VLSI System Design Lab Report

Modelling of Various Digital Circuits in VHDL

Name: Rupom Bora
Roll No: PT-221-839-0001
Session: 2nd Semester

VLSI System Design
(ECE 2324)

Department of Electronics and Communication Engineering
Gauhati University

June 5, 2023

Contents

1 Design a selector/multiplexer

1.1 Block Diagram

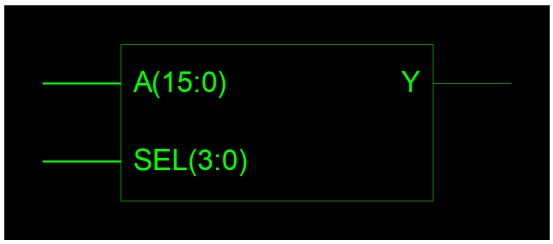

Figure 1: Multiplexer Block Diagram

1.2 VHDL Code

```
-- Rupam Bora
-- SPV, MTech, Gauhati University
-- PT-221-839-0001
-- VLSI System Design (ECE 2324)
-- Selector/Multiplexer design in VHDL


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity SELECTOR is
  port (
       A : in std_logic_vector(15 downto 0);
         SEL : in std_logic_vector(3 downto 0);
         Y : out std_logic);
end SELECTOR;

architecture RTL1 of SELECTOR is
begin
  p0: process (A,SEL)
    begin
      if (SEL = "0000") then
          Y <= A(0);
        elsif (SEL = "0001") then
          Y <= A(1);
        elsif (SEL = "0010") then
          Y <= A(2);
        elsif (SEL = "0011") then
          Y <= A(3);
        elsif (SEL = "0100") then
          Y <= A(4);
        elsif (SEL = "0101") then
          Y <= A(5);
        elsif (SEL = "0110") then
          Y <= A(6);
        elsif (SEL = "0111") then
          Y <= A(7);
        elsif (SEL = "1000") then
          Y <= A(8);
        elsif (SEL = "1001") then
          Y <= A(9);
        elsif (SEL = "1010") then
          Y <= A(10);
        elsif (SEL = "1011") then
          Y <= A(11);
        elsif (SEL = "1100") then
          Y <= A(12);
        elsif (SEL = "1101") then
          Y <= A(13);
        elsif (SEL = "1110") then
          Y <= A(14);
        else
          Y <= A(15);
        end if;
  end process;
end RTL1;

architecture RTL2 of SELECTOR is
begin
  p1: process(A,SEL)
```

```
63      begin
64        case SEL is
65            when "0000" => Y <= A(0);
66              when "0001" => Y <= A(1);
67              when "0010" => Y <= A(2);
68              when "0011" => Y <= A(3);
69              when "0100" => Y <= A(4);
70              when "0101" => Y <= A(5);
71              when "0110" => Y <= A(6);
72              when "0111" => Y <= A(7);
73              when "1000" => Y <= A(8);
74              when "1001" => Y <= A(9);
75              when "1010" => Y <= A(10);
76              when "1011" => Y <= A(11);
77              when "1100" => Y <= A(12);
78              when "1101" => Y <= A(13);
79              when "1110" => Y <= A(14);
80              when others => Y <= A(15);
81          end case;
82       end process;
83  end RTL2;
84
85  architecture RTL3 of SELECTOR is
86  begin
87    with SEL select
88        Y <= A(0) when "0000",
89             A(1) when "0001",
90               A(2) when "0010",
91               A(3) when "0011",
92               A(4) when "0100",
93               A(5) when "0101",
94               A(6) when "0110",
95               A(7) when "0111",
96               A(8) when "1000",
97               A(9) when "1001",
98               A(10) when "1010",
99               A(11) when "1011",
100              A(12) when "1100",
101              A(13) when "1101",
102              A(14) when "1110",
103              A(15) when others;
104 end RTL3;
105
106 architecture RTL4 of SELECTOR is
107 begin
108   Y <= A(conv_integer(SEL));
109 end RTL4;
```

Source Code 1: Multiplexer

1.3 Results

1.3.1 Testbench waveform

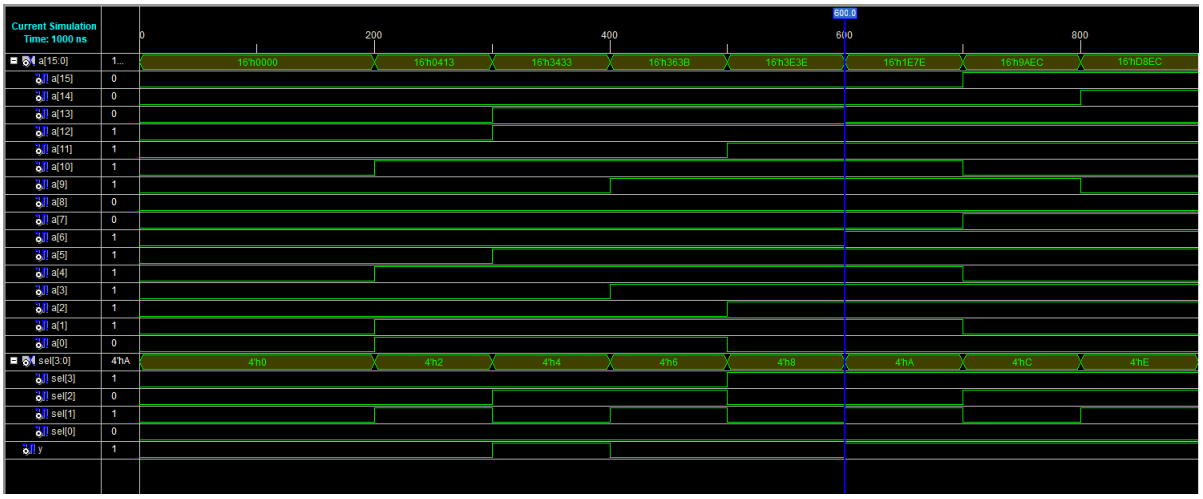

Figure 2: Multiplexer testbench waveform

1.3.2 RTL schematic

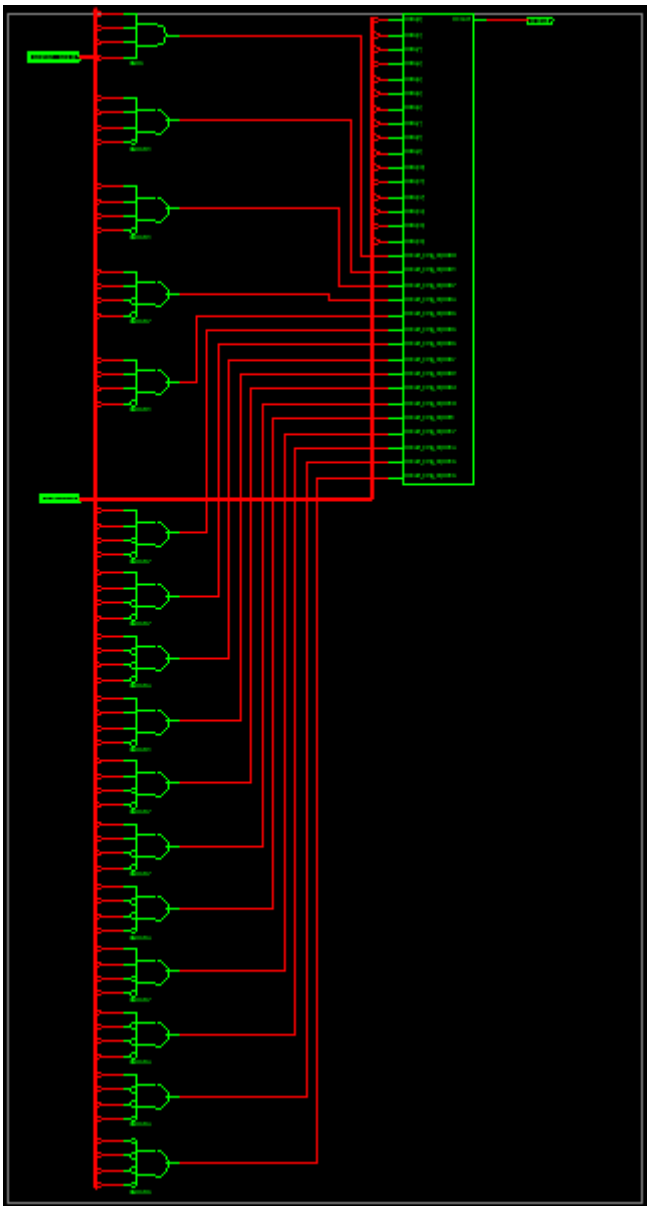

Figure 3: Multiplexer RTL Schematic

2 Design an equality checker/comparator

2.1 Block Diagram

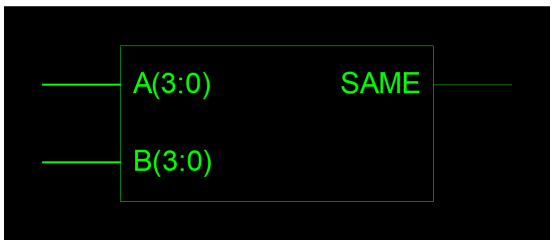

Figure 4: Comparator Block Diagram

2.2 VHDL Code

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity EQUAL is
7    generic(N: in integer :=8);
8      port(
9      A,B : in std_logic_vector(N-1 downto 0);
10     SAME  : out std_logic);
11 end EQUAL;
12
13 architecture RTL_FOR of EQUAL is
14 begin
15   p0: process(A,B)
16       variable SAME_SO_FAR: std_logic;
17     begin
18       SAME_SO_FAR := '1';
19         for i in A'range loop
20           if (A(i)/=B(i)) then
21             SAME_SO_FAR := '0';
22             exit;
23             end if;
24         end loop;
25       SAME <= SAME_SO_FAR;
26     end process;
27 end RTL_FOR;
28
29 architecture RTL_STRUCTURAL of EQUAL is
30   signal EACHBIT: std_logic_vector(A'length-1 downto 0);
31 begin
32   xnor_gen: for i in A'range generate
33       EACHBIT(i) <= not (A(i) xor B(i));
34     end generate;
35     p0: process (EACHBIT)
36       variable BITSAME: std_logic;
37     begin
38       BITSAME:=EACHBIT(i);
39         for i in 1 to A'length-1 loop
40         BITSAME:=BITSAME and EACHBIT(i);
41         end loop;
42         SAME <= BITSAME;
43     end process;
44 end RTL_STRUCTURAL;
45
46 architecture RTL_OPERATOR of EQUAL is
47 begin
48   SAME <= '1' when A=B else '0';
49 end RTL_OPERATOR;
```

Source Code 2: Comparator

2.3 Results

2.3.1 Testbench waveform

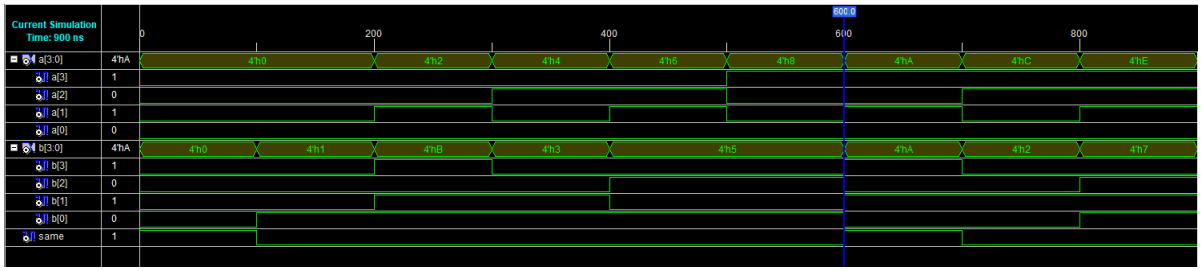

Figure 5: Comparator testbench waveform 1

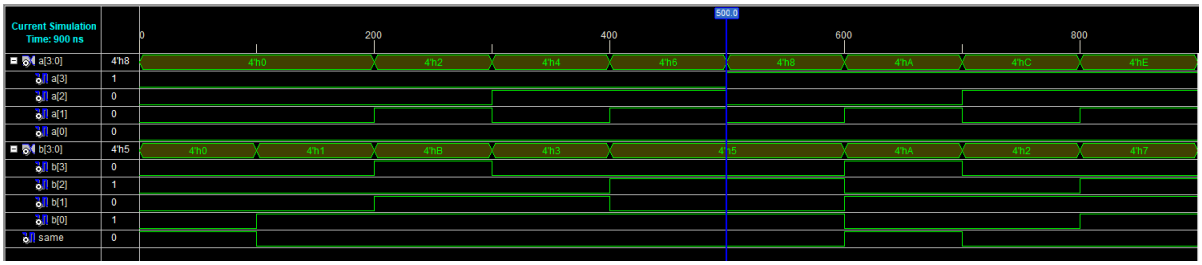

Figure 6: Comparator testbench waveform 2

2.3.2 RTL schematic

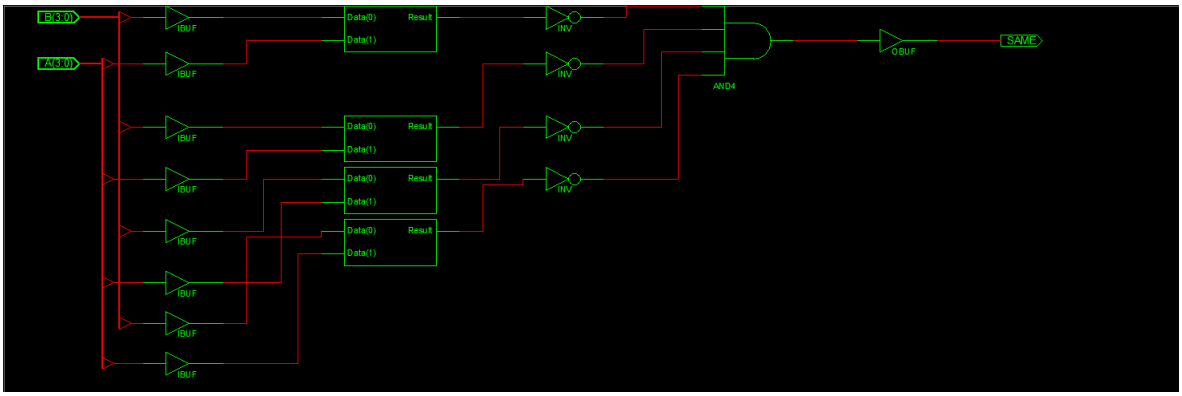

Figure 7: Comparator RTL Schematic

3 Design a half adder, full adder, carry ripple adder and carry look ahead adder

3.1 Block Diagram

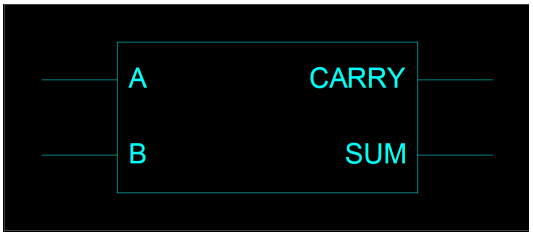

Figure 8: Half Adder Block Diagram

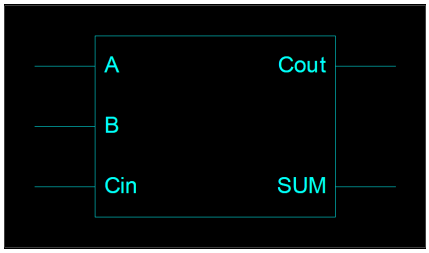

Figure 9: Full Adder Block Diagram

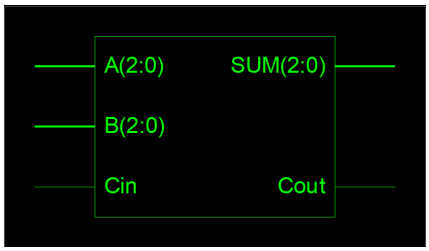

Figure 10: Carry Ripple Adder Block Diagram

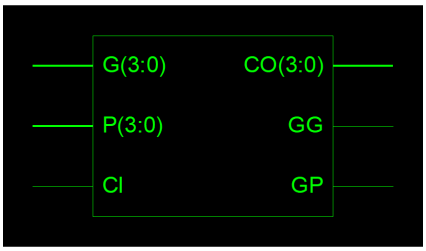

Figure 11: Carry Look Ahead Block Diagram

3.2 VHDL Code

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity HA is
7    port(
8        A,B:in std_logic;
```

```
        SUM,CARRY:out std_logic
    );
end HA;

architecture RTL_dataflow of HA is
begin
  SUM <= A xor B;
    CARRY <= A and B;
end RTL_dataflow;
```

Source Code 3: Half Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FA is
  port(
      A,B,Cin: in std_logic;
        SUM,Cout: out std_logic
    );
end FA;

architecture RTL_dataflow of FA is
begin
  SUM <= A xor B xor Cin;
    Cout <= (A and B)or(A and Cin)or(B and Cin);
end RTL_dataflow;
```

Source Code 4: Full Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity CRA is
  generic(N: in integer:=8);
    port(
      A,B: in std_logic_vector(N-1 downto 0);
        Cin: in std_logic;
        SUM: out std_logic_vector(N-1 downto 0);
        Cout: out std_logic
    );
end CRA;

architecture RTL_component of CRA is
  component FA
    port(
      A,B,Cin: in std_logic;
        SUM,Cout: out std_logic
    );
    end component;
    signal C: std_logic_vector(A'length-1 downto 1);
begin
  gen: for j in a'range generate
      genlsb: if j=0 generate
          fa0: FA port map(A=>A(0), B=>B(0), Cin=>Cin, SUM=>SUM(0), Cout=>C(0));
        end generate;
        genmid: if (j>0) and (j < A'length-1) generate
          fa0: FA port map(A=>A(j), B=>B(j), Cin=>C(j), SUM=>SUM(j), Cout=>C(j+1));
        end generate;
        genmsb: if j= A'length-1 generate
          fa0: FA port map(A=>A(j), B=>B(j), Cin=>C(j), SUM=>SUM(j), Cout=>Cout);
        end generate;
    end generate;
end RTL_component;

architecture RTL_for of CRA is
  procedure fulladder(
      signal A,B,Cin: in std_logic;
        signal SUM,Cout: out std_logic) is
    begin
      SUM<=A xor B xor Cin;
        Cout<=(A and B)or(A and Cin)or(B and Cin);
    end fulladder;
    signal C: std_logic_vector(A'length-1 downto 1);
begin
  gen: for j in A'range generate
      genlsb: if j=0 generate
          fulladder(A=>A(0), B=>B(0), Cin=>Cin, SUM=>SUM(0), Cout=>C(0));
        end generate;
        genmid: if (j>0) and (j < A'length-1) generate
          fulladder(A=>A(j), B=>B(j), Cin=>C(j), SUM=>SUM(j), Cout=>C(j+1));
```

```
54          end generate;
55          genmsb: if j= A'length-1 generate
56             fulladder(A=>A(j), B=>B(j), Cin=>C(j), SUM=>SUM(j), Cout=>Cout);
57          end generate;
58      end generate;
59  end RTL_for;
```

Source Code 5: Carry Ripple Adder

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity CLAU is
7      Port ( P, G : in  STD_LOGIC_VECTOR(3 downto 0);
8             CI : in  STD_LOGIC;
9             CO : out  STD_LOGIC_VECTOR(3 downto 0);
10            GP, GG : out  STD_LOGIC);
11 end CLAU;
12
13 architecture RTL of CLAU is
14
15 begin
16   CO(0) <= CI;
17   CO(1) <= (CI and P(0)) or
18          G(0);
19   CO(2) <= (CI and P(0) and P(1)) or
20          (G(0) and P(1)) or
21          G(1);
22   CO(3) <= (CI and P(0) and P(1) and P(2)) or
23          (G(0) and P(1) and P(2)) or
24          (G(1) and P(2)) or
25          G(2);
26   GG <= (G(0) and P(1) and P(2) and P(3)) or
27        (G(1) and P(2) and P(3)) or
28        (G(2) and P(3)) or
29        G(2);
30   GP <= P(3) and P(2) and P(1) and P(0);
31 end RTL;
```

Source Code 6: Carry Look Ahead Adder

3.3 Results

3.3.1 Testbench waveform

Figure 12: Half Adder testbench waveform

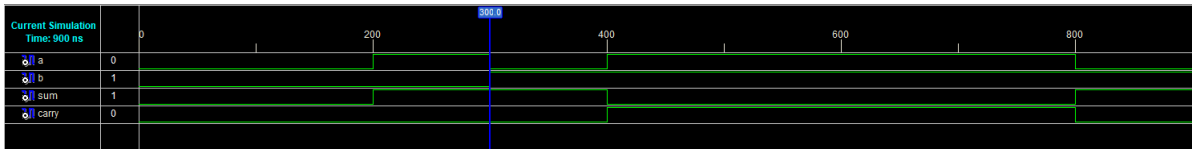

Figure 13: Full Adder testbench waveform

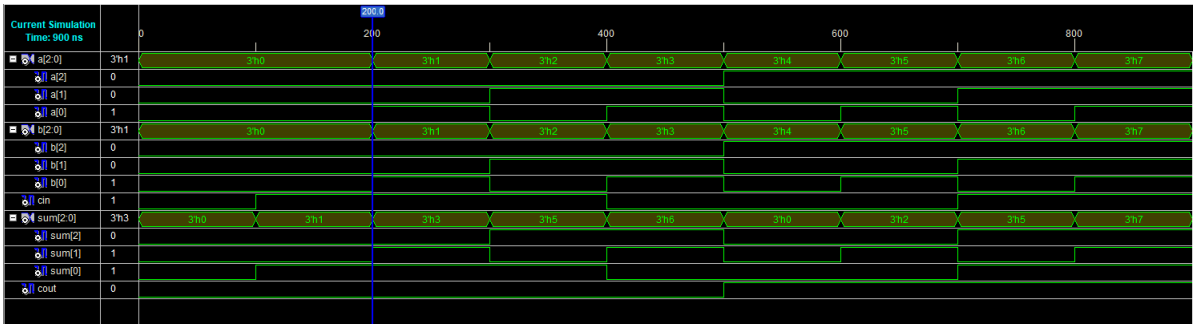

Figure 14: Carry Ripple Adder testbench waveform

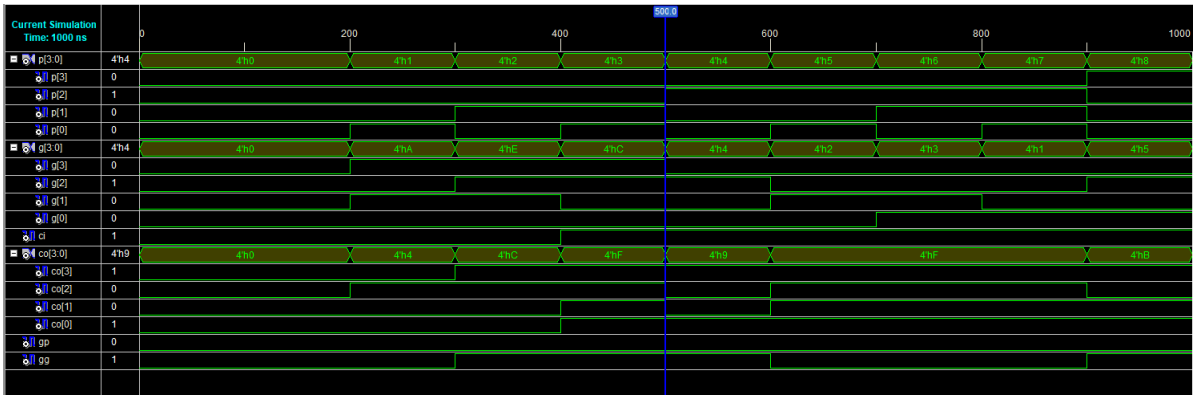

Figure 15: Carry Look Ahead Adder testbench waveform

3.3.2 RTL schematic

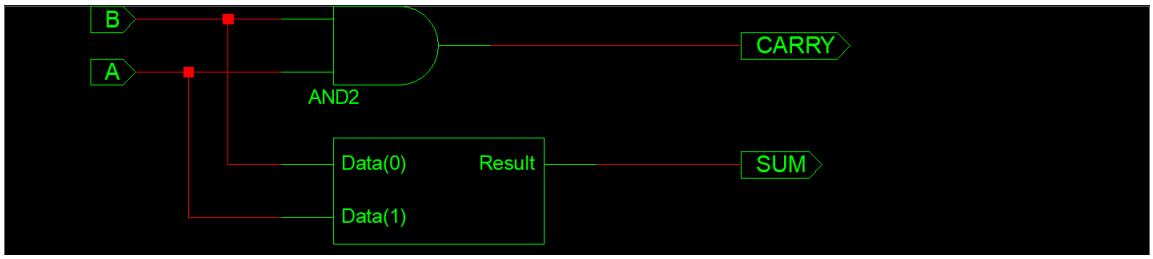

Figure 16: Half Adder RTL Schematic

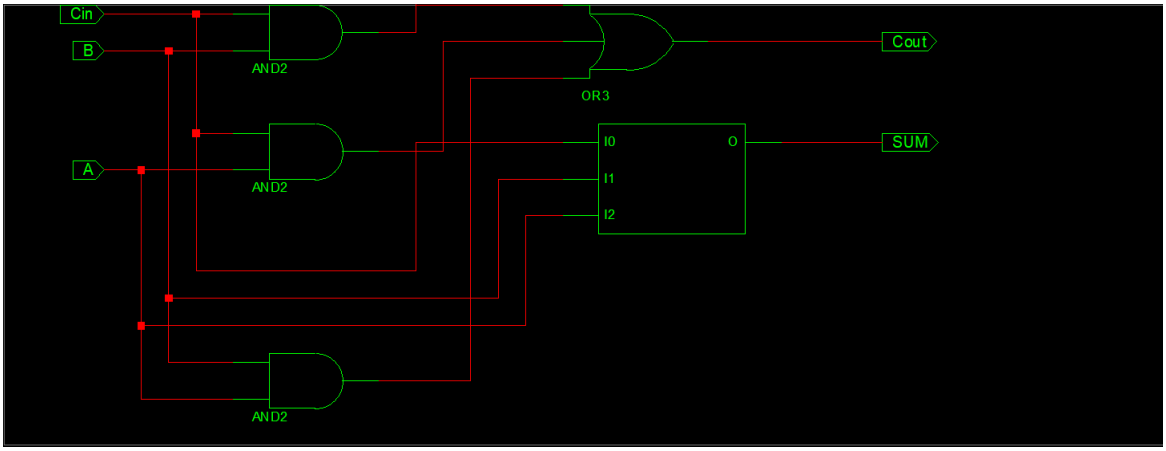

Figure 17: Full Adder RTL Schematic

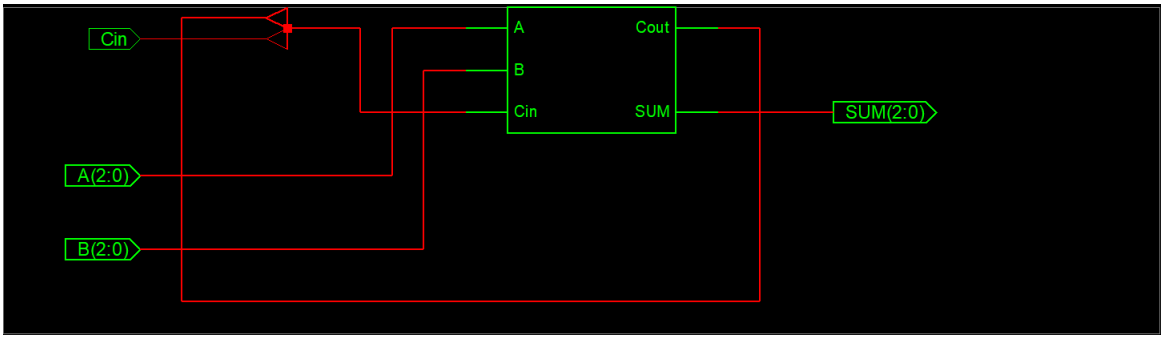

Figure 18: Carry Ripple Adder RTL Schematic

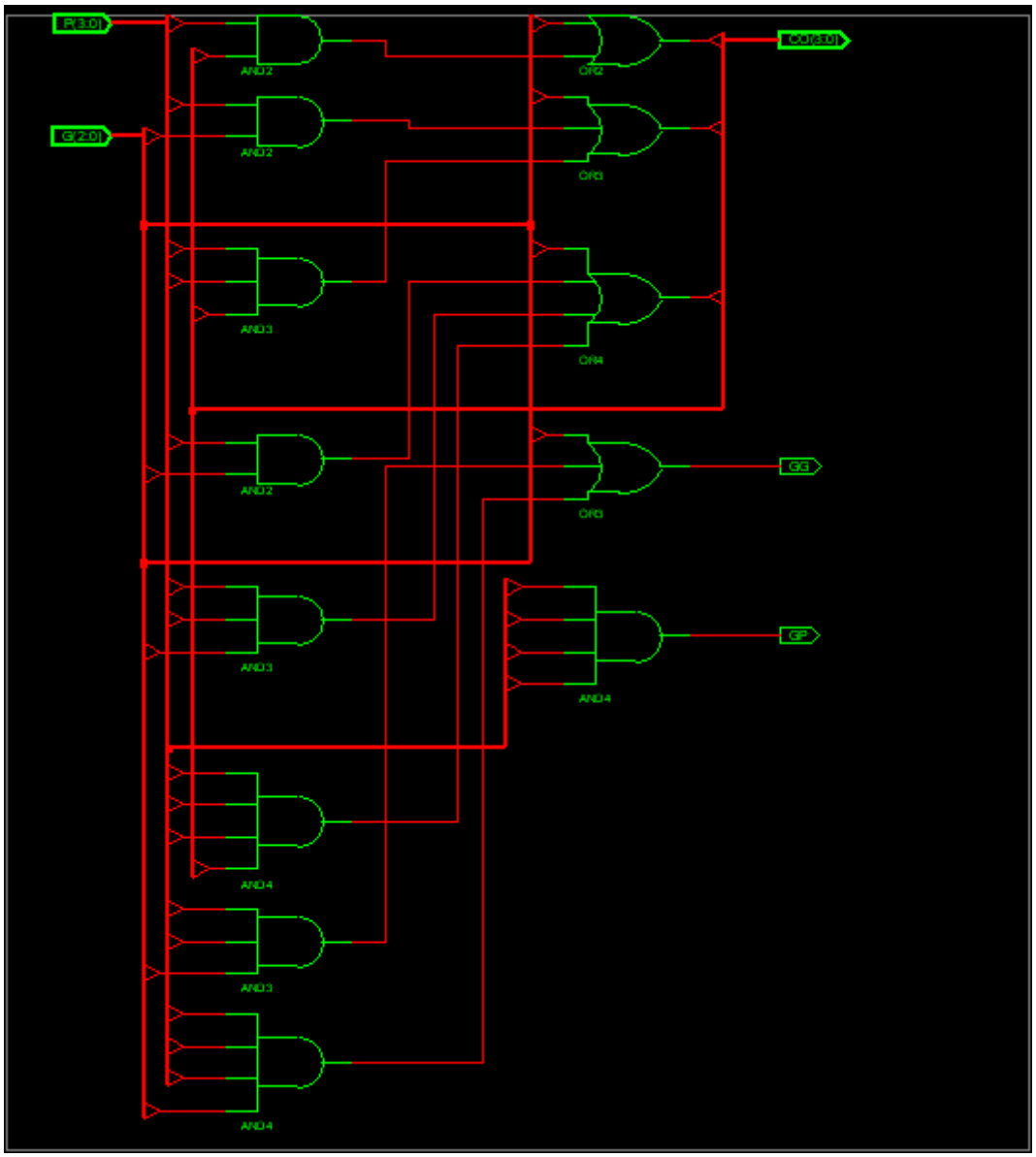

Figure 19: Carry Look Ahead Adder RTL Schematic

4 Design of flipflops

4.1 Block Diagram

Figure 20: D Flipflop Block Diagram

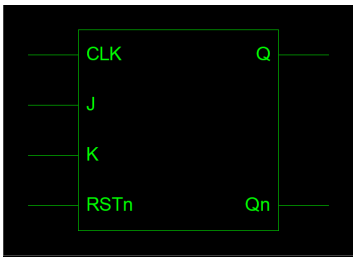

Figure 21: JK Flipflop Block Diagram

4.2 VHDL Code

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity DFFQN is
7    port(
8        CLK, D: in std_logic;
9          Q,Qn: out std_logic
10     );
11 end DFFQN;
12
13 architecture RTL of DFFQN is
14   signal DFF: std_logic;
15 begin
16   seq0: process
17     begin
18       wait until CLK'event and CLK = '1';
19         DFF <= D;
20     end process;
21     Q <= DFF; Qn <= not DFF;
22 end RTL;
```

Source Code 7: D Flipflop

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity JKFF is
7    port(
8        CLK, RSTn, J, K: in std_logic;
9          Q, Qn : out std_logic
10     );
11 end JKFF;
12
13 architecture RTL of JKFF is
14   signal FF: std_logic;
15 begin
16   seq0: process (CLK, RSTn)
17       variable JK: std_logic_vector(1 downto 0);
18     begin
19       if (RSTn = '0') then
```

```
20              FF <= '0';
21          elsif (CLK'event and CLK = '1') then
22              JK := J & K;
23              case JK is
24                  when "01" => FF <= '0';
25                  when "10" => FF <= '1';
26                  when "11" => FF <= not FF;
27                  when others => null;
28              end case;
29            end if;
30      end process;
31      Q <= FF after 2 ns;
32      Qn <= not FF after 2 ns;
33  end RTL;
```

Source Code 8: JK Flipflop

4.3 Results

4.3.1 Testbench waveform

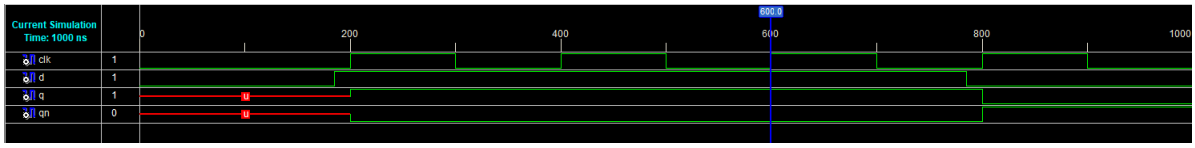

Figure 22: D Flipflop testbench waveform 1

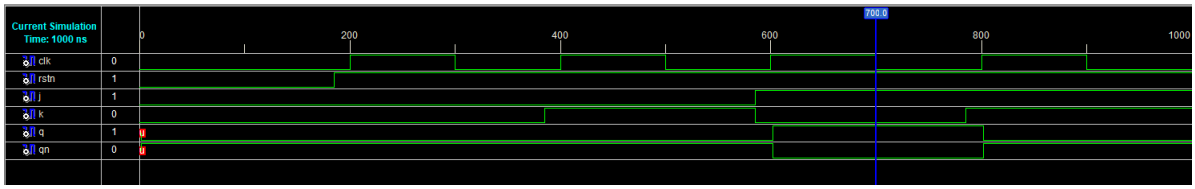

Figure 23: JK Flipflop testbench waveform 2

4.3.2 RTL schematic

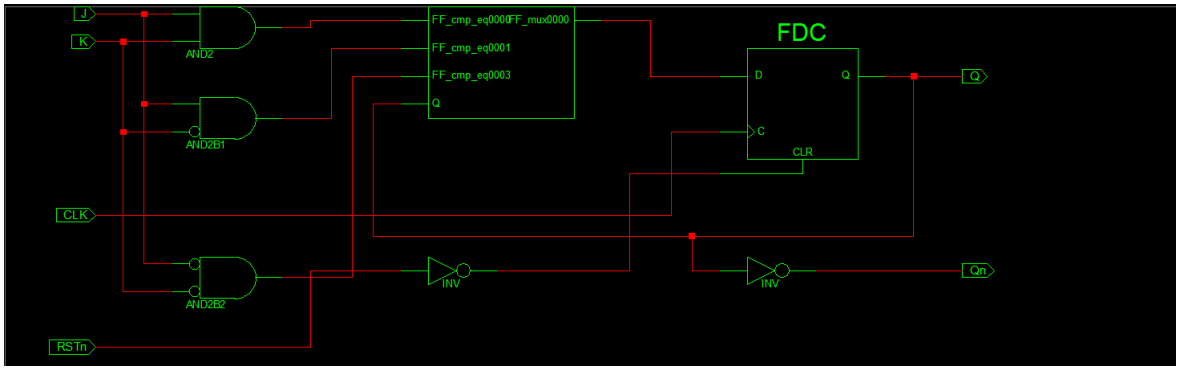

Figure 24: JK Flipflop RTL Schematic

5 Design a parallel to serial and serial to parallel converter

5.1 Block Diagram

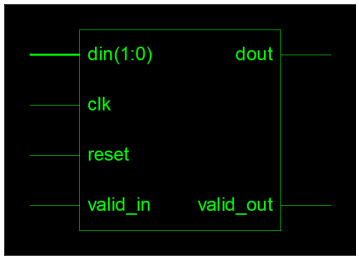

Figure 25: Parallel to Serial Converter Block Diagram

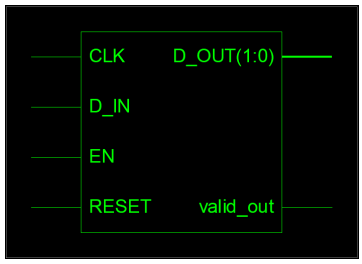

Figure 26: Serial to Parallel Converter Block Diagram

5.2 VHDL Code

```
1  -- library declaration
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  -- entity declaration
8  entity Piso2bit_mod is
9    Port ( clk : in std_logic; -- processing clock
10        reset : in std_logic; -- reset signal
11        valid_in : in std_logic; -- input valid signal
12        din : in std_logic_vector(1 downto 0); -- input data
13        dout : out std_logic; -- output data
14        valid_out : out std_logic); -- output valid signal
15 end Piso2bit_mod;
16
17 -- architecture begins here
18 architecture Behavioral of Piso2bit_mod is
19
20 -- signal declaration
21   type state is (rst, s0, s1);
22   signal ps, ns : state;
23   signal din_s : std_logic_vector(1 downto 0);
24
25   begin
26 -- process for updating the present state
27   process(clk, reset)
28     begin
29     if reset = '1' then
30       ps <= rst;
31       din_s <= "00";
32     elsif rising_edge(clk) then
33       ps <= ns;
34       if valid_in = '1' then
35         din_s <= din;
36       end if;
37     end if;
38   end process;
39
40 -- process for updating the next state
41   process(ps, valid_in)
42   begin
43     case ps is
```

```
    when rst => if valid_in = '1' then
    ns <= s0;
    else
    ns <= rst;
    end if;
    when s0 => ns <= s1;
    when s1 => if valid_in = '1' then
    ns <= s0;
    else
    ns <= rst;
    end if;
    when others => ns <= rst;
    end case;
  end process;

  process(ps)
  begin
    case ps is
    when rst => dout <= '0'; -- Here output is 0
    valid_out <= '0';

    when s0 => dout <= din_s(1); -- Here output is 2nd bit of the input
    valid_out <= '1';

    when s1 => dout <= din_s(0); -- Here output is 1st bit of the input
    valid_out <= '1';

    when others => dout <= '0';
    valid_out <= '0';

  end case;
  end process;

end Behavioral;
```

Source Code 9: Parallel to Serial Converter

```
-- library declaration
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- entity declaration
entity SIPO_2_n is
port(
    CLK : in STD_LOGIC; -- global clk
    D_IN : in STD_LOGIC; -- serial data in
    EN : in STD_LOGIC; -- enable
    RESET : in STD_LOGIC; -- asynchronous reset
    D_OUT : out STD_LOGIC_VECTOR(1 downto 0); -- parallel data out
    valid_out : out std_logic -- valid out
);
end SIPO_2_n;
-- architecture declaration
architecture RTL of SIPO_2_n is
  signal dout_i : std_logic;
  signal tog : std_logic;
begin
-- process to conevert serial to parallel data
  process (clk, reset)
  begin
    if reset = '1' then
      valid_out <= '0';
      tog <= '0';
      dout_i <= '0';
      D_out <= "00";
    elsif rising_edge(clk)then
      if en = '1' then
        tog <= not tog;
      else
        tog <= '0';
      end if;
        dout_i <= D_IN;
    if tog = '1' then
      D_OUT <= dout_i& D_in;
      valid_out <= '1';
    elsif tog = '0' then
      valid_out <= '0';
    end if;
    end if;
  end process;

end RTL;
```

Source Code 10: Serial to Parallel Converter

5.3 Results

5.3.1 Testbench waveform

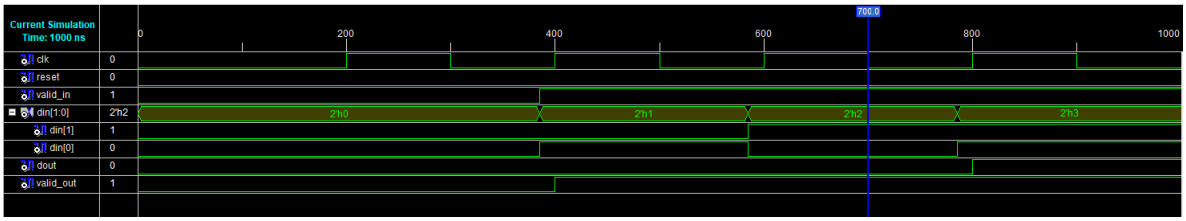

Figure 27: Parallel to Serial Converter testbench waveform

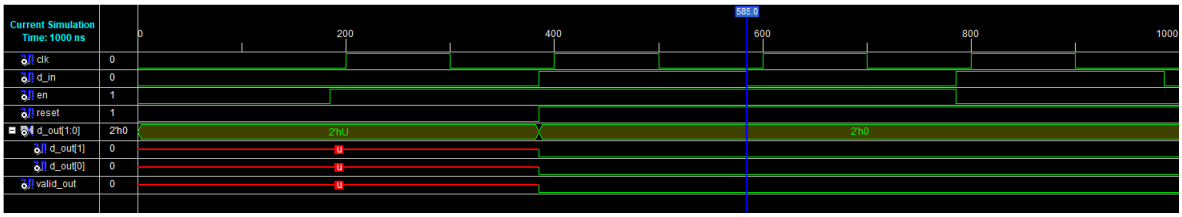

Figure 28: Serial to Parallel Converter testbench waveform

5.3.2 RTL schematic

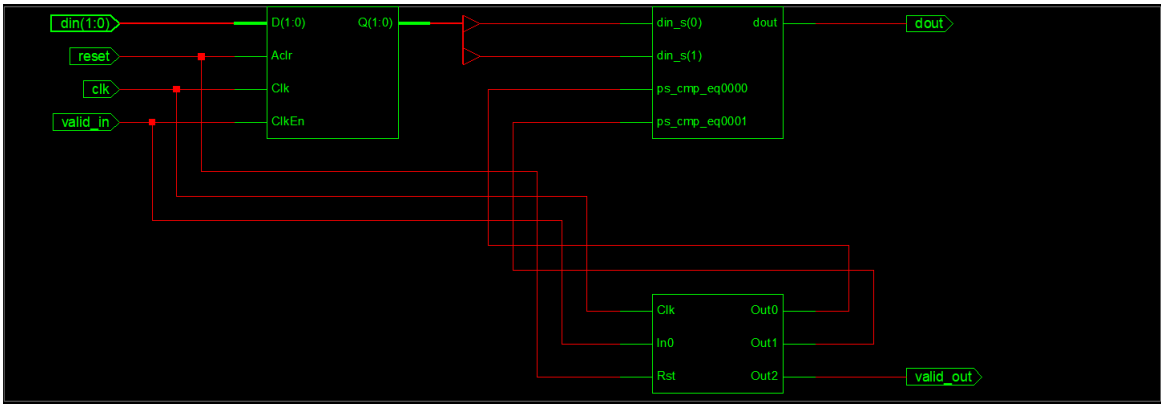

Figure 29: Parallel to Serial Converter RTL Schematic

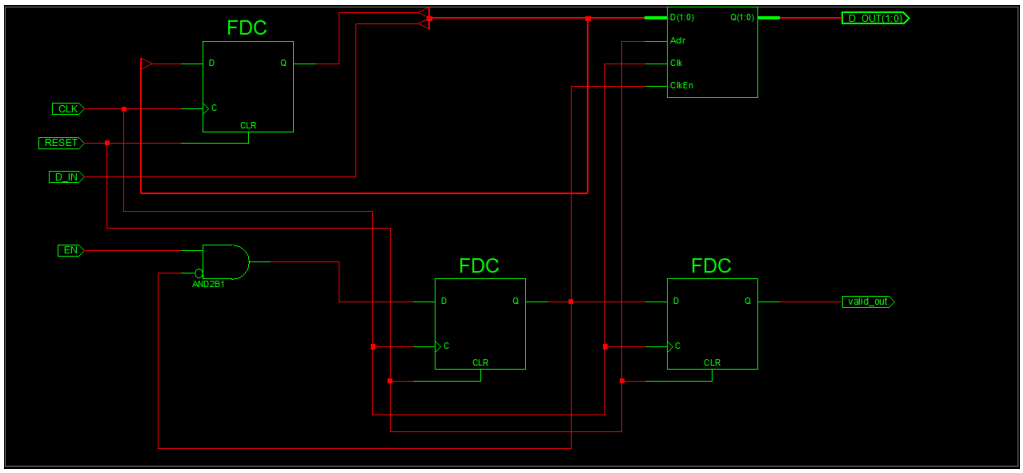

Figure 30: Serial to Parallel Converter RTL Schematic

6 Design of counters

6.1 Block Diagram

Figure 31: Asynchronous Counter Block Diagram

Figure 32: Ring Counter Block Diagram

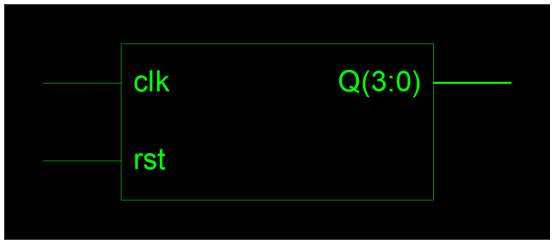

Figure 33: Johnson Counter Block Diagram

Figure 34: Updown Counter Block Diagram

6.2 VHDL Code

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.STD_LOGIC_ARITH.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  entity ACNT is
8    port(
9         RSTn, EN: in std_logic;
10          CNTR: out std_logic_vector(7 downto 0)
11    );
```

```
12  end ACNT;
13
14  architecture RTL of ACNT is
15    component JKFF
16      port(
17        CLK, RSTn, J, K: in std_logic;
18          Q, Qn : out std_logic
19      );
20      end component;
21      signal FFQ: std_logic_vector(8 downto 0);
22      signal FFQn: std_logic_vector(8 downto 0);
23      signal VDD: std_logic;
24  begin
25    VDD <= '1';
26      FFQn(0) <= EN;
27      jk0: for j in 1 to 8 generate
28        b17: JKFF port map(CLK => FFQn(j-1), RSTn => RSTn, J => VDD, K => VDD, Q => FFQ(j), Qn
      => FFQn(j));
29      end generate;
30      CNTR <= FFQ(8 downto 1);
31  end RTL;
```

Source Code 11: Asynchronous Counter

```
1   -- Ring Counter
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.STD_LOGIC_ARITH.ALL;
5   use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7   entity Ring_counter is
8       Port ( CLOCK, RESET : in  STD_LOGIC;
9              Q : out  STD_LOGIC_VECTOR(3  downto 0));
10  end Ring_counter;
11
12  architecture Behavioral of Ring_counter is
13    signal q_tmp: std_logic_vector(3 downto 0):= "0000";
14  begin
15    process(CLOCK,RESET)
16      begin
17      if RESET = '1' then
18        q_tmp <= "0001";
19      elsif Rising_edge(CLOCK) then
20        q_tmp(1) <= q_tmp(0);
21        q_tmp(2) <= q_tmp(1);
22        q_tmp(3) <= q_tmp(2);
23        q_tmp(0) <= q_tmp(3);
24      end if;
25    end process;
26  Q <= q_tmp;
27  end Behavioral;
```

Source Code 12: Ring Counter

```
1   -- Johnson Counter
2   library IEEE;
3   use IEEE.STD_LOGIC_1164.ALL;
4   use IEEE.STD_LOGIC_ARITH.ALL;
5   use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7   entity Johnson_counter is
8       Port ( clk, rst : in  STD_LOGIC;
9              Q : out  STD_LOGIC_VECTOR(3 downto 0));
10  end Johnson_counter;
11
12  architecture Behavioral of Johnson_counter is
13    signal temp: std_logic_vector(3 downto 0):= "0000";
14  begin
15    process(clk,rst)
16      begin
17        if rst = '1' then
18          temp <= "0000";
19        elsif Rising_edge(clk) then
20          temp(1) <= temp(0);
21          temp(2) <= temp(1);
22          temp(3) <= temp(2);
23          temp(0) <= not temp(3);
24        end if;
25    end process;
26    Q <= temp;
27  end Behavioral;
```

Source Code 13: Johnson Counter

```vhdl
-- Up Down Counter
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity updown_counter is
    Port ( clk: in std_logic; -- clock input
           reset: in std_logic; -- reset input
     up_down: in std_logic; -- up or down
           counter: out std_logic_vector(3 downto 0) -- output 4-bit counter
     );
end updown_counter;

architecture Behavioral of updown_counter is
signal counter_updown: std_logic_vector(3 downto 0);
begin
-- down counter
process(clk)
begin
if(rising_edge(clk)) then
    if(reset='1') then
         counter_updown <= x"0";
    elsif(up_down='1') then
         counter_updown <= counter_updown - x"1"; -- count down
  else
   counter_updown <= counter_updown + x"1"; -- count up
    end if;
 end if;
end process;
 counter <= counter_updown;

end Behavioral;
```

Source Code 14: Updown Counter

6.3 Results

6.3.1 Testbench waveform

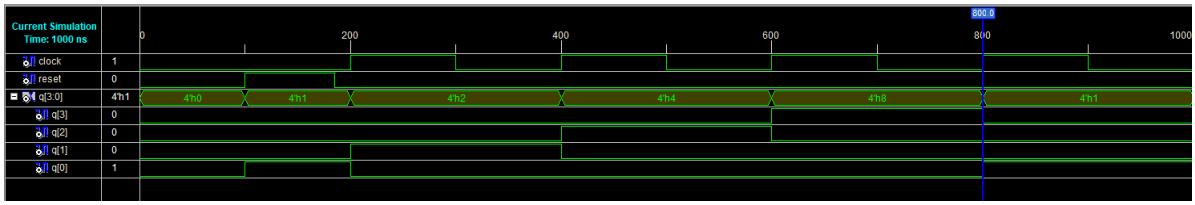

Figure 35: Ring Counter testbench waveform

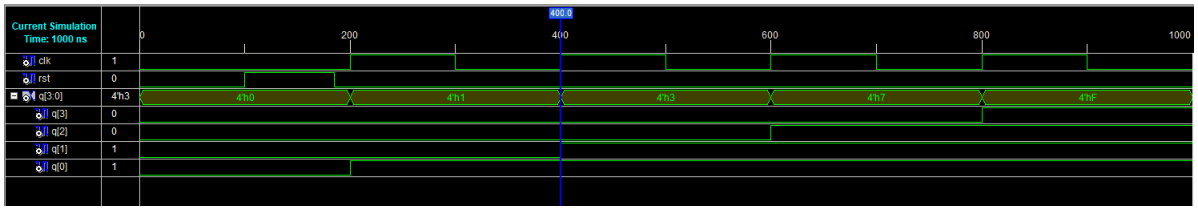

Figure 36: Johnson Counter testbench waveform

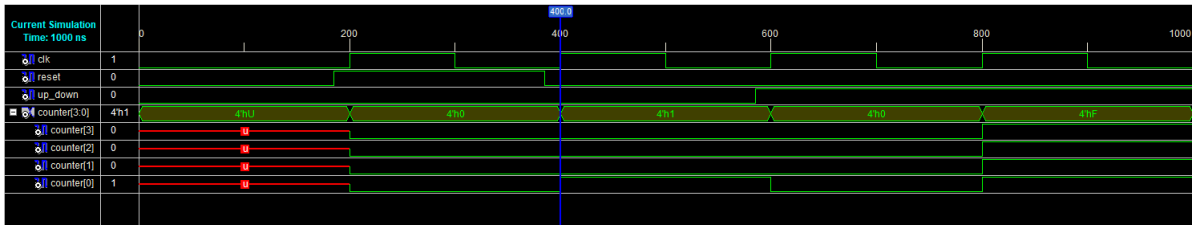

Figure 37: Updown Counter testbench waveform

6.3.2 RTL schematic

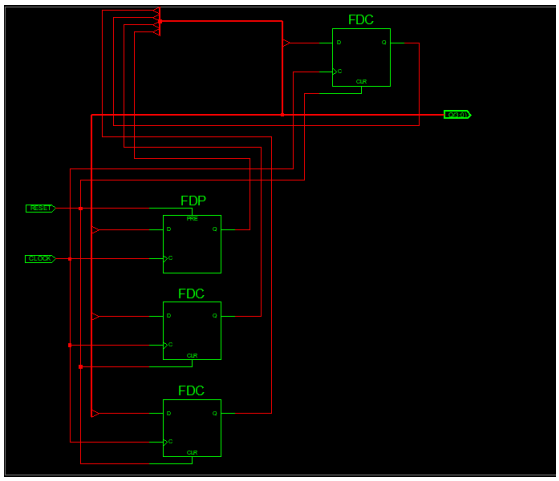

Figure 38: Ring Counter RTL Schematic

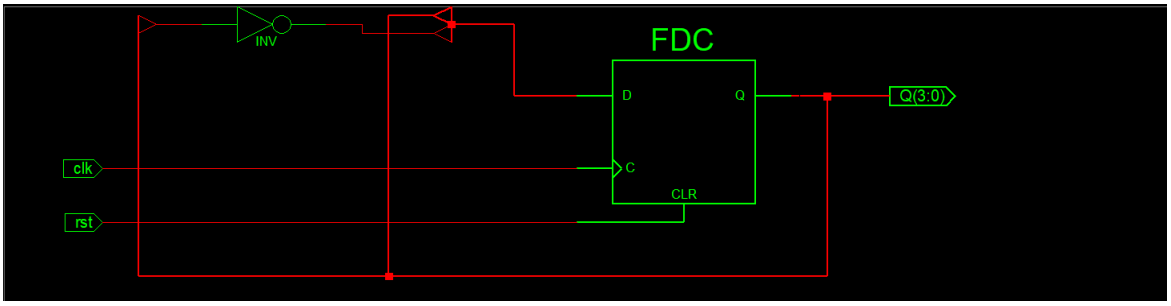

Figure 39: Johnson Counter RTL Schematic

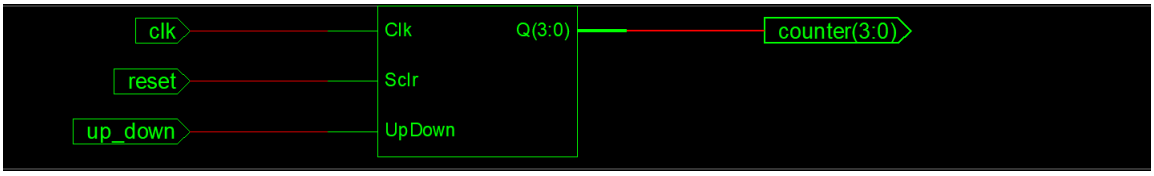

Figure 40: Updown Counter RTL Schematic

7 Design a FSM

7.1 Block Diagram

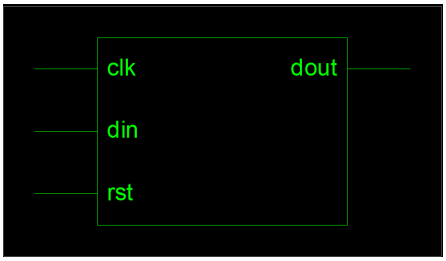

Figure 41: Mealy FSM Block Diagram

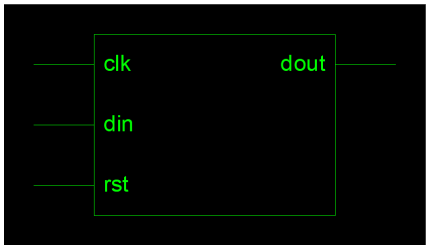

Figure 42: Moore FSM Block Diagram

7.2 VHDL Code

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity mealy is
5    Port ( clk : in STD_LOGIC;
6      din : in STD_LOGIC;
7      rst : in STD_LOGIC;
8      dout : out STD_LOGIC);
9  end mealy;
10
11 architecture Behavioral of mealy is
12   type state is (st0, st1, st2, st3);
13   signal present_state, next_state : state;
14 begin
15
16   syncronous_process : process (clk)
17   begin
18     if rising_edge(clk) then
19       if (rst = '1') then
20         present_state <= st0;
21       else
22         present_state <= next_state;
23       end if;
24     end if;
25   end process;
26
27   next_state_and_output_decoder : process(present_state, din)
28   begin
29     dout <= '0'; case (present_state) is when st0 =>
30     if (din = '1') then
31       next_state <= st1;
32       dout <= '0';
33     else
34       next_state <= st0;
35       dout <= '0';
36     end if;
37     when St1 =>
38       if (din = '1') then
39         next_state <= st1;
40         dout <= '0';
41       else
42         next_state <= st2;
43         dout <= '0';
44       end if;
```

```
    when St2 =>
      if (din = '1') then
        next_state <= st1;
        dout <= '1';
      else
        next_state <= st0;
        dout <= '0';
      end if;
    when others =>
      next_state <= st0;
      dout <= '0';
    end case;
  end process;
end Behavioral;
```

Source Code 15: Mealy FSM

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

entity moore is
  Port ( clk : in STD_LOGIC;
  din : in STD_LOGIC;
  rst : in STD_LOGIC;
  dout : out STD_LOGIC);
end moore;

architecture Behavioral of moore is
  type state is (st0, st1, st2, st3);
  signal present_state, next_state : state;
begin

  synchronous_process: process (clk)
  begin
    if rising_edge(clk) then
      if (rst = '1') then
        present_state <= st0;
      else
        present_state <= next_state;
      end if;
    end if;
  end process;

  output_decoder : process(present_state, din)
  begin
    next_state <= st0;
    case (present_state) is when st0 =>
    if (din = '1') then
      next_state <= st1;
    else
      next_state <= st0; end if; when st1 =>
    if (din = '1') then
      next_state <= st1;
    else
      next_state <= st2; end if; when st2 =>
    if (din = '1') then
      next_state <= st3;
    else
      next_state <= st0; end if; when st3 =>
    if (din = '1') then
      next_state <= st1;
    else
      next_state <= st2; end if; when others =>
      next_state <= st0;
    end case;
    end process;
    next_state_decoder : process(present_state)
    begin
    case (present_state) is when st0 =>
      dout <= '0'; when st1 =>
      dout <= '0'; when st2 =>
      dout <= '0'; when st3 =>
      dout <= '1'; when others =>
      dout <= '0';
    end case;
  end process;

end Behavioral;
```

Source Code 16: Moore FSM

7.3 Results

7.3.1 Testbench waveform

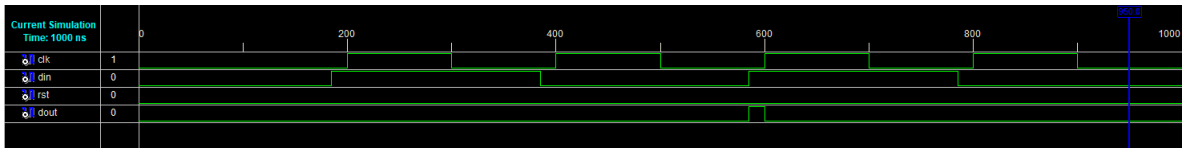

Figure 43: Mealy FSM testbench waveform

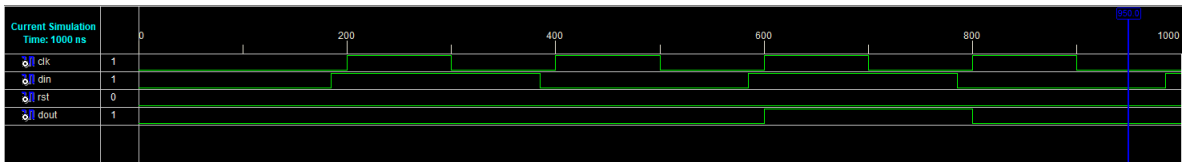

Figure 44: Moore FSM testbench waveform

7.3.2 RTL schematic

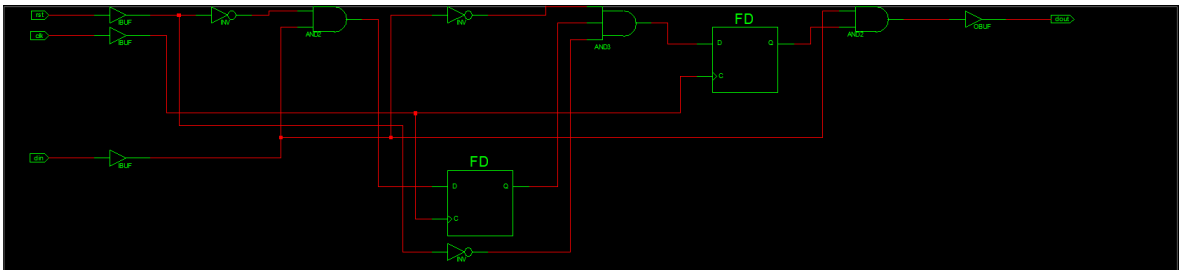

Figure 45: Mealy FSM RTL Schematic

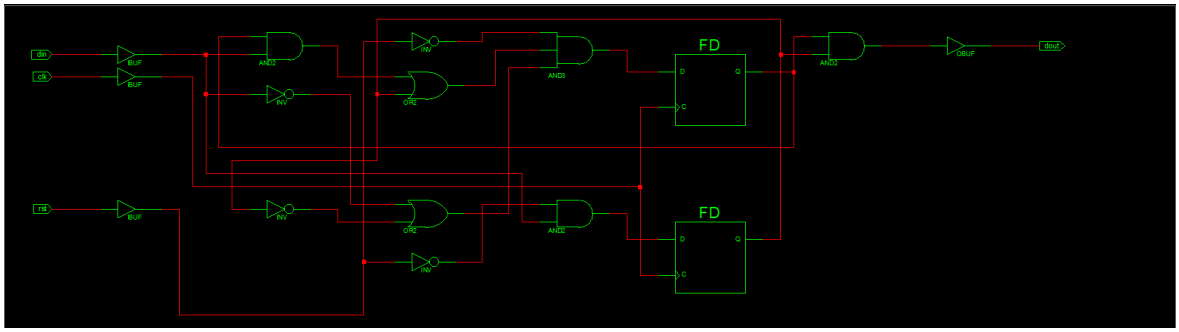

Figure 46: Moore FSM RTL Schematic