VHDL Design Techniques

Adders are fundamental components for digital systems. In this lab, you will first model a single-bit adder. The single-bit adder will then be re-used to define the structural architecture of an N-bit adder, where N is an adjustable generic parameter. A behavioural model for the N-bit adder will also be built. Timing and implementation details will be considered to compare the two design techniques. You are required to develop test cases to verify the timing performance of your design.

Lab Overview

In this lab you will:

- Understand the difference between behavioural and structural VHDL models.
- Design VHDL models at both the Behavioural and Structural levels.
- Build parameterizable VHDL models using the Generic and Generate constructs.
- Learn how a design is implemented in the configurable logic blocks of an FPGA.

Pre-lab

1. Read all of this document before you come to the first session of the lab.
2. Review the lecture notes, the supplementary Lab 1 tutorial files, and the VHDL textbook to become familiar with the relevant VHDL syntax. In particular, you must be familiar with the syntax for the Port Map, Generic and Generate constructs.
3. Generate the truth table for a 1-bit Full-Adder (FA).
4. Construct the schematic for a 1-bit FA using only elementary logic gates.
5. Create a 4-bit Ripple Carry Adder block diagram using your 1-bit full adder symbol. Show all connections.
6. Design a thorough set of test cases for your 12-bit adder.
7. Write your draft VHDL code for the lab. (1 bit FA, N-bit structural adder and N-bit behavioural adder).
Background - Half Adder

The simplest form of adder is called a **Half-Adder (HA)**. The HA performs bit-wise addition between two input bits, with no carry-in bit. Depending on the result of the operation, the HA either sets or clears its outputs, the **Sum** and **Carry** bits. A HA can be expanded to include the logic for carry-in, and the modified unit is called **Full Adder (FA)**. Several FAs can be combined in a cascaded chain, creating a **Ripple Carry Adder (RCA)**, to compute the sum of two N-bit wide vectors. Let's see how we do this in VHDL.

Each VHDL file contains two major sections, the **entity** definition and the **architecture** definition. The entity definition will describe the external interface to your design, and the architecture will define how the module actually functions.

Let's start by examining the interface to the HA in the form of a symbol. The HA symbol pictured below describes the external interface to the HA module. Typically, inputs are on the left and outputs are on the right.

![HA Symbol](image)

The VHDL equivalent of the external interface description is known as an **entity** declaration. The HA entity declaration would appear as follows:

```vhdl
entity ha is
  Port ( X : in std_logic;
         Y : in std_logic;
         Sum : out std_logic;
         Carry : out std_logic);
end ha;
```
Each VHDL file in CMPE 480 will describe a single design module. In other words, there will be only one entity definition per file. Once the entity has been defined, we proceed to describe the behaviour, or architecture, of our module.

Below is the schematic for the HA. This completely defines the behaviour of our design.

![Schematic Diagram]

Using this schematic as a guide, we can see that the VHDL equivalent can be expressed as two concurrent statements within our architecture body:

```vhdl
architecture Behavioral of ha is
begin
    Sum <= X xor Y;
    Carry <= X and Y;
end Behavioral;
```

Now you have a complete HA. The architecture of the HA is described at the **Register Transfer Level (RTL)**. There are a number of methods that can be used to define the architecture of a VHDL module. In this lab exercise we will examine both a structural and behavioral adder.

A structural architecture can be thought of as a system of interconnected components. Thinking back to EE280 you might remember the painful experience of hooking up components on a breadboard to form a complete system. This is analogous (without the pain) to a structural VHDL architecture. Each component in the architecture can be thought of as one of the IC’s on the breadboard.

Continuing with this analogy, we recall that we could not directly connect the pins of one IC to the pins of another IC. To make this connection, we needed jumper wires. In VHDL, jumper wires are called **signals**. When connecting components in a structural architecture, you must use intermediate signals to pass the values from one
component to the next. These signals should not be confused with the external ports of the entity. Ports are analogous to the IC pins.

Lab Procedure

Section A – 1-bit RTL Full Adder

You will design a 1-bit FA with RTL level. This will be a building block of larger adders.

1. Build a VHDL code for a 1-bit FA.

2. Perform a **behavioural** simulation of your module to verify its logical behaviour. Submit an annotated waveform describing your test cases. Multiple test cases can be included on a single waveform printout. Clearly indicate (annotate) what is happening at each point in the simulation. Annotating waveforms will help you to explain the purpose of each test case. Briefly describe your testing strategy and attach all relevant waveforms to your design summary.

3. Synthesize your design.

4. Perform a **post-place and route** simulation and verify the correctness of your implementation.

5. Open the FPGA Editor and analyze the design.

Section B – Structural N-Bit Adder

In this section you will use the 1-bit FA designed in the previous section to build an N-Bit adder. The design will be parameterizable, which means that it can be scaled to N bits by changing a single **generic** parameter in the entity declaration.

1. Use the 1-Bit FA design to build a parameterizable N-Bit ripple-carry adder (**RCA**). The architecture body of the RCA will be made up of multiple FA’s connected together as components to function as a RCA. Before you can use the FA within the architecture of the RCA you must declare the FA as a **component** in the declaration section of your architecture. You will also have to declare some signals to interconnect your components. You will use the **Port Map** and **For-Generate** statements to instantiate multiple copies of the FA within the RCA architecture. Use the following entity declaration for the RCA (notice that the default width is 16 bits):
Section C – Behavioural N-Bit Adder

In this section you will create a behavioural model of an N-Bit adder. As before, the design will be parameterizable, which means that it can be scaled to N bits by changing a single generic parameter in the entity declaration.

1. Use the following entity declaration and code the architecture for the N-Bit adder. Describe your model at the behavioural level and use the “+” VHDL operator to perform the addition.

```vhdl
entity adder is
    generic (width : integer := 16);
    port (a, b: in std_logic_vector(width-1 downto 0);
        sum: out std_logic_vector(width downto 0)
    );
end adder;
```

2. Perform a behavioural simulation on a 12-bit adder using your structural model. Use smart test cases here. Submit an annotated waveform describing your test cases and methodology. Multiple test cases can be included on a single waveform printout. Clearly indicate (annotate) what is happening at each point in the simulation. Annotations will help you describe your test cases. Please make sure that you set the radix of each waveform signal to hexadecimal. Attach all relevant waveforms to your design summary. Briefly describe your testing strategy.

3. Synthesize your design.

4. Static timing information will be available after this step. Static timing analysis measures static worse-case delays between signals based on a net-list implementation, without the need to simulate the timing every case of inputs. Xilinx ISE performs the static analysis every time a design is implemented, and leaves the information in a report.

5. Analyze the final implementation of your design in the FPGA Editor.

6. Perform post-place and route simulations and verify the correctness of your implementations.
2. Perform a **behavioural** simulation on a **12-bit adder** using your behavioural model. **Use smart test cases here.** Submit an annotated waveform describing your test cases and methodology. Multiple test cases can be included on a single waveform printout. Clearly indicate (annotate) what is happening at each point in the simulation. Annotations will help you describe your test cases. Please make sure that you set the radix of each signal to hexadecimal. Attach all relevant waveforms to your design summary. Briefly describe your testing strategy.

3. Synthesize your design.

4. Perform **post-place and route** simulations and verify the correctness of your implementations.

5. Compare the designs of structural N-bit adder and behavioural N-bit adder in FPGA editor.

**Section D**

Repeat the experiments in this lab for a behavioural parameterizable multiplier. Create a test bench for a 12-bit multiplier to start with. What is the largest multiplier that will fit on the FPGA?

**Section E**

Code a behavioural and structural adder supporting different sizes for each port (a, b, and sum). Assume size of sum \( \geq a \geq b \).

**Demonstration**

You are required to demonstrate the waveforms for each section and the FPGA editor for sections B and C.

**Questions**

1. Assume all of the logic gates from your pre-lab schematics have a worst case delay of 0.25 ns (regardless of input size, gate type, etc). What is the worst case delay for your 12-bit ripple-carry adder?
2. Briefly compare the differences between the structural and behavioural VHDL code for the N-Bit adder in terms of level of abstraction, number of lines, etc. Which one works faster?

3. Briefly compare the behavioural simulation and the post-place and route simulation for each of the designs. When you ran a post-place and route simulation you should have observed that the output value goes through a range of incorrect values before stabilizing at the correct one. Why is this?

4. From map report, compare the numbers of slices and LUTs used in 12-bit behavioral adder and 12-bit structural adder. Review how the two 12-bit designs were placed and routed in the FPGA editor. Identify the major differences between the two implementations. You will notice structural and behavioural adder use different FPGA hardware resources. Which one uses less hardware and why? (attach part of the map report that shows number of used resources)

5. Explain what is meant by slices and CLBs in Xilinx FPGAs.

6. For sections A, B and C, answer the following questions: (support your answer by attaching part of the simulator’s timing report that you have used)
   a. Where is the critical path in the circuit?
   b. Find the static timing analysis report created during the implementation process and determine the worse case delay.
   c. Compare the maximum delay for sections B and C. Which one is faster and why?

**Documentation**

You are required to submit a design summary for this lab. Consult the course website for details of what is required.