Lab – 9

ECN – 252

19114001 | Abhinav Saini

Part 1: 4-Bit adder

Design

```
library IEEE;
use IEEE.std_logic_1164.all;
entity NAND_Gate is
Port (a,b: in std_logic; y: out std_logic);
end entity;
architecture NAND_Gate_Structure of NAND_Gate is
begin
     y <= not (a and b);
end architecture;
library IEEE;
use IEEE.std_logic_1164.all;
entity Full_Adder is
Port ( a, b, cin: in std_logic; sum, cout : out std_logic );
end Full_Adder;
architecture Full_Adder_Structure of Full_Adder is
component NAND_Gate Port (a,b: in std_logic; y: out std_logic);
end component;
signal S1, S2, S3, S4, S5, S6, S7: std_logic;
 begin
     NG_1 : NAND_Gate Port map(a,b,S1);
     NG_2 : NAND_Gate Port map(a,S1,S2);
     NG_3 : NAND_Gate Port map(b,S1,S3);
     NG_4 : NAND_Gate Port map(S2,S3,S4);
     NG_5 : NAND_Gate Port map(S4,cin,S5);
     NG_6 : NAND_Gate Port map(S4,S5,S6);
     NG_7 : NAND_Gate Port map(S5,cin,S7);
     NG_8 : NAND_Gate Port map(S6,S7,sum);
     NG_9 : NAND_Gate Port map(S1,S5,cout);
end architecture;
library IEEE;
use IEEE.std_logic_1164.all;
entity Four_Bit_Adder is
     port( a, b      : in  STD_LOGIC_VECTOR(3 downto 0);
           ans       : out STD_LOGIC_VECTOR(3 downto 0);
           cout      : out STD_LOGIC       );
end Four_Bit_Adder;
architecture Four_Bit_Adder_strcuctural of Four_Bit_Adder is
component Full_Adder   port( a, b, cin: in  STD_LOGIC;  sum, cout :
out STD_LOGIC ); end component;
signal c0, c1, c2, c3 : STD_LOGIC;
```

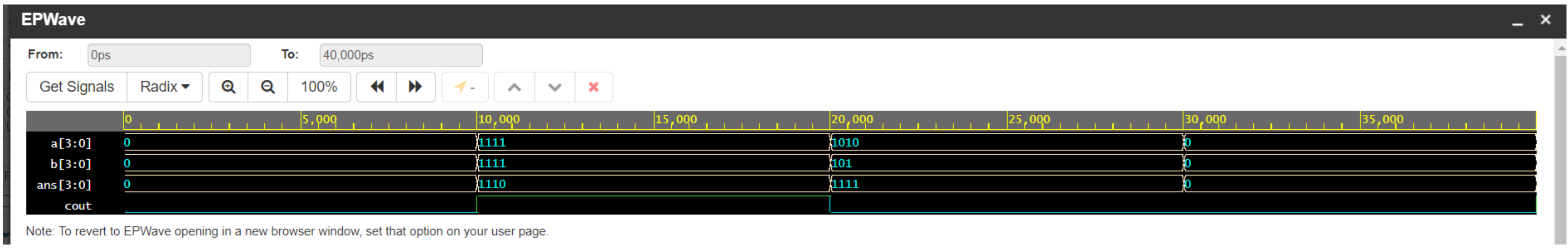
```
begin
      c0 <= '0';
      b_adder0: Full_Adder port map (a(0), b(0), c0, ans(0), c1);
      b_adder1: Full_Adder port map (a(1), b(1), c1, ans(1), c2);
      b_adder2: Full_Adder port map (a(2), b(2), c2, ans(2), c3);
      b_adder3: Full_Adder port map (a(3), b(3), c3, ans(3), cout);
end architecture;
```

Testbench

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Four_Bit_Adder_tb is
end Four_Bit_Adder_tb;
architecture TEST of Four_Bit_Adder_tb is
component Four_Bit_Adder    port( a, b : in  STD_LOGIC_VECTOR(3
downto 0);
                                            ans   : out
      STD_LOGIC_VECTOR(3 downto 0);
                                          cout : out    STD_LOGIC

                                       );
end component;
      signal a, b           : STD_LOGIC_VECTOR(3 downto 0);
      signal ans       : STD_LOGIC_VECTOR(3 downto 0);
      signal cout           : STD_LOGIC;
      begin
      U1: Four_Bit_Adder port map (a,b,ans,cout);
            process
            begin
                  a <= "0000";
                  b <= "0000";
                  wait for 10 ns;
                  a <= "1111";
                  b <= "1111";
                  wait for 10 ns;
             a <= "1010";
             b <= "0101";
             wait for 10 ns;
            end process;
END TEST;
```

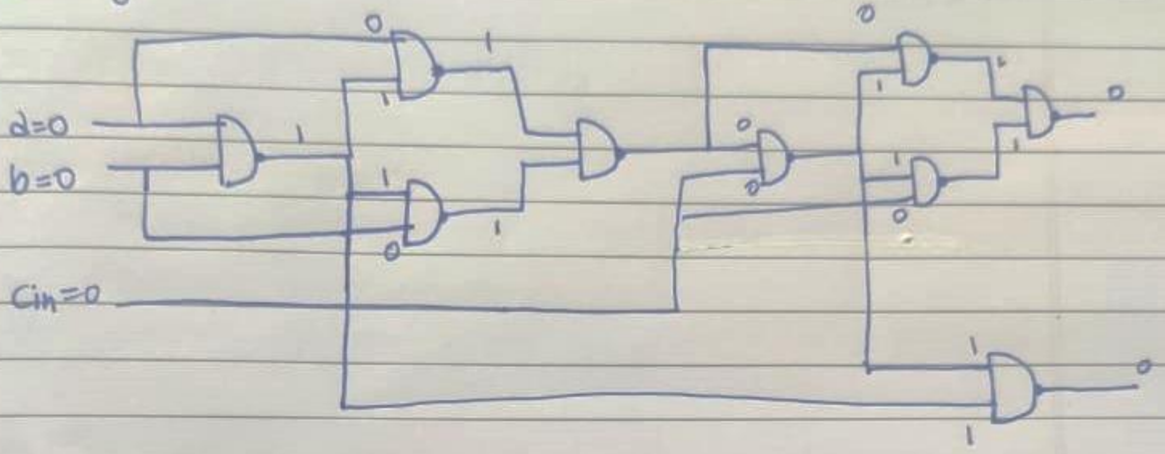
Waveform



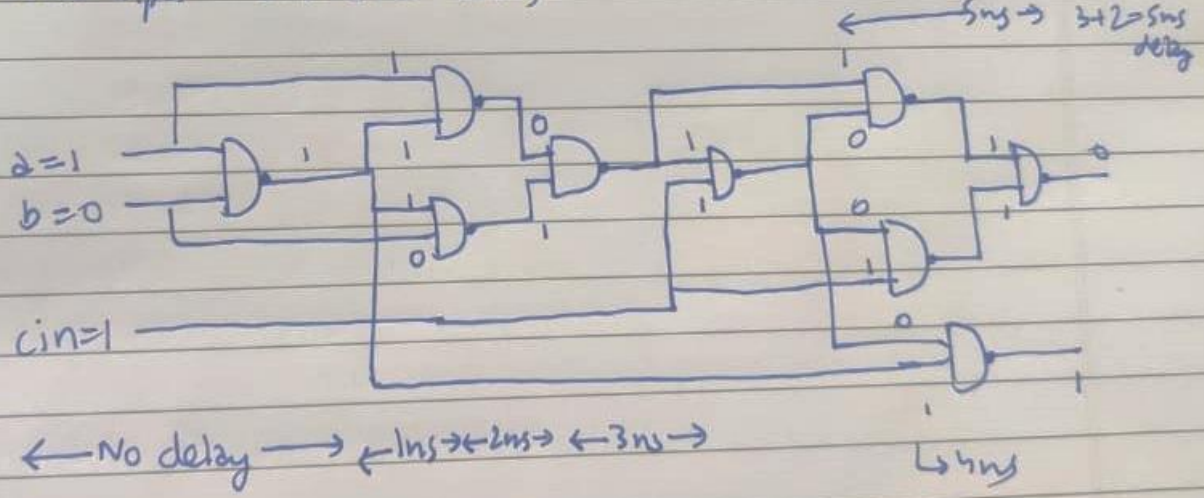
From: 0ps To: 40,000ps

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

Initially.



Now input a=1 b=0 cin=1



Hence, theortically, we get 5ns delay for any bit i.e. the sum bit and 4ns delay for cout bit. Also is the same we obtain by simulations.

1-bit Adder with 1ns delay in Nand Gates

Design

```
library IEEE;
use IEEE.std_logic_1164.all;
entity NAND_Gate is
Port (a,b: in std_logic; y: out std_logic);
end entity;
architecture NAND_Gate_Structure of NAND_Gate is
begin
  y <= not (a and b) after 1 ns;
end architecture;
library IEEE;
use IEEE.std_logic_1164.all;
entity Full_Adder is
Port ( a, b, cin: in std_logic; sum, cout : out std_logic);
end Full_Adder;
architecture Full_Adder_Structure of Full_Adder is
component NAND_Gate Port (a,b: in std_logic; y: out std_logic); end
component;
signal S1, S2, S3, S4, S5, S6, S7: std_logic;
begin
     NG_1 : NAND_Gate Port map(a,b,S1);
     NG_2 : NAND_Gate Port map(a,S1,S2);
     NG_3 : NAND_Gate Port map(b,S1,S3);
     NG_4 : NAND_Gate Port map(S2,S3,S4);
     NG_5 : NAND_Gate Port map(S4,cin,S5);
     NG_6 : NAND_Gate Port map(S4,S5,S6);
     NG_7 : NAND_Gate Port map(S5,cin,S7);
     NG_8 : NAND_Gate Port map(S6,S7,sum);
     NG_9 : NAND_Gate Port map(S1,S5,cout);
end architecture;
```

Testbench

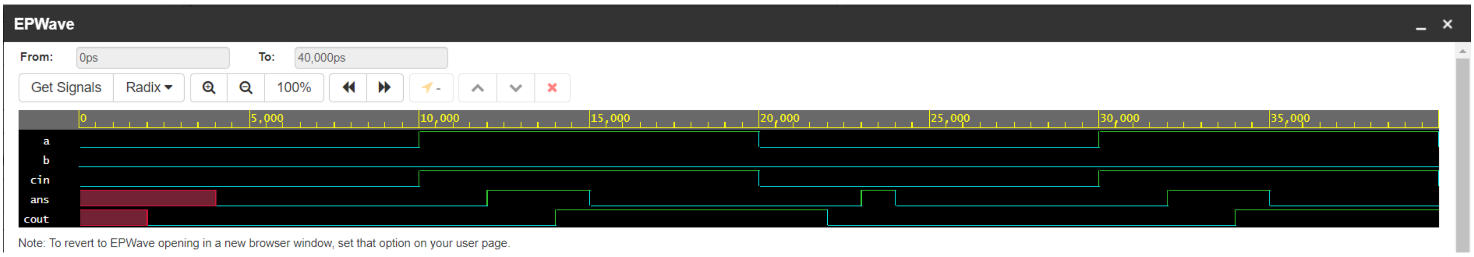
```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Full_Adder_tb is
end Full_Adder_tb;
architecture Testbench of Full_Adder_tb is
component Full_Adder  Port ( a, b, cin: in std_logic; sum, cout :
out std_logic);
end component;
     signal a, b, cin, ans, cout      : STD_LOGIC;
     begin
     U1: Full_Adder port map (a,b, cin, ans,cout);

          process
```

```
           begin

                a <= '0';
                b <= '0';
                cin <= '0';
                wait for 10 ns;
                a <= '1';
                b <= '0';
                cin <= '1';
            wait for 10 ns;
           end process;
END Testbench;
```

Waveform



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

Part 2 : 2-bit Down Counter

Design

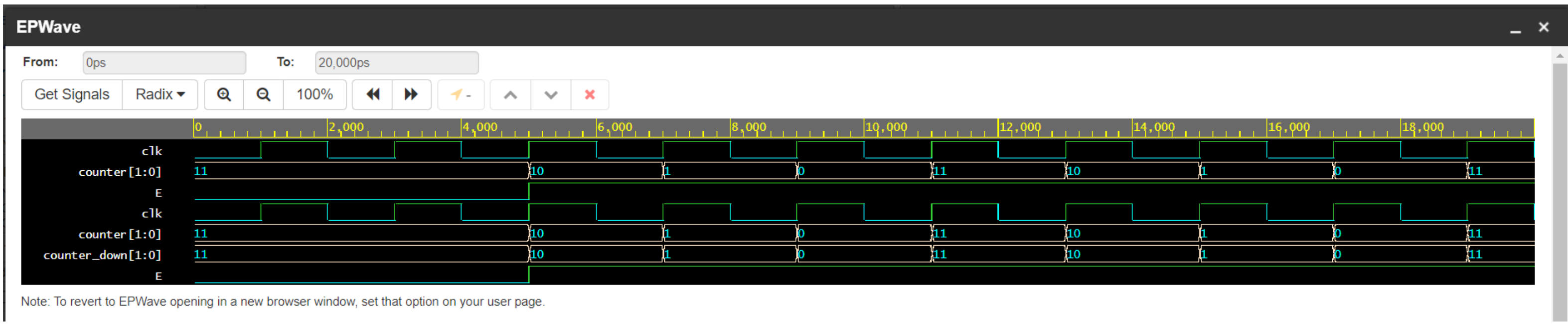
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity Two_Bit_Down_Counter is
    Port ( clk: in std_logic;
           E: in std_logic;
           counter: out std_logic_vector(1 downto 0)
     );
end Two_Bit_Down_Counter;
architecture Down_Behavioral of Two_Bit_Down_Counter is
signal counter_down: std_logic_vector(1 downto 0) := "11";
begin
process(clk)
begin
if(rising_edge(clk)) then
    if(E='1') then
        counter_down <= counter_down - "01";
    end if;
 end if;
end process;
 counter <= counter_down;
end Down_Behavioral;
```

Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity down_tb_counters is
end down_tb_counters;
architecture Down_Behavioral_tb of down_tb_counters is
component Two_Bit_Down_Counter
    Port ( clk: in std_logic; E: in std_logic; counter: out
std_logic_vector(1 downto 0));
end component;
signal E,clk: std_logic;
signal counter:std_logic_vector(1 downto 0);
begin
      dut: Two_Bit_Down_Counter port map (clk => clk, E=>E, counter
=> counter);
   process
      begin
           for i in 0 to 30 loop
              clk <= '0';
              wait for 1 ns;
              clk <= '1';
              wait for 1 ns;
           end loop;
      end process;

   process
      begin
            E <= '0';
         wait for 5 ns;
           E <= '1';
         wait;
      end process;
end Down_Behavioral_tb;
```

Waveform



From: 0ps To: 20,000ps

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