if different |
| axi\_expect\_wd\_burst | Compare burst write data partial transaction with an expected one and produces an error if different |
| axi\_apply\_bresp | Apply BRESP status for write transactions  (00,01,10,11) <=> (OKAY,EXOKAY,SLVERR,DECERR) |
| axi\_expect\_bresp | Compare BRESP status with an expected one and produces error if different |
| axi\_apply\_rresp\_single | Apply RRESP status for single read responses  (00,01,10,11) <=> (OKAY,EXOKAY,SLVERR,DECERR) |
| axi\_apply\_rresp\_burst | Apply RRESP status for burst read responses |
| axi\_expect\_rresp\_single | Compare RRESP status with an expected one for a single read response and produces error if different |
| axi\_expect\_rresp\_burst | Compare RRESP status with an expected for a burst read response one and produces error if different |

## psi\_tb\_axi\_conv\_pkg

This package implements conversions between synthesis-friendly AXI package from ***psi\_common*** and test bench friendly AXI package from ***psi\_tb***. It is done in a separate package to avoid using TB AXI package from having to also include synthesis package

### Dependencies

* psi\_tb\_axi\_pkg
* psi\_common\_axi\_pkg

### List

|  |  |
| --- | --- |
| Procedure | Description |
| axi\_conv\_tb\_synth\_master | Conversion between TB AXI package and synthesis AXI pkg for master side |

## psi\_tb\_i2c\_pkg

TB AXI package allows simulating bus transactions for I2C (Bus Functional Model). It also contains type record used to map the I2C bus.

### Depedencies

* Psi\_tb\_compare\_pkg
* Psi\_tb\_activity\_pkg
* Psi\_tb\_txt\_util
* Psi\_common\_logic\_pkg
* Psi\_common\_math\_pkg

### List

|  |  |
| --- | --- |
| Procedure | Description |
| I2cPullup | Initialization to set scl/sda => ‘H’ |
| I2cBusFree | Initialization to set scl/sda => ‘Z’ |
| I2cSetFrequency | Initialization to I2C clock frequency used in different functions |
| I2cMasterSendStart | Initiates a start from Master, check scl/sda level and assign scl/sda to 0 properly aligned compared to clock |
| I2cMasterSendRepeatedStart | Initiates a repeated start from Master. (*verify the level of scl and if stuck to 0 reapply Z prior to reset to 0)* |
| I2cMasterSendStop | Initiates a stop from Master (*wait for scl equal to 0 to set sda to 0 and set scl to high impedance, check if bus is held to ‘1’)* |
| I2cMasterSendAddr | Initiates master to slave address write |
| I2cMasterSendByte | Initiates master to slave byte write |
| I2cMasterExpectByte | Check expected byte if different an error is produced |
| I2cSlaveWaitStart | Initiates I2C slave wait for start *(check scl/sda level)* |
| I2cSlaveWaitRepeatedStart | Initiates I2C slave repeated start |
| I2cSlaveWaitStop | Initiates I2C slave wait for stop |
| I2cSlaveExpectAddr | Check I2C slave expected address received, an error is produced if different |
| I2cSlaveExpectByte | Check I2C slave expected byte value received, an error is produced if different |
| I2cSlaveSendByte | Initiates I2C send byte after having received address |
| **Function** | **Description** |
| I2cGetAddr | Returns value for specific read or write transaction address |