School of Information and Communications Technology
Dept. of Electronics Systems

Last Revision 110831

Laboration 0

IL2217 Digital Design using HDL

Introduction to VHDL and ModelSim

Name:.....

Personal Number:.....

Date of Approval:.....

Assistant:.....

1. Introduction

Please note, this laboration is not compulsory for passing this course.

This laboration is a simple introduction to the VHDL language and ModelSim, the tool used for simulating the written VHDL code. The goal of this laboration is to give you, the student, a gentle and easy start to the laboration environment used in this course.

All code used in this laboration is available in the laboration directory.

2. Preparations

No preparations are needed for this lab.

3. Setting up your simulation system

Log on to the computer using your username and password.

In windows: Create a lab directory for the course (e.g. IL2217) in your home directory.

Start ModelSim:

Start -> Programs -> ModelSim -> Altera 6.0

When ModelSim has started (this make take a while), change directory to the newly created laboration directory (IL2217):

Example: File -> Change Directory -> ...

Create a project (File->New...->Project...)

Name the project LAB0

Name the project location IL2217/Lab0

The Modelsim tool will now create a directory Lab0 for you.

4. Laboration exercises

4.1. The NAND gate

The NAND gate in figure 1 on page 4 can be implemented as an AND gate followed by an inverter, see figure 2 on page 4.

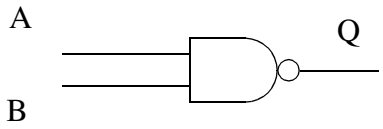

FIGURE 1. NAND GATE

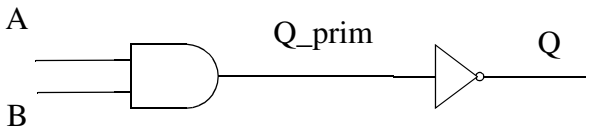

FIGURE 2. AND-NOT IMPLEMENTATION

4.2. The ENTITY construct

VHDL files are basically divided into two parts, the entity and the architecture. We will start with the entity and deal with the architecture in section 4.3.

The entity is basically where the circuits in- and outports are defined. There are a multitude of I/O ports available, but this lab will only deal with the two most usual ones, the INput and OUTput ports. (Other types of ports are for example the INOUT and BUFFER ports.)

The entity of the circuit in figure 1 should look something like below. Please notice that comments in the code are made with a double-dash (--).

```
ENTITY nandgate IS
  PORT(
    A: IN BIT;
    B: IN BIT;
    Q: OUT BIT -- NB! No ';' here!
  );
END nandgate;
```

Now that we have defined the I/O interface to the rest of the world for the NAND gate, we should move on to the architecture of the circuit.

4.3. The ARCHITECTURE construct

While the entity was comparable to figure 1, the architecture is comparable to figure 2. The entity told us nothing about how the circuit was implemented, this is taken care of by the architecture part of the VHDL code.

The architecture of the NAND gate matching the entity above could then be written as something like this...

```
ARCHITECTURE dataflow OF nandgate IS
  SIGNAL Q_prim: BIT;
BEGIN
  Q_prim <= A AND B; -- The AND-operation. (1)
  Q <= NOT Q_prim; -- The inverter. (2)
END dataflow;
```

All statements in the architecture are executed in parallel. If statement (1) and (2) would switch places in the architecture the result would still be the same.

4.4. Your first complete VHDL file

All VHDL files start with an entity and is then followed by (at least) one architecture. More architectures are possible for one entity, but that is beyond the scope of this laboration. The complete VHDL file will look as below:

```
ENTITY nandgate IS
  PORT(
    A: IN BIT;
    B: IN BIT;
    Q: OUT BIT -- NB! No ; here!
  );
END nandgate;
ARCHITECTURE dataflow OF nandgate IS
  SIGNAL Q_prim: BIT;
BEGIN
  Q_prim <= A AND B; -- The AND-operation.
  Q <= NOT Q_prim; -- The inversion.
END dataflow;
```

Save this file as `nandgate.vhd` as an ordinary text-file on your account.

4.5. Compilation of VHDL files in ModelSim

The next step is to compile the file and check the code for syntax errors. If you havent already started ModelSim, do it now. Dont forget to create a work directory as described in Section 3.

To compile the file nandgate.vhd:

version 6.0

click on the leftmost icon in the toolbar (compile) OR

Select from the menus: Compile->Compile...

version 5.7

click on the fourth symbol in the toolbar (compile)

Choose directory by double click on it (the one you created and copied files into earlier)

Select file: nandgate.vhd

(press compile)

Check for errors in the compile window. If you encounter any errors, talk to an assistant. Close the compile windows by clicking Done.

If you succeed this far without getting any compile errors you have successfully compiled your first VHDL file. Congratulations! But... How will you be sure that the operation of the design is the one you intended? For this purpose you need to build a testbench.

4.6. The testbench

The testbench is the most usual way to test VHDL designs. In short, one can say that the testbench uses the design to be tested as a component. The testbench gives you the possibility to feed the tested design with various input signals and to study the output of the circuit to be tested.

The testbench code is given in the appendix (‘*VHDL code for the NAND testbench*’) and in the laboration directory.

Compile the testbench just as you did with the NAND gate.

If no errors occur, you are ready to start simulating the circuit, with help from the testbench.

version 6.0

Simulate -> Start Simulation... (or press the simulation icon)

Click the plus sign for work

Choose tb_nand

Ok

version 5.7

Simulate -> Simulate

Click the plus sign for work

Choose tb_nand

Ok

Except for the use of BIT_VECTOR there are two more new constructs in the test-bench VHDL code, the component and the port map constructs. They will be studied in section 4.8. The BIT_VECTOR type is a type for multi-bit wide signals.

4.7. Simulating the NAND gate

Now, when you have compiled a design it is time to choose which design to simulate:

Before simulation, it can be a good idea to open some windows.

version 6.0

View-> workspace

View -> Debug Window -> Objects

View -> source

version 5.7

View -> structure

View -> signals

View -> source

In the workspace (structure) window you can see the structure of the design. The source code for the module that is chosen in the structure window is displayed in the source window. In the signal window the signals from the chosen design are listed. Play around a little bit with the structure window and watch what happens in the other windows.

Choose what signals to watch. Do the following:

Select the architecture c1: nandgate(dataflow) in the workspace (structure) window,

then in the signals window:

Edit -> Select all

version 6.0

View -> wave

version 5.7

Add-> Wave -> Signals in region

(You may also select signals in the Object (signals) window and drag and drop them into the wave window)

In the wave window you can see four signals, A, B, Q and Q_prim listed.

Start the simulation (the time can be adjusted by changing the number under Run)

version 5.2

Simulate->Run -> run 100 ns

version 5.7

Simulate->Run -> run 100 ns

NB! You set the runtime under Simulate->Runtime Options...

In the wave window, you can see the waveforms for the selected signals.

Is the NAND gate working ok, e.g. does Q equal A NAND B?

4.8. The COMPONENT and the PORT MAP constructs

If you want to use an already created circuit as a part of a bigger circuit you need to use a component and a port map construct. The testbench used the NAND circuit as a component and we will soon use our self-made NAND circuit to design an 2:1 multiplexer.

The component declarations are made in the beginning of the architecture. The port map construct are used in the statement part of the code.

To implement the 2:1 multiplexer we should study figure 3 below.

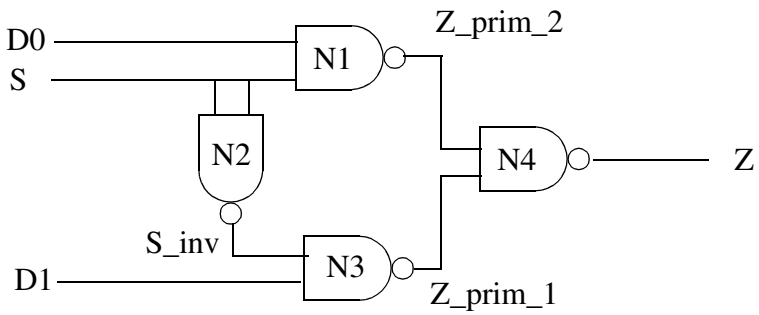

FIGURE 3. 2:1 MUX IMPLEMENTED WITH NAND GATES

4.9. The VHDL code for the 2:1 multiplexer

The **ports** of the **entity** of the multiplexer should consist of the ports D0, D1, S and Z. D0, D1 and S are input signals, Z is the output signal. The type of the ports are BIT.

One type of **component** is used in the multiplexer. The NAND gate we made earlier is used four times in this design.

Four **port map** constructs are used, one for each NAND gate instantiation.

Three wires are not connected to any of the ports so we need three **signals** in the VHDL code. The signals are named S_inv, Z_prim_1 and Z_prim_2. The type of the signals are BIT.

The VHDL code of the multiplexer using the NAND gates as components is as follows:

```
ENTITY mux IS
  PORT(
    D0: IN BIT; -- Input data signal A.
    D1: IN BIT; -- Input data signal B.
    S: IN BIT; -- Control signal.
    Z: OUT BIT -- Output data signal.
  );
END mux;

ARCHITECTURE rtl OF mux IS
  COMPONENT nandgate
    PORT(
      A: IN BIT;
      B: IN BIT;
      Q: OUT BIT
    );
```

```
  END COMPONENT;

  SIGNAL  S_inv, Z_prim_1, Z_prim_2: BIT;

BEGIN
  N1: nandgate PORT MAP(D0, S, Z_prim_2);
  N2: nandgate PORT MAP(S, S, S_inv);
  N3: nandgate PORT MAP(S_inv, D1, Z_prim_1);
  N4: nandgate PORT MAP(Z_prim_2, Z_prim_1, Z);
END rtl;
```

4.10. Simulating the 2:1 MUX

Compile the file.

Compile the testbench in the appendix, '*VHDL code for the MUX testbench*'. The VHDL code is also available in the laboration directory.

To clear ModelSim from the previous simulation, i.e. the simulation of the NAND gate:

Simulate -> End simulation

This will remove most of the open windows. To open all windows do:

View -> All

Load the design:

Simulate -> work -> test -> testmux

In the structure window, click on c1: mux(rtl).

In the signals window, add the signals to view:

View -> Wave -> Selected signals

or

right click-> Add-> Add to Wave

To run the simulation:

Run -> run 100 ns

Inspect the waveforms. Is the multiplexer working ok?

Appendix I. VHDL-Code for the Exercises

1. VHDL code for the NAND testbench

```
-- Laboration 0, Modern Digital Design 2001
--
-- Test bench for a NAND gate

ENTITY test IS END test;

ARCHITECTURE testNand OF test IS

  COMPONENT nandgate
    PORT (
      A: IN BIT;
      B: IN BIT;
      Q: OUT BIT);
  END COMPONENT;

  SIGNAL testvector: BIT_VECTOR(1 downto 0);
  SIGNAL result : BIT;

BEGIN

  C1:nandgate PORT MAP(A => testvector(1), B => testvec-
tor(0) , Q => result);

  testvector <="00",
      "01" AFTER 10 ns,
      "11" AFTER 20 ns,
      "10" AFTER 30 ns,
      "00" AFTER 40 ns;

END testNand;
```

2. VHDL code for the MUX testbench

```
-- Laboration 0, Modern Digital Design 2001
--
-- Test bench for a 2:1 multiplexer component

ENTITY test IS END test;

ARCHITECTURE testMux OF test IS

  COMPONENT mux
    PORT (
```

```
      D0: IN BIT;
      D1: IN BIT;
      S: IN BIT;
      Z: OUT BIT);
  END COMPONENT;

  SIGNAL testvector:BIT_VECTOR (2 downto 0); -- D0, D1 and
S.
  SIGNAL result:BIT;

BEGIN

  C1: mux PORT MAP(D0 => testvector(2), D1 => testvec-
tor(1), S => testvector(0), Z => result);

  testvector<=
    "001",
    "101" AFTER 10 ns,
    "011" AFTER 15 ns,
    "111" AFTER 20 ns,
    "000" AFTER 25 ns,
    "100" AFTER 30 ns,
    "010" AFTER 35 ns,
    "110" AFTER 40 ns;

END testMux
```