Politecnico di Torino

III Facoltà di Ingegneria

Multiplier Verification with UVM

Master degree in Electriconic Engineering

Integrated Systems Architectures

Authors: Group 34

github repository link :

https://github.com/alessionaclerio22/Integrated-Systems-Architecture-Labs

Simone Di Blasi, Stefano Floridia, Alessio Naclerio

Contents

CHAPTER 1

Simple Adder UVM Test

The aim of this first part of the laboratory is to become more familiar with the *UVM* environment. For this purpose, the test of a simple behavioral 32-bit adder, implemented in *adder.sv*, has been carried out exploiting the given *.sv* files. As a matter of fact, these files allow the proper setup of a *UVM* based testbench. The working directory for this part of the laboratory is *cap4*, which has been organized in three main sub-folders, *src*, *tb* and *sim*, respectively containing the files related to the DUT, the *UVM* testbench and the simulation.

As requested by the assignment, the *packet_in.sv* file has been properly modified in order to accomodate the required constraints on both the adder operands. For this purpose, the *inside* statement has been exploited. The operand A must be in the range 100 to 1000, while the operand B must be lower than 10 times A, which leads to the following constraint:

Listing 1.1: Required operands constraint using *inside* operator

```
constraint range { A inside {[100:1000]}; B inside {[0:10*A]};}
```

Then, the simulation has been carried out in the *sim* folder using the commands suggested in the assignment, which are *vlog -sv ../tb/top.sv*, *vsim top* and *run 4 us*. 1.2 shows an example of match between the DUT and the reference module values.

Listing 1.2: Example of match between DUT and refmod values for constrained adder input simulation

```
# adder: input A = 586, input B = 3063, output OUT = 3649
# adder: input A = 00000000000000000000001001001010,
input B = 00000000000000000000101111110111,
output OUT = 00000000000000000000111001000001
# refmod: input A = 586, input B = 3063, output OUT = 3649
# refmod: input A = 00000000000000000000001001001010,
input B = 00000000000000000000101111110111,
output OUT = 00000000000000000000111001000001
```

As last step, the *refmod.sv* file has been modified substituting the "+" operator with the "-" one. This change allowed mismatches between the results of the DUT (*adder.sv*) and the ones produced by the reference model (*refmod.sv*) to arise. In 1.3, the *UVM* summary generated for this simulation is reported. 1.4 shows an example of mismatch.

Listing 1.3: UVM Report Summary for modified refmod simulation

```
# --- UVM Report Summary ---
#
# ** Report counts by severity
# UVM_INFO :  207
# UVM_WARNING :  101
# UVM_ERROR :    1
# UVM_FATAL :    0
# ** Report counts by id
# [Comparator Mismatch]    101
# [MISCMP]    202
# [Questa UVM]      2
# [RNTST]      1
# [TEST_DONE]      1
# [env]      2
```

Listing 1.4: Example of mismatch between DUT and refmod

```
# adder: input A = 586, input B = 3063, output OUT = 3649
# adder: input A = 00000000000000000000001001001010,
input B = 00000000000000000000101111110111,
output OUT = 00000000000000000000111001000001
# refmod: input A = 586, input B = 3063, output OUT = -2477
# refmod: input A = 00000000000000000000001001001010,
input B = 00000000000000000000101111110111,
output OUT = 11111111111111111111011001010011
```

CHAPTER 2

MBE Dadda-Tree Multiplier UVM Test

As requested by the second point of the assignment, the *MBE Dadda-Tree Multiplier* for unsigned numbers developed during laboratory 2 has been tested. The working directory for this section is *cap4_1*, which has been organized in three sub-folders. These are *src*, *tb* and *sim*, respectively containing the MBE Multiplier design along with the the DUT-related files provided by the assignment, the *UVM* testbench and the files used during simulation.

In order to properly test the current DUT, the *UVM* testbench has been adapted by properly modifying some of the given files, in particular *DUT.sv*, *dut_if.sv*, *refmod.sv*, *packet_in.sv* and *packet_out.sv*. The *DUT.sv* has been changed by substituting the adder with the MBE component instance, while the interface in *dut_if.sv* has been properly adapted to support the DUT inputs and output width. As regards the reference model, the addition operator in *refmod.sv* has been replaced with the multiplication one in order to emulate the DUT behavior. Moreover, the input operands generated in *packet_in.sv* have been constrained using the *inside* statement in order to be in the range 0 to 10000. Finally, in order to support the 64-bit-wide DUT output, *packet_out.sv* has been properly modified.

In order to launch the simulation, the compilation of the *VHDL* files composing the MBE Multiplier design has been run before issuing the commands given in the assignment, which are *vlog -sv ../tb/top.sv* and *vsim top*. 2.1 shows an example of match between the DUT and the refmod values.

Listing 2.1: Example of match between DUT and refmod for MBE Dadda-Tree Multiplier simulation

```
# dadda_mul: input A = 9177, input B = 9725, output OUT = 89246325
# dadda_mul: input A = 00000000000000000010001111011001,
input B = 00000000000000000010010111111101,
output OUT = 00000000000000000000000000000000000010101010001110010101001110101
# refmod: input A = 9177, input B = 9725, output OUT = 89246325
# refmod: input A = 00000000000000000010001111011001,
input B = 00000000000000000010010111111101,
output OUT = 00000000000000000000000000000000000010101010001110010101001110101
```

CHAPTER 3

Floating-point Multiplier UVM Test

As requested by the last point of the assignment, the *Floating-point Multiplier* using the *MBE Dadda-Tree Multiplier* in its stage 2 has been tested. For this purpose, the *UVM* testbench has been adapted to this design. The working directory for this last part is called *cap4_2*, which is internally divided in the sub-folders *src*, *tb* and *sim*. The list of files contained in each of these folders is equal to the one mentioned in the previous section, even though *src* also contains the Floating-Point Multiplier design. The main files that have been modified are:

- *dut_if.sv* has been modified to properly support the parallelism of the multiplier inputs and output;
- *DUT.sv* has been changed in order to take into account the pipelined architecture of the floating-point multiplier. For this purpose, the FSM wrapper processing the valid-ready signals has been extended in order to support a trivial 5-stage pipeline. This is crucial in order to let the DUT compute the right result out of the received inputs. The FSM evolves through the following states:
  - INITIAL: it is the initial state of the FSM;
  - WAIT_1: it is the state devoted to the handling of the inputs which, if valid, are printed and delivered to the DUT;
  - from WAIT_2 to WAIT_4: these are states used in order to emulate the pipeline stages;
  - WAIT_5: it is the state in which the output of the DUT is printed and its validity is asserted;
  - SEND: it is the final state in which, if the output of the DUT is ready, the DUT is able to receive new operands and restart from WAIT_1.
- *packet_in.sv*: since *SV* does not support the creation of random floating-point numbers, the A and B inputs for the DUT has been declared as *rand bit[31:0]*. Moreover, a constraint has been employed in order to set the exponent of the number to a fixed value. Hence, bits from 23 to 30 of A and B, the ones corresponding to the exponent, have been constrained to different values for each simulation that has been performed;
- *packet_out.sv*: it has been modified to properly accomodate the parallelism of the DUT output;
- *refmod.sv*: it is the reference module performing the multiplication on the two DUT operands. Since these values are declared as *bit[31:0]*, a cast to *shortreal* is needed in order to compute the right result. The function *$bitstoshortreal()* has been exploited for this purpose. Once

the *shortreal* result has been calculated, it has been converted to *bit* again, using the function *$shortrealtobits()*. Moreover, a display of the inputs and the output is performed.

Before issuing the given commands, all the Floating-Point Multiplier files have been compiled. Then, more than one simulation has been carried out in the *sim* folder, each with a different constraint on the floating-point number exponent. Some examples of the results obtained in the four simulations are showed in 3.1, 3.2, 3.3 and 3.4.

Listing 3.1: Floating-point Multiplier simulation with exponent set to 128

```
# refmod: input A = 11000000000001111010000001111000,
input B = 01000000010101110101101100111101
# refmod: input A = -2.119169, input B = 3.364944
# refmod: output OUT = 11000000111001000011000000110110, out_sr : -7.130885
# fpmul: input A = 11000000000001111010000001111000,
input B = 01000000010101110101101100111101
# fpmul: input A = -2.119169, input B = 3.364944
# fpmul: output OUT = 11000000111001000011000000110110, OUT_f = -7.130885
```

Listing 3.2: Floating-point Multiplier simulation with exponent set to 130

```
# refmod: input A = 11000001000001111010000001111000,
input B = 01000001010101110101101100111101
# refmod: input A = -8.476677, input B = 13.459775
# refmod: output OUT = 11000010111001000011000000110110, out_sr : -114.094164
# fpmul: input A = 11000001000001111010000001111000,
input B = 01000001010101110101101100111101
# fpmul: input A = -8.476677, input B = 13.459775
# fpmul: output OUT = 11000010111001000011000000110110, OUT_f = -114.094162
```

Listing 3.3: Floating-point Multiplier simulation with exponent set to 132

```
# refmod: input A = 11000010000001111010000001111000,
input B = 01000010010101110101101100111101
# refmod: input A = -33.906708, input B = 53.839100
# refmod: output OUT = 11000100111001000011000000110110, out_sr : -1825.506626
# fpmul: input A = 11000010000001111010000001111000,
input B = 01000010010101110101101100111101
# fpmul: input A = -33.906708, input B = 53.839100
# fpmul: output OUT = 11000100111001000011000000110110, OUT_f = -1825.506592
```

Listing 3.4: Floating-point Multiplier simulation with exponent set to 142

```
# refmod: input A = 11000111000001111010000001111000,
input B = 01000111010101110101101100111101
# refmod: input A = -34720.468750, input B = 55131.238281
# refmod: output OUT = 11001110111001000011000000110110, out_sr : -1914182435.892944
# fpmul: input A = 11000111000001111010000001111000,
input B = 01000111010101110101101100111101
# fpmul: input A = -34720.468750, input B = 55131.238281
# fpmul: output OUT = 11001110111001000011000000110110, OUT_f = -1914182400.000000
```