GH09-implementation example

This document describes the usage of the Stimulus vector generator and generation of the stimulus vector generator memories. The initial values of the memories are generated by python scripts.

The package consist of an example design for each GH09 test case, stimulus generator for each case (which needs a stimulus generation), parser for XML-input stimulus, parser for XML-reference output and example XML-files generated for 10 MHz tick rate.

The directory structure of the package is following:

- When package is extracted, it creates GH09 directory, which is the root directory of the package.
- Example XML-files are located in <root>/XML-files/<Case_number>/
  where <Case_number> is B1, B2, B3, B4, B5 or B6. These XML-files are generated by GH09-tool with default input, except the tick rate of the system is changed to 10 MHz.
- Python parsers are located in <root>/python/
- Memory images for the default(case w.0) XML-stimulus are located in <root>/MEM_IMAGES/<Case_number>/
- Example HDL descriptions of the cases are located in <root>/VHDL/CASE/<Case_number>/
- Stimulus generators are located in <root>/VHDL/CASE/TB_<Case_numer>/
- common files for the test generators are located in <root>/VHDL/CASE/TB_COMMON/

Power measurement setup

The power measurement setup consists of power measurement system, stimulus vector generator, and design-under-test (DUT) as illustrated in Figure 1.

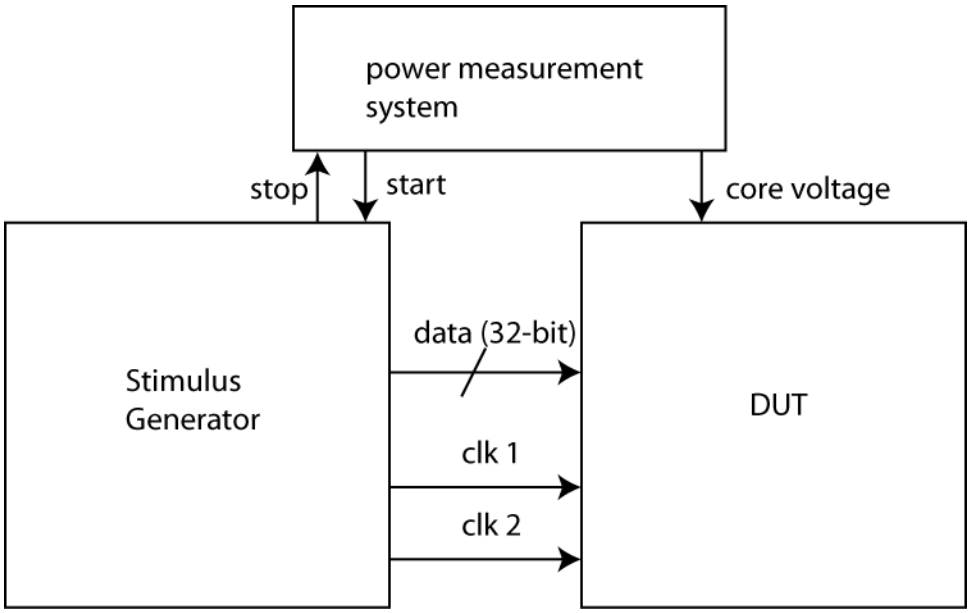

Figure 1 Power measurement setup

Power measurement system is used to measure the power consumption of the DUT and there are several ways to do it. The best way is to measure the core voltage and current drawn by the core. If measuring only the current, the core voltage should be stabilized to the fixed level, such that the supply voltage does not drop even if the DUT draws large amount of current.

The stimulus vector generator can be controlled by power measurement setup, although it is not necessary. The start signal indicates when the stimulus vector generator starts feeding the input stimulus and stop signal indicates when the test is completed. These signals are active high.

Stimulus vector generator

In order to measure the power power consumption of DUT with given stimulus (workload) generated by GH09 tools, the stimulus data needs to be fed to the DUT. This is done by the stimulus vector generator unit. The stimulus vector generator unit can also verify the correctness of the output of the DUT. The stimulus vector generator can be implemented on an FPGA, which is then connected to DUT , e.g. Altera DE2 FPGA-board, which can provide 36-inputs and read 36-outputs to the DUT. The inputs of the DUT can be either regular output or clock signals from the DE2 board.

The structure of the basic stimulus vector generator unit is based on memories shown in Figure 2. There is memory for time (when event happens in workload), value (value of input vector, or control bits for

input memory at corresponding event) and input (value of input, controlled by value memory). The principal structure of the state machine controlling the stimulus vector generator is shown in Figure 3.

Verification of the DUT output is an optional feature, as shown in a dotted line in figure 2. When developing DUT functionality, the functionality of the system has to be verified thoroughly using other sophisticated verification methods.

In the GH09 Case B.1 and B.2, the verification is not implemented on the example testbench and, in B.1. case, it is assumed to have “perfect master” testbench, which means that Stimulus generator will generate button presses and responses to interrupts regardless of DUT outputs.

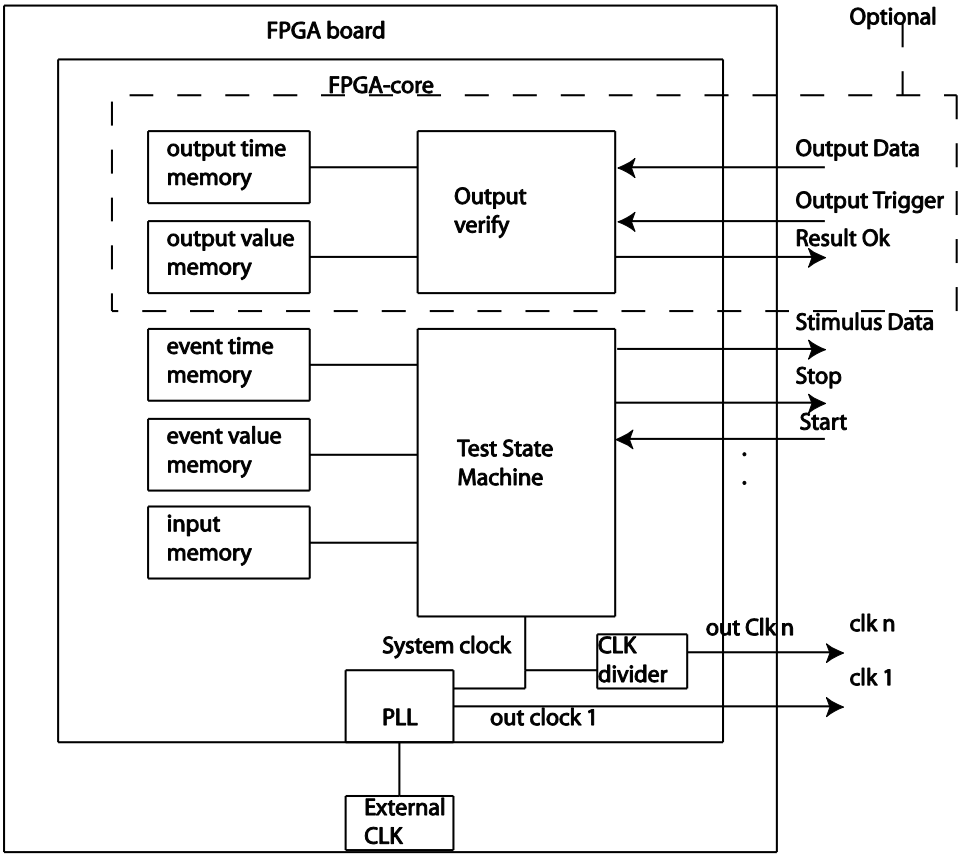

Figure 2 Structure of Stimulus vector generator

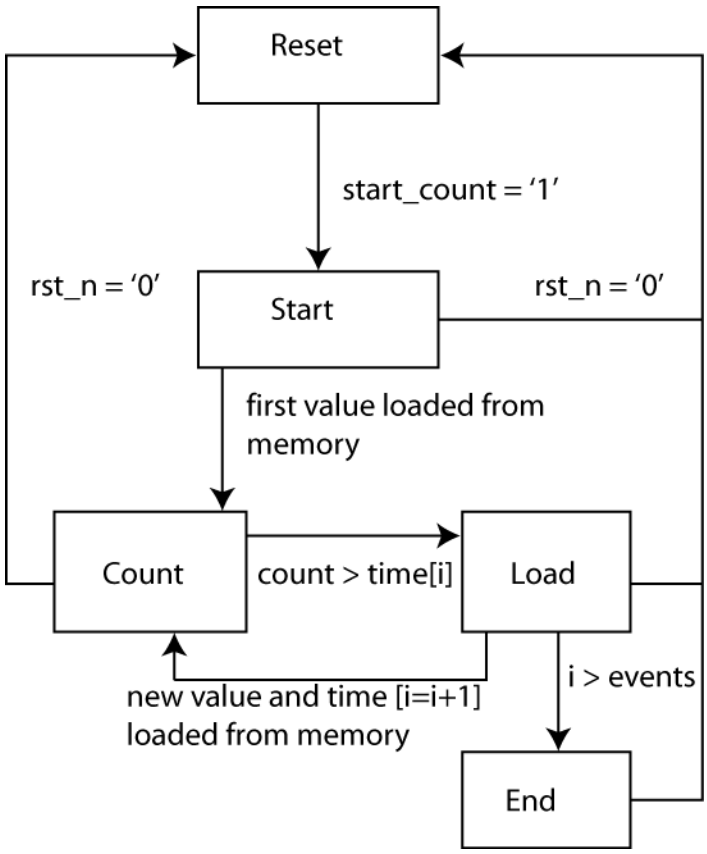

Figure 3 Stimulus vector generator control state machine

Usage of the created test suite

Test suite consists of stimulus vector generator and XML-parser to create the stimulus vectors for the stimulus vector generator.

Stimulus vector generator

The XML-parser is a set of python scripts (python version 2.6.4, other version should work as well) which generate memory components needed by stimulus vector generator. In order to use the scripts python interpreter needs to be installed (following commands works on Unix/Linux machines; in windows machines, e.g., CYGWIN needs to be installed with python interpreter)

Step 1: Generate XML-stimulus with GH09-toolset This can be done by following the README of the GH09-toolset.

Step 2:

In this step XML-stimulus is modified to input/output vectors.

Go to the directory where parse_xml.py is located (<PATH_EXTRACTED>/GH09/python/)

Run parse_xml.py inside python interpreter as following:

python parse_xml.py <PATH_OF_XML_FILES> <NAME_OF_XML> <OUTPUT> <GH09_CASE>

<PATH_OF_XML_FILES> : path where XML-stimulus generated by GH09-tools is located

<NAME_OF_XML> : the name of the input XML-file ought to be parsed (original naming of the XML-files generated by GH09-tools must be used, since output is parsed at same command and it's assumed to be out_<NAME_OF_XML>)

<OUTPUT> : name of the output files.

Several files are generated starting with the <OUTPUT> prefix (generated input vectors for the DUT)

<OUTPUT>_time.txt: Time vector indicating when an event happens

<OUTPUT>_value .txt : value vector controlling what happens when the given time vector is reached

<OUTPUT>_input.txt: some cases have separated input vectors controlled by the value vector.

<OUTPUT>_input_key.txt: Case B3 has also input key vector.

two files are generated starting with the out_<OUTPUT> prefix (reference output of the DUT)

out_<OUTPUT>_time : Output time vector which defines earliest possible time when output should be ready

out_<OUTPUT>_value : The value of supposed output when output time Vector is triggered.

<GH09_CASE> : Which case is ought to be parsed, B.1, B.2, B.3, B.4, B.5, or B.6.

Step 3:

Use generate_mem_image.py to generate memory initialization files for RTL-simulation of the stimulus generator as follows.

generate_mem_image.py <FOLDER> <FILE> <WIDTH> <TYPE_OF_OUTPUT> <TYPE_OF_INPUT> {LOOPING}

<FOLDER> : folder where just parsed files are located (usually ./)

<FILE> : which file ought to be generated to memory image

<WIDTH> : bit width of the output

<TYPE_OF_OUTPUT>: Radix of the output, HEX,BIN or UNS (unsigned INT)

<TYPE_OF_INPUT>: radix of the input, HEX,INT or BIN.

{LOOPING} : describes if input is wider than the output, so input is divided to multiple lines of the output used in generation of the input vectors. If argument is not given extra bits in input is discarded. Parameter is optional. (parameter value does not matter: if not given value is 0 and if given value is 1)

Step 4:

Use generate_mif.py to generate memory initialization files for the FPGA-board. In this case supported boards are boards which can import mif files, for example Altera FPGA boards. (the script can be easily changed to support other formats)

generate_mif.py <file_name> <width_of_data> <depth_of_memory> <start_point> <radix_out> <radix_in> {division}

<file_name> describes file ought to be modified to .mif file (generated by generate_mem_image.py)
<width_of_data> width of the input and output data
<depth_of_memory> elements in memory. When this is lower compared to length of file, the parsing ends when this value is reached. When value is larger than input file, other values are filled with zeros
<start_point> in which line of the input file parsin is started
<radix_out> describes in which format the output is written
<radix_in> describes in which format the input is written
{division} Optional parameter, which describes in which value the input is divided to generate the output, works only with integer input .

Step 5:

Copy generated images and mif files to corresponding directories, for example ../MEM_IMAGES/B5/ so the stimulus generator can find the files in order to simulate the behavior of the system or run it in the FPGA board.

There are complete example scripts for the Steps 2-5 in <PATH_EXTRACTED>/GH09/python/example_scripts/. The usage of the python scripts can be studied with the provided example scripts or the example scripts can be used to generate memory images and mif files for the example XML files, which are generated with GH09-tool with default values.

In addition to these steps, a constant table for key input in case B.3 needs to be generated and separated io direction package for the case B.1. This is done with following command:

generate_vhdl_pkg.py <FOLDER> <FILE> <NAME_OF_PACKAGE> <NAME_OF_CONSTANT> <LENGTH>

<FOLDER> : folder where key image generated by memory image generator is located(usually ./)
<FILE> : file name of the generated key input VHDL package.
<NAME_OF_PACKAGE> name of the package that generator create.In the case of B.3 key_pkg and in the case of B.1. io_pkg
<NAME_OF_CONSTANT> name of the constant inside the package. In the case of B.3 KEY_IN_CONST and inthe case of B.1 IO_DIR_CONST
<LENGTH> length of the constant. In the case of B.3. 128 and in the case of B.1 16.

Step 6:

Compile Stimulus vector generator and simulate it or synthesize the stimulus vector generator for the FPGA-board.

Files needed to compile (for simulation):

<root>/VHDL/Case/TB_<case-number>/tb_<case_number>_fpga.vhdl
<root>/VHDL/Case/TB_<case-number>/<case_number>_output.vhdl (except cases B.1 and B.2)
<root>/VHDL/Case/TB_<case-number>/tb_toplevel.vhdl
<root>/VHDL/Case/TB_common/pll_tb.vhd
<root>/VHDL/Case/TB_common/write_hex.vhdl
<root>/VHDL/Case/TB_common/write_hex.vhdl
<root>/VHDL/Case/TB_common/lcd/*.vhdl
<root>/VHDL/Case/TB_common/sim_memories/<CASE>/*.vhd (memories assume that simulation directory is three levels upper compared to the MEM_IMAGE directory)

For the caseB.3 :
Key_pkg.vhdl from where is has been copied after generation.

For the case B.1:
io_pkg.vhdl from where is has been copied after generation.

In FPGA-simulation, memory models have to be replaced with suitable memories provided by FPGA-board, and there are already memory models for Altera boards, which are located in:

<root>/VHDL/Case/TB_common/altera/<CASE>/*.vhd

Also the phase-locked loop (PLL) needs to be changed corresponding the used board, for Altera DE2-board there is a complete PLL for generation of 100 MHz and 10 MHz clocks located in:

<root>/VHDL/Case/TB_common/pll.vhd

The clock signal of the system is created at PLL, so when changing the tick-rate of the measurement remember to change the PLL.

The Stimulus vector generator is made for the Altera DE2 board, thus it may need modifications for other target boards.

Device-Under-Test

Following steps are needed to configure the DUT for FPGA

Step 7:

Map the pins of the stimulus vector generator to corresponding locations. (GPIO_0 is the input stimulus to the DUT and GPIO_1 is the output of the DUT). Compile and program the FPGA board.

Step 8:

Connect Stimulus vector generator and DUT

Step 9:

Start the stimulus vector generator

Example designs

In the following, the files used in each example design are listed.

B1

<root>/VHDL/Case/B1/B1_top.vhd
<root>/VHDL/Case/B1/B1.vhd

B2

<root>/VHDL/Case/B2/B2_top.vhd
<root>/VHDL/Case/B2/B2.vhd

B3

<root>/VHDL/Case/B3/b3_top.vhdl
<root>/VHDL/Case/B3/aes/*.vhdl

B4

<root>/VHDL/Case/B4/b4_top.vhdl
<root>/VHDL/Case/B4/b4_table.vhdl
<root>/VHDL/Case/B4/b4_mem.vhdl

B5

<root>/VHDL/Case/B5/b5_top.vhdl
<root>/VHDL/Case/B5/b5.vhdl

B6

<root>/VHDL/Case/B6/b6_top.vhdl
<root>/VHDL/Case/B6/mem_addr_gen.vhdl