Lab Assignment 9

Name – Md Junaid Mahmood

Enrolment Number – 19116040

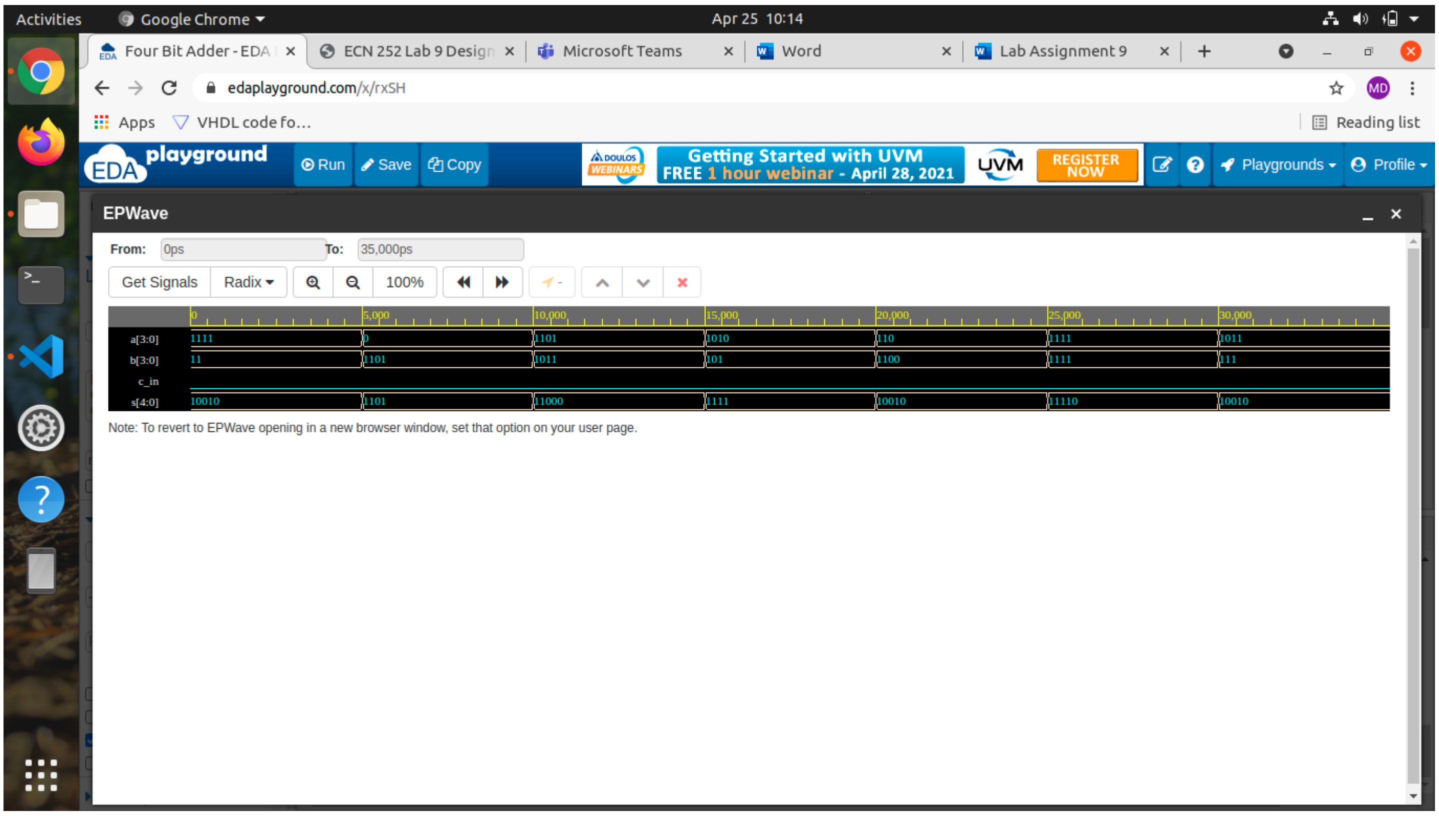
Department – Computer Science & Engineering

Year – 2nd Year

Part 1) For the 4-bit full adder the design file (design.txt) and the testbench file (testbench.txt) is placed inside *4-bit Full Adder* folder. The *4-bit Full Adder* folder is kept inside *Part 1* folder in the zip file. Now, in the screenshot attached, we have adopted following convention:

- **a** represents an input 4-bit number.
- **b** represents an input 4-bit number.
- **c_in** represents input carry to the first full adder, that is, that adder which is responsible for adding a_0 and b_0. It is kept 0 throughout the simulation, as we are doing addition only.
- **s** represents an output 5-bit sum.

<u>Screenshot of the waveform:</u>



```
From: 0ps   To: 35,000ps

          0       5,000   10,000  15,000  20,000  25,000  30,000
a[3:0]    1111    0       1101    1010    110     1111    1011
b[3:0]    11      1101    1011    101     1100    1111    111
c_in
s[4:0]    10010   1101    11000   1111    10010   11110   10010
```

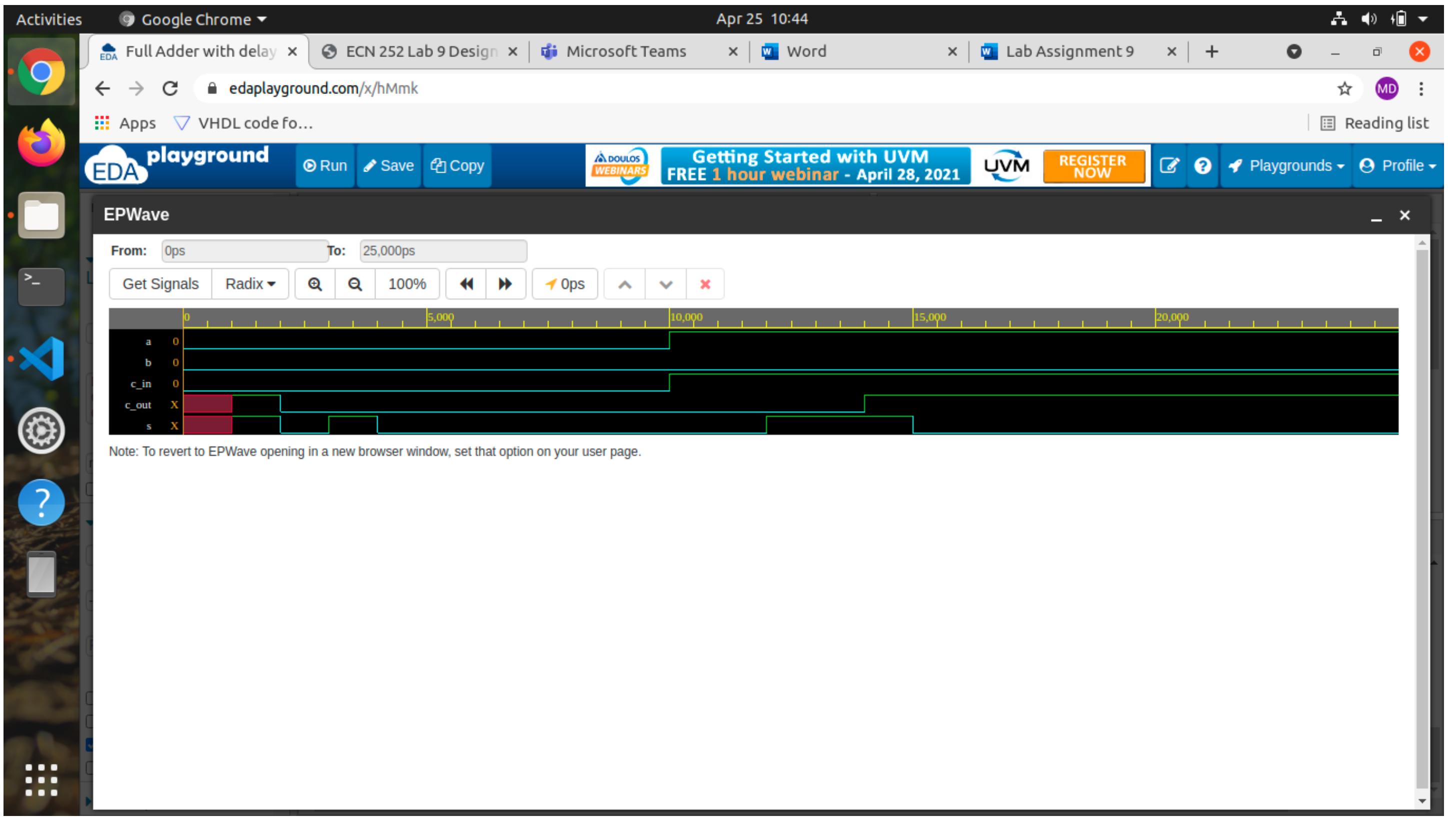
Note: To revert to EPWave opening in a new browser window, set that option on your user page.

The above simulation was run for about 35ns.

The above 4-bit full adder was designed without considering any delay in the NAND gates. Now 1ns delay is introduced into the NAND Gate. The design file (design.txt) and the testbench file (testbench.txt) for 1-bit full adder is placed inside *NAND Gate with Delay* folder. The *NAND Gate with Delay* folder is kept inside *Part 1* folder in the zip file. Now, in the screenshot attached, we have adopted following convention:

- a represents an input bit
- b represents an input bit
- c_in represents an input carry
- c_out represents an output carry
- s represents the output sum

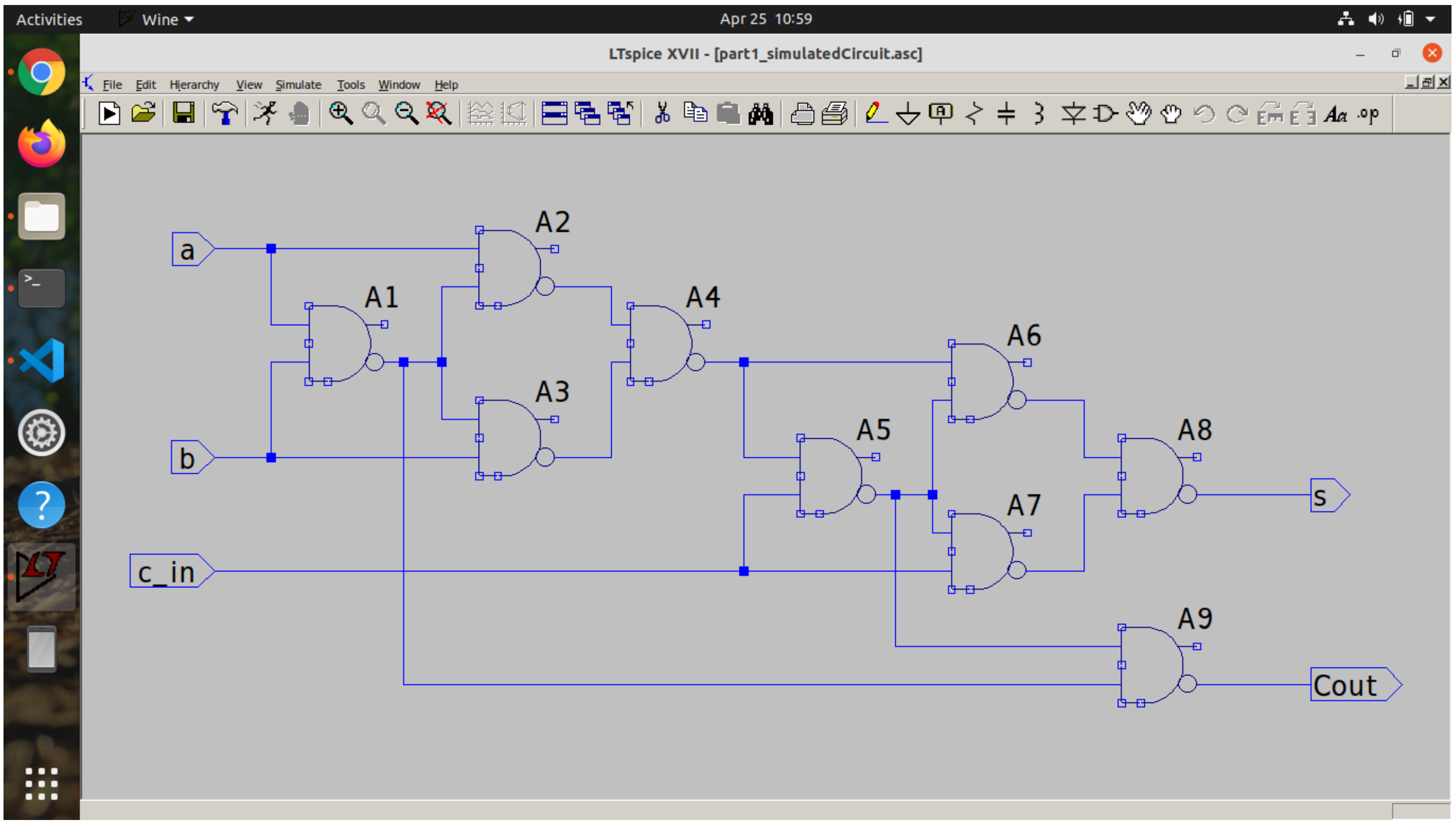
<u>Screenshot of the waveform:</u>



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Above simulation was carried out for 25ns. At 0ns, a = 0, b = 0 and c_in = 0 were applied. Due to the architecture design of the NAND Gate, both c_out and s have undefined value for 1ns. Then, some garbage value are shown as output until input to all the 9 NAND gates used in the full adder does not become well defined. Thus, in total it took 2ns for c_out to become stable and 4ns for s to become stable.

Now at 10ns, input transition from a = 0, b = 0 and c_in = 0 to a = 1, b = 0 and c_in = 1 was applied. After transition, it took 4ns for c_out to become stable and 5ns for s to become stable. Now for explaining the delay, let us consider the following circuit diagram for full adder:


Activities
Wine
Apr 25 10:59
LTspice XVII - [part1_simulatedCircuit.asc]
File Edit Hierarchy View Simulate Tools Window Help
a
b
c_in
A1
A2
A3
A4
A5
A6
A7
A8
A9
s
Cout

1. **For c_out**
    - Since b = 0, as a result output for A1 and A3 is known with certainty to be 1 after 1ns. However, previous value of A1 and A3 was 1 only. So, there is no transition and hence there is no delay associated with it.
    - However, a = 1. Thus, it will take 1ns to determine the output of A2 as 0, using unchanged output of A1 and input bit a. Thus, by the end of 1ns that is at an instant t = 11ns, output of A1, A2 and A3 are known.
    - Since, output of A3 was 1, thus output of A4 cannot be determined until output of A2 is known. Since, output of A2 is known at the end of 1ns, thus, output of A4 was determined to be 1 at the end of 2ns, at an instant t = 12ns.
    - Since c_in was 1, output of A5 cannot be determined until output of A4 is known. Since, output of A4 is known at the end of 2ns, thus, output of A5 was determined to be 0 at the end of 3ns, at an instant given by t = 13ns.

- Since output of A1 was 1, c_out cannot be determined until output of A5 is known. Since, output of A5 is known at the end of 3ns, thus, c_out was determined to be 1 with certainty at the end of 4ns, at an instant t = 14ns.
- Thus, total time delay associated with calculation of c_out in the given transition is 4ns and output of c_out becomes stable from the time t = 14ns onwards.

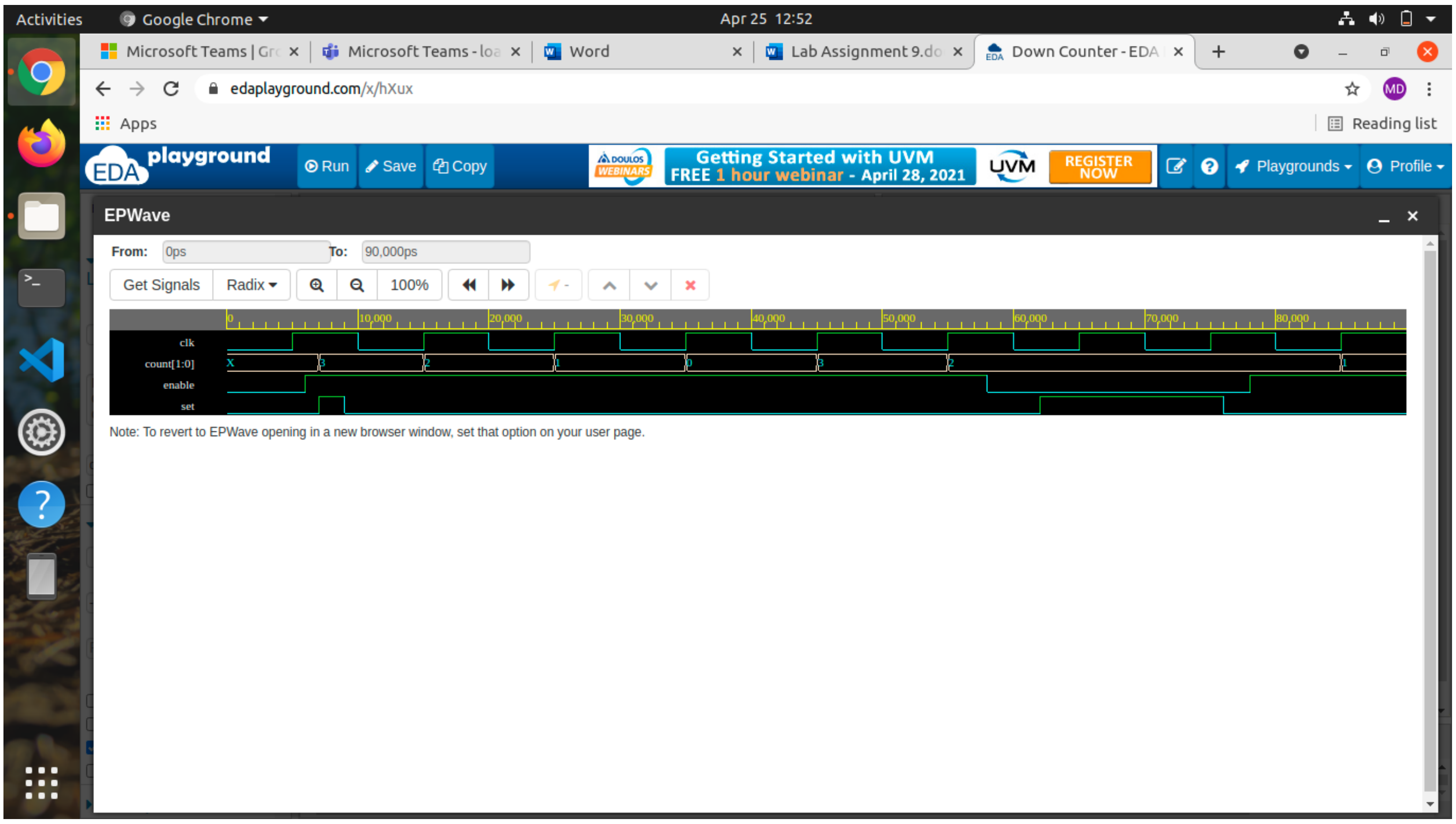
2. **For s**
   - The previous output of A5 was 1. Until 3ns, there was no change in the output of A5. Now, c_in = 1. Thus, at the end of 1ns, that is at instant t = 11ns, output of A7 is 0.
   - Now, this change of output of A7, will take 1ns to get reflected in the output s. Thus, at the end of 2ns, that is at instant t = 12ns, output of s is 1.
   - Now, output of A5 becomes 0 at t = 13ns. Thus, at t = 14ns, after 1ns delay output of A6 and A7 will become 1 with certainty. Thus, with another 1ns delay, that is at time instant of t = 15ns, output of s will become 0.
   - Based on above reasoning, it can be shown that the output of s is 1 from t = 12ns to t = 15ns. After t = 15ns, that is time delay of 5ns, output of s becomes 0 which is correct and it remains stable throughout.

Part 2) To design and demonstrate a two-bit down counter.

For the 2-bit down counter the design file (design.txt) and the testbench file (testbench.txt) is placed inside *Part 2* folder inside zip file. Now, in the design of the down counter, we have adopted following convention:

- clk represents clock signal and has the period of 10ns
- count is a 2-bit output number for representing output of the counter
- when enable = 1, the counter is enabled and vice versa
- when set = 1, the count is set to $(11)_2$ if enable = 1, irrespective of the value of the clk

Screenshot of the waveform:



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Explanation of the waveform: The above simulation was run for 90ns. Enable was set to 1 at 6ns and set was put to 1 at 7ns. Thus, at 7ns count is $(11)_2$. At 9ns, set was put to 0 keeping enable signal as it is. Thus, at every rising clock edge, value of counter behaved as desired. It is shown by following output:

- At 15ns, count is $(10)_2$.
- At 25ns, count is $(01)_2$.
- At 35ns, count is $(00)_2$.
- At 45ns, count is $(11)_2$.
- At 55ns, count is $(10)_2$.

Now, at 58ns, enable is set to 0. Thus, value of count does not change either at the rising clock edge or when set is put to 1. Now, at 78ns, enable is again set to 1. Thus, at 85ns when there was a rising clock edge, value of count became $(01)_2$.

Thus, the down counter is functioning correctly.