ECSE 325 - Lab Report 1

Group 42

Michael Frajman 260863814
Shi Tong Li 260857759

March 10, 2019

Introduction

The goal of this lab is to learn the basics of VHDL code and the operation of the Quartus environment by designing a synchronous counter with reset, enable and dual rate inputs and an 8-bit output. Once the code for this counter is written and compiled, the compilation report, chip planner and RTL viewer designs outputted by Quartus are to be observed and analyzed.

VHDL code for counter

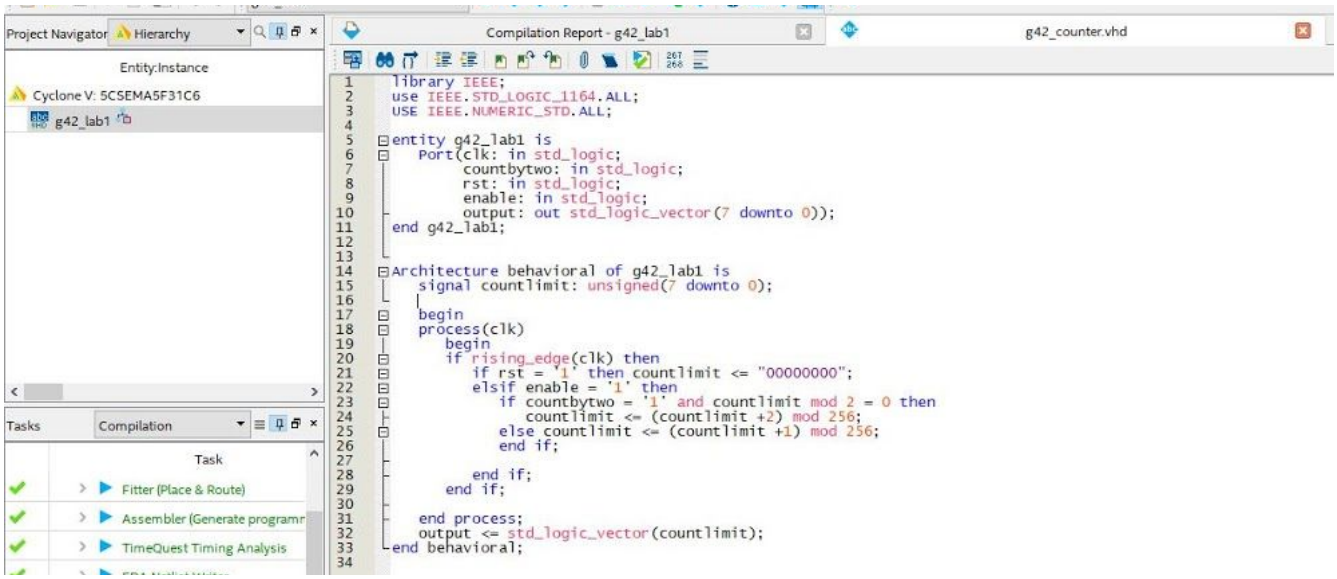

Image 1: VHDL code for our 8-bit counter. The Appendix section at the end of this lab report contains a text version of this code.

The above image shows the code for our counter as specified in the lab 1 assignment. Additionally a text version of our vhd file, g42_counter.vhd is included in the appendix at the end of our report.

The entity of the 8-bit counter includes a reset, an enable, and a toggle for the ‘countbytwo’ mode. For the architecture of the counter, it consists of a single process block which contains all of the logic for the circuit. The synchronous reset condition is first checked when the clock edge rises. Otherwise, as long counter is enabled, ‘1’, we either count up by two or count up by one depending on the state of ‘countbytwo’, with ‘1’ causing the counter to go up in increments of two. The ‘mod 256’ statement both takes care of overflow and ensures that when the counter counts by ‘2s’ it only does so for even numbers. Finally, the number inside the counter is cast from an unsigned to a standard logic vector which is sent to the output. The main signal within the architecture, ‘countlimit, is made an unsigned as this allows for an easier high level description of a decimal-based counter.

Compilation report

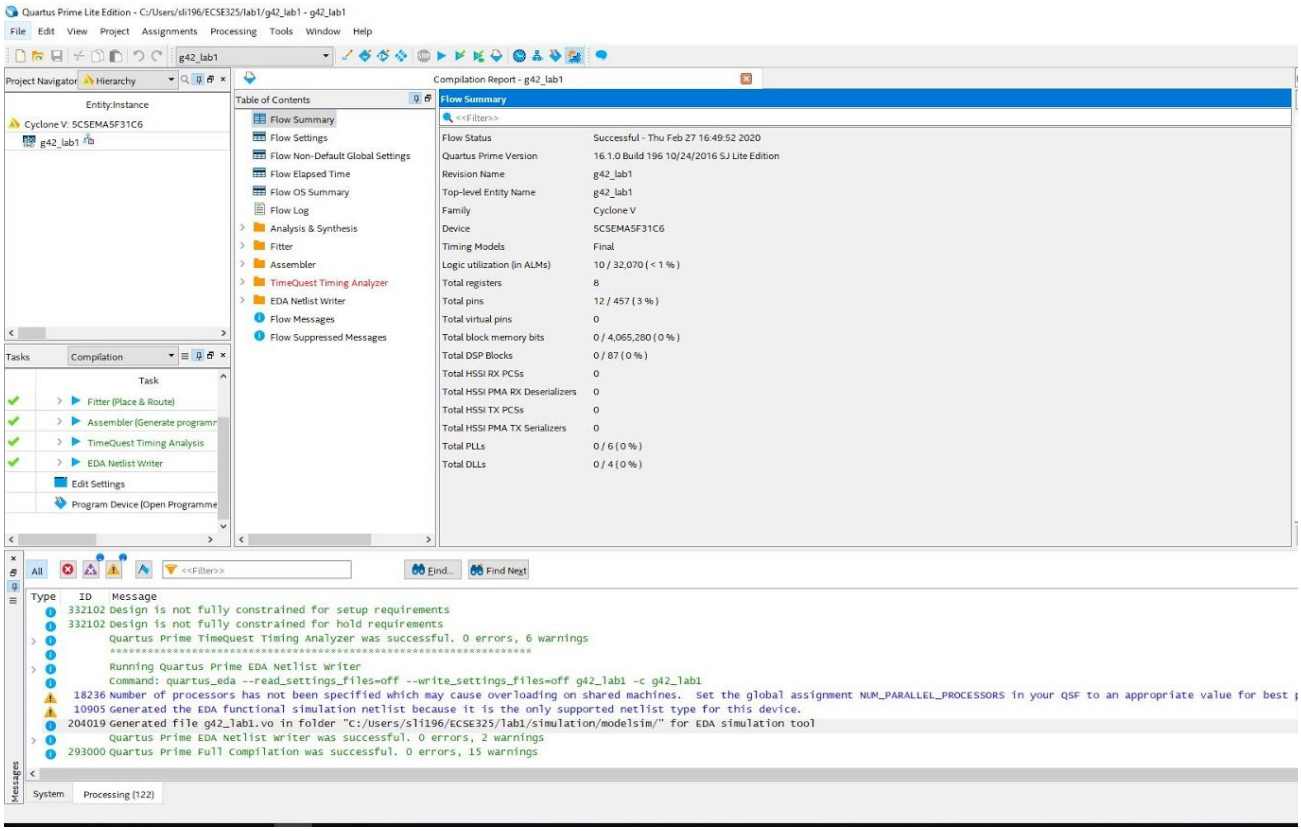

Image 2: Compilation report/flow summary of our counter design.

Resource utilization

According to the compilation report, image 2, there are 8 registers used for an 8-bit counter. Since the number of registers used is directly proportional to the number of bits in the counter in a one-to-one way, we can expect a one-to-one increase of registers if we increase the number of bits of the counter's output (eg. 16 registers for a 16-bit counter).

Chip planner

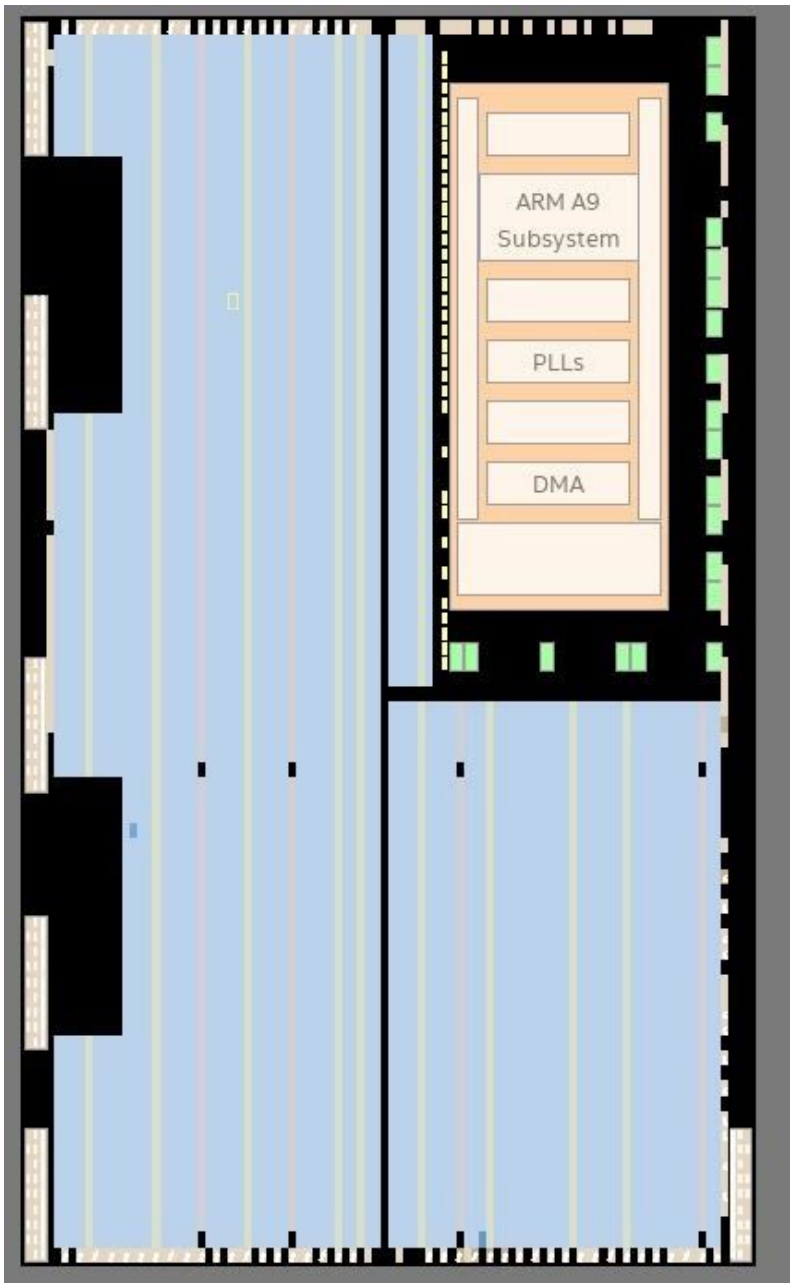

Image 3: Overview of the FPGA chip from the chip planner. Used resources/look-up-tables are represented by darkened blue rectangles (eg. one such rectangle is in the middle of the chip toward the far left side).

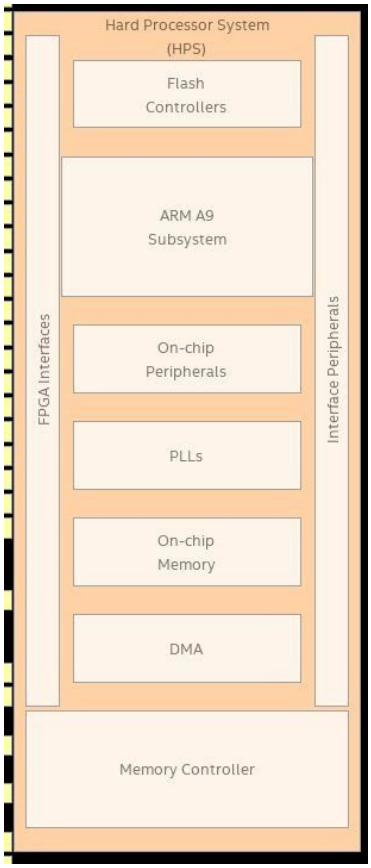

Image 4: Closeup of the processor of the FPGA board from the chip planner.

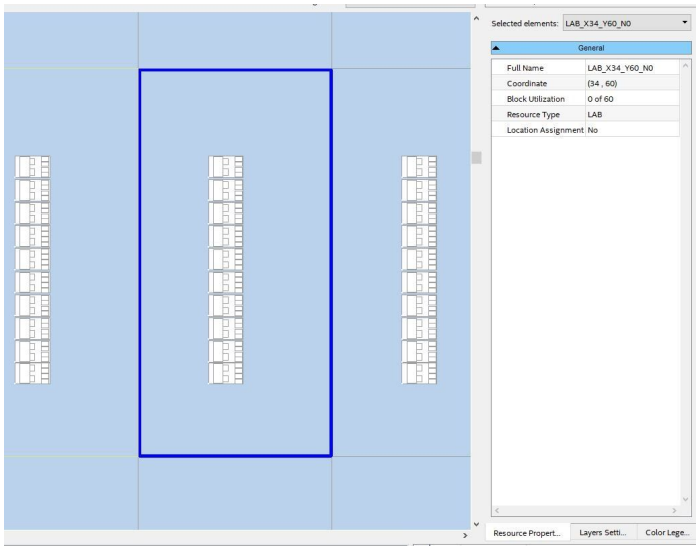

Image 5: Closeup of a generic look-up-table on the FPGA board from the chip planner.

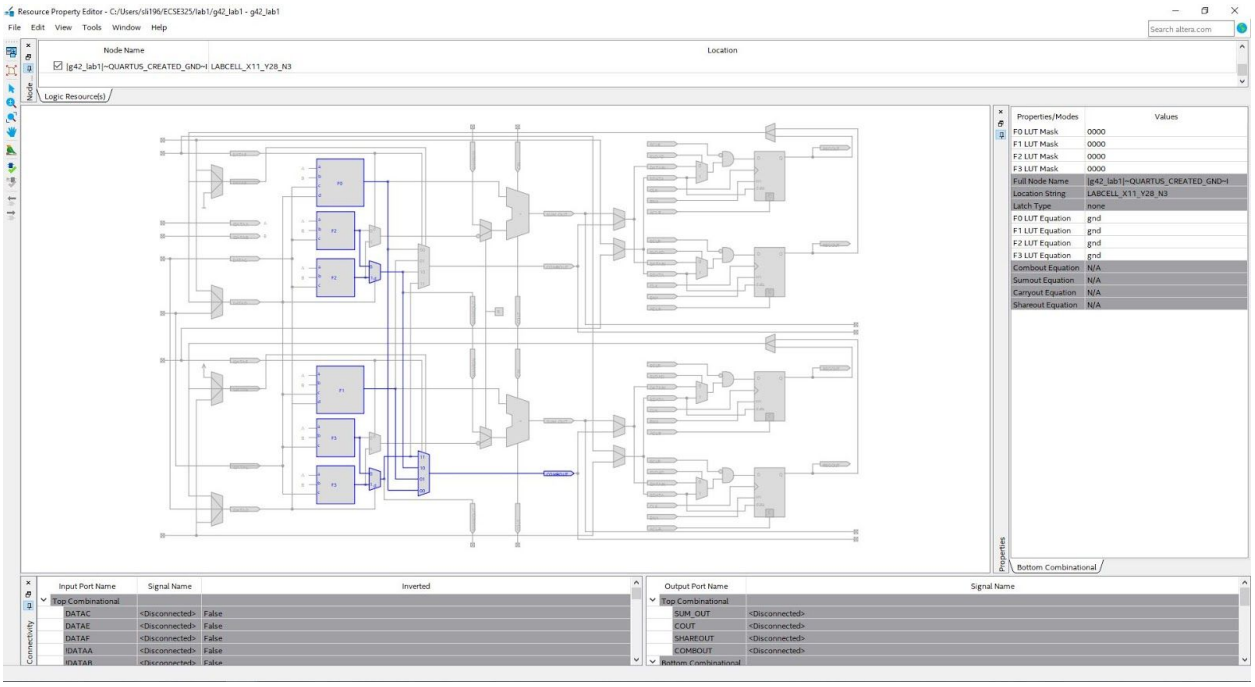

Image 6: Logic circuit layout of a look-up-table being used by the FPGA board to produce our counter. This image is from the chip planner as well.

RTL view

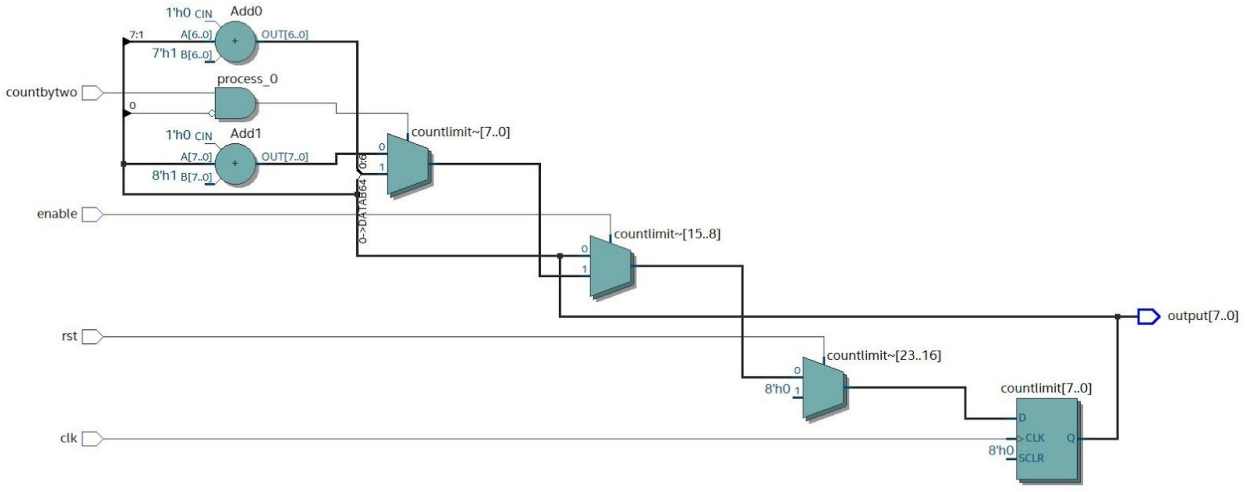

Image 7: The RTL/logic diagram of our counter design.

From the RTL viewer screenshot, image 7, it can be seen that we are using two adders, one AND gate, 3 multiplexers, and one register. The control signals on the multiplexers are used to implement the enable,

reset, and counting by two signals. The register at the end is used to store the 8-bit number of the counter in order to output it.

Conclusion

In conclusion, through this lab we established a better understanding of the capabilities of the Quartus environment while learning basic VHDL coding patterns. Additionally, we learned how to analyze our FPGA resource utilization which can serve to help avoid bloat in future larger scale design projects.

Appendix

g42_counter.vhd - code for our counter:

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

entity g42_lab1 is
	Port(clk: in std_logic;
		countbytwo: in std_logic;
		rst: in std_logic;
		enable: in std_logic;
		output: out std_logic_vector(7 downto 0));
end g42_lab1;


Architecture behavioral of g42_lab1 is
	signal countlimit: unsigned(7 downto 0);

	begin
	process(clk)
		begin
		if rising_edge(clk) then
			if rst = '1' then countlimit <= "00000000";
			elsif enable = '1' then
				if countbytwo = '1' and countlimit mod 2 = 0 then
					countlimit <= (countlimit +2) mod 256;
				else countlimit <= (countlimit +1) mod 256;
				end if;

			end if;
		end if;

	end process;
```

```
        output <= std_logic_vector(countlimit);
end behavioral;
```