VHDL Assignment #3: Design and Simulation of Adder Circuits

1 Instructions

- Due date: Friday, October 23, 2020 by 11:59pm.
- 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 lab you should know how to:

- Use VHDL statements to implement different adder circuits
- Write efficient VHDL codes
- Perform functional simulation
- Perform basic timing analysis

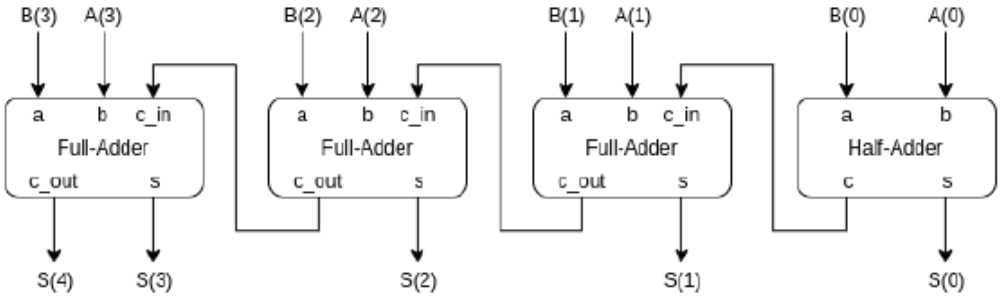
You will also have a better understanding of the concept of synthesis of logic circuits.

3 VHDL Description of Adder Circuits

In this assignment, you will be asked to perform the design and simulation of the following two adder circuits: (a) a 4-bit ripple-carry adder; and (b) a one-digit binary coded decimal (BCD) adder. Details of the assignments are described below.

3.1 Ripple-Carry Adder (RCA)

In this section, you will implement a structural description of a 4-bit ripple-carry adder using basic addition components: half-adders and full-adders.



3.1.1 Structural Description of a Half-Adder in VHDL

A half-adder is a circuit that takes two binary digits as inputs, and produces the result of the addition of the two bits in the form of a sum and carry signals. The carry signal represents an overflow into the next digit of a multi-digit addition. Using the following entity definition for your VHDL code, implement a structural description of the half-adder.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity half_adder is
  port (a: in std_logic;
```

```
        b: in std_logic;
        s: out std_logic;
        c: out std_logic);
end half_adder;
```

After you have described your structural style of the half-adder in VHDL, you are required to test your circuit. Write a testbench code and perform an exhaustive test of your VHDL description of the half-adder.

3.1.2 Structural Description of a Full-Adder in VHDL

Unlike the half-adder, a full-adder adds binary digits while accounting for values carried in (from a previous stage addition). Write a structural VHDL description for the full-adder circuit using the half-adder circuit that you designed in the previous section. Use the following entity declaration for your structural VHDL description of the full-adder.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity full_adder is
  port (a: in std_logic;
        b: in std_logic;
        c_in: in std_logic;
        s: out std_logic;
        c_out: out std_logic);
end full_adder;
```

After you have described your circuit in VHDL, write a testbench code and perform an exhaustive test of your VHDL description of the full-adder.

3.1.3 Structural Description of a 4-bit Ripple-Carry Adder (RCA) in VHDL

Using the half-adder and full-adder circuits implemented in the two previous sections, implement a 4-bit carry-ripple adder. Write a structural VHDL code for the 4-bit RCA using the following entity declaration.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity rca_structural is
port ( A: in std_logic_vector (3 downto 0);
       B: in std_logic_vector (3 downto 0);
       S: out std_logic_vector (4 downto 0));
end rca_structural;
```

Note that $S(4)$ contains the carry-out of the 4-bit adder. After you have described your circuit in VHDL, write a testbench code and perform an exhaustive test of your VHDL structural description of the 4-bit RCA.

3.1.4 Behavioral Description of a 4-bit RCA in VHDL

In this part, you are required to implement the 4-bit RCA using behavioral description. One way to obtain a behavioral description is to use arithmetic operators in VHDL (*i.e.*, "+"). Write a behavioral VHDL code for the 4-bit RCA using the following entity declaration for your behavioral VHDL description.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity rca_behavioral is
port ( A: in std_logic_vector (3 downto 0);
       B: in std_logic_vector (3 downto 0);
       S: out std_logic_vector (4 downto 0));
end rca_behavioral;
```

After you have described your circuit in VHDL, write a testbench code and perform an exhaustive test of your VHDL behavioral description of the 4-bit RCA.

3.2 VHDL Description of a One-Digit BCD Adder

In this section, you will implement a one-digit BCD adder in VHDL. A one-digit BCD adder adds two four-bit numbers represented in a BCD format. The result of the addition is a BCD-format 4-bit output, representing the decimal sum, and a carry that is generated if this sum exceeds a decimal value of 9 (see slides of Lecture #11).

3.2.1 Structural Description of a BCD Adder in VHDL

In this part, you are required to implement the BCD adder using structural description. You are allowed to use either behavioral or structural style of coding for your implementation. Using the following entity definition. Use the following entity declaration.

```
library IEEE;
use IEEE.STD_LOGIC_11164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity bcd_adder_structural is
port ( A: in std_logic_vector (3 downto 0);
       B: in std_logic_vector (3 downto 0);
       S: out std_logic_vector (3 downto 0);
       C: out std_logic);
end bcd_adder_structural;
```

After you have implemented the one-digit BCD adder in VHDL, you are required to test your circuit. Write a testbench code and perform an exhaustive test of your VHDL structural description of the one-digit BCD adder.

3.2.2 Behavioral Description of a BCD Adder in VHDL

In this part, you are required to implement the BCD adder using behavioral description. You are encouraged to base your code on the VHDL code in Section 5.7.3 of the textbook, so that you learn about conditional signal assignments (these are explained in detail in the same section as well as in the Appendix in Section A.7.4). Use the following entity declaration.

```
library IEEE;
use IEEE.STD_LOGIC_11164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity bcd_adder_behavioral is
port ( A: in std_logic_vector (3 downto 0);
B: in std_logic_vector (3 downto 0);
S: out std_logic_vector (3 downto 0);
C: out std_logic);
end bcd_adder_behavioral;
```

After you have implemented the one-digit BCD adder in VHDL, you are required to test your circuit. Write a testbench code and perform an exhaustive test for your VHDL behavioral description of the one-digit BCD adder.

3.3 Timing Analysis and Critical Path

In this part you will learn how to perform timing analysis in the EDAPlayground platform. You will also be able to find the critical path of a given circuit.

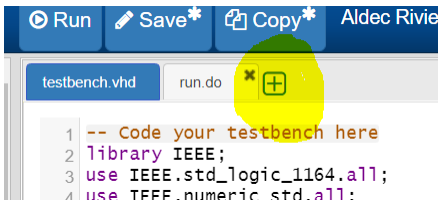
The critical path is the longest path in the circuit which defines the speed of the circuit. The speed of a digital circuit is measured in terms of latency and throughput. Latency is the time needed for the circuit to produce an output for a given input (*i.e.*, the total propagation delay (time) from the input to the output), and it is expressed in units of time. Alternatively, throughput refers to the rate at which data can be processed. In this assignment, we only consider the latency as a metric to measure the speed of the circuit.

In general, digital circuits are subject to timing constraints dictated by the target application. Whether a circuit meets these timing constraints can only be known after the circuit is synthesized. After synthesis is performed, the designer can analyze the circuit to determine whether timing constraints were satisfied using the term **slack**.

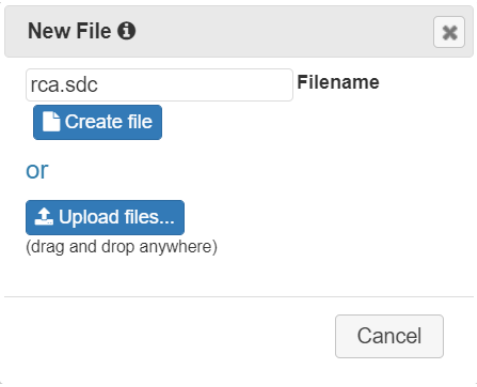
Slack is the margin by which a timing requirement is met or not met; it is the difference between the required arrival time and the actual arrival time. A positive slack value indicates the margin by which a requirement was met. A negative slack value indicates the margin by which a requirement was not met.

To simulate timing constraints, we can add time constraint files to our design. Here, we will use the 4-bit RCA that we designed in Section 3.1.3 (structural description) as an example. **Please note that for your assignment submission, you will be required to analyze the timing of the BCD adder, NOT the RCA that is used here as an example only.**

Create a new file by clicking on the (+) icon (next to testbench.vhd).



Name the file **rca.sdc**. Note that sdc stands for Synopsis Design Constraints.



The maximum delay can be specified in the sdc file using the following command:

```
set_max_delay -from [get_ports <name_of_input_port>] -to [get_ports <name_of_output_port>] <time
    in nano seconds>
```

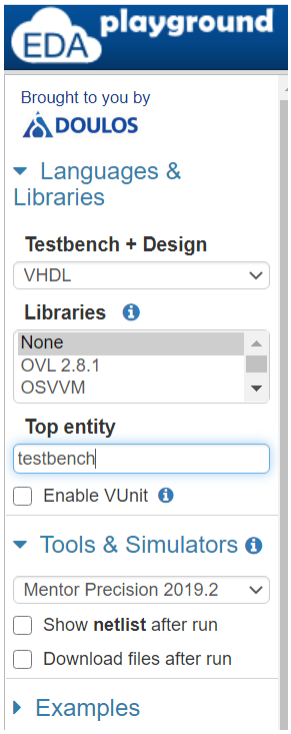
In this example, we set the delay to be 20 nsec (nano seconds) from the input A to the output S (for all possible paths), and 15 nsec from the input B to the output S (for all possible paths).

```
set_max_delay -from [get_ports A[*]] -to [get_ports S[*]] 20
set_max_delay -from [get_ports B[*]] -to [get_ports S[*]] 15
```

Update your **run.do** file to include time delay related information. The updated **run.do** file should look like this:

```
add_input_file rca.sdc
setup_design -manufacturer Xilinx -family Artix-7 -part 7A100TCSG324
foreach arg $::argv {
  add_input_file $arg
}
compile
synthesize
auto_write precision.v
report_output_file_list
report_area
report_timing
exec cat precision.v
```

Make sure that your system settings look like this:



Click run (remember to set the simulator to *Mentor Precision 2019.2* for this part), and you will be able to see the timing analysis information in the compiler log. In this example, the log looks like this:

Log | Share

```
# Info: Critical path #1, (path slack = 15.019):, Logic Levels = 3
# Info: SOURCE CLOCK: name: <not found>  Path is min/max delay constrained
# Info: NAME                GATE       DELAY    ARRIVAL DIR  FANOUT LEVEL
# Info: A(3)              (port)                 0.000   dn
# Info: A(3)              (net)        0.000                    1     0
# Info: A_ibuf(3)/I       IBUF                   0.000   dn
# Info: A_ibuf(3)/O       IBUF         1.298     1.298   dn
# Info: A_int(3)          (net)        0.335                    2     1
# Info: ix48511z57698/I1 LUT5                    1.633   dn
# Info: ix48511z57698/O  LUT5          0.124     1.757   dn
# Info: nx48511z1         (net)        0.333                    1     2
# Info: S_obuf(4)/I       OBUF                   2.090   dn
# Info: S_obuf(4)/O       OBUF         2.891     4.981   dn
# Info: S(4)              (net)        0.000                    0     3
# Info: S(4)              (port)                 4.981   dn
# Info:                 Minmax delay constraint:     20.000
# Info:                 Source clock delay:       -    0.000
# Info:                 Dest clock delay:         +    0.000
# Info:                                          -----------
# Info:                 Edge separation:             20.000
# Info:                 Setup constraint:         -    0.000
# Info:                                          -----------
# Info:                 Data required time:          20.000
# Info:                 Data arrival time:        -    4.981   ( 86.59% cell delay, 13.41% net delay )
# Info:                                          -----------
# Info:                 Slack:                       15.019
```

For example, the critical path from input *A(3)* to output *S(4)* exhibits a positive slack of 15.019 ns, which means that the circuit meets the requirement.

4 Deliverables

Once completed, you are required to submit a report explaining your work in detail, including answers to the questions related to this lab. You will find the questions in the next section. You must also submit all of your VHDL, run.do, and .sdc files along with Log content of the synthesized circuits and timing analysis. You must also submit all of your testbench files. If you fail to submit any part of the required deliverables, you will incur grade penalties as described in the syllabus.

5 Questions

Please note that even if some of the waveforms may look the same, you still need to include them separately in the report.

1. Briefly explain your VHDL code implementation of all circuits.
2. Show representative simulation plots of the half-adder circuit for all possible input values.
3. Show representative simulation plots of the full-adder circuit for all possible input values.
4. Show representative simulation plots of both behavioral and structural descriptions of the 4-bit RCA for all possible input values.
5. Show representative simulation plots of both behavioral and structural descriptions of the one-digit BCD adder circuit for all possible input values.
6. Perform timing analysis of the one-digit BCD adder and find the critical path(s) of the circuit. What is the delay of the critical path(s)? Assuming that an additional delay of 10 ns can be added to the critical path(s) due to wiring, what should be the set_max_delay command to ensure a positive slack of 5 ns for the circuit (write the complete set_max_delay command).
7. Report the number of pins and logic modules used to fit your BCD adder designs on the FPGA board.

	RCA		One-digit BCD adder	
	Structural	Behavioral	Structural	Behavioral
Logic Utilization (in LUTs)				
Total pins				