# Production Test Suite

## User’s Manual

# Revision Table

| Revision | Date | Author | Comments |
|---|---|---|---|
| 0.1 | 01/06/2011 | Samuel IGLESIAS GONSALVEZ, CERN | Initial version |
| 0.2 | 20/06/2011 | Renan LETHIECQ, CERN | Add pictures, fix documentation |
| 0.3 | 05/08/2011 | Samuel IGLESIAS GONSALVEZ, CERN | Added how to create a test file |

# Table of contents

Table of contents ..... 3
Introduction ..... 4
  Architecture ..... 4
Installation procedure ..... 5
How to use PTS ..... 6
How to create a test file ..... 9

# Introduction

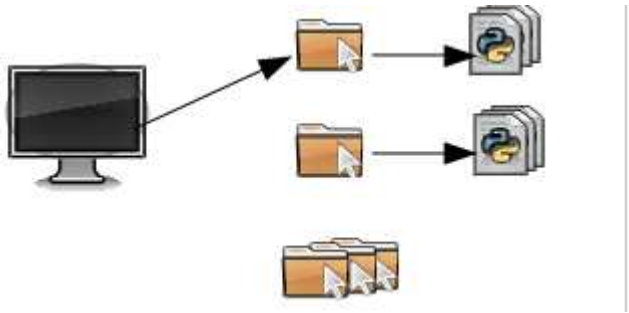
Production Test Suite (as knows as *PTS*) is an environment to check the connectivity of the boards produced in a factory. This environment was originally designed to test the boards specifically designed for Open Hardware Repository[1] but it can be adapted to test other boards.

This project started to cover the needs of CERN groups which want to be sure that the boards produced by external companies follow a minimum quality in matters of soldering, mounting and fabrication process of the PCB.

Note: This test is functional. It is not intended to cover verification & validation tests of the design.

## Architecture

The environment has an architecture described in this section.



The Production Test Suite (PTS) contains a main program and separate directories where are placed the testing files for each kind of board. It provides a layer (test library) to communicate between test files and a generic driver used to talk to the board's firmware.

| Main program |
|---|
| Test files |
| Test library |
| Driver |
| VHDL |
| Hardware |

Production Test Suite uses a generic driver called *gnurabbit* which allows the communication between the board and the test programs. This *gnurabbit* driver only allows to communicate with only one PCI-board of the computer.

[1] http://www.ohwr.org

PTS generates a bunch of log files which contains the results for each test and for each execution. When there is a critical error, this message is also displayed on the screen. Sometimes the error allows to continue the execution or, even, ask to the user what to do.

# Installation procedure

Before to use the PTS environment, there are some requirements that needs to be fully accomplished:

- GNU/Linux operating system with kernel 2.6.24 or higher.
- GCC, kernel sources and related stuff to compile the *gnurabbit* driver.
- Python 2.4 or higher.[2]
- Download the latest version of the Production Test Suite from the official website OHWR: http://www.ohwr.org

Before executing the PTS program, the user should compile the provided *gnurabbit* driver for the first time.

All you need is gcc, the headers of your current Linux kernel. Go into $PATH_GNURABBIT/kernel and compile using *make.*

Once it is compiled, the user should load the compiled driver in the target machine using *insmod* command as root, each time you boot[3]:

# insmod $PATH_DRIVER/gnurabbit.ko

After that, the user can run PTS. For example in a terminal execute: $PATH_PTS/pts.py -h

Next step is to check if the board to test is properly plugged, the corresponding test files are downloaded and the required elements are satisfied.

[2]Check the documentation provided for test each board. For SPEC boards is recommended to use Python 2.7 or higher because it needs some features that are integrated in that version.
[3] It is also possible to load the kernel module automatically during the boot process. You are encouraged to search on Internet how to do it.


# How to use PTS

The PTS program is a command-line utility. Its behavior is determined by the command-line arguments supplied. As a bare minimum, the user must specify the board type to test, its serial number for identification purposes, and paths to the directories for test programs and log outputs. Moreover, a sequence of test case numbers has to be provided.

With this information at hand, PTS proceeds to run the enumerated tests in the specified order (or a random order if the --randomize option was provided), until successful completion. However, if any of the tests fails, PTS will, according to the severity of the error:

- abort the suite immediately.
- report the error and continue with the following tests.
- ask the user what to do.

In addition, a repetition number can be provided with the –ntimes option, causing the sequence to be performed a prescribed number of times.

A typical (actually, quite complete) run of PTS is started as follows:

```
$ ./pts.py -b SPEC -s 0314159265 -t ./tests/ -l ./logs/
                    -r -n 100 00 02 test04 test05 01
```

This test will refer to a SPEC board (-b SPEC) with serial number 0314159265. The test programs for this run will be found under ./tests/, and should be named test00, test02, test04, test05 and test01. The prefix test is optional in the list, but the filenames must have this structure. The specified sequence of programs will be run in a random permutation (option -r) one hundred times (-n 100). Logs of the run will be found under ./logs/.

During the run, no information will be printed to the console, except in case of test case failure. Such failures are reported to the console as explained above.

The complete log of the execution of tests and their results is found under the specified log path with a name pattern:

```
pts_run_{runid}_{timestamp}_{board}_{serial}.txt
```

and the standard outputs of the test programs with a pattern:

```
pts_tst_{runid}_{timestamp}_{board}_{serial}_{testnumber}.txt
```

like

```
pts_run_f5cd678_20110527.102727.243708_SPEC_000000.txt
pts_tst_f5cd678_20110527.102727.248371_SPEC_000000_51.txt
```

This makes spotting of particular test runs quite straightforward in directory listings.

A complete explanation of the command-line options of PTS follows:

`-h, --help`
: Show a help message and exit.

`-c CONFIG, --config=CONFIG`
: Config file name. Default options will be read from the config file. Command-line options override the settings specified in it.

`-w, --write-config`
: Write configuration data to config file.

`-C, --cli`
: Enter command-line interpreter.

`-b NAME, --board=NAME`
: This specifies the board name. The parameter is mandatory.

`-s SERIAL, --serial=SERIAL`
: This specifies the serial number of the board being tested. This parameter is mandatory.

`-t PATH, --test-path=PATH`
: Path where test files will be looked for (see below).

`-l PATH, --log-path=PATH`
: Path where the whole log set will be written to. Write permission to the (already existing) PATH should be granted; otherwise, the whole suite will abort. Log files stored here will belong to three categories: a run log file (named pts_run_*).

`-n NUMBER, --ntimes=NUMBER`
: Number of times to repeat the batch of tests.

`-r, --randomize`
: Run the batch in random order.

`test, test...`
: The remaining command-line arguments are interpreted as test cases, and are specified either by a two-digit number (e.g., 05) or by a string testXX. Files named testXX must be present in the test directory; otherwise, the suite will be aborted.

`-w, --write-config`

Write configuration data to config file.

`-y, --yes`

Assume all user interventions are affirmative.

`-e SERIAL, --extra_serial=SERIAL`

Another board serial number [Optional].

# How to create a test file

Firstly, you need to develop the corresponding firmware file (if needed) to the target board. The firmware has defined some registers using the Wishbone bus that could be accessed by the test files.

It is recommended to create a folder called $PATH_PTS/test/<board>/ with the following structure inside:

- firmwares/ -- Folder to save the .bin files and the program to load the FW file (fpga_loader or other).
- python/ -- Folder to save the python test programs.
- doc/ -- Optional folder to save needed documentation.
- utils/ -- Optional folder to add needed tools, utilities, external programs.

To create a test file for the PTS environment, you have an example in $PATH_PTS/test/example/ that shows a test that prints “Hello world” on the screen and raise an exception. Also, there are the tests of the SPEC board that could be interesting to read.

It is recommended the use of classes for each functionality (I2C, SPI...). It’s also recommended than the test can run standalone without the need of the pts.py program. See examples in SPEC test files.

The structure of the python script is, usually, something like:

- Definition of the needed classes.
- Main function:
    - Initialization of the Gennum (if it is PCI board) or other.
    - Initialization of the needed components (I2C addresses, SPI, init registers…)
    - Access to the board using the provided functions from the API of the test library (rr.py or other)
    - Print error or raise a PTS exception if needed.