Lab Worksheet 02

LWS02 LWS02

Part 1:

Main Objective:

To Learn Recursive Circuit Design using VHDL.

Sub-Objectives:

1) To learn how to specify the architecture when instantiating an Entity.
2) To design a N-bit Parity Detector using an iterated FOR statement.
3) To design a N-bit Parity Detector forming a tree recursively.

Part 2:

Main Objective:

To become familiar with the internal structure of Intel FPGA families.

Sub-Objectives:

1) To practice using device datasheets to understand FPGA structures.
2) To learn how to synthesise and view synthesis output using Quartus.
3) To practice creating reasonable documentation of synthesis results.

Specification:

An N-bit Odd-Parity Detector:

The Odd-Parity Detector should:

- should only use `std_logic` or `std_logic_vector` for its interface signals, (to be compatible with the types created by Quartus' netlist)
- be a purely combinational circuit,
- have an N-bit input, determined by a GENERIC parameter,
- have an output that is '1' when the parity of the input is Odd.

```
Entity OddParity is
Generic ( N : natural := 8 );
Port ( X : in std_logic_vector( N-1 downto 0 );
       IsOdd : out std_logic );
End Entity OddParity;
```

Procedure: (Part 1)

Part 1:

Writing multiple ARCHITECTUREs for a single ENTITY.

We can include more that one ARCHITECTURE for a single ENTITY in a file. To accomplish this we simply write the extra ARCHITECTUREs after the ENTITY giving each ARCHITECTURE a unique name.

We specify the desired ARCHITECTURE each time that we instantiate the ENTITY by placing the ARCHITECTURE name inside parentheses.
ie.

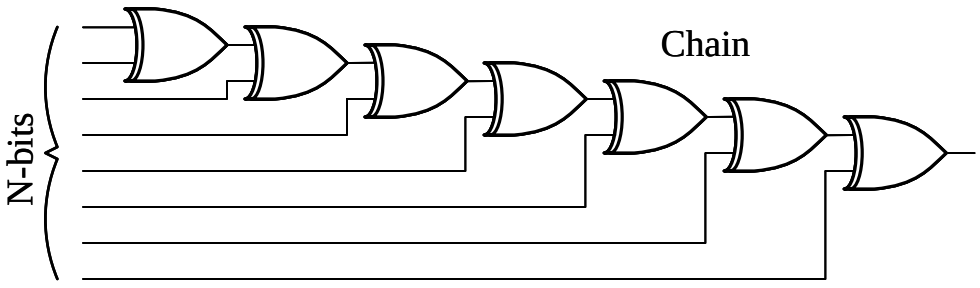
```
c1: entity    Work.MyEntity( FirstArchitectureName )    port map ( … );
c2: entity    Work.MyEntity( SecondArchitectureName)    port map ( … );
```

Directly instantiating an ENTITY requires identifying the library. As an alternative, it is possible to instantiate an ENTITY indirectly using a COMPONENT statement. If this is the case, the compiler must know how to bind a COMPONENT to an ENTITY . Binding is specified using a CONFIGURATION.

Generating repetitive structures during elaboration.

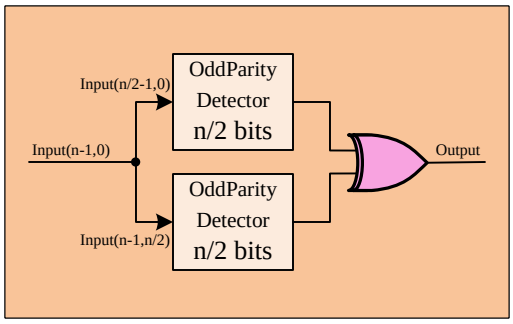
The VHDL compiler can be instructed to create many concurrent statements using a GENERATE statement. (google “VHDL generate”)

1) Create an XOR gate with 2-inputs , as an explicit primitive ENTITY.
   Use the ieee-library XOR-function within the ARCHITECTURE to implement the xor gate.
2) Create a “chain” ARCHITECTURE for OddParity using instances of your XOR primitive.

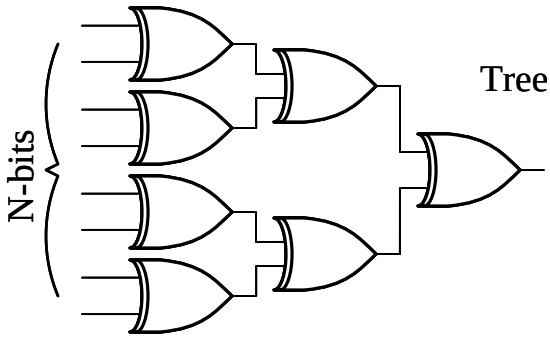


3) Add the VHDL SourceCode to the Quartus project and set OddParity as the top-level.
   (select the file in the project navigator and right-click).
4) Run the synthesiser and fitter, (crtl-l) and inspect the circuit created by the synthesiser using the RTL viewer. (tools->Netlist Viewers->RTL Viewer)
5) Inspect the circuits created by the fitter, using the post-map viewer and post-fit viewers.
   (tools->Netlist Viewers->Technology Map Viewer)
6) Change the statement that instantiates the XOR primitive to the ieee.library XOR function. Now rebuild and observe the changes in the resulting circuits. (using all netlist viewers).
   Can you explain why these results are different?

VHDL allows circuits to be defined recursively. The ARCHITECTURE of the OddParity ENTITY called “Tree" shown below can be defined recursively.
Here is the idea. A parity detector of width 32 can be determined by xoring two parity detectors of width 16.



7) Create a “tree” ARCHITECTURE of ENTITY OddParity.
8) Within the “tree” ARCHITECTURE , instantiate two instances of ENTITY OddParity. Make sure that these two instances are of half the width by passing a value to the GENERIC parameter. Then produce the output with an instance of you XOR primitive.
9) Add a conditional GENERATE statement around these instantiations to prevent them from elaborating when the parameter becomes too small. You should only elaborate the recursive statements when the parameter is 2 or greater.
10) Finally, add a conditional GENERATE statement that elaborates the correct NON-recursive statements for the case when the GENERIC parameter is 1. Be careful when dividing the parameter by 2. Integer division truncates. (reference: **Handout-(NM)10-252-1167.pdf**)



11) Using N = 7, synthesise the new circuit and using the netlist viewers, check that you created a tree.

Studying FPGA Devices: Part 2

Becoming familiar with Intel FPGA device families:

At this point you should have noticed that even though you explicitly defined two different structures (chain and tree) for the architectures, Quartus created a "tree" as the final circuit.

Now let's investigate how the synthesized structures map to FPGA resources.
Let's consider FOUR different FPGA devices.

1) Cyclone IV – EP4C-E6-F17-C7
2) Cyclone V – 5CSE-B-A2-U23-C7
3) MAX 10 – 10M-04-DA-F256-C7G
4) ARRIA II – EP2AGX-45-C-U17-C6

1) Review the following pages that describe the FPGAs. You should study the diagrams and review the resources tables so that you understand the basic structure of typical FPGA devices.
Make yourself familiar with

- the general features available within the device family,
- the resources contained within specific versions of a device,
- the part numbering scheme for specifying a particular device,
- the general architecture of a device family,
- the structure of Logic array blocks, (LABs)
- the structure of Logic Elements, (LEs) or Adaptive Logic Modules. (ALMs)

"Cyclone IV – Device Handbook-Vol1"	pages 15-18, 26, 30 & 33
"Cyclone V – Device Overview"	pages 4, 5, 12, 19 & 29
"Cyclone V – Device Handbook-Vol1"	pages 10, 14 & 15
"MAX 10 - Architecture"	pages 4, 5 & 8
"MAX 10 – Device Overview"	pages 4, 5, 6 & 9
"Arria II – Device Handbook-Vol1"	pages 15-17, 20, 28, 35 & 36.

You do not memorize the details, you look them up when you need them.
You do however memorize the general structure and overall features of the devices.

You can change the target FPGA device using the menu, "Assignments->Device..."

Procedure: (Part 2)

Part 2:

Comparing Resource Utilization across the FOUR selected devices.

Now it's time to observe the synthesis results for different sized circuits implemented on the four target FPGAs.

2) Document both the RTL circuit and the post-fit circuit for both N=7, tree and N = 7 chain, implemented on a Cyclone 4E, EP4C-E6-F17-C7.
Make sure that all entities in the RTL circuit are expanded.
Refer to the document "LWS-02.x-Documentation(synthesis)-350-1251.pdf" for information about producing acceptable documentation.

3) Now synthesize the circuit (4 * 3) = 12 times, for each value of N and for all four devices. Complete the table showing resource utilization. Enter the number of LE/ALMs used and the %utilization in each cell of the table.

Logic Resources Used	N = 24	N = 80	N = 128
EP4C-E6-F17-C7			
5CSE-B-A2-U23-C7			
10M-04-DA-F256-C7G			
EP2AGX-45-C-U17-C6			

4) For the four N = 128 circuits, document the device floorplan, and document one example of a used LE/ALM.

- You can see the allocated floorplan of the device using the Chip Planner, "Tools->Chip Planner".
- To see an allocated LE/ALM, select and zoom-in on one element, then double-click to open the resource property editor. Capture the window of the resource property editor and resize so that you can place two images on a portrait page. Show the four captured windows with labels on two pages so that it is easy to compare the structure of the circuitry across the four devices. To capture the active window type "alt-PrnScr"

You should study these results. You should understand the basic FPGA technology and how combinational circuits are implemented using Look Up Tables. (LUTs)

Lab Worksheet 02

LWS02 LWS02

Task ID	Show to TA during Lab Period for feedback and Lab Tick	**Demonstration** Required Signup for a demo time on Canvas	**Documentation** Required Canvas Submission	**Documentation** Required for feedback and Lab Tick
LWS-02-P1 (01-06)	☒	☒	☒	☒
LWS-02-P1 (07-10)	☒	☒	☒	☒
LWS-02-P1 (11)	☑	☒	☒	☒
LWS-02-P2 (01)	☒	☒	☒	☒
LWS-02-P2 (02-04)	☒	☒	☒	☑