VHDL Assignment #4: Critical Path and Getting Started with Altera DE1-SoC Board

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, October 28, 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:

- Use Quartus Timing Analyzer tools to find the critical path of a ripple-carry adder designed in a previous assignment
- Use CAD tools to design and implement a BCD to 7-segment LED decoder on the Altera DE1-SoC board
- 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 7-segment LED display on the Altera DE1-SoC board to display the output of your circuits

3 Introduction

In this assignment, you will use a previously designed BCD adder circuit, you will implement and test the circuit on the Altera DE1-SoC board. You will learn how to work with the Altera DE1-SoC board, use switches, and the 7-segment LED display.

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).

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.

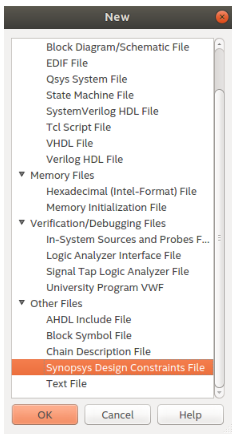
4 Critical Path of Digital Circuits

In this part, you will learn how to use the Quartus CAD tool to determine the delay of a given path in digital circuits. To this end, in this section, we use the ripple-carry adder circuit (that you designed in VHDL assignment #3) as the "circuit under examination".

Follow the instructions described in VHDL Assignment #1 to create a project. Make sure to select the **Cyclone V** family of FPGAs, with the following part number: **5CSEMA5F31C6** when creating a project. Once created, import the VHDL description of your digital circuit into the project and compile it to make sure there are no syntax errors in your design.

The critical path is the longest path in the circuit and limits the speed of the circuit speed. 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 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.

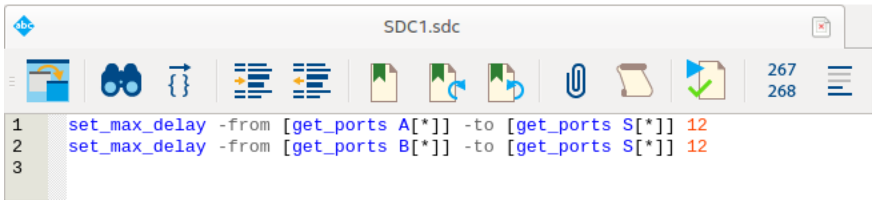
To insert timing constraints in Quartus, select "Synopsys Design Constraints File" from the "File–>New" menu.


New
Block Diagram/Schematic File
EDIF File
Qsys System File
State Machine File
SystemVerilog HDL File
Tcl Script File
VHDL File
Verilog HDL File
Memory Files
Hexadecimal (Intel-Format) File
Memory Initialization File
Verification/Debugging Files
In-System Sources and Probes F...
Logic Analyzer Interface File
Signal Tap Logic Analyzer File
University Program VWF
Other Files
AHDL Include File
Block Symbol File
Chain Description File
Synopsys Design Constraints File
Text File
OK
Cancel
Help

The maximum delay can be specified in the Synopsys Design Constraints File using the following command:

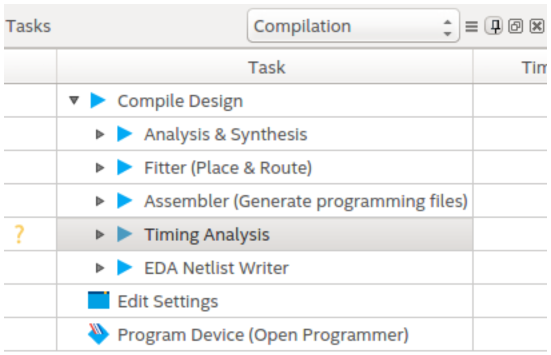
set_max_delay -from [get_ports <name_of_input_port>] -to [get_ports <name_of_output_port>] <time in nano seconds>

For example, we can specify a maximum delay of 12 ns for all the possible paths from the inputs of the ripple-carry adder to its outputs as shown below.

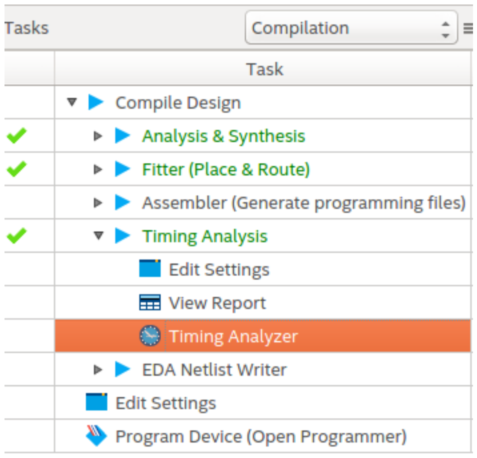


```
set_max_delay -from [get_ports A[*]] -to [get_ports S[*]] 12
set_max_delay -from [get_ports B[*]] -to [get_ports S[*]] 12
```

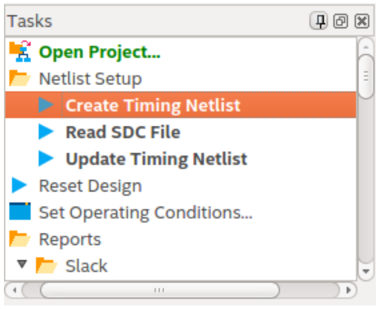
Once the timing constraints are inserted, save the file with the name "firstname_lastname_sdc.sdc". Recompile your design by double clicking on "Timing Analysis" in the Tasks window of Quartus. *Before recompilation, make sure that the .sdc file is added to the project.*



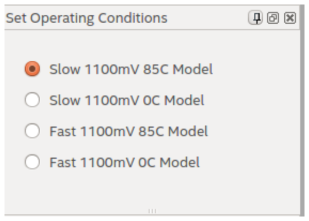
The Timing Analyzer will read the .sdc file and use the constraint information when performing timing analysis. Once a green check mark appears next to “Timing Analysis”, double click on “Timing Analyzer” under “Timing Analysis” to open the Timing Analyzer tool.



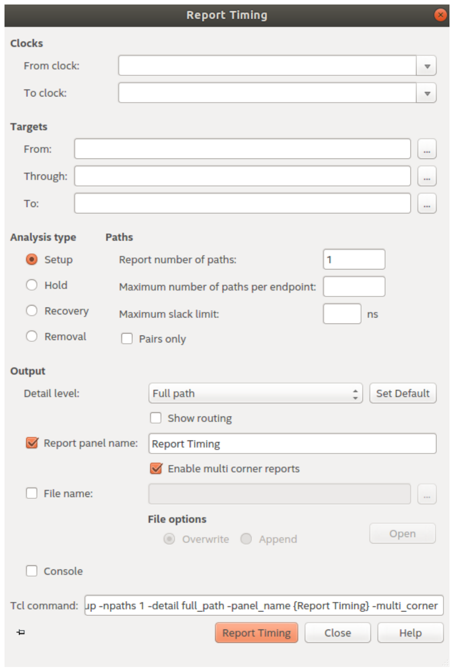
In the Tasks window of the Timing Analyzer tool, double click on “Create Timing Netlist”, “Read SDC File” and “Update Timing Netlist” icons, respectively.



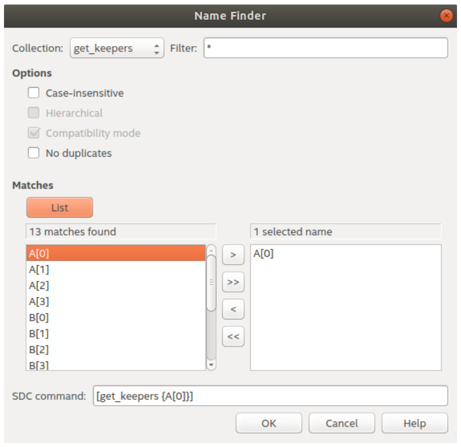
Once completed, the colors of the icons will become green. Before measuring the delay of your design, you should specify the operating conditions of the FPGA device. This can be simply set by selecting one of the possible operating conditions listed in the “Set Operating Conditions” in the Timing Analyzer tool.



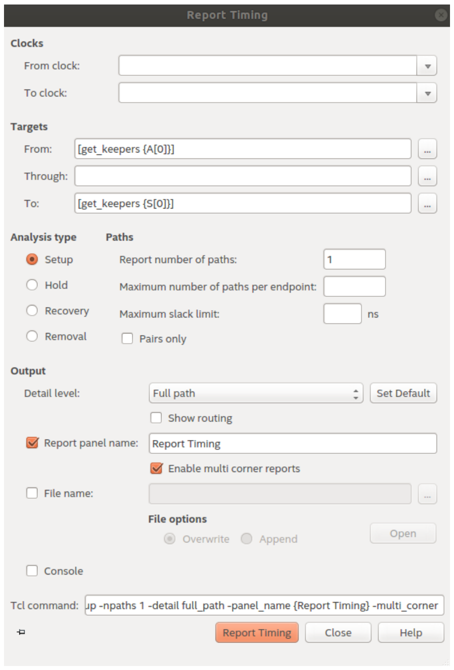
To report the delay of different paths from inputs to outputs, select “Report Timing . . . ” from the “Reports –> Custom Reports” menu.



Since no clock signal is associated with your design, we only specify the beginning of the target path(s) by clicking on “From” and the end of the target path/paths by clicking on “To” under the section labeled as “Targets”. In the “Name Finder” window that pops up, click on “List” to list all the I/O signals of your design. Select (double click on) a signal (or signals) to determine the beginning of the path that you want to examine and click “OK”. Repeat the same procedure to determine the end of the path.



As an example, we can examine the path from the LSB of the input A (*i.e.*, A(0)) to the LSB of the output S (*i.e.*, S(0)) as shown below.



Report Timing

Clocks

From clock:

To clock:

Targets

From: [get_keepers {A[0]}]

Through:

To: [get_keepers {S[0]}]

Analysis type: [✓ Setup] Hold Recovery Removal

Paths

Report number of paths: 1

Maximum number of paths per endpoint:

Maximum slack limit: ns

Pairs only

Output

Detail level: Full path

Show routing

[✓] Report panel name: Report Timing

[✓] Enable multi corner reports

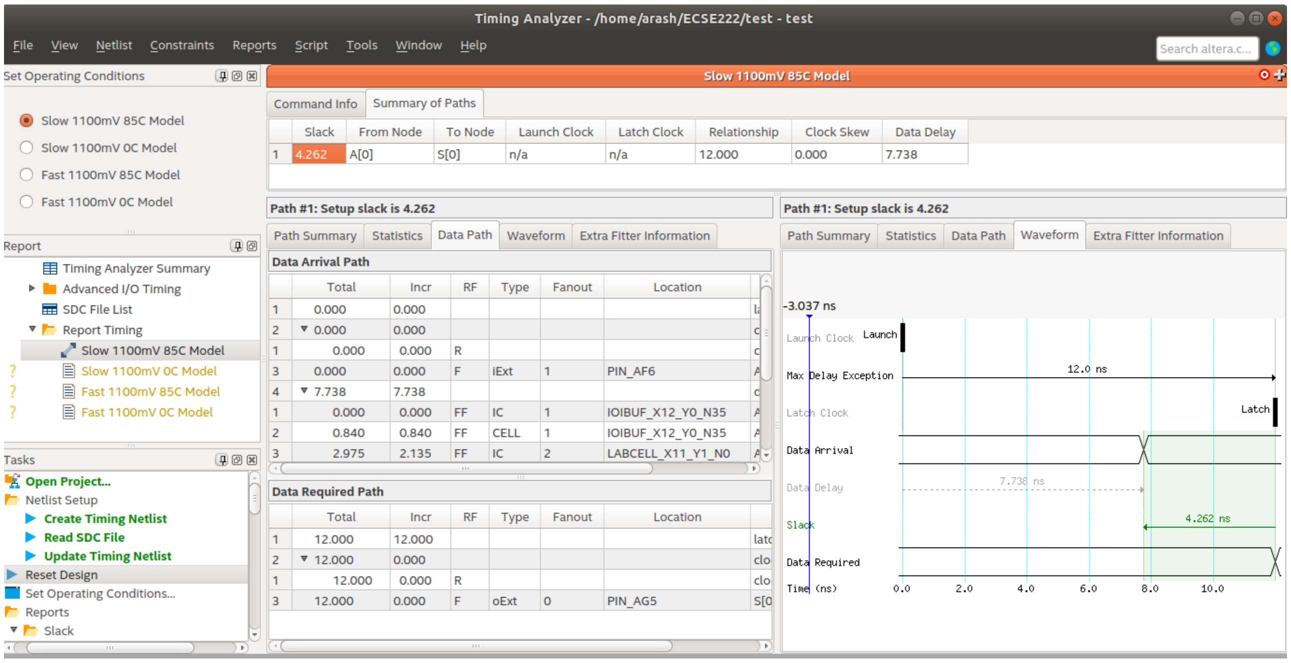
File name:

File options: Overwrite Append

Console

Tcl command: ıp -npaths 1 -detail full_path -panel_name {Report Timing} -multi_corner

Now, click on "Report Timing" to obtain timing information for the specified path. The delay of the specified path is denoted "Data Delay" in the section entitled "Summary of Paths". The positive value of the slack denotes the difference between the delay of the path (*i.e.*, Data Delay) and the timing constraint inserted in the .sdc file (*i.e.*, 12 ns). This information is also visualized under the "waveform" tab.



Timing Analyzer - /home/arash/ECSE222/test - test

Set Operating Conditions

[✓ Slow 1100mV 85C Model] Slow 1100mV 0C Model Fast 1100mV 85C Model Fast 1100mV 0C Model

Slow 1100mV 85C Model

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	4.262	A[0]	S[0]	n/a	n/a	12.000	0.000	7.738

Path #1: Setup slack is 4.262

Data Arrival Path

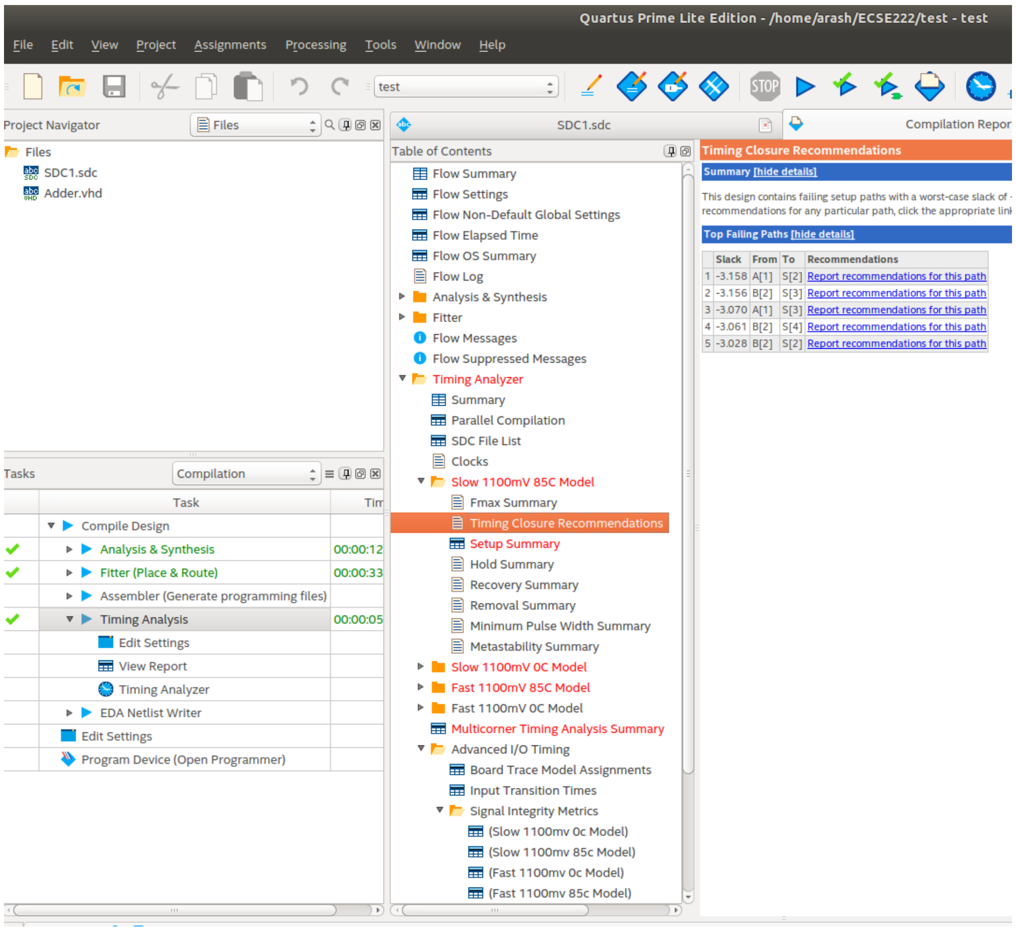
	Total	Incr	RF	Type	Fanout	Location	
1	0.000	0.000					la
2	▼ 0.000	0.000					c
1	0.000	0.000	R				c
3	0.000	0.000	F	iExt	1	PIN_AF6	A
4	▼ 7.738	7.738					c
1	0.000	0.000	FF	IC	1	IOIBUF_X12_Y0_N35	A
2	0.840	0.840	FF	CELL	1	IOIBUF_X12_Y0_N35	A
3	2.975	2.135	FF	IC	2	LABCELL_X11_Y1_N0	A

Data Required Path

	Total	Incr	RF	Type	Fanout	Location	
1	12.000	12.000					latc
2	▼ 12.000	0.000					clo
1	12.000	0.000	R				clo
3	12.000	0.000	F	oExt	0	PIN_AG5	S[0

Path #1: Setup slack is 4.262

To find the critical path of your design, you should examine all possible paths from all inputs to all outputs and find the one with the longest delay. However, using this "exhaustive search" method is very time consuming as the number of I/O ports increases. To limit the number of paths under examination, we reduce the target delay value in the .sdc file so that a timing violation occurs. For instance, we reduce the target delay constraint of the "circuit under examination" from 12 ns to 5 ns and recompile the design by double clicking on "Timing Analysis" in Quartus. In case of a timing violation, Quartus determines the violating path(s) in the compilation summary of the "Timing Analyzer" tool.


Quartus Prime Lite Edition - /home/arash/ECSE222/test - test
File Edit View Project Assignments Processing Tools Window Help
Project Navigator
Files
SDC1.sdc
Adder.vhd
SDC1.sdc
Compilation Report
Table of Contents
Timing Analyzer
Slow 1100mV 85C Model
Timing Closure Recommendations
Setup Summary

Timing Closure Recommendations

Summary [hide details]

This design contains failing setup paths with a worst-case slack of -
recommendations for any particular path, click the appropriate link

Top Failing Paths [hide details]

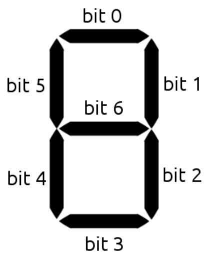
	Slack	From	To	Recommendations
1	-3.158	A[1]	S[2]	Report recommendations for this path
2	-3.156	B[2]	S[3]	Report recommendations for this path
3	-3.070	A[1]	S[3]	Report recommendations for this path
4	-3.061	B[2]	S[4]	Report recommendations for this path
5	-3.028	B[2]	S[2]	Report recommendations for this path

	Task	Time
	Compile Design	
✓	Analysis & Synthesis	00:00:12
✓	Fitter (Place & Route)	00:00:33
	Assembler (Generate programming files)	
✓	Timing Analysis	00:00:05
	Edit Settings	
	View Report	
	Timing Analyzer	
	EDA Netlist Writer	
	Edit Settings	
	Program Device (Open Programmer)	

In this approach, our search for the critical path will now be limited to the violating paths. Examining the violating paths in the "Timing Analyzer" tool will, therefore, determine the critical path.

5 BCD to 7-Segment LED Decoder

A 7-segment LED display includes 7 individual LED segments, as shown below. By turning on different segments together, we can display characters or numbers. There are six of these displays on the DE1-SoC board, which you will use later to display the result of your full implementation of the adder.



We will need a circuit to drive the 7-segment LEDs on the DE1 board, called 7-segment decoder. It takes as input a 4-bit BCD-formatted code representing the 10 digits between 0 and 9, and generates the appropriate 7-segment display associated with the input code. For many LED displays, including the ones on the DE1 board, segments turn on when their segment inputs are driven low, that is "0" means on and "1" means off. This scheme for the inputs is called "active low." The VHDL code for the 7 segment decoder is provided below.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity seven_segment_decoder is
port (  code            : in std_logic_vector(3 downto 0);
segments_out    : out std_logic_vector(6 downto 0));
end seven_segment_decoder;

architecture decoder of seven_segment_decoder is
begin
WITH code SELECT
segments_out <=
"1000000" WHEN "0000", -- 0
"1111001" WHEN "0001", -- 1
"0100100" WHEN "0010", -- 2
"0110000" WHEN "0011", -- 3
"0011001" WHEN "0100", -- 4
"0010010" WHEN "0101", -- 5
"0000010" WHEN "0110", -- 6
"1111000" WHEN "0111", -- 7
"0000000" WHEN "1000", -- 8
"0010000" WHEN "1001", -- 9
"1111111" WHEN others;
end decoder;
```

In the next section, you will use the 7-segment LED decoder to display the inputs/outputs of the one-digit BCD adder.

6 Wrapper Design

In this part of the lab, you will design a circuit (that is called “wrapper” circuit) that performs addition of two 4-bit BCD-formatted inputs A and B, and displays the inputs as well as the result of the addition in BCD format using the 7-segment LEDs on the DE1-SoC board. You will need, therefore, to use four 7-segment LEDs on the board: the first two 7-segment LEDs will be used to display the inputs A and B, while the other two 7-segment LEDs will be used to display the result of the addition. We need two LED displays for the output as the result may be two BCD digits long. Note that you should use the binary-to-7-segment LED decoder to obtain appropriate 7-segment display codes.

Use the following entity declaration to write a VHDL description of the wrapper circuit. Note that you will need four instances of the `seven_segment_decoder` component and one instance of `firstname_lastname_bcd_adder` component in your VHDL description. You can use the BCD adder that one of the members of your group designed in the previous assignment.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity firstname_lastname_wrapper is
Port (  A, B            : in  std_logic_vector(3 downto 0);
decoded_A       : out std_logic_vector(6 downto 0);
decoded_B       : out std_logic_vector(6 downto 0);
decoded_AplusB  : out std_logic_vector(13 downto 0));
end firstname_lastname_wrapper;
```

where `firstname_lastname` in the name of the entity is the name of one of the students in your group.

The wrapper circuit has two 4-bit inputs, A and B, each representing a BCD digit. The outputs are: a 7-bit display code for A, a 7-bit display code for B, and two 7-bit display codes for the sum A+B.

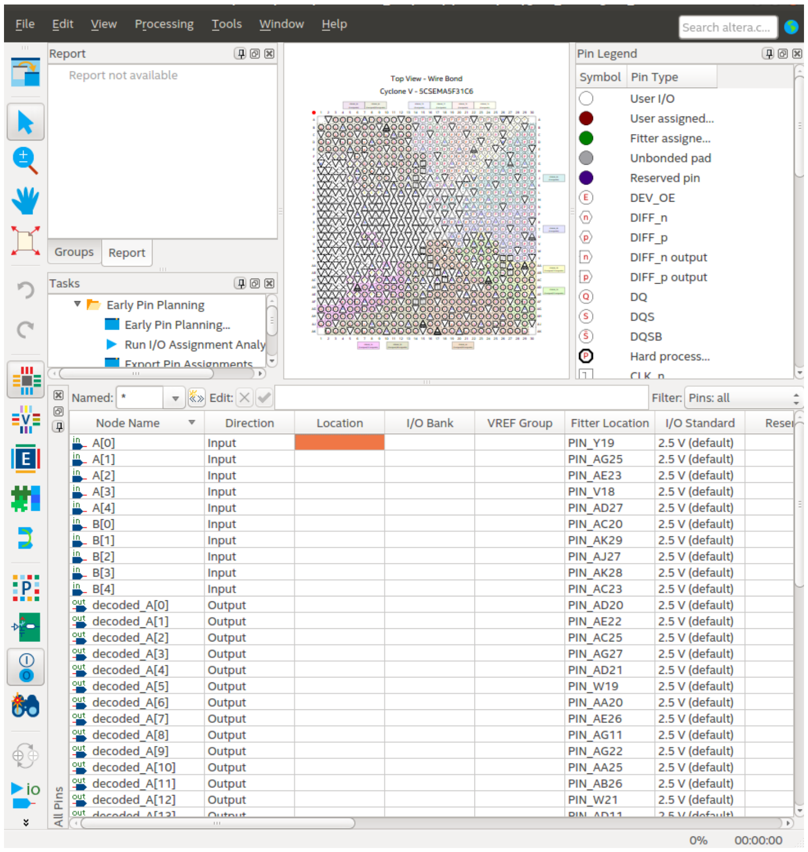
7 Testing the Wrapper on the Altera Board

You will now test the wrapper circuit you designed in Section 6. Follow the instructions described in Assignment #1 to create a project. Make sure to select the **Cyclone V** family of FPGAs, with the following part number: **5CSEMA5F31C6** when creating a project. Once created, import the VHDL description of the wrapper circuit and its components into the project and compile to make sure there are no syntax errors in your design. Once you have compiled the wrapper circuit, it is time to map it onto the target hardware, in this case the Cyclone V chip on the Altera DE1-SoC board.

Since you will now be working with an actual device, you have to consider to which device package pins (physical pins) the various inputs and outputs of the project are connected. The FPGA chip on the board communicates with the outside world by sending and receiving binary signals using physical pins. You can think of these pins as the ports of an VHDL entity. Some of the input pins are connected to the sliding switches on the board, and some of the output pins are connected to the LED displays. In particular, you will want to connect the LED segment outputs from the instances of the `seven_segment_decoder` circuit (*i.e.*, the outputs of the wrapper circuit) to the corresponding pins of the four 7-segment LED displays on the board. See Section 3.6.2 of the DE1-SoC User Manual. The mapping segments of the each 7-segment LED on the board to the pins on the Cyclone FPGA device, is listed in Table 3-9 on page 27 of the DE1-SoC User Manual.

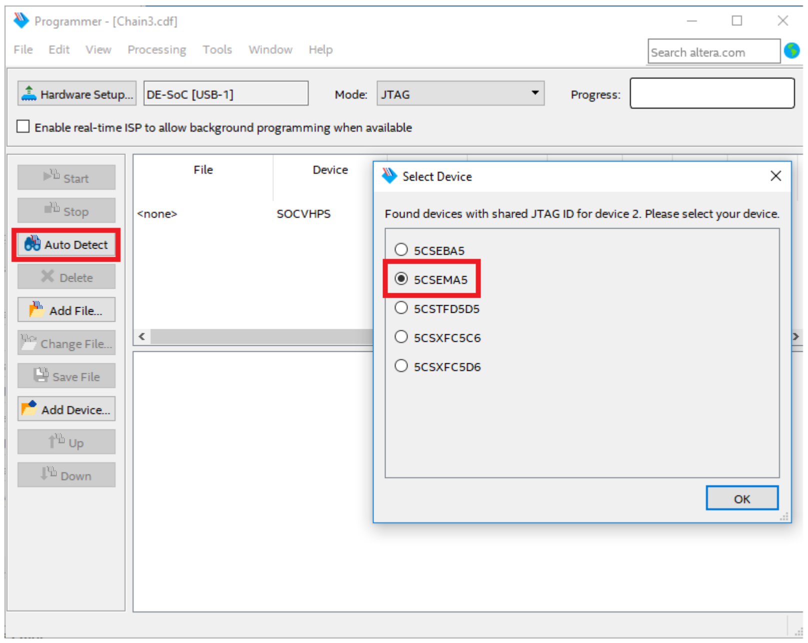
You will also want to connect, for testing purposes, four of the slide switches on the DE1-SoC board to the input A, and another four switches to the input B of the wrapper circuit. See Section 3.6.1 of the DE1-SoC User Manual. The mapping of the slide switches to the FPGA pins is listed in Table 3-6 on page 25 of the DE1-SoC user manual.

You must now notify the compiler to which pins the input and output signals of your wrapper circuit should be connected. For example, if B[0] is connected to switch SW0 (See Fig. 3-15 of the manual on page 24), you should notify the compiler that B[0] is assigned to pin PIN_AB12. You specify the pin assignments for your inputs and outputs by using the **Pin Planner**, which can be done by choosing the Pins item in the **Assignments** menu. See example below. Note that this figure is just an example, and does not correspond to the wrapper circuit that you are supposed to design in this assignment.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reser
A[0]	Input	🟧			PIN_Y19	2.5 V (default)	
A[1]	Input				PIN_AG25	2.5 V (default)	
A[2]	Input				PIN_AE23	2.5 V (default)	
A[3]	Input				PIN_V18	2.5 V (default)	
A[4]	Input				PIN_AD27	2.5 V (default)	
B[0]	Input				PIN_AC20	2.5 V (default)	
B[1]	Input				PIN_AK29	2.5 V (default)	
B[2]	Input				PIN_AJ27	2.5 V (default)	
B[3]	Input				PIN_AK28	2.5 V (default)	
B[4]	Input				PIN_AC23	2.5 V (default)	
decoded_A[0]	Output				PIN_AD20	2.5 V (default)	
decoded_A[1]	Output				PIN_AE22	2.5 V (default)	
decoded_A[2]	Output				PIN_AC25	2.5 V (default)	
decoded_A[3]	Output				PIN_AG27	2.5 V (default)	
decoded_A[4]	Output				PIN_AD21	2.5 V (default)	
decoded_A[5]	Output				PIN_W19	2.5 V (default)	
decoded_A[6]	Output				PIN_AA20	2.5 V (default)	
decoded_A[7]	Output				PIN_AE26	2.5 V (default)	
decoded_A[8]	Output				PIN_AG11	2.5 V (default)	
decoded_A[9]	Output				PIN_AG22	2.5 V (default)	
decoded_A[10]	Output				PIN_AA25	2.5 V (default)	
decoded_A[11]	Output				PIN_AB26	2.5 V (default)	
decoded_A[12]	Output				PIN_W21	2.5 V (default)	

Once you have assigned all of the inputs and outputs of your circuit to appropriate device pins, re-compile your design. Your design is now ready to be downloaded to the target hardware. You may want to go over Section 4.1 of the DE1-SoC user manual for information on configuring (programming) the Cyclone V FPGA on the board. You will be using the JTAG mode to configure the device. Take the board out of the kit box, connect the USB cable to the USB port of the computer and to the USB connector on the board. Next, select the Programmer item from the Tools menu. Click Auto Detect and then select the correct device (5CSEMA5), as shown below. Both FPGA device and HPS should be detected.



DE-SoC [USB-1]
Mode: JTAG
Progress:
Enable real-time ISP to allow background programming when available

<none> SOCVHPS

Select Device
Found devices with shared JTAG ID for device 2. Please select your device.
5CSEBA5 [✓ 5CSEMA5] 5CSTFD5D5 5CSXFC5C6 5CSXFC5D6

Next, double-click the FPGA device (5CSEMA5), from the window that pops up, add the `.sof` file created by Quartus. Finally, check the "Program/configure" box beside the 5CSEMA5 device, and then click "Start". See also Section "Configuring the FPGA in JTAG Mode" on p. 12 of the manual. Now, you should be able to use the slide switches to insert values for inputs A and B. The 7-segment LEDs should also display inputs and outputs in BCD format.

8 Questions

1. Briefly explain your VHDL code implementation of all circuits.
2. Perform timing analysis and find the critical path(s) of the one-digit BCD adder circuit for the Fast 1,100 mV 85C Model. Show the obtained timing waveform(s) of the critical path(s) that you found.
3. Provide the VHDL code that you wrote for the wrapper circuit.
4. Show a representative photo of the board displaying the result of the addition of A and B, where A is the last (rightmost) digit of the McGill ID number of one of the students in your group and B is the last (rightmost) digit of the McGill ID number of the other student in the group (if there are three members in your group, choose two IDs and state which were chosen). Clearly indicate which 7-segment LEDs/sliding switches are assigned to which inputs/outputs of the circuit on the photo.
5. Report the number of pins and logic modules used to fit your designs on the FPGA board.

9 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.