EE 5143 – M6 Lab – Dr Rahman

Labs so far have given you practice with entering VHDL code, compiling it, fixing any bugs, making pin assignments, checking RTL and programming the FPGA on the DE0-Nano board. The VHDL used so far has been in the behavioural/data-flow style. This week we look at the structural style of VHDL where you can 'wire' components together (using VHDL text) just as you could wire components on a breadboard. Using this alternative style you can make a circuit in exactly the same way as a real circuit with real components. While structural VHDL is intrinsic to the language, QP also provides you the ability to perform schematic entry i.e. wire up a circuit using components in a completely visual fashion. This is no different from wiring up a virtual circuit in any other schematic entry program. For instance, using logic gates, graphically interconnected to each other, you can build up a multiplexer, decoder, counter etc. as a typical schematic circuit diagram. This week we'll also learn how to perform schematic entry using QP. We'll build the same circuit using structural VHDL and then using schematic entry. This will be a simple AND-OR-INVERT (AOI) circuit. To get started create a folder called M6 to hold this week's projects then create two separate folders called A and B inside this folder to contain the two projects mentioned above. Let's get started with the first project.

Using the new project wizard in QP, create a new project. First navigate to the /M6/A project folder as the project directory. Next, name the project Comp1 (for the first component – we'll first create two components and then will connect them together later on). This will also become the name of the so-called top-level entity. Thus, we'll have to call this entity Comp1. Set the device as usual and then start a new VHDL text entry file (this will have to be saved as Comp1.vhd). Type the following entity/architecture pair in that text file:

```
----------------------------------
-- Entity Declaration --
----------------------------------
-- here we describe the I/O of the design

entity Comp1 is
	port(
		a, b : in bit;
		c : out bit
	);
end entity;

----------------------------------
--	Architecture Definition  --
----------------------------------
--	here we put the description code of the design

architecture gate of Comp1 is

begin

c <= a AND b;
```

```
end gate;
```

As you can see, this simply describes a 2-input AND gate. This will serve as our first component. Although, this example is extremely simple, a component can have any degree of complexity. It is 'exposed' by its I/O ports (declared in the entity). After VHDL code entry, compile the code and then click on Save All and then Close Project (these are under the file menu on the left). Note that we do not perform any pin assignments at this stage.

To describe the second component, we have to repeat the above steps again (you do this for each new component). Using the new project wizard in QP, create a new project. First navigate to the /M6/A project folder as the project directory (notice that all components share the same directory folder – this is very important). Next, name the project Comp2 (for the second component). This will also become the name of the so-called top-level entity. Thus, we'll have to call this entity Comp2. Note that QP may give you a warning that the folder already contains a previous project – ignore that warning by clicking on 'No'. Set the device as usual and then start a new VHDL text entry file (this will have to be saved as Comp2.vhd). Type the following entity/architecture pair in that text file:

```
---------------------------------
-- Entity Declaration --
---------------------------------
-- here we describe the I/O of the design

entity Comp2 is
        port(
                a, b : in bit;
                c : out bit
        );
end entity;

---------------------------------
--      Architecture Definition  --
---------------------------------
--      here we put the description code of the design

architecture gate of Comp2 is

begin

c <= NOT(a OR b);

end gate;
```

This code simply describes a 2-input NOR gate. This will become our second component. Again, after VHDL code entry, compile the code and then click on Save All and then Close Project. At this stage your M6/A folder will contain files for both these architecture/entity pairs (you may want to verify this by looking at the contents of that folder).

Once you have got the components in place, you need a so-called 'top-level' project where you can interconnect your components. This idea of a top-level entity/architecture pair whose function (mostly) is to connect lower level components is extremely important and quite widespread so don't be surprised when you see it very frequently. This is how most real VHDL projects of professional complexity are built – you either develop your top-level first and then develop your required components (top-down approach) or you develop your components first and then develop your top-level (bottom-up approach). The latter one is more common, as implemented here. You can make a rough sketch of your digital system on a piece of paper, showing the various components and their interconnections then you write VHDL for components, followed by interconnecting them either using structural VHDL or a schematic diagram. The interconnection of components can be done either textually through structural VHDL or visually through QP's built-in schematic editor, as we see next. Although in this lab's projects we are only connecting primitive logic gates as components to build our very simple circuit, it is possible to use your very own components (described in behavioral style) and interconnect them through structural VHDL or using the schematic editor. The idea of first writing behavioral VHDL code to describe components and then interconnecting them using the QP schematic editor is the most powerful and professional approach to developing all kinds of digital systems using VHDL. Remember this. After the next two labs you will know most details of how to perform that type of design.

Next, repeat all the steps to create a new project within the /M6/A folder giving it the name AOI. Complete the rest of the wizard, as usual. Open a new VHDL file and type in the following VHDL code in it:

```
---------------------------------
-- Entity Declaration --
---------------------------------
-- here we describe the I/O of the design

entity AOI is
        port(
                i1, i2, i3, i4 : in bit;
                z : out bit
        );
end entity;

---------------------------------
--      Architecture Definition  --
---------------------------------
--      here we put the description code of the design

architecture structural of AOI is

signal temp1, temp2 : bit; -- these are (internal) signals that carry outputs from 2 Comp1 components to the inputs of the Comp2 component
```

component comp1 port(a, b : in bit; c : out bit); end component; -- declaration of Comp1 (written on a single line)

component comp2 port(a, b : in bit; c : out bit); end component; -- declaration of Comp2 (written on a single line)

begin

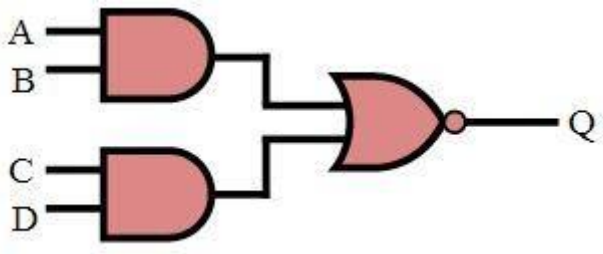
MyAND1: Comp1
port map(a => i1, b => i2, c => temp1);

MyAND2: Comp1
port map(a => i3, b => i4, c => temp2);

MyORINVERT: Comp2
port map(temp1, temp2, z);

end structural;

There are some new things here. Our circuit will eventually look like this:



We see a need for five I/O signals and you see them declared in the AOI entity. The declarative part of the architecture first declares two internal signals temp1 and temp2. These are internal signals, not visible from the outside, used for connecting one component to another (you see them in the above diagram as the connections from the output of the AND gates to the input of the NOR gate. After that, two components are declared. The main purpose of doing this is to make the architecture aware of their existence. These are only declarations – their definitions could be in other folders contained within the same top-level folder (as done here) or can even be in a library package. Study the construction of a line such as:

component comp1 port(a, b : in bit; c : out bit); end component;

Like all component descriptions, it starts with the VHDL reserved word 'component'. The rest is very similar to an entity declaration except that there is no 'is' used and after end you write component instead of entity. The actual port declaration can in fact be copied verbatim from the entity declaration of this component – you can do a copy and paste here. Declaration of the other component is similar.

After the begin keyword of the architecture you see three concurrent 'instantiation' statements. These serve to instantiate i.e. create components. Look at the first such statement:

```
MyAND1: Comp1
        port map(a => i1, b => i2, c => temp1);
```

Comp1 refers to the AND gate component whereas MyAND1 is an instance label and is required in this case. This statement is said to create an instance of the Comp1 component. You can have any number of instances of the same component – all of them will need their own instantiation statements and will have different instance labels. Thus MyAND1 and MyAND2 are two distinct AND gates in your design which are otherwise exactly the same (both are Comp1). The rest is a port mapping that attaches signals to the various I/Os of the component. This should be obvious from the above: AND gate MyAND1's input node a connects to the input signal i1 etc. This type of port mapping is called mapping by association and you can write the mappings in any order, not necessarily a, b, c.

MyAND2 is dealt with in the same way.

MyORINVERT is similar except that it uses mapping by order i.e. the signal names that appear in the bracket connect to the instance's nodes in the same order as the port order in the instance declaration. In this case, it is important to put the signal names in the correct order. The different styles have been shown here just to illustrate the fact that two different types of port mappings are possible.

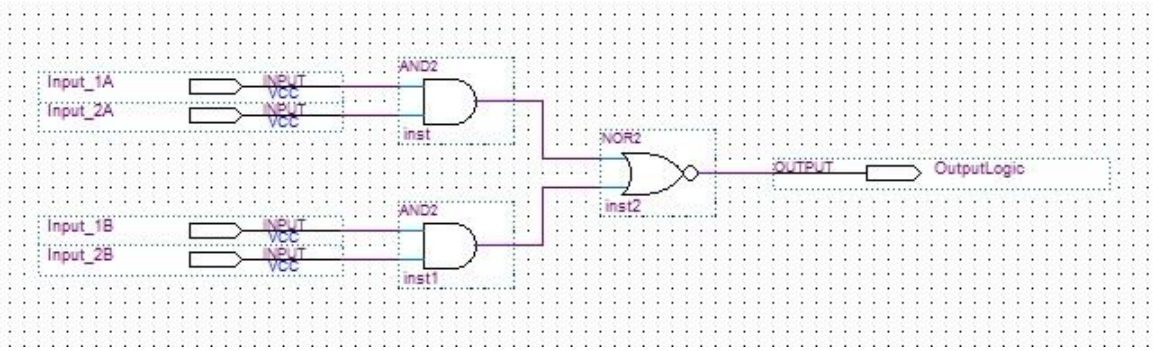
That is all that is needed. Note that you can have other usual types of concurrent statements, processes etc. in this code but it is quite usual to have structural VHDL code which only interconnects components, as above.

Save your code, compile and debug it. This time perform a pin assignment after first compilation using the pin planner, as usual. Connect the four input nodes to the four slide switches on your DE0-Nano board. These switches connect to FPGA pins M1, T8, B9 and M15. Connect the z output to an LED, such as led[0] at pin A15. After pin assignment, compile the code again to bind your pin assignments to the I/O nodes in your code. Then you can download your code to your Cyclone IV FPGA as before. Be careful to select the AOI.sof file for the programmer. The output folder will also contain Comp1.sof and Comp2.sof files. These are not used for programming here.

Next we do the same thing but through a circuit diagram approach.

To proceed further, start yet another new project through the QP New Project Wizard. This time use the /M6/B as the project directory/folder. Use Diagram as the project's name. Complete the wizard as usual. This time instead of creating a new VHDL text file create a schematic file (File -> New -> Block Diagram/Schematic File). A graphical space with a dotted grid appears. This is where you can draw your circuit diagram. Notice the tool bar that appears at the immediate top of the schematic area. Click on the AND gate icon there – it holds all components that are available for interconnection. In the window that appears double click the top line of libraries (at this time there will only be one line of library). Double click primitives, then double click logic and finally click and2. A preview of a 2-input AND gate appears in the preview space to the right.

Click on OK to close this window. Now a 2-input AND gate appears attached to your cursor and moves with it. Take the cursor to a suitable location in the schematic entry space and click once to place an AND gate there. The cursor still has the symbol attached to it so further AND gates can be placed. Place another instance of 2-input AND gate a little below the first AND gate. Next, pick up a nor2 gate component and place an instance of it to the right of the two AND gates. Each time you want to get rid of the instance attached to the cursor just click on the arrow cursor on the tool bar. Next make the connections between the AND gates and the OR gate. Click on the Orthogonal Node Tool icon and then use the cursor to click on the output of each AND gate and drag it to the inputs of the NOR gate and click there. If you need further guidance then read the description on pages 62 and 63 of your DE0-Nano user manual. You will also need to add two input pins and one output pin explicitly to your circuit diagram. These pins appear under the pins tool icon on the tool bar. Connect these pins to the rest of your circuit as appears below.



Double click (twice) on each input and output pin's name and type in your own names, such as input_1A etc. as shown above.

Save your schematic diagram file. It gets saved as Diagram.bdf (bdf stands for block diagram file).

Next compile your project and then make pin assignments to the same pins as you did in the structural VHDL project above. Re-compile and program your FPGA.

In each case the behavior of your FPGA will be the same and can be verified by setting the slide switches to different positions. For moving the tiny slides on the switches use something small and pointed such as a compass point, a tweezer, a very small screw driver etc.

For the first project, place your code, pin planner (after pin assignment) and RTL windows in a Word file. Note that in the RTL view you can click on any + sign to see what is inside that block.

For the second project, place your circuit diagram, pin planner (after pin assignment) and RTL windows in the same Word file.

You can submit either the Word file or the corresponding pdf file through the Blackboard.

Try clicking on Tools -> Chip Planner this time to see where physically your logic has been placed inside the FPGA (it shows quite a large black area although only a single logic element is all that is needed to implement this very simple design).