VHDL Assignment #5: Sequential Statements and a 4-Bit Comparator

1 Instructions

- TA in charge: Arish Yaseen (arish.yaseen@mail.mcgill.ca) - please utilize discussion boards on myCourses for questions as much as possible.
- Due date: Friday, November 11, 2022 by 11:59 pm EDT.
- Submission is in teams using myCourses (only one team member submits). In the report, provide the names and McGill IDs of the team members.
- Late submissions will incur penalties as described in the course syllabus.

2 Learning Outcomes

After completing this assignment you should know how to:

- Describe a circuit using sequential assignment statements
- Test digital circuits on the Altera DE1-SoC board
- Use the sliding switches on the Altera DE1-SoC board to specify the inputs to your circuits
- Use the LEDs on the Altera DE1-SoC board to visualize the outputs of your circuit
- Describe sequential elements in VHDL
- Design a clock divider circuit
- Design a 3-bit counter in VHDL
- Perform functional simulation of the counter using ModelSim

3 Introduction

In this VHDL assignment, you will describe a 4-bit comparator circuit in VHDL and test it on the Altera DE1-SoC board using LEDs as outputs and sliding switches as inputs. You will also learn how to describe storage elements and sequential logic circuits in VHDL. Then, you will design a 3-bit up-counter and simulate the counter using ModelSim.

If you need any help regarding the lab materials, you can

- Ask the TA for help during lab sessions and office hours.
- Refer to the text book. In case you are not aware, Appendix A “VHDL Reference" provides detailed information on VHDL.
- You can also refer to the tutorial on Quartus and ModelSim provided by Intel (click here for Quartus and here for ModelSim).
- Refer to the DE1-SoC User Manual (in the Content tab on myCourses).

It is highly recommended that you first try to resolve any issue by yourself (refer to the textbook and/or the multitude of VHDL resources on the Internet). Syntax errors, especially, can be quickly resolved by reading the error message to see exactly where the error occurred and checking the VHDL Reference or examples in the textbook for the correct syntax.

4 Sequential Assignment Statements

In the previous VHDL assignments, you have learned several types of assignment statements such as simple assignment statements, selected assignment statements, and conditional assignment statements. All of these statements have the property that the order in which they appear in the VHDL code does not affect the meaning of the code, that is, it does not affect the synthesized circuit. For this reason, these statements are usually referred to as concurrent assignment statements.

VHDL also has a second category of statements, called sequential assignment statements, for which the order of the statements may affect the meaning of the code. In the following, we briefly discuss this new kind of statements. *They are described in more detail in Sections 6.6.6 and 6.6.7 of the textbook. Please read these sections before you attempt this assignment.* Even more details can be found in Appendix A.9 of the textbook.

The two main types of sequential assignment statements are: if-then-else statements and case statements. These sequential assignment statements must be placed inside a block, called a *process* block. The PROCESS block starts with the keyword "PROCESS". It has an optional label and a sensitivity list. Following the PROCESS keyword is the statement "BEGIN". Any statement between "BEGIN" and the "END PROCESS label;" are sequential statements.

```
label: PROCESS (sensitivity list)
BEGIN
--sequential statements
END PROCESS label;
```

The sensitivity list contains signals. Unlike concurrent statements which are executed all the time, the process block is activated only when one of the signals in its sensitivity list changes its value. Once activated, the statements inside the process block are executed *sequentially* in the order they appear in the block. There are two things to note: *(i)* Any assignments made to signals inside the process are not visible outside the process until all of the statements in the process have been evaluated; *(ii)* in case of multiple assignments to the same signal, only the last one determines the final value of the signal. Also note that VHDL allows multiple processes to be described within the same architecture.

4.1 IF-THEN-ELSE Statements

IF-THEN-ELSE statements are used to modify the behavior of your function depending on whether one or more conditions hold. The syntax of IF-THEN-ELSE statements is shown below. Note that the "END IF" must be separated by a space.

```
IF condition THEN
-- sequential statements
ELSE
-- sequential statements
END IF;
```

You may have nested IF and ELSE as follows:

```
IF condition THEN
-- sequential statements
ELSIF condition THEN
-- sequential statements
ELSIF condition THEN
-- sequential statements
ELSE
-- sequential statements
END IF;
```

In this structure only one of the branches is executed depending on the condition. Even if there are several conditions which are true in this structure only the first TRUE condition will be followed. After the execution of the sequential statements within the first true condition the statements after the END IF will be executed next. Therefore it is very important to write the order of IF blocks and ELSIF blocks according to your desired behavior.

4.2 CASE Statements

CASE statements consider all of the possible values that an object can take and execute a different branch depending on the current value of the object as shown next.

```
CASE object IS
WHEN value1 =>
-- statements
WHEN value2 =>
-- statements
WHEN value3 =>
-- statements
-- etc.
WHEN OTHERS =>
-- statements
END CASE;
```

Note that the CASE statement must include a WHEN clause for each of the possible values of object. This necessitates a WHEN OTHERS clause if some of possible values of object are not covered by WHEN clauses.

5 4-Bit Comparator

Comparators are a useful type of arithmetic circuits that compare the relative sizes of two binary numbers. In this assignment you will implement a 4-bit comparator circuit that takes two 4-bit unsigned inputs A and B, and determines which one of the cases $A = (B+1)$, $A < (B+1)$, $A \leq (B+1)$, $A > (B+1)$, $A \geq (B+1)$ holds. It should detect the occurrence of overflow when performing $B+1$, i.e., detect if $B+1$ requires more than 4 bits. Use the following entity declaration for your implementation of the comparator circuit.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity firstname_lastname_comparator is
Port (  A, B              : in  std_logic_vector(3 downto 0);
AgtBplusOne       : out std_logic;
AgteBplusOne      : out std_logic;
AltBplusOne       : out std_logic;
AlteBplusOne      : out std_logic;
AeqBplusOne       : out std_logic;
overflow          : out std_logic);
end firstname_lastname_comparator;
```

Note that in case of overflow when performing $B+1$, the comparator circuit outputs 1 for the `overflow` signal while the remaining signals (i.e., `AgtBplusOne`, `AgteBplusOne`, `AltBplusOne`, `AlteBplusOne` and `AlteBplusOne`) are set to 0. Otherwise, the circuits outputs proper values for `AgtBplusOne`, `AgteBplusOne`, `AltBplusOne`, `AlteBplusOne` and `AeqBplusOne` signals according to $A > (B+1)$, $A \geq (B+1)$, $A < (B+1)$, $A \leq (B+1)$, $A = (B+1)$, respectively, while the `overflow` signal is set to 0. For example, if $A = 9_{10}$ and $B = 5_{10}$ then both `AgtBplusOne`, `AgteBplusOne` should be 1, while the rest (including `overflow`) should be 0. The `firstname_lastname` in the name of the entity is the name of one of the students in your group.

To describe your comparator in VHDL, use sequential statements in a single process block. Once you have described your circuit in VHDL, you should test your design on the FPGA board. Similar to the VHDL Assignment #4, the inputs of the comparator circuit are provided using the slider switches; you will need to use eight slider switches. To visualize the output signals of the comparator circuit, use the LEDs located right above the slider switches on the FPGA board. Note that you will need to use six LEDs (one for each output signal). When an output signal is set to '1', the corresponding LED turns on; otherwise, it will be off. Afterwards, perform the pin assignments and program the FPGA by following the instructions provided in VHDL Assignement #4. Note that the pin locations of LEDs are different from those of 7-segment LEDs. The pin assignments for the LEDs are given in Figure 3.17 on p. 25 of the Altera board manual. Once completed, test the functionality of your comparator circuit for different input values using the LEDs and slider switches.

6 Questions

1. Briefly explain your VHDL code implementation of all circuits.
2. Given that $A = 5_{10}$, provide a **separate** simulation plot that demonstrates all possible cases for the 4-bit comparator, including a separate plot for the case where overflow occurs. A total of four plots should be included. Explain each plot and mark all inputs and outputs clearly.
3. Perform timing analysis (slow 1,100 mV model) of the 4-bit comparator and find the critical path(s) of the circuit. What is the delay of the critical path(s)?
4. Report the number of pins and logic modules used to fit your 4-bit comparator design on the FPGA board.

7 Deliverables

You are required to submit the following deliverables on MyCourses. Please note that a single submission is required per group (by one of the group members).

- Lab report. The report should include the following parts: (1) Names and McGill IDs of group members, (2) an executive summary (short description of what you have done in this VHDL assignment), (3) answers to all questions in previous section (if applicable), (4) legible figures (screenshots) of schematics and simulation results, where all inputs, outputs, signals, and axes are marked and visible, (5) an explanation of the results obtained in the assignments (mark important points on the simulation plots), and (6) conclusions. Note - students are encouraged to take the reports seriously, points will be deducted for sloppy submissions. Please also note that even if some of the waveforms may look the same, you still need to include them separately in the report.
- Project files. Create a single .zip file named `VHDL#_firstname_lastname` (replace `#` with the number of the current VHDL assignment and `firstname_lastname` with the name of the submitting group member). The .zip file should include the working directory of the project.