Experiment 0: Combinational Circuits 1

Prasann Viswanathan, Roll Number 190070047
EE-214, WEL, IIT Bombay
January 27, 2021

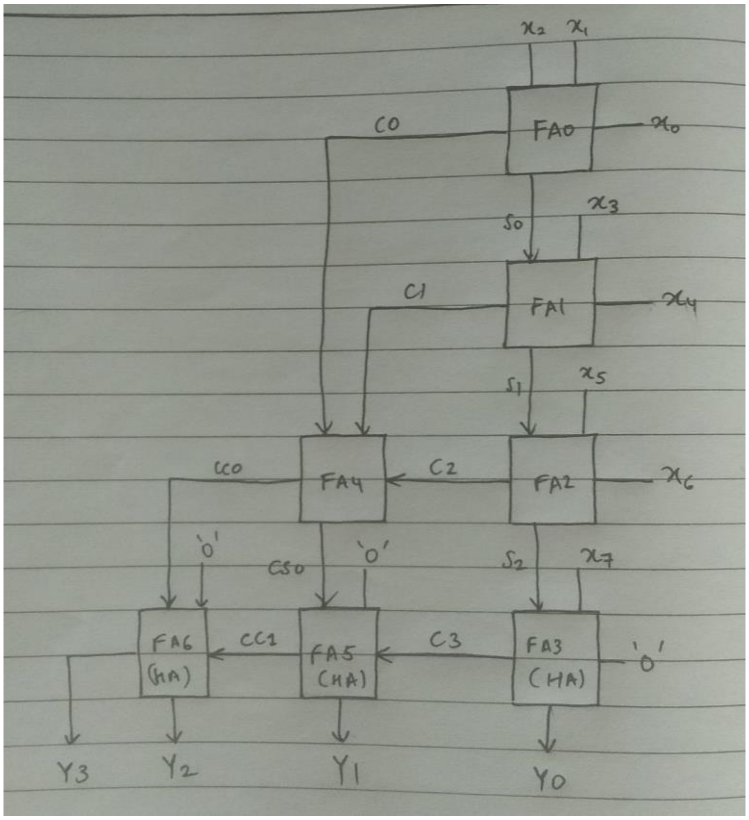
Overview of the experiment:

The purpose of the experiment was to design a logic circuit in Quartus which takes 8 inputs (x_0, …, x_7) and gives 4bit binary number, indicating the number of 1s in the 8 inputs given. I implemented this using Full Adders as components, which themselves used AND, NOT, XOR and OR gates.

I started by ideating a hand sketch to get an idea of the structure of the gates and what the rough framework should be. Since a single full adder adds 3 bits, I built up on this logic to end at a 7 full adder (4 full + 3 half adder) implementation. As a result of this, I needed only one VHDL file to code with logic, named 'BitCounter'. The remaining 'Gates' and 'FullAdder' remained the same and 'Testbench' required a change in only number of inputs and outputs. Then I finished the DUT file to suit my needs. Finally, I made a python script to generate the testcases in 'TRACEFILE.txt' which I added to simulations>modelsim. I added Testbench to the simulation and ran RTL as well as Gate Level to see if all cases passed.

The 2 simulations show the waveform of the input vector of size 8 and output vector of size 4. The output waveform corresponds to the number of 1's in the input represented as a binary number.

Approach to the experiment:



Using the logic that a full adder can add three bits, I split the problem statement into adding 8 bits whilst simultaneously taking care of carries and summing them up as well. In the end I came up with this structure which was the simplest solution that I could think of.
Everything has been labelled and is self-explanatory. Keeping track of carries was the tricky part as it needed many additional intermediate signals. Also note, the last three full adders are effectively half adders.

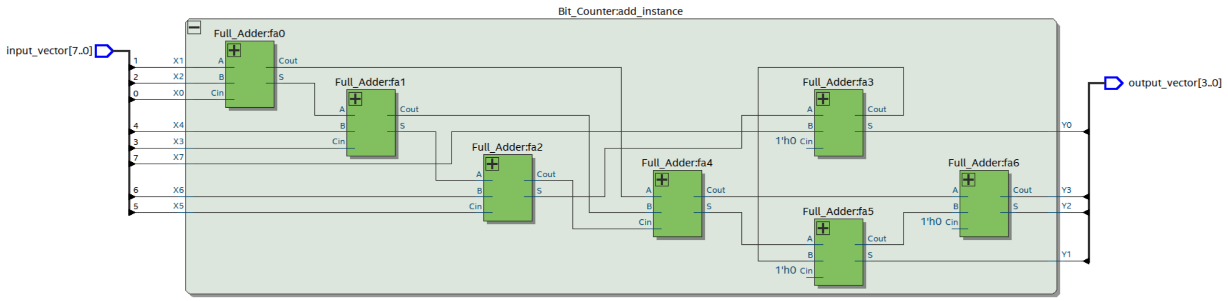
Design document and VHDL code if relevant:

DUT: Wraps the Bit_Counter entity and converts the inputs and outputs to be std logic vectors.
Testbench: Compares outputs of TRACEFILE.txt and output of implementation.
Full_adder: Acts as a full adder to add 3 bits
Bit_Counter: The main logic of the design which instantiates component Full_adder and gives output as the number of 1s in the input as a 4 bit binary number.

Architecture of Bit_Counter:

```
architecture Struct of Bit_Counter is

        component Full_Adder is
                port (A, B, Cin: in std_logic; S, Cout: out std_logic);
        end component Full_Adder;

        -- Signal names are self-sufficient to understand when paired with component port
maps
        signal S0, S1, S2, C0, C1, C2, C3, CS0, CC0, CC1: std_logic;

begin
        -- component instances
        fa0: Full_Adder
                port map (A => X1, B => X2, Cin => X0, S => S0, Cout => C0);

        fa1: Full_Adder
                port map (A => S0, B => X4, Cin => X3, S => S1, Cout => C1);

        fa2: Full_Adder
                port map (A => S1, B => X6, Cin => X5, S => S2, Cout => C2);

        fa3: Full_Adder
                port map (A => S2, B => X7, Cin => '0', S => Y0, Cout => C3);

        fa4: Full_Adder
                port map (A => C0, B => C1, Cin => C2, S => CS0, Cout => CC0);

        fa5: Full_Adder
                port map (A => CS0, B => C3, Cin => '0', S => Y1, Cout => CC1);

        fa6: Full_Adder
                port map (A => CC0, B => CC1, Cin => '0', S => Y2, Cout => Y3);

end Struct;
```

RTL View:



DUT Input/Output Format:

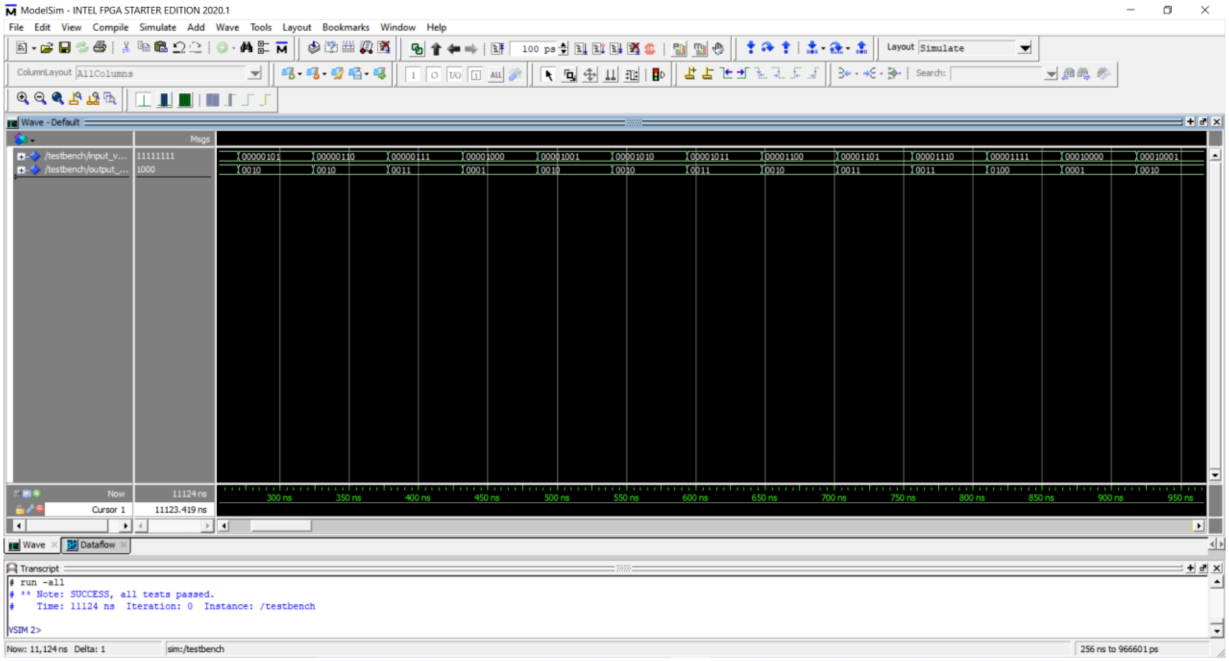
Input is a std logic vector of size 8. The LSB of the input is x0 and the MSB of the input is x7.
Output is a std logic vector of size 4. The LSB of the output is y0 and the MSB of the output is y3.

Few Testcases:

00010000 0001 1111
00010001 0010 1111
00010010 0010 1111
00010011 0011 1111
00010100 0010 1111
00010101 0011 1111
00010110 0011 1111

(Taken directly from TRACEFILE.txt)

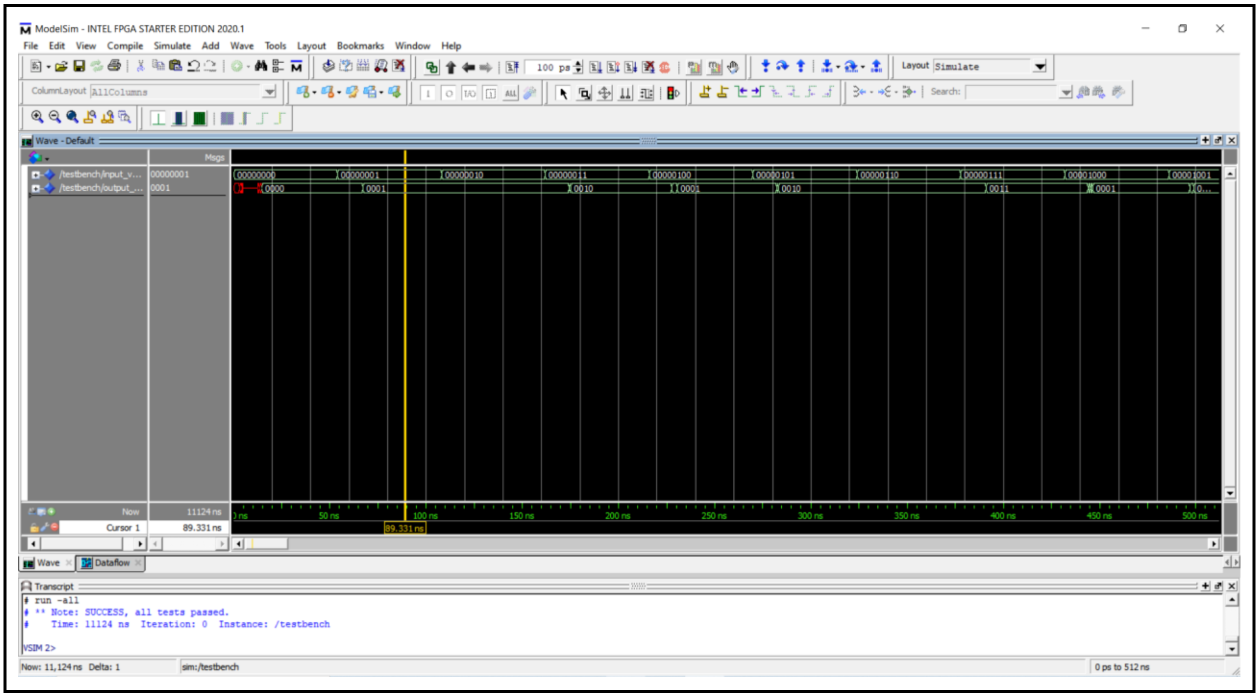
RTL Simulation:



```
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 11124 ns  Iteration: 0  Instance: /testbench

VSIM 2>
```

Now: 11,124 ns Delta: 1   sim:/testbench   256 ns to 966601 ps

Gate-level Simulation:



```
# run -all
# ** Note: SUCCESS, all tests passed.
#    Time: 11124 ns  Iteration: 0  Instance: /testbench

VSIM 2>
```

Now: 11,124 ns Delta: 1 sim:/testbench 0 ps to 512 ns

Krypton board*:

Map the logic circuit to the Krypton board and attach the images of the pin assignment and output observed on the board (switches/LEDs).

Observations*:

You must summarize your observations, either in words, using figures and/or tables.

References:

NA

* To be submitted after the tutorial on” Using Krypton.