Politecnico di Torino

III Facoltà di Ingegneria

Lab 4 Report
Integrated Systems Architecture

Master degree in Computer Engineering

Authors: ISA36

Nicole Dai Prà s274501, Leonardo Izzi s278564

March 29, 2021

Many thanks to Prof. Mariagrazia Graziano for providing us with this template.

Contents

CHAPTER 1

Introduction

In this laboratory session we have learned the principles behind UVM and how to test both combinational and sequential circuits. As required, there is a GitHub repository available at the following link: `https://github.com/leoizzi/isa_labs/tree/main/lab4`.

There are three top folders, which are:

- `adder`, where the test of the given adder has been performed.
- `fpmult`, where the test of the whole floating point multiplier has been performed.
- `mult`, where the test of the MBE-Dadda tree multiplier has been performed.

Each of these folders has the same structure, which is:

- `sim`, the folder containing all the transcripts produced by QuestaSim.
- `src`, the folder containing the files related to the DUT.
- `tb`, the folder containing all the files related to the verification.

CHAPTER 2

Testing

2.1 Adder

To gain confidence with the UVM environment, we have run the simulation with the provided files first. The resulting transcript is `adder_transcript`, that shows the correctness of the circuit.

Then, we have modified `packet_in.sv` to support input constraints. We have used the suggested ranges for the operands, which are $100 \leq A \leq 1000$ and $B < 10 * A$. The transcript related to this simulation is called `constraint_transcript`. This lesson has been very useful for the testing of the multipliers.

As a final step, we have changed the reference model described in `refmod.sv` to verify what happens when QuestaSim detects a mismatch in the result. Specifically, we have turned the golden model adder into a subtractor. The results of this simulation are stored in `mismatch_transcript`. As expected, there are only mismatches. This can be visualized at the end of the transcript, where the UVM Report Summary is written.

2.2 MBE-Dadda tree multiplier

To test the multiplier we used the adder infrastructure as a starting point and then we changed few files. The first step was to add all the VHDL files developed for the lab2, which are `ha.vhd`, `fa.vhd`, `dadda.vhd`, `mbe.vhd` and `mult.vhd`.

The first modification has been done in `DUT.sv`. Here, we have replaced the adder's instantiation with the multiplier's one. This change is not enough though, because the result of the adder fits in 32 bits, while the multiplier's one fits in 64. Hence, we changed the size of `data` in `dut_if.sv` from 32 bits to 64 bits. This is all what concerns the changes in `src/`.

In the `tb/` folder, we have changed the reference model from an adder to a multiplier. Then, we have added two constraints in `packet_in.sv`, which are $A \geq 0$ and $B \geq 0$. This step was mandatory, because the multiplier works only on unsigned numbers. Moreover, we have changed the output's data type of `packet_out.sv` from an integer to a longint.

Since the multiplier is written in VHDL while the testbench is written in System Verilog, we have solved the integration by compiling first our multiplier and then the testbench. The result of the simulation is stored in `transcript_mult`. As expected, there were no mismatches.

2.3 Floating point multiplier

We have used the same strategy of the integer multiplier to have a baseline test infrastructure. We have then imported in `src/` all the required files. Since the FP multiplier is a sequential circuit, it was not enough to change the circuit instantiation in `DUT.sv`, but we needed to change also the FSM used by the testbench. In particular, we have declared an integer variable called `count` to count in the `WAIT` state the iterations. We remain in the `WAIT` state until `count == 6`, because after 5 cycles the circuit has produced the final output. We've also sampled the input values when `count == 0` due to a print mismatch (which had no influences on the results).

In `packet_in.sv` we generate the random numbers as integers for convenience, but we have added few constraints to avoid mismatches due to the different handling of NaN, infinities and denormal numbers. The checks are:

- $(((A\&32'h7f800000) >> 23) + ((B\&32'h7f800000) >> 23) - 127) > 0$
- $(((A\&32'h7f800000) >> 23) + ((B\&32'h7f800000) >> 23) - 127) < 253$
- $((A\&32'h7f800000) >> 23) > 0$
- $((A\&32'h7f800000) >> 23) < 255$
- $((B\&32'h7f800000) >> 23) > 0$
- $((B\&32'h7f800000) >> 23) < 255$

The bit shifts and AND operations are used to extract the exponents from A and B. In fact, the numbers passed to the reference model and the DUT are nothing more than the binary representations of these number. The first two checks are used to avoid denormal numbers and overflows in the product. In the second check we have considered the fact that the exponent may increase due to the rounding and normalizations. The other checks are used to avoid special numbers in the inputs.

In `refmod.sv` we convert the integers into shortreal by using the directive `$bitstoshortreal`, then we perform the floating point multiplication, and finally we store the final result as an integer through the `shortrealtobits` directive.

The result of the simulation is stored in `transcript`.

In `sim/` a script that can be used to launch the simulation in QuestaSim is also contained.