

IP Core
Specification

*Author: Sinx*

*sinx@opencores.org*

**IP rev. (SVN rev) 26**

**Doc rev. 0.3**

**September 21, 2019**

*This page has been intentionally left blank.*

**Revision History**

| **Rev.** | **Date** | **Author** | **Description**  |
| --- | --- | --- | --- |
| 0.1 | 2018-07-21 | Sinx | First Draft  |
| 0.2 | 2019-09-21 | Sinx | Removed some typos. Minor changes |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Contents**

[1 Introduction 1](#_Toc19977594)

[2 Architecture 2](#_Toc19977595)

[Workflow 2](#_Toc19977596)

[Module hierarchy 3](#_Toc19977597)

[3 Operation 5](#_Toc19977598)

[wishbone\_bfm\_pkg functions 5](#_Toc19977599)

[tb\_top wishbone address decoder 5](#_Toc19977600)

[4 Clocks 6](#_Toc19977601)

[5 How to get it run 7](#_Toc19977602)

## Introduction

*For the development of IP cores a test bench is needed. The given project provides a test bench written in VHDL which controls the stimulus, the verifier and the DUT via a common wishbone bus function model. The bfm acts like a virtual CPU performing read and write accesses to the control wishbone bus.*

* *tc-files contain the test sequence and use wb\_bfm procedures*
* *the wb\_bfm\_procedures generate transactions on the wishbone bus*
* *the wishbone bus connects to*
	+ *the stimulator, which writes or generates DUT inputs*
	+ *the verifier, which reads or checks DUT outputs*
	+ *the DUT, which can be configured*

*In contrast to Ken Campbells "The VHDL Test Bench" (*[*https://opencores.org/project/vhld\_tb*](https://opencores.org/project/vhld_tb)*) we do not use an interpreter for the bfm functions. We use plain VHDL which requires VHDL knowledge for test definitions, but also provides more flexibility.*

*The given project uses VHDL 2008.*

*For the development of this project I used*

* *the free ModelSim - Intel FPGA Starter Edition 10.6c (Quartus Prime Pro 18.0)*
* *Notepad ++V7.5.7 with the AutoSave plugin*

## Architecture

### Workflow

The following figure illustrates the workflow of the test bench. Test cases define via register write accesses how the simulator, the DUT and the verifier work. In both templates for the simulator and verifier a wb register process allows write accesses from the test case to be stored. The simulator connects some of those registered bits to inputs of the DUT. In the verifier outputs from the DUT are connected to a read address decoder of the wb bus, allowing the test case to read those outputs.

The test case writes to the stimulator either the DUT input patterns or stars a more complex generator. The verification can either be done in the test case by comparing to expected values or inside the verifier for more complex operations.

Stimulator

DUT

Verifier

Testcase

Wishbone bus

Figure 1: Workflow

### Module hierarchy

tb\_top.vhd

stimulator.vhd

tc\_top.vhd (entity only)

tc\_xxxx.vhd (architecture)

core\_top.vhd (DUT)

verifier\_top.vhd

Figure 2: source file hierarchy



Figure 3: Design unit hierarchy

The highest simulation file is tb\_top.vhd it instantiates the stimulator, core\_top, verifier and tc\_top. It also connects the wb bus with tc\_top as master and all other modules as slaves. It generates the basic wb bus clock.

The core.vhd is your DUT.

The stimulator.vhd holds the input signal generator for the DUT. Its output is connected to the DUT.

The verifier.vhd contains the output signal monitor for the DUT. Its input is connected to the DUTs outputs.

The tc\_top.vhd together with the tc\_xxx.vhd files (only one in the template but more if you like) contain the entity and the architecture of the testcase. In the tc\_xxx.vhd the wb test sequences are described. It makes use of the wishbone\_bfm\_pkg.vhd which defines write and read functions.

This project is wishbone compatible.

## Operation

### wishbone\_bfm\_pkg functions

In wishbone\_bfm\_pkg.vhd three basic procedures are provided.

* wb\_write writes data to a given wb address
* wb\_read (format 1) reads data from wb and compares to a given value
* wb\_read (format 2) reads data from wb and delivers it to tc\_xxx

See the comments in the wishbone\_bfm\_pkg.vhd file for details about the parameters and default values.

### tb\_top wishbone address decoder

In tb\_top an address decoder decodes the address from the controlling wb bus into segments. It decodes the upper four bits of the address. The segments are connected to the wb\_slaves\_in\_s and wb\_slaves\_out\_s array of wb bus signals. How many slave busses are existing can be defined with the parameter number\_of\_wb\_slaves\_c. Connect those slave signals to further modules like this:

 port map(

 wb\_i => wb\_slaves\_in\_s(0),

 wb\_o => wb\_slaves\_out\_s(0),

## Clocks

*In tb\_top the parameter g\_wb\_clock\_period defines the clock period for the wishbone bus. The default value is 20ns =>50 MHz*

*Further clock for your DUT can be created similarly.*

## How to get it run

In order to get a local copy of this tb to run do the following

1. Downlaod Modelsim from the Altera/intel site. You might need to register first. The Modelsim is part of the free Quartus web edition.
2. Install Modelsim. Be aware that modelsim cannot cope with spaces in the installation path.
3. Get a copy of all vhdl\_wb\_tb files.
4. Adjust the paths in the vhdl\_wb\_tb\trunk\rtl\_sim\run\sim.mpf file. Since modelsim cannot handle mpf-file related relative paths, you have to adjust all file paths in the sim.mpf file manually. Be aware that on windows those paths must use /-slashes instead of \-slashes.
5. Start modelsim.
6. Open the sim.mpf project file.
7. Go to the transscript window and type **do ../bin/init.to**. This script creates the work library and compiles all files into it. It also sets the alias “s” to run the ../bin/s.do script file. The init.do script needs to be run only once for a project.
8. Go to the transscript window and type “s” (without quotes) to run the simulation
9. When you leave modelsim and open it later again the aliases are forgotten. Therefore the s command is no longer recognized. To run the simulation again, type do ../bin/s.do. This also sets the “s” alias again, so that you can re-run the simulation with “s” again in this session.
10. When you pulled new signals to the wave window or set cursors or do other settings you need to save this in the wave.do file. To do this the aliases b (bookmark write) and w (write wave window) have been set by the s.do script. You need to start b first followed by w to have the cursors saved.
11. Now you are ready to modify the vhdl files and simulate it.

If you succeed you should see the following transcript output:

do ../bin/init.do

# Bookmark saving is disabled.

# The 1 bookmark in window "Wave" has been discarded

# End time: 15:17:39 on Jul 21,2018, Elapsed time: 0:01:21

# Errors: 0, Warnings: 1

# Model Technology ModelSim - Intel FPGA Edition vmap 10.6c Lib Mapping Utility 2017.07 Jul 26 2017

# vmap work work

# Modifying D:/OpenCoresOrg/vhdl\_wb\_tb/vhdl\_wb\_tb/trunk/rtl\_sim/run/sim.mpf

# Model Technology ModelSim - Intel FPGA Edition vmap 10.6c Lib Mapping Utility 2017.07 Jul 26 2017

# vmap -del work

# Removing reference to logical library work

# Modifying D:/OpenCoresOrg/vhdl\_wb\_tb/vhdl\_wb\_tb/trunk/rtl\_sim/run/sim.mpf

# Model Technology ModelSim - Intel FPGA Edition vmap 10.6c Lib Mapping Utility 2017.07 Jul 26 2017

# vmap work work

# Modifying D:/OpenCoresOrg/vhdl\_wb\_tb/vhdl\_wb\_tb/trunk/rtl\_sim/run/sim.mpf

# Compile of convert\_pkg.vhd was successful.

# Compile of my\_project\_pkg.vhd was successful.

# Compile of wishbone\_pkg.vhd was successful.

# Compile of wishbone\_bfm\_pkg.vhd was successful.

# Compile of tc\_top.vhd was successful.

# Compile of tb\_pkg.vhd was successful.

# Compile of stimulator.vhd was successful.

# Compile of verifier.vhd was successful.

# Compile of tc\_xxxx.vhd was successful.

# Compile of core\_top.vhd was successful.

# Compile of top.vhd was successful.

# Compile of tb\_top.vhd was successful.

# All compile dependencies have been resolved.s

# vsim -t 1fs tb\_top

# Start time: 15:17:55 on Jul 21,2018

# Loading std.standard

# Loading std.textio(body)

# Loading ieee.std\_logic\_1164(body)

# Loading ieee.numeric\_std(body)

# Loading ieee.std\_logic\_arith(body)

# Loading work.convert\_pkg(body)

# Loading work.my\_project\_pkg(body)

# Loading work.wishbone\_pkg(body)

# Loading work.wishbone\_bfm\_pkg(body)

# Loading work.tb\_top(rtl)

# Loading work.tb\_pkg

# Loading std.env(body)

# Loading work.tc\_top(tc\_xxxx)

# Loading work.core\_top(rtl)

# Loading work.stimulator(rtl)

# Loading work.verifier(rtl)

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 490 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 490 ns Note -- tc\_xxxx: ADD\_DESCRIPTION\_HERE --

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 490 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 490 ns Note --configure stimulator

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 490 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 510 ns Note --configure verifier

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 510 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 550 ns Note --configuration done

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 550 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 550 ns Note --starting stimulator

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 550 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 710 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 710 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 710 ns Note -- All fine till here. Now we provoke error messages for illustration.

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 730 ns Error error: expected data at address 0x1000\_0008 was: 0x0000\_0006 but read: 0x0000\_0005

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 770 ns Error error: expected data at address 0x1000\_0008 was: 0x0000\_0006 but read: 0x0000\_0005

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 790 ns Note : read data at address 0x1000\_0008 was: 0x0000\_0005

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 810 ns Note <TEXT>: read data at address 0x1000\_0008 was: 0x0000\_0005

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 830 ns Note : read data at address 0x1000\_0008 was: 0x0000\_0005

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 850 ns Error error: expected data at address 0x1000\_0008 was: 0x0000\_0006 but read: 0x0000\_0005 diff: 0000\_0005

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 850 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 850 ns Note --check results

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 850 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 7870 ns Note -- tc\_xxxx finished

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 7870 ns Note test case tc\_xxxx completed successfully

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 7870 ns Note -----------------------------------------------------------------

# \*\* [ Instance: /tb\_top/tc\_top\_inst] 7870 ns Note -----------------------------------------------------------------

# Break in Process tc\_xxxx\_proc at D:/OpenCoresOrg/vhdl\_wb\_tb/vhdl\_wb\_tb/trunk/bench/vhdl/tc\_xxxx.vhd line 147

The wave window should look like this:



In the red ellipse you can see signals controlled by tc\_xxx.vhd

In the transcript you can see error messages generated by unexpected read values.