Politecnico di Torino

III Facoltà di Ingegneria

Laboratory 3
RISCV Processor

Master degree in Electronic Engineering

Authors: Group 21

Dilillo Nicola S284963
Moncalvo Stefano S290315
Carrano Lorenzo S281565

GitHub Repository

Contents

CHAPTER 1

Introduction

During the verification steps of HDL design different testbenches have been used to check the correctness of implementation. Each of them has been manually modified to verify the correct behavior of the design under different inputs, and no automatic tools had been used in order to automatize this process.

The UVM is the latest technique used to create modular testbenches. It's a free framework based on System Verilog and exploits the Object-Oriented paradigm. This paradigm entails the separation of concerns while building the testbench architecture and the specific test that can be performed: multiple tests can be applied to the same architecture just by modifying the input stimuli sequence (and constraints) and by selecting different Design Under Test (DUT) properties to be observed. Alternatively, the same tests can be applied to different testbench architectures.

In this laboratory the generation of different input has been random and the right output to compare has been elaborated based on a reference model, implemented through the UVM class, properly created according to the DUT. In this way the only operation to performed in order to change the testbench are:

- choose the proper DUT;
- select a range for input values;
- modify the model reference and the FSM to follow the design behavior.

CHAPTER 2

Adder

The first step has been to test the correctness of the behavior of the System Verilog description of an adder by using QuestaSim. In order to evaluate the performance, in terms of coherence with the expected behavior, it is needed to check the amount of matches and mismatches. For each input sequence, it was observed that the output of the adder description to be tested and the output of the gold model were coincidental, thus we can conclude that the overall behavior of the described adder is correct.

2.1 Changing the parallelism

Changing the parallelism means to modify both the interface of the unit under test and size of each input/output sequence. Even in this case, no mismatches have been detected by comparison with the golden model.

2.2 Constraints Definition

It is possible to define some constraints over the randomly generated input sequences employed in the test, for example using the inside operator, to define a range of values for one of the inputs of the adder. Even in this case, no mismatches have been observed.

2.3 Incorrect Gold Model

At this point, a golden model with an incorrect behavior has been employed for the test, thus in this case it is expected to observe mismatches inside each record of the transcript. For each mismatch, a warning is signaled in the transcript, and an error is reported inside the final summary.

2.4 Forcing Overflow Condition

At this step the behavior of the unit under test in case of overflow has been tested. In order to do that, it has been necessary to set proper constraints over the input sequences. It has been possible to observe that the adder correctly handles overflow conditions.

CHAPTER 3

Integer Multiplier

In this section the MBE-Dadda tree multiplier, from lab 2, has been tested.

3.1 Compilation

Now, in order to compile and simulate the files, that are no more written in system verilog, a dedicated bash file has been written to execute this task automatically.

Listing 3.1: sim/sim.sh

```
source /software/scripts/init_questa10.7c

rm -rf work
vlib work

vcom ../src/fpuvhdl/adder/*
vcom ../src/fpuvhdl/common/*
vcom ../src/fpuvhdl/multiplier/*
vcom ../src/MBE/*
vlog -sv ../tb/top.sv

vsim top
```

3.2 Change interface

The first things that have to be changed are the interface. For the input one there is no problem, both are still on 32 bits, while the output has the double bits and must be changed from 32 to 64 bits.

```
interface dut_if(input clk, rst);
    logic [31:0] A, B;
    logic [63:0] data;
    logic valid, ready;

    modport port_in (input clk, rst, A, B, valid, output ready);
    modport port_out (input clk, rst, output valid, data, ready);
endinterface
```

3.3 Change reference model

The last things that must changed is the reference model. Now the executed operation is a multiplication and no more addition and the reference must be changed properly.

Listing 3.2: Inside refmode.sv

```
...

forever begin
    in.get(tr_in);
    tr_out.data = tr_in.A * tr_in.B;
    $display("refmod: input A = \%d, input B = \%d, output OUT = \%d", tr_in.A, tr_in
    $display("refmod: input A = \%b, input B = \%b, output OUT = \%b", tr_in.A, tr_in
    out.put(tr_out);
end

...
```

3.4 Testbench Run

During the testbench run no mismatches occurred, as shown in the transcript file.

MBE: input A = 9177, input B = 9725, output OUT = 89246325
MBE: input A = 00000000000000000010001111011001, input B = 00000000000000000010010111111101,
output OUT = 00000000000000000000000000000000000001010101000111001010011101010
refmod: input A = 9177, input B = 9725, output OUT = 89246325
refmod: input A = 00000000000000000010001111011001, input B = 00000000000000000010010111111101,
output OUT = 00000101010100011100101001110101
UVM_INFO @ 45: uvm_test_top.env_h.comp [Comparator Match]
MBE: input A = 9536, input B = 1307, output OUT = 12463552
MBE: input A = 00000000000000000010010101000000, input B = 00000000000000000000010100011011,
output OUT = 00000000000000000000000000000000000001011111100010110111000000
refmod: input A = 9536, input B = 1307, output OUT = 12463552
refmod: input A = 00000000000000000010010101000000, input B = 00000000000000000000010100011011,
output OUT = 00000001011111100010110111000000
UVM_INFO @ 75: uvm_test_top.env_h.comp [Comparator Match]
MBE: input A = 3035, input B = 2330, output OUT = 7071550
MBE: input A = 00000000000000000000101111011011, input B = 00000000000000000000100100011010,
output OUT = 00000000000000000000000000000000000001101011111001110011110
refmod: input A = 3035, input B = 2330, output OUT = 7071550
refmod: input A = 00000000000000000000101111011011, input B = 00000000000000000000100100011010,
output OUT = 00000000110101111100111001111110
UVM_INFO @ 105: uvm_test_top.env_h.comp [Comparator Match]
MBE: input A = 2880, input B = 3695, output OUT = 10641600
MBE: input A = 00000000000000000000101101000000, input B = 00000000000000000000111001101111,
output OUT = 00000000000000000000000000000000000001010001001100000011000000
refmod: input A = 2880, input B = 3695, output OUT = 10641600
refmod: input A = 00000000000000000000101101000000, input B = 00000000000000000000111001101111,
output OUT = 00000000101000100110000011000000

```
# UVM_INFO @ 135: uvm_test_top.env_h.comp [Comparator Match]
```

CHAPTER 4

Floating Point Multiplier

4.1 Testbench Modifications

The floating point multiplier, in which the MBE multiplier tested in the previous section is used for the multiplication of the significands, is a pipelined architecture. This means that the results of the multiplications are ready some clock cycles after the generation of the inputs. In order to compensate for the added latency, the verification has to be delayed with respect to the generation of the inputs. After an initial latency of six clock cycles necessary to perform the first multiplication, the testbench handles the multiplier in a pipelined way, performing and verifying one operation every three clock cycles. This is possible by saving the generated inputs in a chain of two registers; since the FSM is composed of three states, this is equivalent to delaying the inputs of six clock cycles. When a new result is ready, it is confronted with the correct one computed using the inputs present in the second register of the chain. The FSM is composed of three states:

- Initial: all control signals are initialized, then the FSM moves on to the WAIT state.
- Wait: the inputs are saved in order to be delayed, and the results of the multiplications are displayed. Two flags are set, in_inter.ready = 0 to stop the generation of new inputs, and out_inter.valid = 1 to signal the presence of a new valid output to confront with the reference one.
- Send: when the results have been compared, in_inter.ready is set to 1 to generate a new input and out_inter.valid = 0.

4.2 Input Generation

Since it is not possible to generate random single-precision floating point numbers, random 32-bits sequences are generated. Using the functions $shortrealtobits() and $bitstoshortreal() it is possible to convert from floating point numbers to their binary representation and viceversa. To make the results more readable, some constraints have been applied to the input generation: in particular, the bits from 30 to 23, which represent the exponent of the floating point number, are forced to be equal to 128 for both generated inputs.

4.3 Testbench Run

During the testbench run no mismatches occurred, as shown in the transcript file.

FPMUL: input A = 3.210777, input B = -2.400374
FPMUL: input A = 01000000010011010111110101011111, input B = 11000000000110011001111110111001
FPMUL: output OUT = -7.707065
FPMUL: output OUT = 11000000111101101010000001000110
refmod: input A = 3.210777, input B = -2.400374, output OUT = -7.707065
refmod: input A = 01000000010011010111110101011111, input B = 01000000010011010111110101011111, output OUT = 11000000111101101010000001000110
UVM_INFO @ 3015: uvm_test_top.env_h.comp [Comparator Match]
FPMUL: input A = 3.084183, input B = -3.686763
FPMUL: input A = 01000000010001010110001100111111, input B = 11000000011010111111001111101110
FPMUL: output OUT = -11.370651
FPMUL: output OUT = 11000001001101011110111000110000
refmod: input A = 3.084183, input B = -3.686763, output OUT = -11.370651
refmod: input A = 01000000010001010110001100111111, input B = 01000000010001010110001100111111, output OUT = 11000001001101011110111000110000
UVM_INFO @ 3045: uvm_test_top.env_h.comp [Comparator Match]
UVM_INFO verilog_src/uvm-1.1d/src/base/uvm_objection.svh(1267) @ 3045: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO ../tb/env.sv(42) @ 3045: uvm_test_top.env_h [env] Reporting matched 101
#
— UVM Report Summary —
#
** Report counts by severity
UVM_INFO : 106
UVM_WARNING : 0
UVM_ERROR : 0
UVM_FATAL : 0
** Report counts by id
[Comparator Match] 101
[Questa UVM] 2
[RNTST] 1
[TEST_DONE] 1
[env] 1