Test Production Suite facility

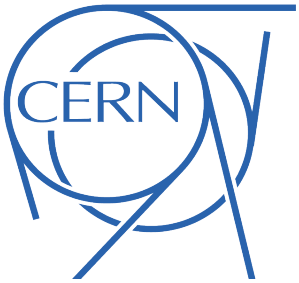
Specification document

Version	Date	Author/s
0.1 - draft	26/01/2011	Samuel I. Gonsálvez, BE/CO/HT
1.0	01/02/2011	Samuel I. Gonsálvez, BE/CO/HT

Beam Department, Control group, Hardware & Timing section

BE/CO/HT

For internal purpose



Test Production Suite facility

Introduction

BE/CO/HT and manufacturers need a form to test and debug the boards developed in their projects.

For that reason, it is required to develop a production test suite for the manufacturers and developers of Simple PCIe FMC carrier, ADCs and others.

These tools should be generic enough to use with several use cases and to test the connections of each board.

Basic architecture

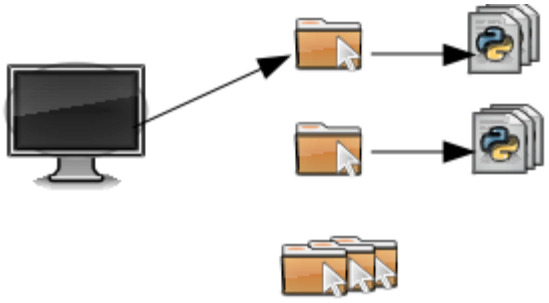


Illustration 1: Diagram of the architecture

The Test Production Suite (TPS) should contain a main program and separate directories where the testing files for each board to test.

It also provides a layer to communicate between test files and the driver used to talk to the board.

Main program
Test files
Test library
Driver
VHDL
Hardware

Main program

1. The main program should provide an interface with the user to select the board to test and the options needed to run these tests.
    1. The interface should be done in command-line and should be simple and usable.
2. The program should receive the necessary data from:
    1. Config file.
    2. Command-line parameters which override the setup from the config file.
3. The data information received should be:
    1. Name of the directory where load the test files.
    2. The test files that the user wants to run and the order. Example: run “test01.py test05.py test22.py” in that order.
    3. Also, the user should have the possibility to setup sequences (repeat N times one batch of tests) in the specified order or randomly.
4. The user could have the possibility to change the selected board and other parameters in run time.
5. The user can save the sequence of test files configured to run in a config file from the Main program prompt.
6. The user should be reported of the errors and warnings present during the test.
7. The program should indicate, in case of error, the following options: continue with the next test in the sequence, stop and exit, jump to an specific test.
8. The program should create a log file to indicate the test programs done and their results.
9. Once the user selects the board, the main program should open the corresponding directory (one for each board) and read the available test files.
10. Using the information provided, it should load the necessary test files and run these actions:
    1. Run each program in the sequence provided by the user. If no sequence is provided, it should be sequential using the name of the tests (test00, test01...) as reference.
    2. The user could select that one batch of tests which will be repeated a concrete number of times.
    3. If the test file required is not present, the main program should print an error and exit.

Directories

1. The name of the directories should be explicit of the board who want to test and/or the test case of the board. Example: /SPEC-VME/, /SPEC-PCI/, /SPEC-NOT-IRQ_CONTROLLER/... and so on.

Test files

1. Their mission is to test the connections and the functionality of each board. Each test file should be created by the test engineers.

2. Test files should be saved in one directory for each board, or for each test case (if there are more than one) for each board. The name of each test should be starting with “test” word and using a sequential number following with the extension of the file. Example: “test01.py”, “test02.py”
3. Test files should have a common interface with the Main Program (to be specified with the developer).
4. The error messages should include the connection line or the functionality that was being tested at the moment of the fail.

Test library

1. Testing library should provide a set of functions that will be used by the test files:
    1. Open/close the device.
        1. Select another device.
    2. Read/write from a register. Parameters: device, offset, size, address_space.
    3. Read/Write using a DMA operation. The user can specify the DMA mode wanted using the address modifier.
        1. Parameters: address, offset, address_modifier, size, data length.
    4. Read_and_check: Read and compare the data with the one provided.
        1. Parameters: device, mask, offset, size, address_space, expected_data, error message.
    5. Write_and_check: Write and compare the data written with the one provided.
        1. Parameters: device, mask, offset, size, address_space, data_to_write, error message.
    6. Wait for interrupt. It should have the possibility to wait a specific time (timeout) and exit if the IRQ has not arrived after the timer expired.
        1. Parameters: device, timeout.
    7. Provide a call to an existing and external FPGA firmware loader program.
        1. Parameter: name of the firmware file to load.
2. Test library should report error messages and indicate its state to the Main Program.
3. The test library should use the corresponding driver interface to communicate with the underlying hardware.

Driver

The driver is the bottommost software layer inside of the front-end.

There are drivers for each bus port of interest: VME and PCI-Express at this moment.

It should provide an interface to execute the corresponding functions of the test library.

VHDL

Each VHDL file should be created by the test engineers to enable/disable the corresponding cores that allow the tests to be completed.

Others

As far as possible, the whole TPS should be designed with simplicity, flexibility and compactness in mind.

It should use free software tools as much as possible, to allow future modifications without the requirement of proprietary licenses.

It should have free software license to allow to be used/modified by the Open Hardware Community and others.

The source code of the whole environment should be published in a public repository available on Internet. Test files are not mandatory but recommended to be published.