Lab Worksheet 02

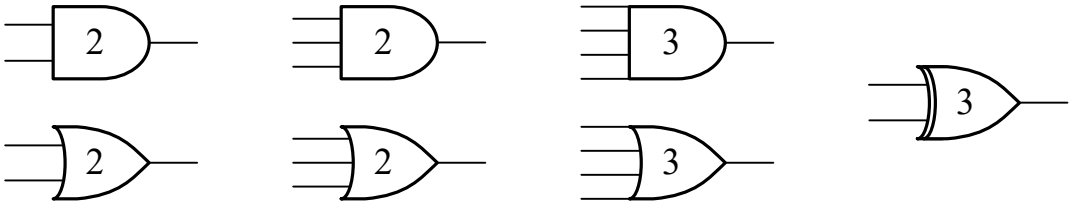
Main Objective:

1) To enhance your ability to code Hierarchical Circuits in VHDL.

2) To verify (using simulation) the timing of various Addition Circuits.

Start the program ModelSim. Create a new folder to store your project files before creating a new project (called "LWS02"). Create a file called LuckyGates.vhd. When the "create" window appears you should add the existing file called "LuckyGates.vhd". Compile the code so that the entities appear in the Work Library. Check that the ENTITIES are now inside the Work library.

The following questions require that you create structural descriptions for various circuits. The circuits are to be built from the following primitive gates.



I have already created the ENTITY/ARCHITECTURE definitions for these objects in a file called "LuckyGates.vhd". You should build your ENTITIES from these primitive gates. **DO NOT use VHDL operators** to create basic gates.

Please demonstrate to the Teaching Assistant the following.

FOR PART 1: (*deadline: beginning of Lab on Feb 7th 2020*)

Chain1: VHDL architecture / Circuit Diagram (drawn in your lab notebook)
Tree1: VHDL architecture / Circuit Diagram
Chain2: VHDL architecture / Circuit Diagram
Tree1: VHDL architecture / Circuit Diagram
Simulation results showing the output signal waveforms of all four circuits in ONE diagram.

FOR PART 2: (*deadline: beginning of Lab on Feb 14th 2020*)

Ripple: VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.
BookSkip: VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.
GoodSkip: VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.
Brent-Kung: VHDL architecture / Circuit Diagram and waveforms with expanded sum bits.
Table of predicted and measured timing. (drawn in your lab notebook)

All circuit diagrams must be complete showing every element of the VHDL architecture.

- All internal signals must be labelled identically to the VHDL code.
- Do not omit elements simply because a structure is repetitive.
- If the circuit is recursive you should draw TWO circuit diagrams; one for the recursive pattern and one for the stopping condition. Both circuits should be labelled identically to your VHDL code.

Please refer to the Handout-10-252-1167.pdf for further information hierarchical circuit descriptions.

Lab Worksheet 02

LWS02

PART 1:

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.

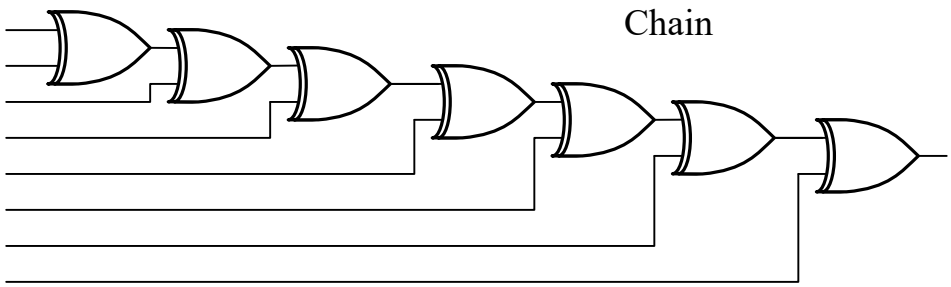
ie.

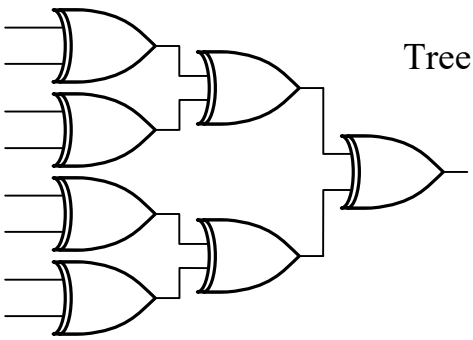
```
c1: entity    Work.MyEntity( FirstArchitectureName )    port map ( … );
c2: entity    Work.MyEntity( SecondArchitectureName)    port map ( … );
```

a) Using the ENTITY, "ParityCheck" that I have supplied in a file called "Part1.vhd", Create two architectures called "tree1" and "chain1" according to the following diagrams.

b) Simulate your designs for 700 nsec using the testbench file "TestPart1.vhd". If you are convinced that the ARCHITECTUREs are correct then submit your circuit diagrams, VHDL code.

PART 1: 1) Continued





2) **Generating repetitive structures during elaboration.**

The VHDL compiler can be instructed to create many concurrent statements using a GENERATE statement. (google "VHDL generate") The code that I have provided also contains examples demonstrating the use of GENERATE statements.

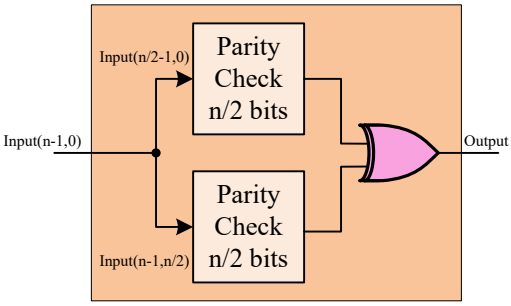
a) Create a third ARCHITECTURE for the xor chain. Use a GENERATE statement to build a third ARCHITECTURE for the ENTITY ParityCheck.
Call this new ARCHITECTURE "chain2"

b) Add an instance of this new ARCHITECTURE to the testbench file "TestPart1".
Run the simulation and verify that the signals Chain1Out and Chain2Out are identical.

Notice that the ENTITY declaration for ParityCheck contains a clause GENERIC by it's PORT definition. The GENERIC clause is a way to pass values to instances of an ENTITY. The value specified in the ENTITY **declaration** is simply the default value of the GENERIC parameter. We pass a specific value of the parameter to an ENTITY object whenever we instantiate the ENTITY.

ie. c1: entity Work.MyEntity(MyArchitecture) generic map (MyValues) port map (…);

c) Build a recursive ARCHITECTURE of the ParityCheck ENTITY called "Tree2". We shall define this ARCHITECTURE recursively. Here is the idea. a parity check of width 32 can be determined by xoring two parity checks of width 16.



Inside the ARCHITECTURE for ENTITY ParityCheck, instantiate two instances of ENTITY ParityCheck. 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 an xor2 ENTITY. Add a conditional GENERATE statement around these instantiations to prevent then from elaborating when the parameter is becomes too small. You should try 3 or Less. Finally, add two conditional GENERATE statements that instantiate correct NON-recursive ENTITYs for the cases when the GENERIC parameter is either 3 or 2. Be careful when dividing the parameter by 2. Integer division truncates. It is also possible to use a width = 1 as a stopping condition. Does this work?

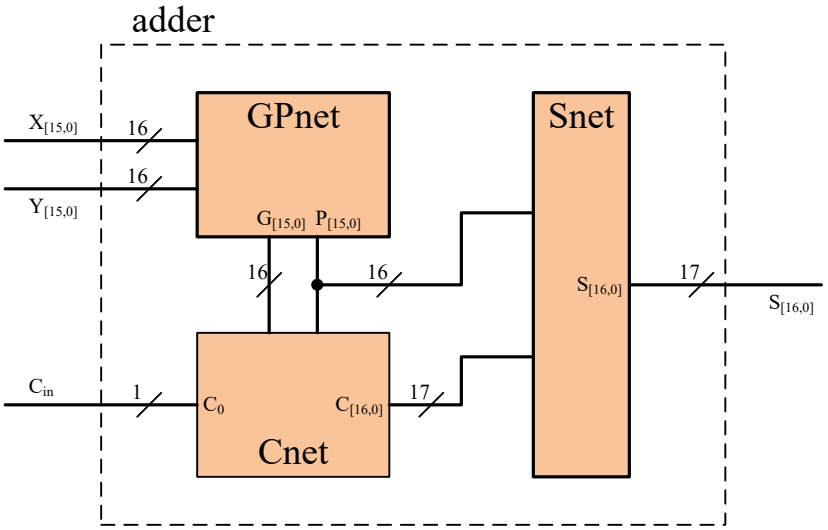
d) Add an instance of this new ARCHITECTURE to the testbench. Make sure that the ENTITY is passed a width of 8. Run the simulation and check that we have the same result as Tree1Out.

e) Submit the waveform window that shows the results of all ARCHITECTUREs.
Submit your Circuit Diagrams and VHDL code in the file "Part1.vhd". Submit fully labelled circuit diagrams for all four ARCHITECTUREs. DO NOT expanded the circuit diagram for the Recursively defined circuit. **Your circuit diagrams should match EXACTLY to the lines of your VHDL code**.

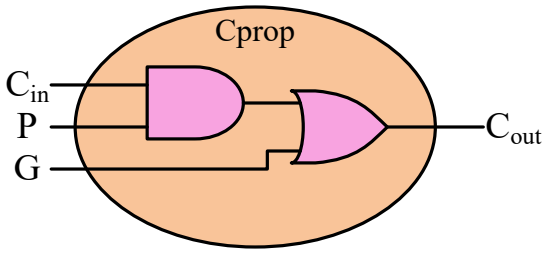
PART 2:

3) **Addition Circuits.**

We are now going to study techniques for **designing addition circuits**. I have provided the basic structure of the circuit in the file "part2.vhd". Read this file carefully and make sure that you understand the code. You are expected to be able to create this code yourself. The file "part2.vhd" declares an ENTITY **adder16** and then provides a structural description that places three ENTITYs inside the ARCHITECTURE. The **GPnet** and **Snet** ENTITY/ARCHITECTUREs are provided inside the file "Part2.vhd". Note the use of GENERATE statements.

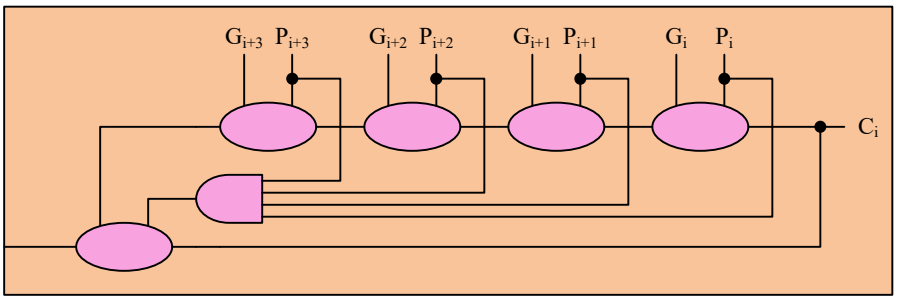


a) Add the file "Cnet.vhd" to your project. This file contains the ENTITY declaration for the carry network. Design an ARCHITECTURE called "Ripple" for a ripple carry network.
You can find details from your Lecture Notes. I have included a simple building block ENTITY called "Cprop" that you should use for most of your Cnet ARCHITECTUREs.
You should also declare an internal signal of type std_logic_vector to connect the carries. This signal will be useful in other ARCHITECTUREs.



b) Now add the file "TestPart2.vhd" to your project and compile the code. An ENTITY called TestPart2 should now be in the Work Library. This file instantiates the ENTITY adder and tests the design by stimulating the inputs with various values of the operands. Start a new simulation and select the TestPart2 ENTITY from the Work Library. Add all the signals in the region to the wave window and then run a simulation for 350 ns. Expand the Bits of signal SR so that you can see the detailed timing of the carry propagation. Submit the waveform.

c) Now it is time to compare the ripple network design to a skip network design.
Build a 16-bit carry network with a skip path for every 4-bits. The design of a block of 4-bits is given in Parhami, page 183, Figure 10.7. Try implementing skip blocks using the Cprop ENTITY as shown below. Call the ARCHITECTURE "BookSkip"



d) Run the simulation of TestPart2 again. This time you should see waveforms for both SR and SBS signals. Expand the bits of these signals and compare them to see if the skip paths are working correctly. You should find that the improvement in minimal. You can verify that the skip paths are working correctly by checking bits 4, 8, 12 & 16. There is a flaw in the design. Submit the simulation waveform with the bits of SBS expanded.

e) You now need to correct the error in the design. This involves adding a few more "or2" ENTITYs. Create a new ARCHITECTURE called "GoodSkip". You may find it easy to simply copy & paste the BookSkip ARCHTECTURE and then edit the copy.

f) Run the simulation again. You should see valid data on the signal line SGS. With the improvement that one would expect when using a skip trick.

g) Finally implement a Brent-Kung Look-Ahead network (See Lecture Notes) using the recursive method of definition. (You may assume that the width is a power of 2)

h) The testbench simulates five additions. Make a table that has five rows, one for each addition. Label each row with the values of X, Y and C_{in}. The table should have four columns, one for each Cnet design. Enter both the predicted (from theory) and the measured (from simulation) time delays for output value to stablize, for all circuits.