Microelectronic Systems

Lab 1

Parametric VHDL and synthesis

March 8, 2023

1 Introduction

During this laboratory session you will recall some simple VHDL structures and will synthesize them to trace and constrain their performance. All the needed files are available inside the following folder:

/home/repository/ms/cap1/

Before you proceed, generate the new directory **cap1** in your home workspace. Then, create inside it two sub-directories, named **vhdlsim** and **syn**. Copy all the files cited above into the **vhdlsim** folder through the Shell command:

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

The laboratory is organized so that for each block you first perform a simulation on it and then you synthesize it in order to verify the results of your code. In particular, when you switch from a simulation phase to a synthesis, copy the VHDL entity files from the **vhdlsim** directory to the **syn** one.

It is suggested to use two different work spaces in your virtual machine (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 Lab0 instructions) to set the simulator environment variables ***(setmentor)*** and to create a work library ***(vlib work)*** in the **vhdlsim** directory. In the same way but IN ANOTHER TERMINAL and IN ANOTHER WORK SPACE, set the Synopsys' environment variables ***(setsynopsys)*** and create the work directory ***(mkdir work)*** inside **syn** directory. In there, copy also the file **.synopsys_dc.setup**.

2 Parametric VHDL for a 2 to 1 Multiplexer

You have considered in Lab0 an example of a simple mux. So, you are expected to finish the following first task in a few minutes. In particular, you are required to parameterize your mux on a generic NBIT parallel version. In order to do that:

1. Copy the testbench file **tb_mux21_generic.vhd** and use the declaration of the mux component inside it as a basis for **both** a behavioral and a structural description of your parametric multiplexer.
2. Simulate it, in both the implementations, using the given inputs and check its behaviour.

Summary of what is requested

Generic structural and behavioral Mux VHDL netlist.

Hints&Tips: What is a netlist? It is a description of your electronic circuit, so it contains all the components, the signals and the nodes they are connected to.

3 Basic memory elements

Let's now consider the definition of memory registers:

1. Copy the files **fd.vhd**, which describes a **D-TYPE FLIP-FLOP**, and the related test bench **tb_fd.vhd**.

As you can observe, inside the *fd.vhd* file there are two distinct architectures, which differ for the RESET semantics *(synchronous vs. asynchronous)*. Consequently, two different instances are mapped in the test bench file.

Hence:

2. Transform these blocks in two registers with a parametric definition for their parallelism and simulate them. As a starting point, consider the test bench used before for the simulation of the generic Mux.

Summary of what is requested

Edge triggered register VHDL netlist and test bench.

4 Synthesis of parametric structures

In this section, you are going to synthesise your previous RTL blocks. You will use a **standard cell flow** through the powerful Synopsys' Design Compiler.
Remember that for elaborate the parametric structure it is necessary to modify the command provided in the lab0, in the following way:
`elaborate <entity_name> -architecture <architecture_name> -library WORK -parameter <name_param=value>`,
where `name_param` is equal to the parameter name in the VHDL file and `value` is the assignment to the variable.

4.1 Synthesis of the generic MUXes

In order to synthesize your multiplexers, you have to:

1. Copy from your **vhdlsim** directory all your VHDL files describing your entities. Remember that test benches are only used for simulations and they can't be synthesized.
2. Follow the same steps detailed inside Lab0 for the definition of the *environment variables*, for launching *design_ vision* and for proceeding with the *analysis*, the *elaboration* and the *compilation* of the design.

You should force the number of bits you want in your design during the synthesis phase, otherwise the default value will be used.

3. Repeat the same steps for both the generic MUX architectures you have and analyze the different results.

Summary of what is requested

Post synthesis VHDL netlists of the parametric MUXes, generated by Synopsys' Design Compiler.

4.2 Synthesis of the sequential circuits

Now, it is the turn of the designed registers:

1. Perform the same steps as before for the two flip flop based registers, previously considered.
2. Analyze the deriving results.

Remember that delays are not synthesizable: remove them before executing the synthesis process.

Summary of what is requested

Post synthesis VHDL netlists of registers.

5 Characterization of a given Ripple Carry Adder

Here, you have to consider an arithmetic circuit, the RCA, first to simulate it and then to define its new parametric version by yourself.
Let's proceed with the characterization of the circuit:

1. Copy and read carefully the files **fa.vhd**, **rca.vhd**, **lfsr.vhd** and **tb_rca.vhd**, which contain the definitions of a structural RCA, a LFSR and a test bench respectively. Please, note that:
   - A LFSR (Linear Feedback Shift Register) is used for the random generation of the inputs. For the moment, do not consider the internal description of this block.
   - The **generic map** 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**).
   - The RCA is defined through two different implementations. The first one is structural and it uses a behavioral Full-Adder as a basic block. Remember the meaning of the generic parameters. The second one is completely behavioral instead. Note that something is missing in its definition:

Question 1
What can you do to solve the problem?

2. Now, notice the use of the two RCA instances inside the test bench. Compile and run the simulation for a small time interval *(50 ns)* and check if the results of the sum are correct. For this control, you can change the display format of the various signals in the waveform viewer. To do that, right-click for example on the wave "A" and then select: *Format→Radix→Unsigned*[1].

Question 2
Which is the difference among the three output "S1", "S2", "S3"?
Question 3
Zoom the waves in the range 24.5ns - 24.6ns: what's happening to the three outputs?

5.1 Generating a completely parametric RCA

Now, use the previous RCA for generalizing the parallelism of the data, following the same steps considered for the multiplexer and register implementations. Remember that you have to generalize both the structural and behavioral implementations and DO NOT parametrize the LFSR.

Summary of what is requested

VHDL parametric netlist of the structural and behavioral RCAs.

5.2 Synthesis

At this point, you have to synthesize the RCA circuit:

1. Copy the file **RCA.vhd** into the **syn** directory.

[1] You can access the same option from the top menu.

2. Execute the same steps for the synthesis, as before. Synthesize the circuit and analyze the timing performance and the critical path of both the structural and behavioral implementations.

Remember 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, and the derived area and timing reports.

6 Accumulator

Along this section, you are going to design and synthesize an accumulator in both a structural and a behavioral implementation, reusing the structures you built and simulated in the previous steps. The expected execution time for this exercise is *around 40 minutes*.

6.1 Structural accumulator

The architecture you have to implement is depicted in Figure 1. Among the provided files for this laboratory session, you can find a simple test bench, named **tb_acc.vhd**, with an ACC component instantiated. So:

1. Use the entity inside *tb_ACC.vhd*, as a reference for the definition of your own structural architecture.
2. Use the MUX, REG and ADDER you have described in the previous exercises as components for the design of your accumulator and, if necessary, adapt them to your needs.
3. Choose a parametric configuration using *generics*.
4. Modify the test bench instantiating the structural architecture, defining the data parallelism to 64 bits, and simulate the circuit (Note: the use of an enable signal is optional).

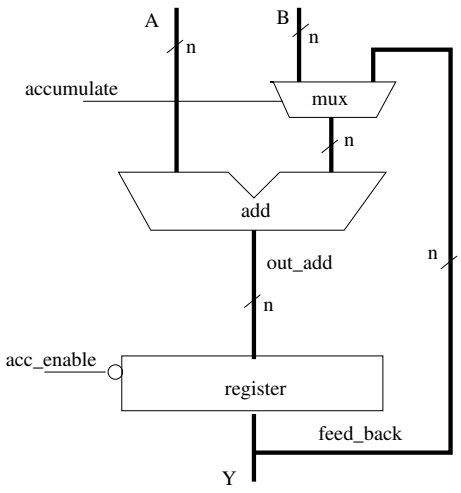

Figure 1:

For an easier and faster compilation phase you can use the file **compile.bash** which is a simple script containing the instructions for compiling all the needed files for this exercise. However, pay attention

and change the file names inside this script accordingly to your ones.
Then, you can run it by typing:

prompt> **./compile.bash**

Summary of what is requested

VHDL netlist of the accumulator structure and analysis of the meaningful waveforms.

6.2 Behavioral accumulator

It is the time to instantiate also a behavioral implementation of the accumulator:

1. Inside the same file of your structural design, define a second architecture with a behavioral description. Use three concurrent processes to describe the three component functions.
2. Simulate it (again, the use of an enable signal is optional).
3. Now instantiate inside the test bench a second 64 bit accumulator and simulate it.

Summary of what is requested

VHDL netlist of the parametric and behavioral-described accumulator.

6.3 Synthesis

The final step consists in the synthesis of the accumulator architectures you have designed:

1. Copy the VHDL entity files, with the exception of the test benches, into the **syn** directory.
2. Inside there, use **design_vision** to synthesize both the structural and the behavioral implementations of the accumulator. Just to remind you, the main steps to follow are: *Analyze*, *Elaborate* and *Compile*).

When a design is hierarchical, so it is defined through different components, you must analyze the corresponding files in the correct order: from bottom to top. Furthermore, when an entity owns more than an architecture, during the elaboration phase, you must choose which configuration will be processed by the synthesizer.

3. Analyze the differences between the two architectures by exploring the hierarchy, down to the leaf cells. For 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.

7 Arithmetic and Logic Unit: A behavioural implementation

For this last exercise, you are required to design and synthesize a behavioural *Arithmetic and Logic Unit (ALU)*. The latter has to operate on **unsigned numbers**.

7.1 ALU description

Among the provided files, you can find the entity of an ALU (**alu.vhd**). Starting form its structure, you have to define its implementation details by providing it the following capabilities:

- all operations are combinational;
- ADD/SUB on N bits operands;
- MULTIPLY on N/2 bits operands (the least significant part of the input operands), so as the result can be correctly represented on N bits;
- bitwise AND, OR, XOR on N bits operands;
- Logical Shifts: Left, Right, Rotate Left and Rotate Right. For this particular logic block, you are free to choose between the following possibilities:
  - Version A *(mandatory)*: the first operand is shifted by one position
  - Version B *(not mandatory)*: the first operand is shifted by a number of positions defined by the value of the second operand *(Barrel Shifter)*

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 inside the file **type_alu.vhd**. Therefore, detail this type and fill the operations as suggested.

When you have designed the entire structure of your ALU, use the test bench **tb_alu.vhd** to simulate it.

Summary of what is requested

ALU behavioral VHDL netlist and relative test bench.

7.2 Synthesis of the ALU

Once you have verified the correct functioning of your ALU, you can proceed with the synthesis process. So:

1. Copy your **alu.vhd** file in the **syn** directory and synthesize it. It is suggested to use a small number of bits (e.g. 4) for an easier inspection of the synthesis results.

Hints&Tips: You can also synthesize only one function at a time, by commenting the others. In this way, you can appreciate how each functional block is effectively synthesized.

2. At the end of your analyses, run a final synthesis for all the functions.
3. Report timing and area.
4. Generate the synthesized netlist.

Summary of what is requested

Synthesis script, timing and area report files. Please provide a .txt file commenting your results.