ECSE 222 Fall 2022

VDHL Assignment 4 Report: Critical Path and Getting Started with Altera DE1-SoC Board

Part 1 – Participants

Karim Elgammal 260920556

Lutming Wang 261006348

Part 2 – Executive Summary

Through this investigation, we have been able to explore the use of timing constraints, more specifically, we limit our discussion to the limitation posed by the latency metric of a critical path. We executed timing analysis to the previously implemented ripple carry adder one-digit BCD adder and were able to obtain timing waveforms of the critical paths that were identified during the timing analysis section. Furthermore, we were able to conduct the FPGA board implementation of our BCD adder; by using a wrapper entity which essentially mapped the signals/output LED to the values that are being used/tested on. Through this implementation, we were able to compose a BCD adder that inherited inputs from FPGA board switches and display the digits and sum on the LED display available on the board.

Part 3 & 4 – Implementation and Results

Critical Path of Digital Circuits

The BCD one-digit adder was implemented using the previous' lab code. This is done using a structural implementation for all components of the BCD adder (implicitly, the Ripple Carry Adder, Full Adder and Half Adder are all structural implementations). We incorporated the steps from the previous assignment to implement a structural implementation of both the BCD adder and the RCA. As mentioned within the VHDL3 report, the structural implementation is chosen to avoid error when it comes to synchronizing the specific wires to their inputs/outputs. We performed timing analysis of our BCD one-digit adder using a constraint of 5ns within our .sdc file (Fig. 1A). We then used the Timing Closure Recommendations to identify the critical path of our circuit, using the slack measure of the respective paths. We came to find that our critical path is in fact from A[0] (the LSB of one of the inputs to the BCD adder), to S[1], one of our sum output signals. This is the critical path as it is the one with the highest (magnitude) slack of -3.08 (Fig. 1B).

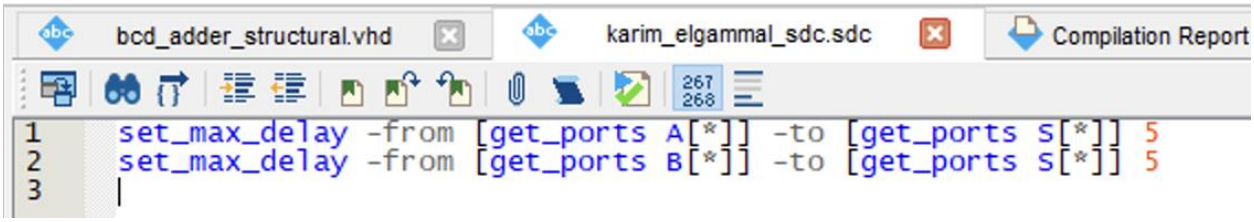

Figure 1A - SDC File

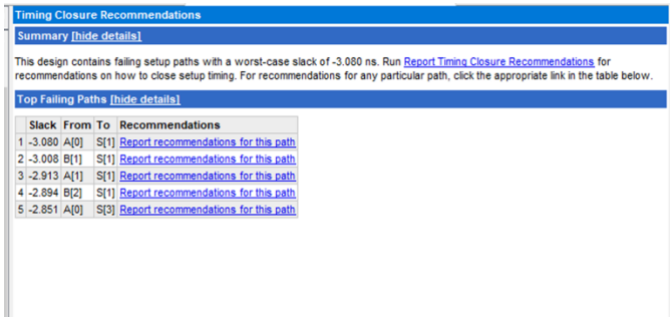

Figure 1B - Timing Closure Recommendations

Once, we identified the critical path of the BCD one-digit adder, we were able to perform timing analysis to arrive at the waveform presented in figure 1C. We completed the timing analysis in the Fast 1,100 mV 85C Model, and then used the custom report tool to arrive at the waveform of the path from A[0] to S[1].

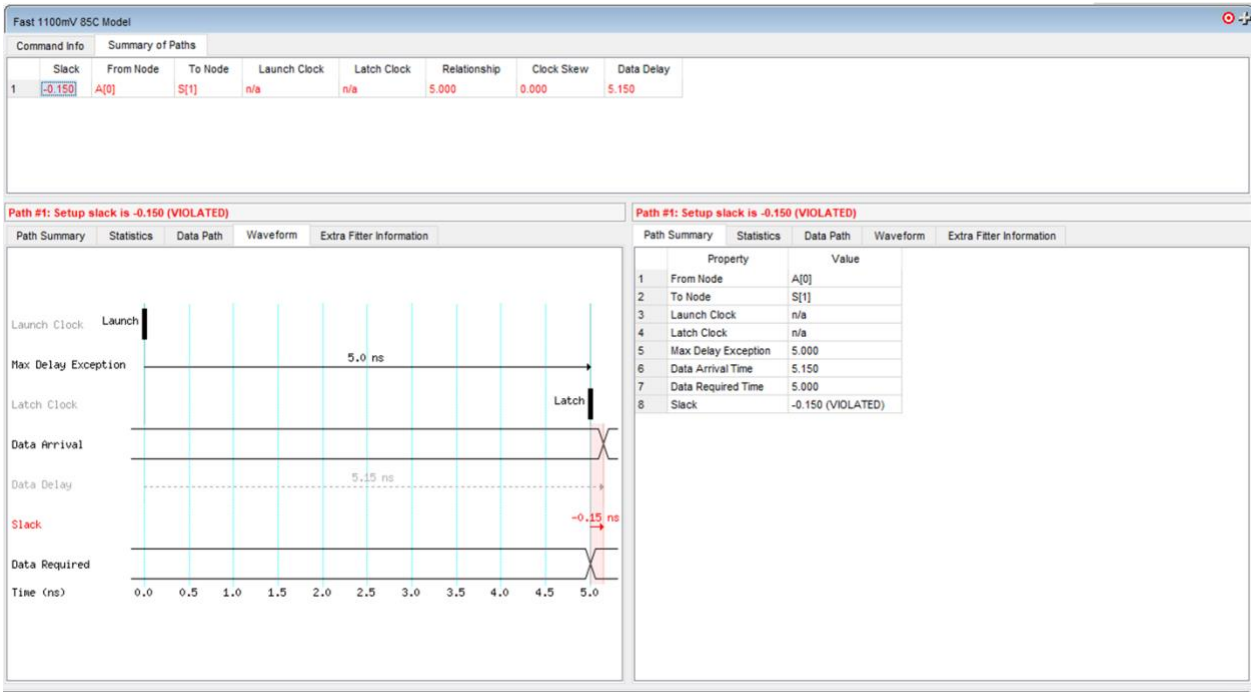

Figure 1C – Waveform Output of the Fast 1,100 mV 85C Model (path under examination: A[0] to S[1])

Wrapper Design

The seven-segment decoder was taken from the VHDL 4 assignment description. The bcd adder was implemented in a structural model, which was based on VHDL 3. The wrapper class includes four instantiations of the seven-segment decoder to decode all of signal A, B, the overflow signal, and the sum. We also instantiated once instance of the BCD adder, and that is to perform the actual summation. We then used four temporary transfer signals in order to connect and concatenate the signals between the instantiations of our units. We concatenate the carry signal with “000” as the MSBs to represent 1 when the result is higher than 9, and 0 when the result is 9 or less. Finally, we concatenate the concatenated overflow signal with the decoded sum into the output signal of decoded_A_plus_B.

VHDL Script for Wrapper Circuit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity karim_elgammal_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_A_plus_B : out std_logic_vector(13 downto 0));
end karim_elgammal_wrapper;

architecture wrapper of karim_elgammal_wrapper is

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

component bcd_adder_structural
   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 component;

signal overflow_signal : std_logic;
signal sum_signal : std_logic_vector(3 downto 0);
signal decoded_overflow : std_logic_vector(6 downto 0);
signal decoded_sum : std_logic_vector(6 downto 0);


begin

decode_a : seven_segment_decoder port map( code => A,
                                           segments_out => decoded_A);

decode_b : seven_segment_decoder port map( code => B,
                                           segments_out => decoded_B);


bcd_adder : bcd_adder_structural port map( A => A,
                                           B => B,
                                           S => sum_signal,
                                           C => overflow_signal);

decode_overflow : seven_segment_decoder port map( code => "000" & overflow_signal,
                                                  segments_out => decoded_overflow);

decode_sum : seven_segment_decoder port map( code => sum_signal,
                                             segments_out => decoded_sum);

decoded_A_plus_B <= decoded_overflow & decoded_sum;

end wrapper;
```

Once these implementations were completed, we configured the FPGA board with Quartus, in order to realize the different sets of switches and inputs that needed to be assigned. The use of the decoder in this sense allowed us to simply relay signals to the assigned representations of the numbers within the FPGA LED display. Seen below in figure 2, is the FPGA board of our implementation in action. More specifically, within this image, we are adding 6 (Karim Elgammal's last digit of his McGill ID) and 8 (Lutming Wang's last digit of his McGill ID), which generates a correct sum of 14. The associations of the LED display outputs and the switch association with bit position can be found in figure 2B; where the switch number refers to the switch number visible on the FPGA board in the image (Fig. 2A).

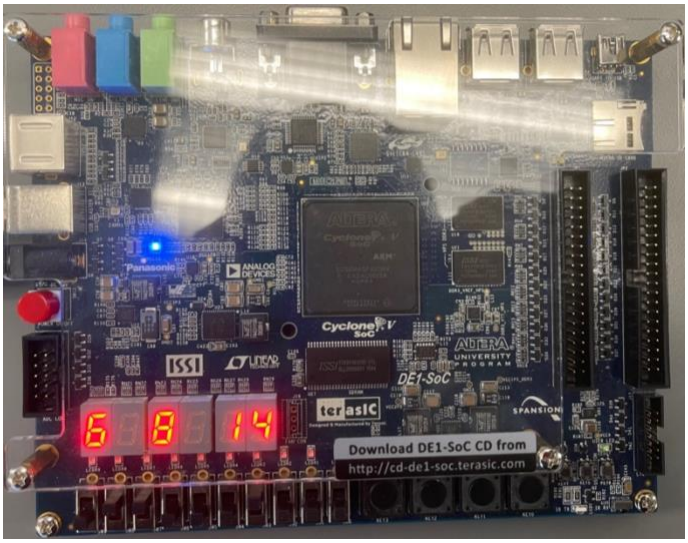

Figure 2A – Demo of Summation

FPGA LED Outputs & Switch Associations
A = HEX5
B = HEX3
decoded_A_plus_B = HEX1 (carry digit) & HEX0 (sum digit)
SW[9] - SW[6] = A[3] – A[0], respectively
SW[3] – SW[0] = B[3] – B[0], respectively

Figure 2B – Switch and Seven-Segment Display Associations

	bcd_adder_structural	karim_elgammal_wrapper
Logic Modules	7 / 32,070 (< 1 %)	17 / 32,070 (< 1 %)
Pins	13 / 457 (3 %)	36 / 457 (8 %)

Figure 2C – Pin and Logic Module Table

Part 5 – Conclusions

Throughout this lab we have been able to execute timing analysis charts and plots for both the slow and fast variations; moreover we have been able to identify the critical path of our design using the timing analysis features of Quartus. We come to find that the BCD one-digit adder's critical path according to the paths which violate the restrictions given in the first section of the assignment, is the path from A[0] to S[1]. We also were successful in implementing a wrapper class which guided the outputs/inputs to the decoder which allowed us to display the digits within the operations accurately. Finally, we successfully implemented said design into the FPGA board. What deems interesting is when we observe the number of pins and logic utilizations (Fig. 2C), we come to find that the wrapper entity required the most logic modules and pins out of all our designs. This is quite plausible as per the instantiations of the various components of the wrapper, and as we get through this course it seems that the more complex the designs are the higher the logic utilizations.