CHAPTER 1

Parametric VHDL and synthesis

In this lab session you will recall some simple VHDL structures and will sinthesize them so that you will be able to trace and constraint their performance. The files for this lab are availabe from: **/home/repository/ms/cap1/** Generate a new directory **cap1** in your home. Create in it other two directories: **vhdlsim** and **syn** Copy all the files in directory vhdlsim: remember that the syntax is

prompt> **cp /home/repository/ms/cap1/* .**

The lab is organized so that for each block you first simulate it and then you synthesize it to check the result of your code. When you switch from a simulation phase to a synthesis, copy the vhdl entity files from the **vhdlsim** to the **syn** directory as well.

It is suggested to use two different work spaces in your system (click on another square in the bottom right rectangle of your display), one for managing the simulations and the other for the synthesis. Before simulating remember (see setup lab instructions) to set the simulator environment variables (setmentor) and to create a work library (vlib work) in the **vhdlsim** directory.
In the same way, IN ANOTHER TERMINAL (in another work space), set the synopsys environment variables (setsynopsys) and create the work directory (mkdir work) in the **syn** directory. Copy also the file .synopsys_dc.setup in there.

1.1 Parametric VHDL

You have seen in the setup lab an example of a simple mux. Now it's up you. You are expected to finish this task in a few minutes. Your first simple task is to parametrize your mux on a generic NBIT parallel version. Copy the test bench *tb_mux21_generic.vhd* and use it as a basis for organizing your structural and behavioral block. Simulate it using the given inputs and check the behavior.

Summary of what is requested

Generic Mux VHDL netlist.

1.2 Basic memory elements

Now copy the files **fd.vhd** and **tb_fd.vhd** which describe a **D-TYPE FLIPFLOP**. You have two different architectures described in *fd.vhd* with a difference in RESET semantics. Consequently two different instances are mapped in your test bench file.

Now transform this block in a register with a parametric definition and simulate it. Use the test bench used for the generic MUX as a starting point. This task again should be done very rapidly.

Summary of what is requested

Edge triggered register VHDL netlist and test bench.

1.3 Synthesis of parametric structures

You are going to synthesise blocks that you have already used in previous sections. You will use a standard cell flow based on one of the most powerful synthesis tool diffused in industry.

1.3.1 Synthesis of the generic MUX

Copy from your VHDLSIM directory all your entities vhdl files. Remember that test benches are used only for simulations and are note synthesized. Now follow the same step as in the setup lab (environment variables, design_vision, analyze, elaborate, compile). Pay attention: you shoud force during the synthesis phase the number of bits you want, otherwise the default value will be used. Repeat the same steps for the other generic MUX architectures and analyze the different results.

Summary of what is requested

Post synthesis VHDL netlists of the parametric MUX.

1.3.2 Synthesis of the sequential ports

Now perform the same steps as before for both your latch and the flip flop based registers. Analyze the results and the different implementations for the synchronous and asynchronous reset.

Summary of what is requested

Post synthesis VHDL netlists of registers.

1.4 A basic block for your next exercises

1.4.1 A starting point: a given RCA

Copy and read carefully the files **fa.vhd**, **rca.vhd**, **lfsr.vhd** and **tb_rca.vhd** (structural RCA, LFSR and test bench). Notice the **generic map** that is used in this case to assign two different delays, one for the sum **S** bit (**DRCAS**) and one for the carry **C** bit (**DRCAC**). Notice also that a LFSR (Linear Feedback Shift Register) is used for the input random generation (for the moment neglect the internal description of the LFSR).
Notice that the RCA is defined by means of two architectures. The first is structural and uses a component that is a given behavioral Full-Adder: Remember the generic parameter definition and usage. The second architecture is a behavioral one and note that something is missing in it. What can you do to solve the problem?

In the test bench now, notice the use of the two RCA instances. Compile now and run the simulation for a few (50 ns) and check that the sum is correct. For this check you can change the signal format in the waveform viewer: select for example the wave "a" and select from the menu: *Format→Radix→Unsigned*. Which is the difference among the three output "s1", "s2", "s3"? Zoom the waves in the range 24.5ns - 24.6ns: what's happening to the three outputs?

Generating a completely parametric RCA

Use the previous RCA for generalizing the number of bits, following the multiplexer and register implementations. Remember that you have to generalize both the structural and behavioral implementation. DO NOT parametrize the LFSR.

Summary of what is requested

RCA vhdl parametric netlist.

1.4.2 Synthesis

Again copy file RCA.vhd in the **syn** directory. Execute the same steps as before. Synthesize it and analyze the timing performance and the critical path. Rememebr that the delays must be commented as the synthesizer does not support the time type.

Summary of what is requested

Adder VHDL netlist, adder post synthesis VHDL netlist, area and timing report.

1.5 Accumulator

You are going to create and synthesize an accumulator both structural and behavioral using the structures you built and simulated in previous labs.
The expected execution time for this first exercise is around 40 min.

1.5.1 Structural accumulator

The structure is in figure 1.1. You have a simple test bench *ACC_tb.vhd* with an ACC component instance. Use the *ACC.vhd* entity for describing your structural architecture. Use the MUX, REG and ADDER you described in previous exercises, and if necessary adapt them to your needs. Choose a parametric configuration using generics. Modify the previous bench instantiating the structural architecture and defining 64 as bit number and simulate it (the use of an enable signal is optional).

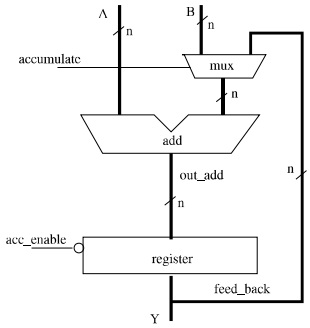

Figure 1.1:

For an easy compilation phase you can use the file **compile.bash** which is simply a script containing the instructions for compiling all the files of this exercise (change the the file names in this script accordingly to your ones). You can run it by typing:

prompt> **./compile.bash**

Summary of what is requested

VHDL netlist of the whole structure, meaningful waveform.

1.5.2 Behavioral accumulator

Add a second architecture with a behavioral description. Use three concurrent processes to describe the three component functions and simulate it (the use of and enable signal is optional). Now instanciate in the test bench a second 64 bit accumulator and simulate it.

Summary of what is requested

VHDL parametric behavioral netlist.

1.5.3 Synthesis

Exactly as in previous class copy the entities vhdl file (not test benches) in the **syn** directory. Go in there and use **design_vision** to synthesize both the structural and the behavioral accumulators. Phases are *Analyze*, *Elaborate* and *Compile*). When a structure has a hierarchical level, then you must analyze files in the correct bottom up order. Furthermore, if you remember, when a structure has more than an architecture you can choose during the elaboration session which configuration you want. Analyze the difference between the two architectures by exploring the hierarchy down to the leaf cells. In all the compiled versions generate the synthesized vhdl netlist.

Summary of what is requested

Synthesized VHDL netlists of the accumulator block in both the architectural implementations.

1.6 ALU: Behavioral

This first exercise requests to generate an Arithmetic-Logic Unit structure in a behavioral way and to synthesize it. The expected execution time for this first exercise is less than one hour.

1.6.1 ALU description

You are given the entity of an ALU (*alu.vhd*) which should have the following capabilities:

- all operations are combinational
- ADD/SUB on N bits operands
- MULTIPLY on N/2 bits operands (Least Significant Part of), result on N bits.
- bitwise AND, OR, XOR on 32 bits operands.
- Logical Shift Left, Right, Rotate Left, Rotate Right.

Note the use of the construct CASE which switches among different operation types: FUNC is of type TYPE_OP, and the possibilities are enumerated in the suggested structure. This type must be defined in a new package in file *type_alu.vhd*. Thus: define this type and fill the operations as suggested. Use the *tb_alu.vhd* test bench and simulate your ALU.

Summary of what is requested

ALU behavioral VHDL netlist, ALU test bench.

1.6.2 ALU synthesis

Copy your alu.vhd file in the syn directory and synthesize it. It is suggested that you use a small number of bits (e.g. 4) to understand the synthesis result. Furthermore, you can also synthesize only one function step by step by commentig the others, so that you can appreciate how each operation is synthesized. At the end of your analysis run a final synthesis for all the functions. Report timing and area. Generate the synthesized netlist and the verilog one.

Summary of what is requested

Synthesis script, timing and area report files.