Introduction

In this demo we will demonstrate that the specification coverage package can handle running of multiple testcases as separate simulator runs. The output from each of the simulation runs will be stored to individual files, and the run_spec_cov.py script will combine the output from all the test runs when evaluating the requirements.

Background Information

This example of the Specification Coverage concept is slightly more advanced than the example located in the basic_usage folder. This example will demonstrate the multiple output files and sub-requirements feature of the Specification Coverage concept. Similar to the basic_usage example, the testbench is based on a simplified version of the one available in the bitvis_uart example. The UART DUT is located under *bitvis_uart/src/*. For this example, the following requirements from the "customer" are used:

Requirement	Description
FPGA_SPEC_1	The default register values of the module shall be as follows: - RX_DATA: 0x00 - TX_READY: 0x01 - RX_DATA_VALID: 0x00
FPGA_SPEC_2	Data written to the TX_DATA register shall be transmitted by the UART TX interface.
FPGA_SPEC_3	Data received by the UART RX interface shall be made available in the RX_DATA register, accessible over SPI.
FPGA_SPEC_4	The module shall handle simultaneous operation of UART transmit and receive.

The first requirement, FPGA_SPEC_1, is broader than it should be. As it is now, it contains three individual, testable requirements. In order to get the desired configuration where one requirement is tested by one testcase, we divide the requirement into three sub-requirements:

FPGA_SPEC_1.a – The default register value of RX_DATA shall be 0x00
FPGA_SPEC_1.b – The default register value of TX_READY shall be 0x01
FPGA_SPEC_1.c – The default register value of RX_DATA_VALID shall be 0x00

In addition, the customer follows a strict development procedure where all testcases must be defined before implementation, and it must be demonstrated that all requirements will be covered by the verification. In these cases, it is common to create a testcase to requirement mapping. For this example, the testcase to requirement mapping can be seen in the table below.

Requirement	Description	Test case
FPGA_SPEC_1.a	The default register value of RX_DATA shall be 0x00.	T_UART_DEFAULTS
FPGA_SPEC_1.b	The default register value of TX_READY shall be 0x01.	T_UART_DEFAULTS
FPGA_SPEC_1.c	The default register value of RX_DATA_VALID shall be 0x00.	T_UART_DEFAULTS
FPGA_SPEC_2	Data written to the TX_DATA register shall be transmitted by the UART TX interface.	T_UART_TX

FPGA_SPEC_3	Data received by the UART RX interface shall be made available in the RX_DATA register, accessible over SPI.	T_UART_RX
FPGA_SPEC_4	The module shall handle simultaneous operation of UART transmit and receive.	T_UART_SIMULTANEOUS

The information in this table is added to the req_list_advanced_demo.csv file.

Running the demo

The demo can be run by executing the python script *run_advanced_demo.py* from the script/ directory:

```
>>python run_advanced_demo.py
```

Or from the sim/ directory:

```
>>python ../script/run_advanced_demo.py
```

Note that Python 3.x is required to run this demo-script. The script will compile all the VHDL sources and execute each testcase as a separate run in the simulator. Since some simulators are locked to a fixed number of licenses, the simulations will not be started in parallel. However, this could be efficient if there are multiple available simulation licenses.

Once all the VHDL testcases have completed, the *run_advanced_demo.py* script will call the *run_spec_cov.py* script automatically. The inputs and outputs to the script are read from the file */demo/advanced_usage/config_advanced_demo.txt.* The script will parse the partial coverage files from the VHDL simulations, now located under:

- */sim/partial_cov_advanced_demo_T0.csv*
- */sim/partial_cov_advanced_demo_T1.csv*
- */sim/partial_cov_advanced_demo_T2.csv*
- */sim/partial_cov_advanced_demo_T3.csv*

The script will also read the requirement map list located in the CSV file */demo/advanced_usage/req_map_advanced_demo.csv*.

After reading all the input files, the script will go through the data and evaluate each requirement as compliant or non-compliant. The results of this evaluation are written to the output file, which is stored under */sim/spec_cov_advanced_demo.csv*.