# NANO-PROCESSOR_LAB-REPORT

## INDEX NO :- 180048D
## PARTNER INDEX NO :- 180711F

## HA

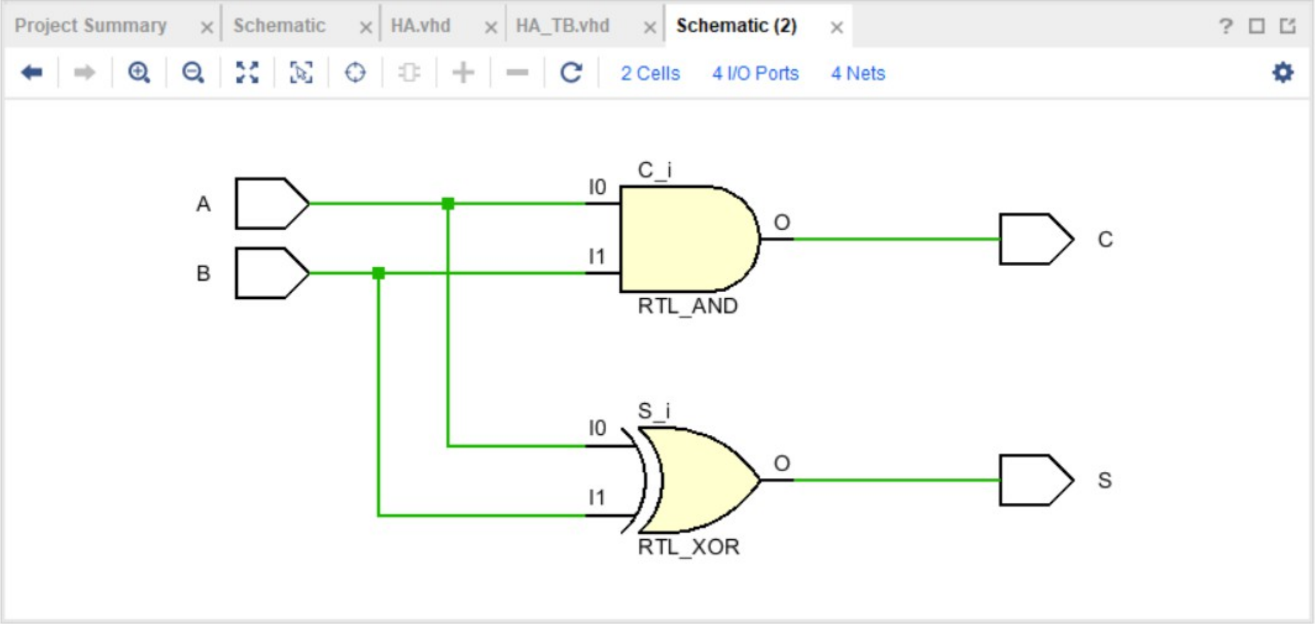
### Tables

| A | B | C | S |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

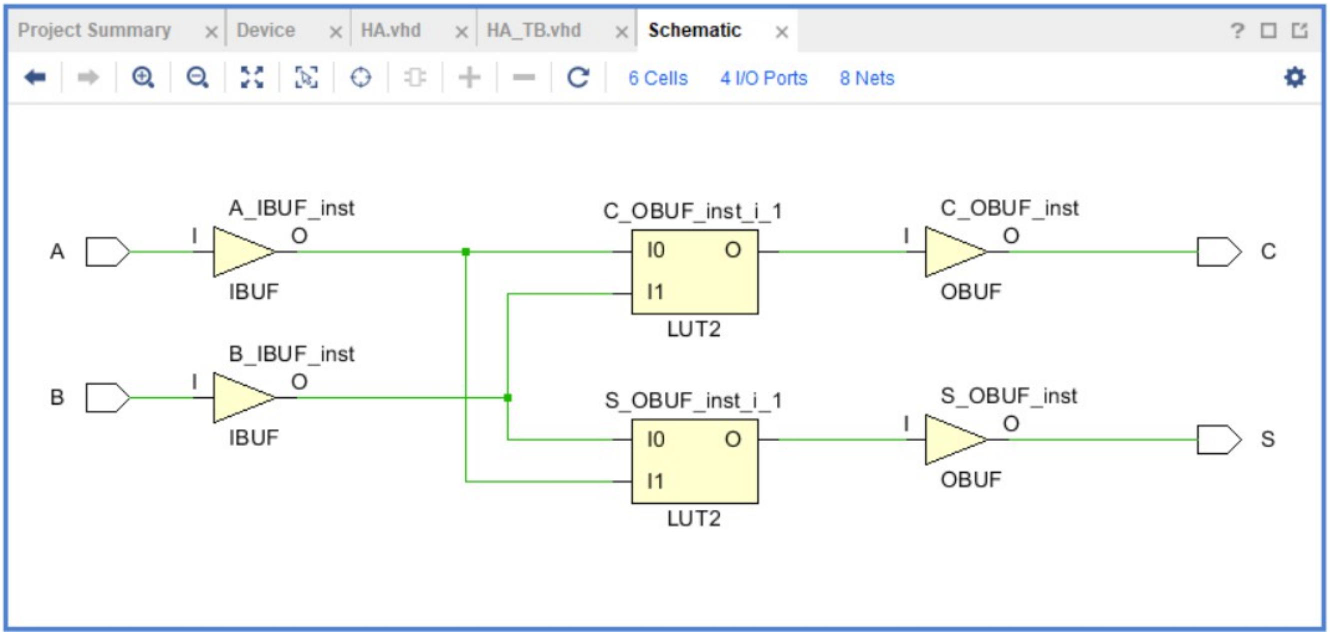
### Code

```
entity HA is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          C : out STD_LOGIC;
          S : out STD_LOGIC);
end HA;

architecture Behavioral of HA is

begin
   S <= A XOR  B;
   C <= A AND B;

end Behavioral;
```

### RTL



## Synthesized Design

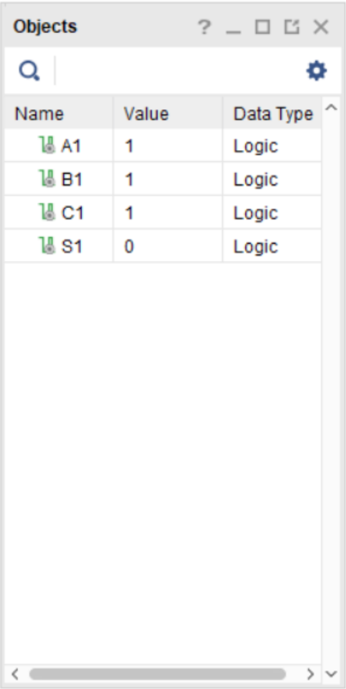


## Simulation_Code

```
entity HA_TB is
--  Port ( );
end HA_TB;

architecture Behavioral of HA_TB is

COMPONENT HA
```

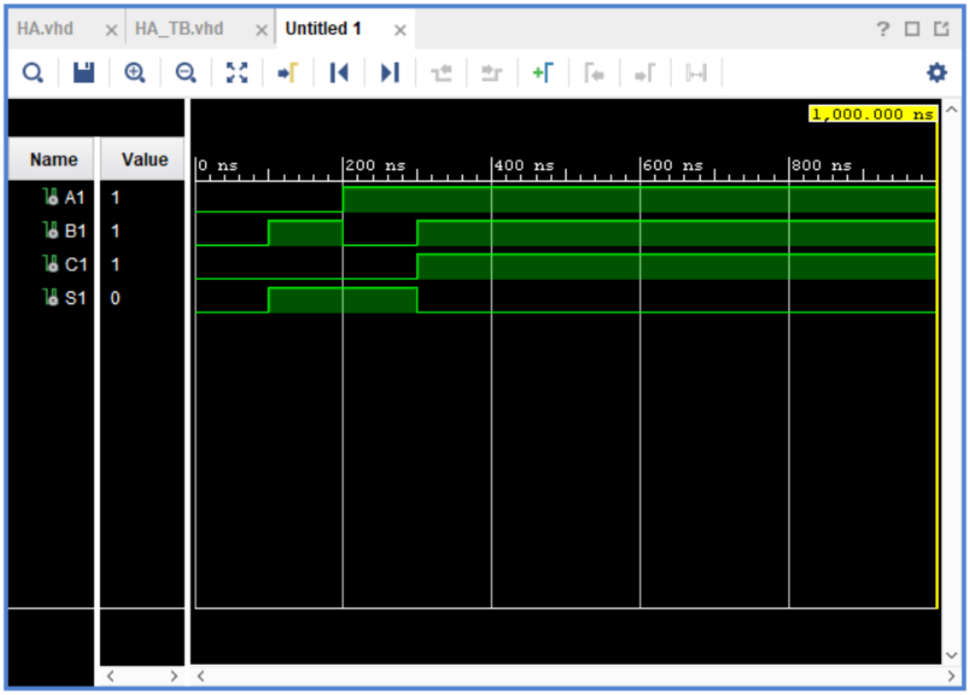
```
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          C : out STD_LOGIC;
          S : out STD_LOGIC);
END COMPONENT;

SIGNAL A1,B1,C1,S1 :STD_LOGIC;


begin

UUT: HA PORT MAP (
   A => A1,
   B => B1,
   S => S1,
   C => C1);
SIM: PROCESS
BEGIN

   A1 <='0';
   B1 <='0';
   WAIT FOR 100 NS;
   A1 <='0';
   B1 <='1';
   WAIT FOR 100 NS;
   A1 <='1';
   B1 <='0';
   WAIT FOR 100 NS;
   A1 <='1';
   B1 <='1';
   WAIT;


END PROCESS;

end Behavioral;
```

**Behavioral Simulation**



Objects

| Name | Value | Data Type |
|---|---|---|
| A1 | 1 | Logic |
| B1 | 1 | Logic |
| C1 | 1 | Logic |
| S1 | 0 | Logic |



| Name | Value |
|---|---|
| A1 | 1 |
| B1 | 1 |
| C1 | 1 |
| S1 | 0 |

# FA

## Tables

| A | B | Cin | S | Cout |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

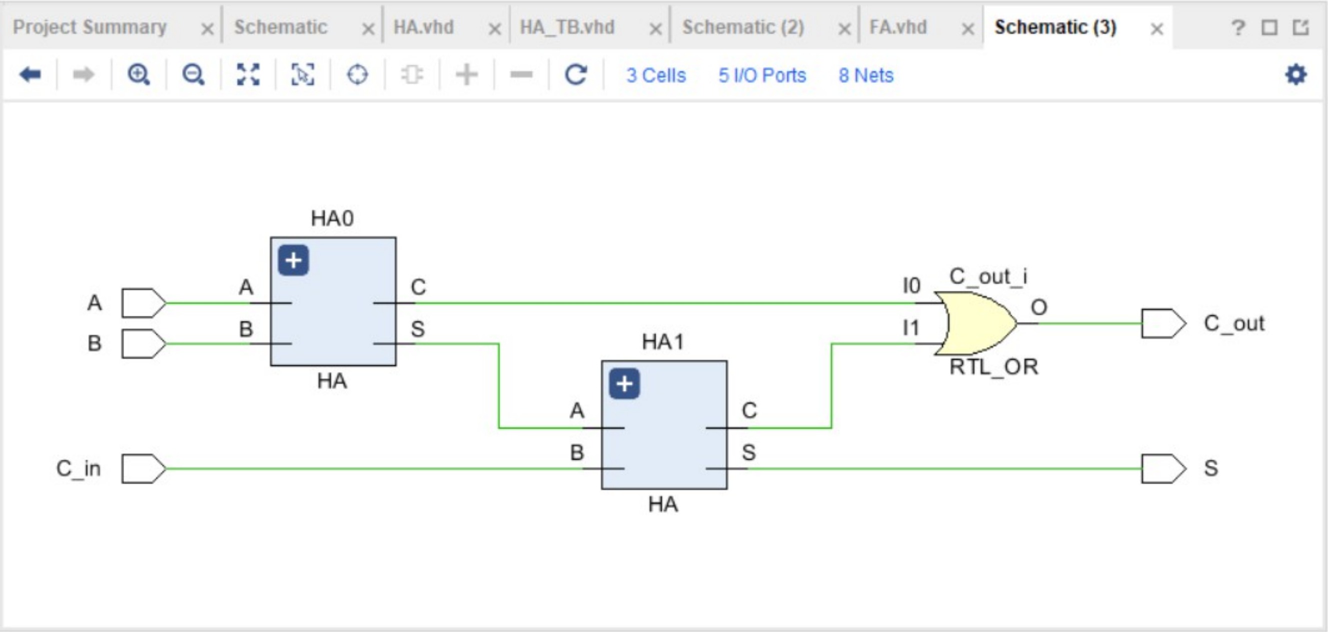
S= (A xor B) xor C_in
C= (A and B) or ((A xor B) and C_in)

## Code

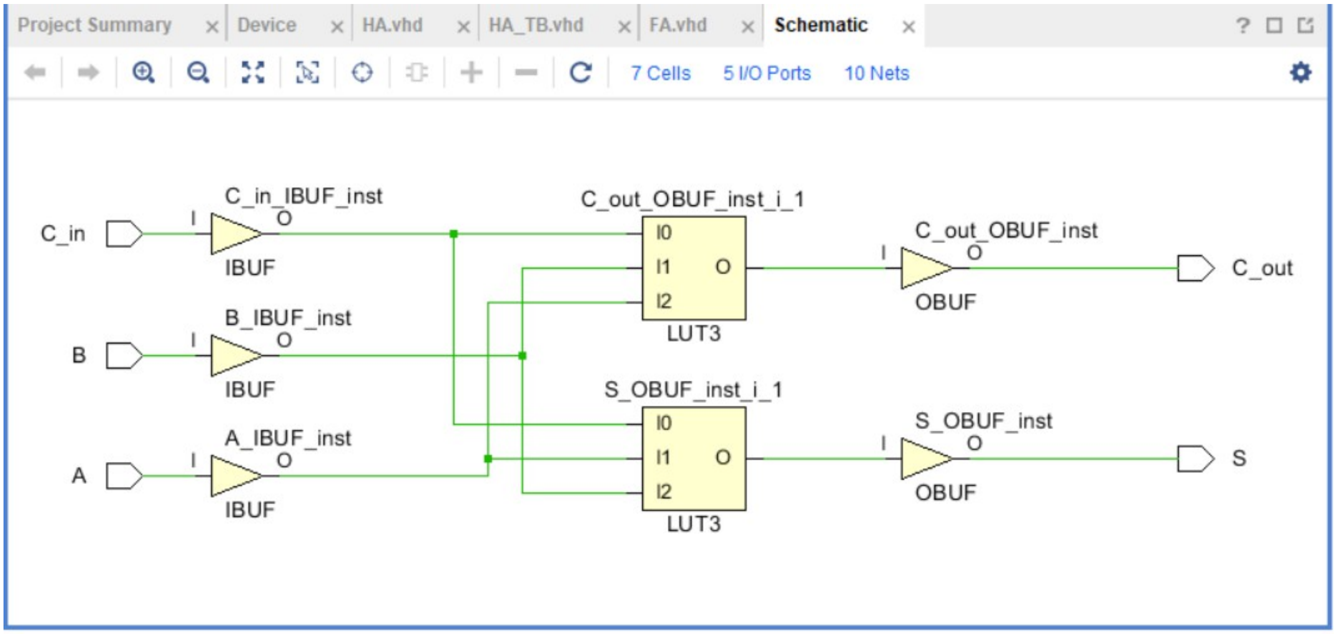
```
entity FA is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          C_in : in STD_LOGIC;
          S : out STD_LOGIC;
          C_out : out STD_LOGIC);
end FA;
```

```
architecture Behavioral of FA is
component HA
     Port ( A : in STD_LOGIC;
            B : in STD_LOGIC;
            C : out STD_LOGIC;
            S : out STD_LOGIC);
end component;

signal ha0_s, ha0_c, hal_c :std_logic;
begin

HA0 : HA PORT MAP (
    A => A,
    B => B,
    C => ha0_c,
    S => ha0_s);
HA1 : HA  PORT MAP (
    A => ha0_s,
    B => C_in ,
    C => hal_c,
    S => S);

  c_out <= ha0_c or hal_c;
end Behavioral;
```

## RTL



## Synthesized Design



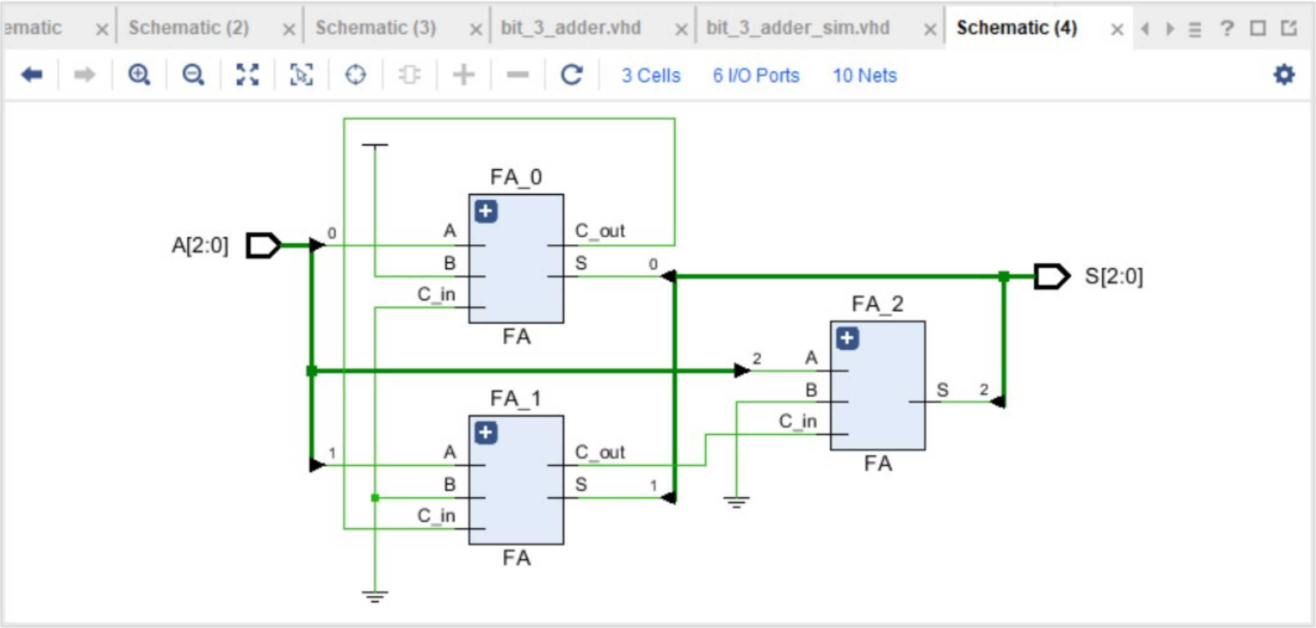
# 3-BIT_ADDER

## Code

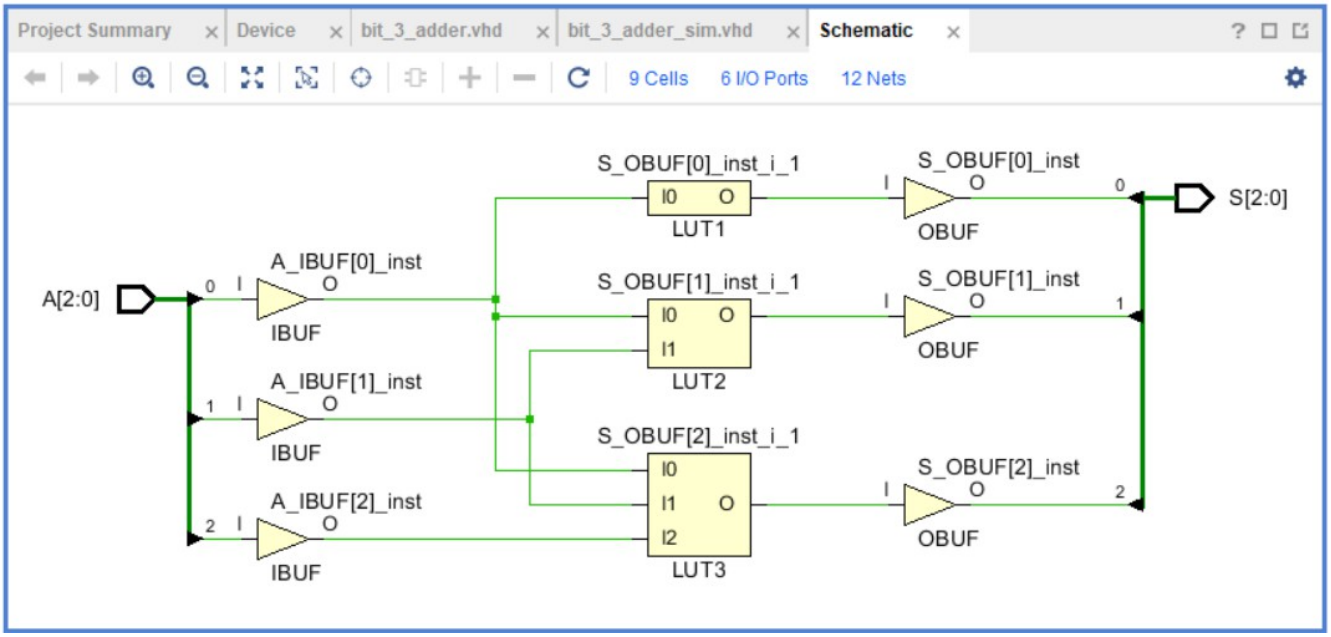
```
entity bit_3_adder is
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0)
           );
end bit_3_adder;

architecture Behavioral of bit_3_adder is
component FA
    port (
        A: in std_logic;
        B: in std_logic;
        C_in: in std_logic;
        S: out std_logic;
        C_out: out std_logic);
end component;SIGNAL
FA0_S, FA0_C, FA1_S, FA1_C, FA2_S, FA2_C, FA3_S, FA3_C ,C_out1
: std_logic;
begin
FA_0 : FA
    port map (
        A => A(0),
        B => '1',
        C_in => '0',
        S => S(0),
        C_Out => FA0_C);
FA_1 : FA
```

```
    port map (
        A => A(1),
        B => '0',
        C_in => FA0_C,
        S => S(1),
        C_Out => FA1_C);
FA_2 : FA
    port map (
        A => A(2),
        B => '0',
        C_in => FA1_C,
        S => S(2),
        C_Out => C_out1);


end Behavioral;
```

## RTL



## Synthesized Design

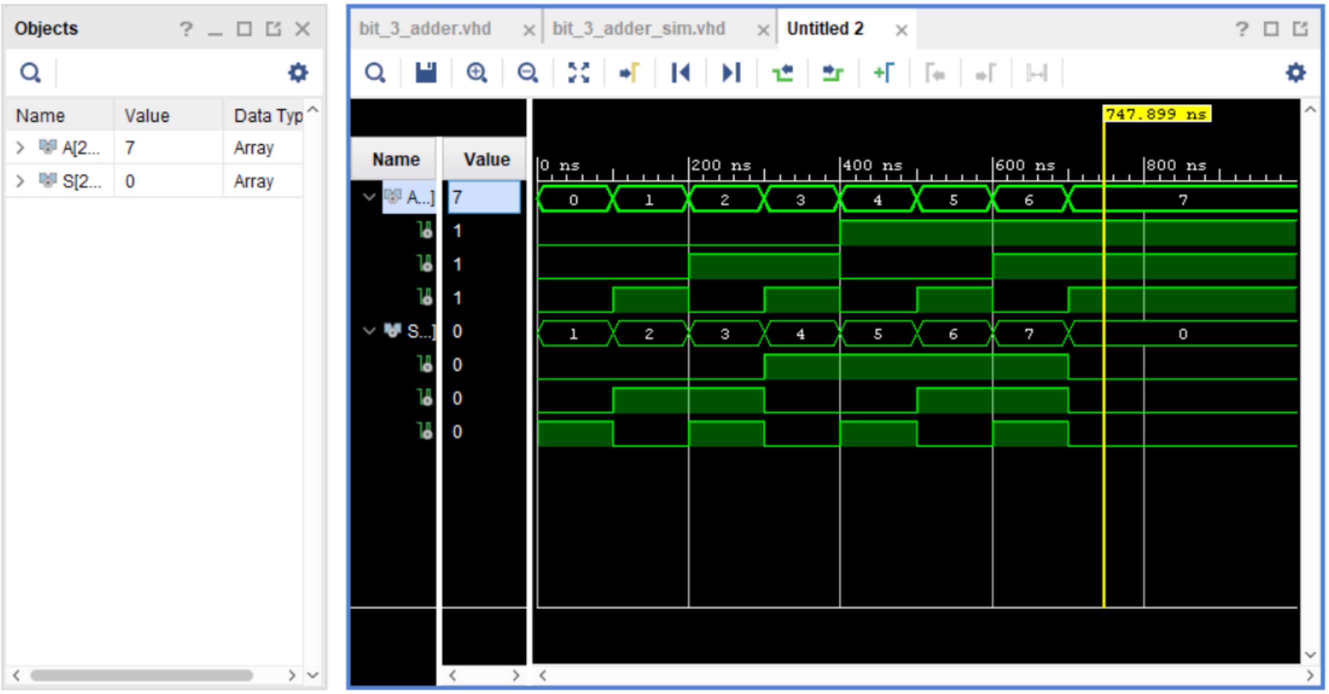


**Simulation_Code**

```
entity bit_3_adder_sim is

end bit_3_adder_sim;

architecture Behavioral of bit_3_adder_sim is
 component bit_3_adder
    Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
           S : out STD_LOGIC_VECTOR (2 downto 0));
end component;


signal A : std_logic_vector (2 downto 0 ) := (others => '0');
signal S : std_logic_vector (2 downto 0 );


begin
uut : bit_3_adder port map
( A=> A,
S => S);
stiM_PROC: PROCESS
BEGIN

A<= "000";
wait for 100 ns;
A<= "001";
wait for 100 ns;
A<= "010";
```

```
wait for 100 ns;
A<= "011";
wait for 100 ns;
A<= "100";
wait for 100 ns;
A<= "101";
wait for 100 ns;
A<= "110";
wait for 100 ns;
A<= "111";
wait;

end process;
end;
```

## Behavioral Simulation



| Name | Value | Data Typ |
|---|---|---|
| A[2... | 7 | Array |
| S[2... | 0 | Array |

| Name | Value |
|---|---|
| A...] | 7 |
| | 1 |
| | 1 |
| | 1 |
| S...] | 0 |
| | 0 |
| | 0 |
| | 0 |

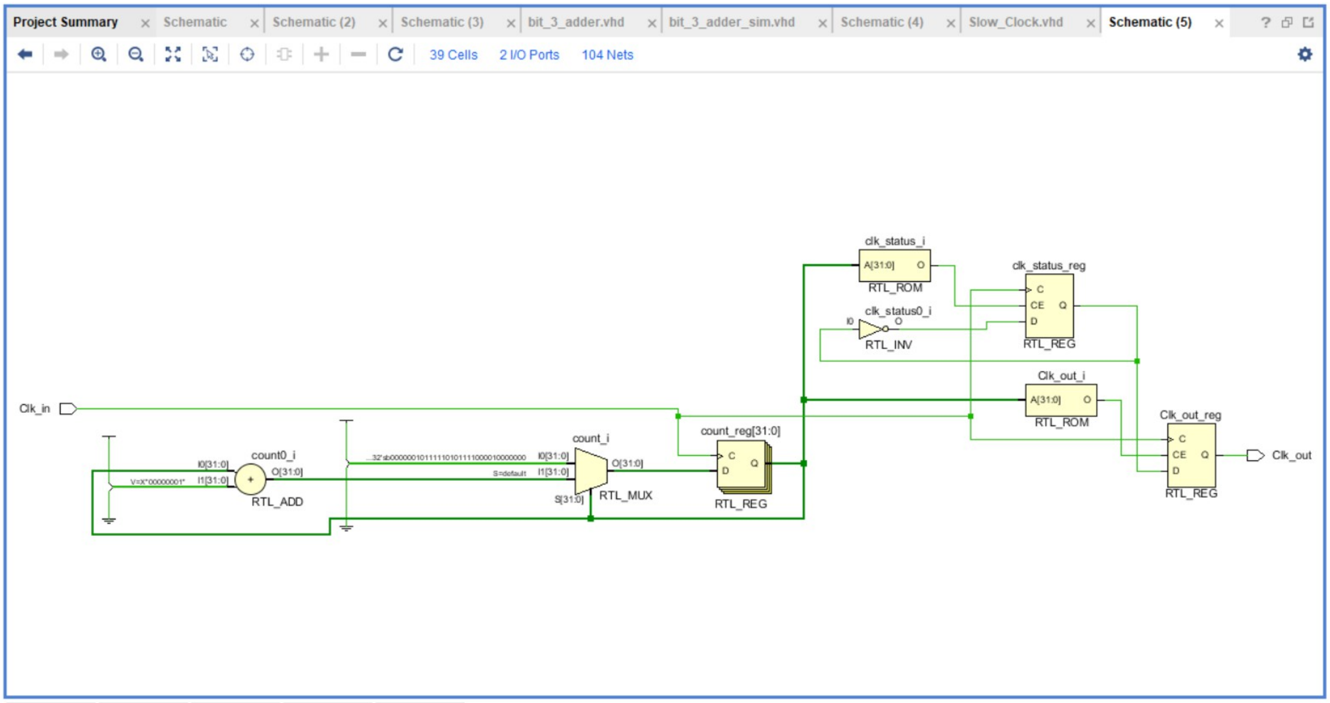
# Slow_Clock

## Code

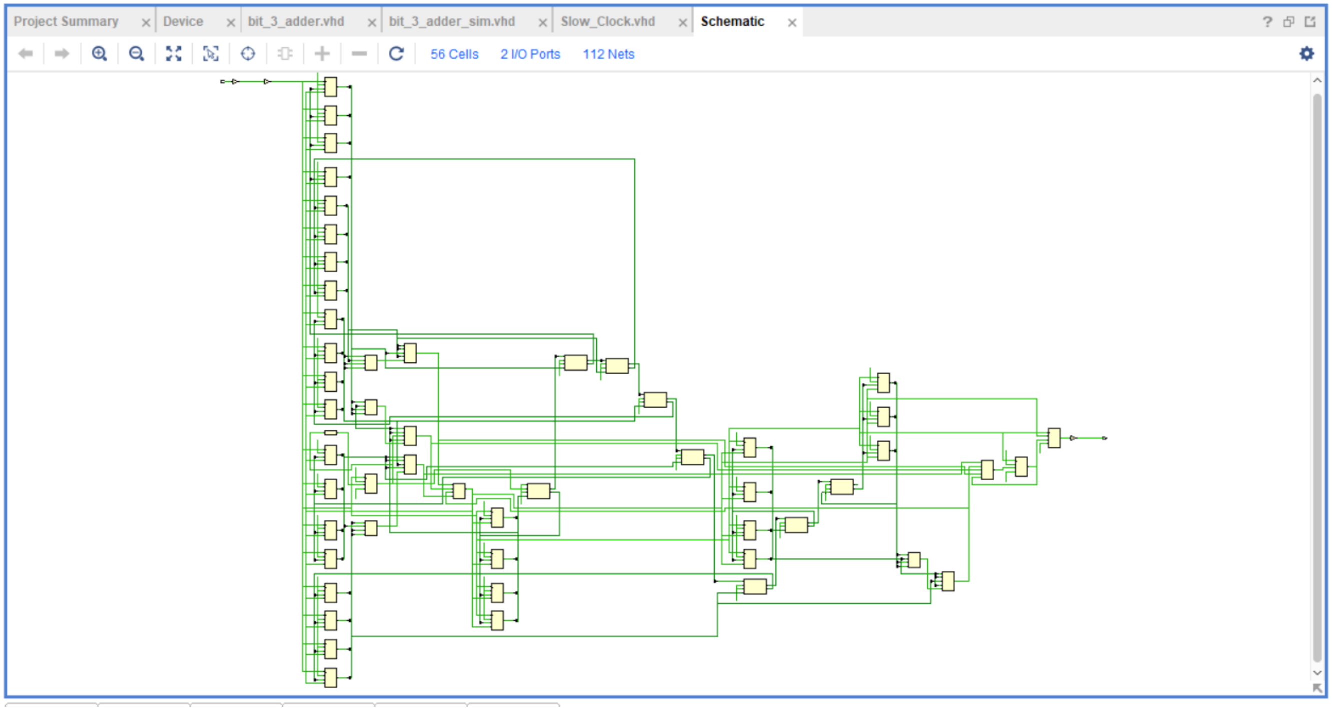
```
entity Slow_Clock is
   Port ( Clk_in : in STD_LOGIC;
          Clk_out : out STD_LOGIC);
end Slow_Clock;

architecture Behavioral of Slow_Clock is
```

```
signal count : integer := 1;
signal clk_status : std_logic :='0';

begin
   process (Clk_in) begin
      if (rising_edge(Clk_in)) then
         count <= count +1;
         if (count = 50000000) then
            clk_status <= not clk_status;
            Clk_out <= clk_status;
            count <= 1;
         end if;
      end if;
    end process;

end Behavioral;
```

## RTL



## Synthesized Design



## 2 to 1 multiplexer

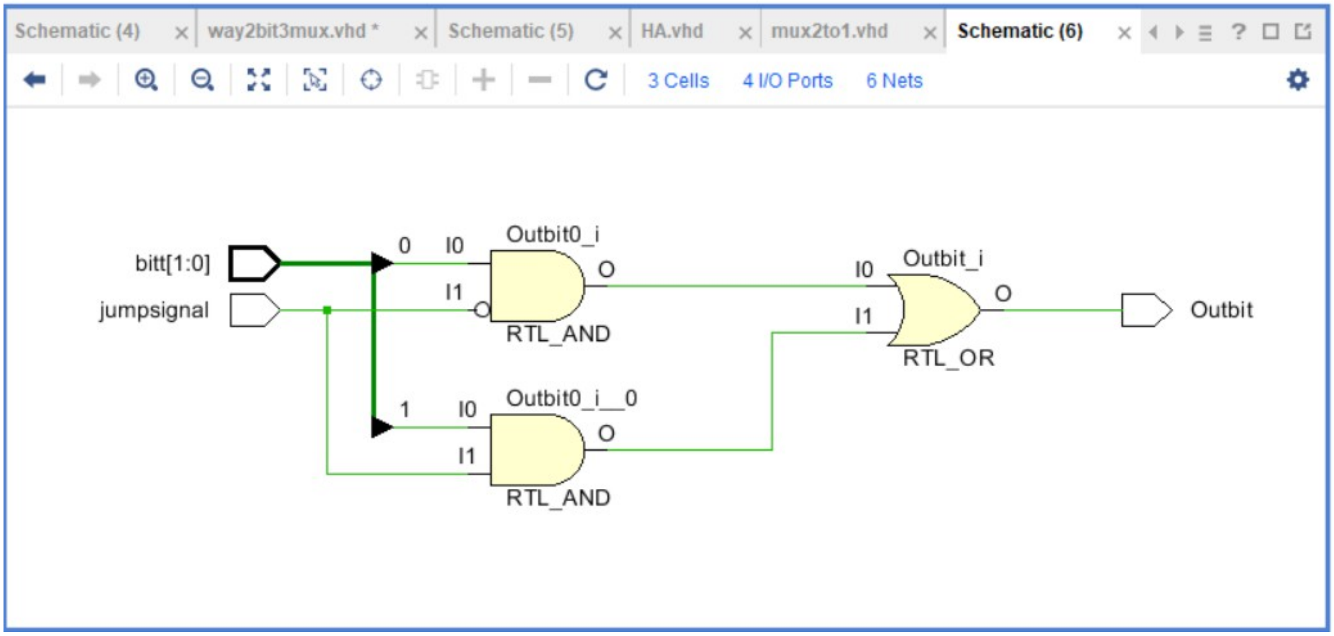
### Tables

| S | D |
|---|---|
| 0 | D(0) |
| 1 | D(1) |

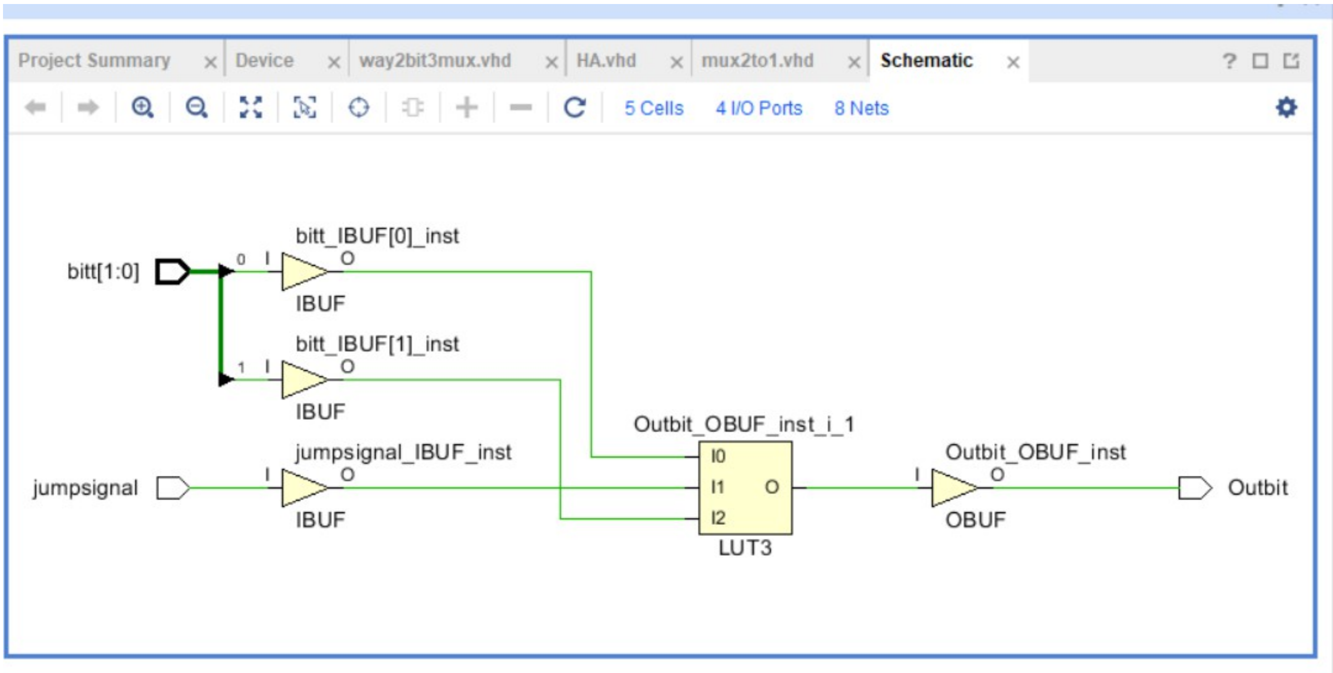
### Code

```
entity mux2to1 is
    Port ( jumpsignal : in STD_LOGIC;
           bitt : in STD_LOGIC_VECTOR (1 downto 0);
           Outbit : out STD_LOGIC);
end mux2to1;

architecture Behavioral of mux2to1 is

begin

outbit <= (bitt(0)and (not jumpsignal)) or (bitt(1) and jumpsignal);


end Behavioral;
```

### RTL



## Synthesized Design



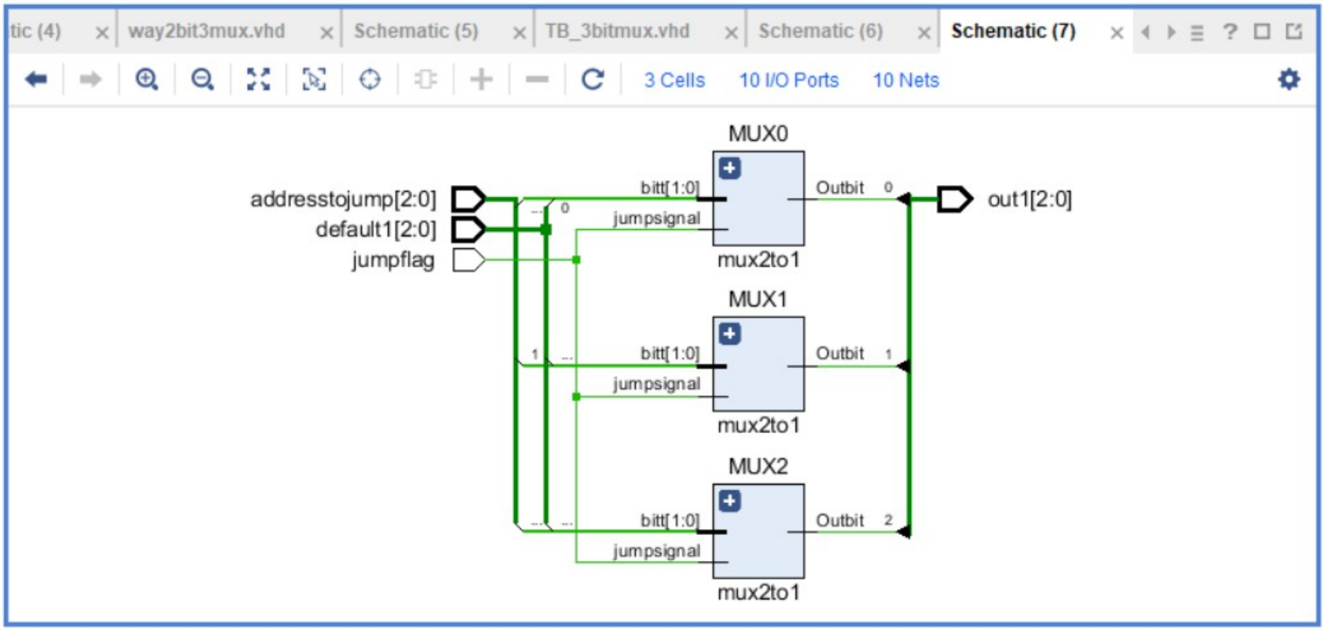
# 2 WAY 3 BIT MUX

## Code

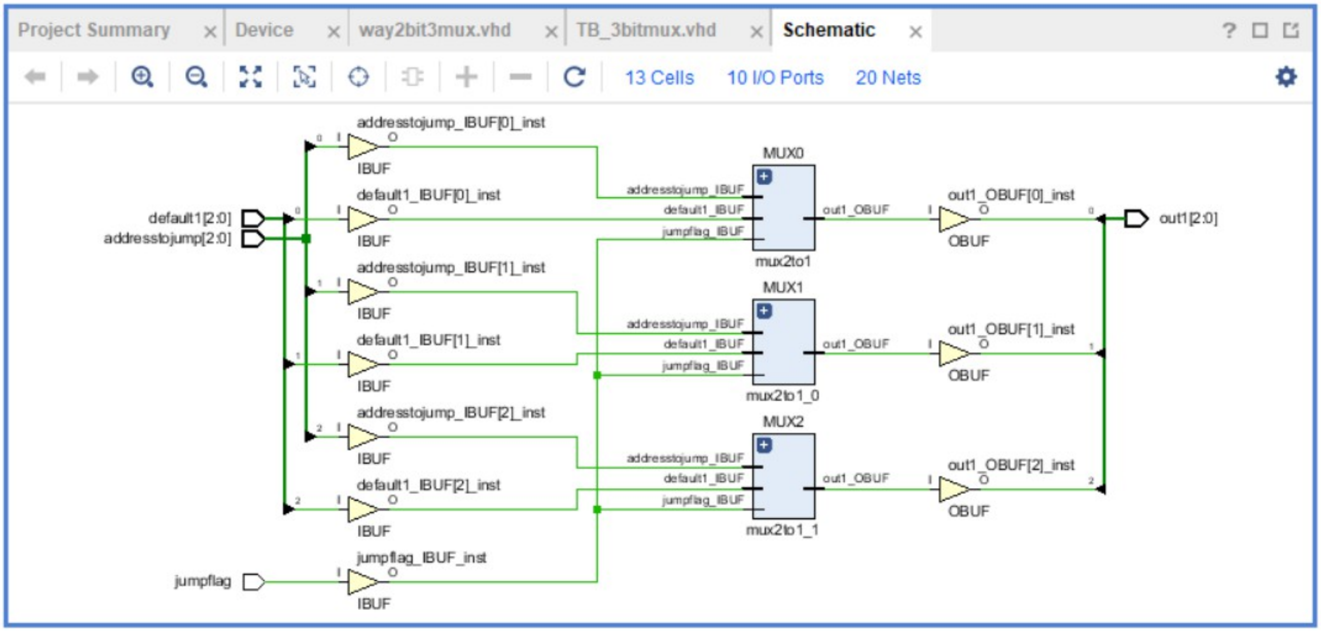
```
entity way2bit3mux is
    Port ( jumpflag : in STD_LOGIC;
           addresstojump : in STD_LOGIC_VECTOR (2 downto 0);
           default1 : in STD_LOGIC_VECTOR (2 downto 0);
```

```
        out1 : out STD_LOGIC_VECTOR (2 downto 0));
end way2bit3mux;

architecture Behavioral of way2bit3mux is

component mux2to1
   Port ( jumpsignal : in STD_LOGIC;
        bitt : in STD_LOGIC_VECTOR (1 downto 0);
        Outbit : out STD_LOGIC);
end component;

signal ha0_s, ha0_c, hal_c :std_logic;
begin

MUX0 : mux2to1 PORT MAP (
   jumpsignal => jumpflag,
   bitt(0) => default1(0),
   bitt(1) => addresstojump(0),
   outbit  => out1(0));
MUX1 : mux2to1 PORT MAP (
      jumpsignal => jumpflag,
      bitt(0) => default1(1),
      bitt(1) => addresstojump(1),
      outbit  => out1(1));
MUX2 : mux2to1 PORT MAP (
         jumpsignal => jumpflag,
         bitt(0) => default1(2),
         bitt(1) => addresstojump(2),
         outbit  => out1(2));


end Behavioral;
```

**RTL**



## Synthesized Design

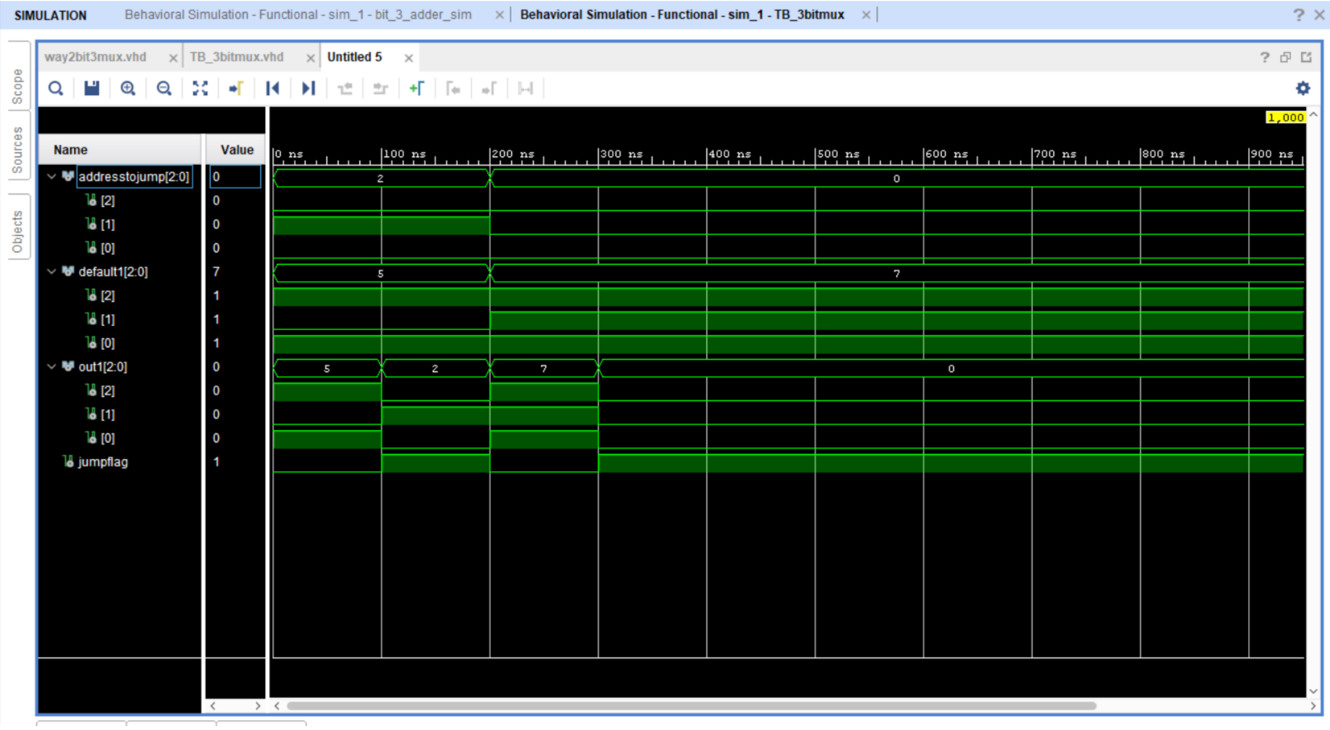


## Simulation_Code

```
entity TB_3bitmux is
--  Port ( );
end TB_3bitmux;

architecture Behavioral of TB_3bitmux is
 component way2bit3mux
```

```
   Port ( jumpflag : in STD_LOGIC;
          addresstojump : in STD_LOGIC_VECTOR (2 downto 0);
          default1 : in STD_LOGIC_VECTOR (2 downto 0);
          out1 : out STD_LOGIC_VECTOR (2 downto 0));
end component;


signal addresstojump : std_logic_vector (2 downto 0 ) := (others => '0');
signal default1 : std_logic_vector (2 downto 0 );
signal out1 : std_logic_vector (2 downto 0 );
signal jumpflag : std_logic;


begin
uut : way2bit3mux port map
( jumpflag => jumpflag,
  addresstojump => addresstojump,
  default1 =>default1,
  out1 => out1);
stiM_PROC: PROCESS
BEGIN

jumpflag<='0';
addresstojump<="010";
default1<="101";
wait for 100 ns;

jumpflag<='1';
addresstojump<="010";
default1<="101";
wait for 100 ns;

jumpflag<='0';
addresstojump<="000";
default1<="111";
wait for 100 ns;

jumpflag<='1';
addresstojump<="000";
default1<="111";
wait;

end process;
end;
```

## Behavioral Simulation

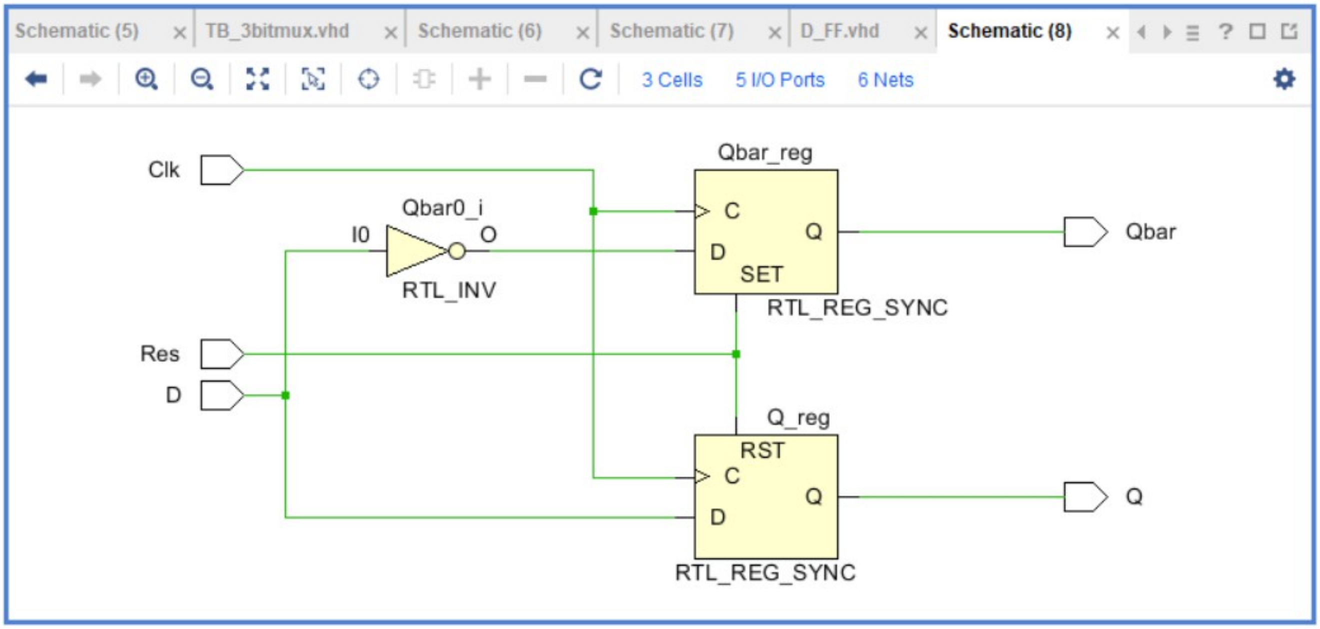


# D_FF

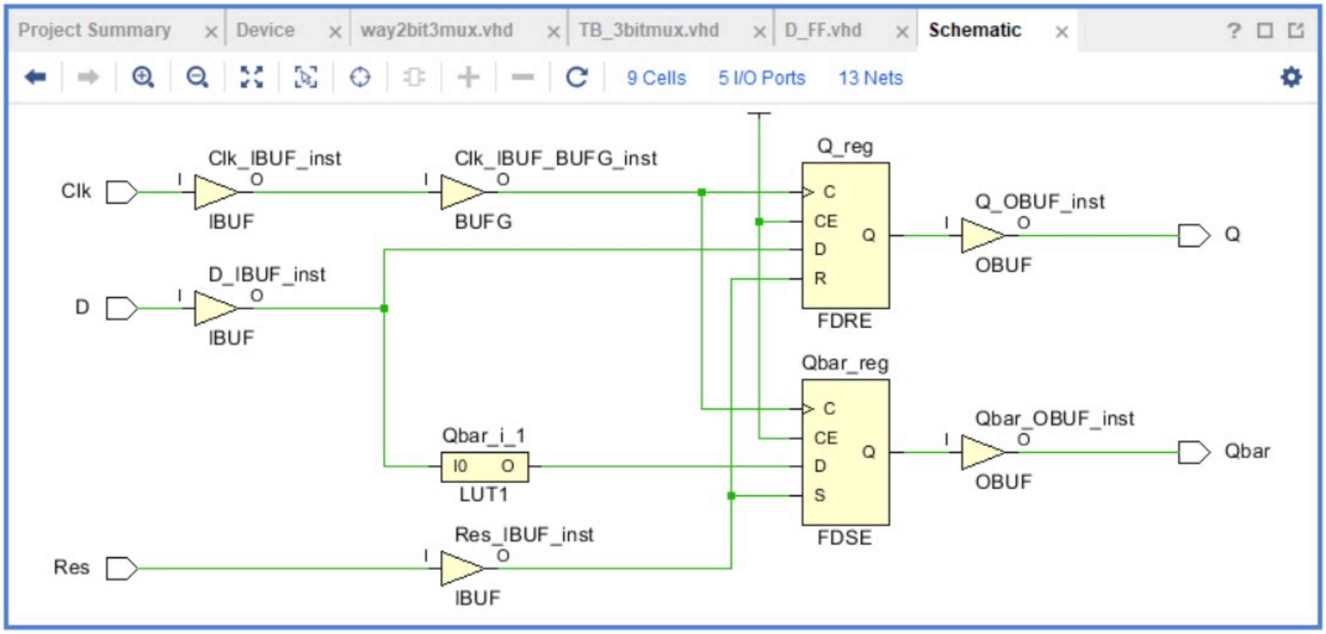
## Code

```
entity D_FF is
    Port ( D : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
end D_FF;

architecture Behavioral of D_FF is

begin
process (Clk)
    begin
        if (rising_edge (Clk)) then
            if Res ='1' then
                Q <= '0';
                Qbar <= '1';
            else
                Q <=D;
                Qbar <= not D;
            end if;
        end if;
end process;
end Behavioral;
```

## RTL



## Synthesized Design

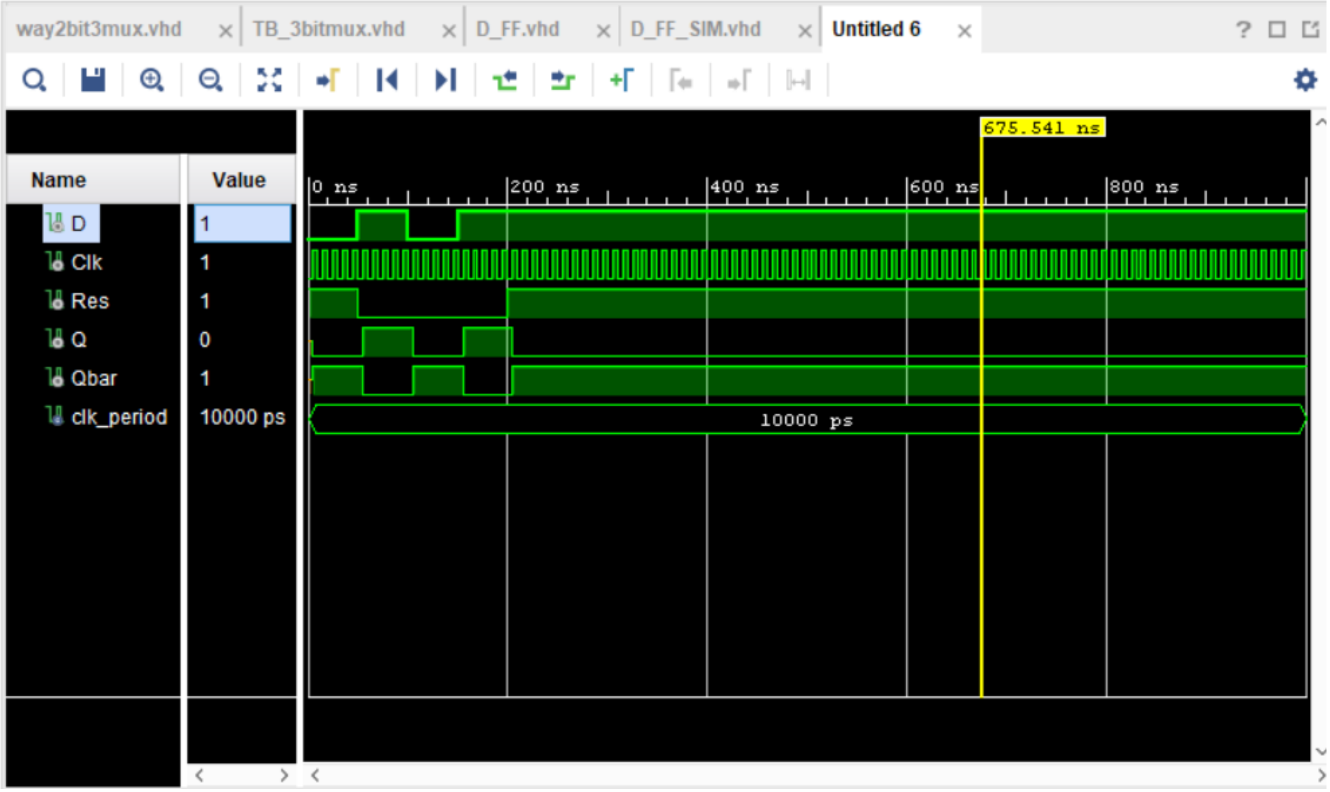


**Simulation_Code**

```
entity D_FF_Sim is
--  Port ( );
end D_FF_Sim;

architecture Behavioral of D_FF_Sim is

component D_FF
Port ( D : in STD_LOGIC;
          Res : in STD_LOGIC;
          Clk : in STD_LOGIC;
          Q : out STD_LOGIC;
          Qbar : out STD_LOGIC);
end component;

signal D : std_logic := '0';
signal Clk : std_logic := '0';
signal Res : std_logic := '1';
signal Q,Qbar : std_logic;

constant clk_period : time := 10 ns;

begin
uut: D_FF port map(
   D => D,
   Res => Res,
   Clk => Clk,
   Q => Q,
   Qbar => Qbar);
clk_process :process
begin
 Clk <='0';
 wait for clk_period/2;
 Clk <='1';
 wait for clk_period/2;
 end process;
stim_proc : process
  begin
   Res <='1';
   wait for 50 ns;
   Res <='0';
   D <='1';
   wait for 50 ns;
   Res <='0';
   D <='0';
   wait for 50 ns;
   Res <='0';
   D <='1';
```

```
    wait for 50 ns;
    Res <='1';
    D <='1';
    wait;
  end process;

end Behavioral;
```

### Behavioral Simulation



| Name | Value |
|---|---|
| D | 1 |
| Clk | 1 |
| Res | 1 |
| Q | 0 |
| Qbar | 1 |
| clk_period | 10000 ps |

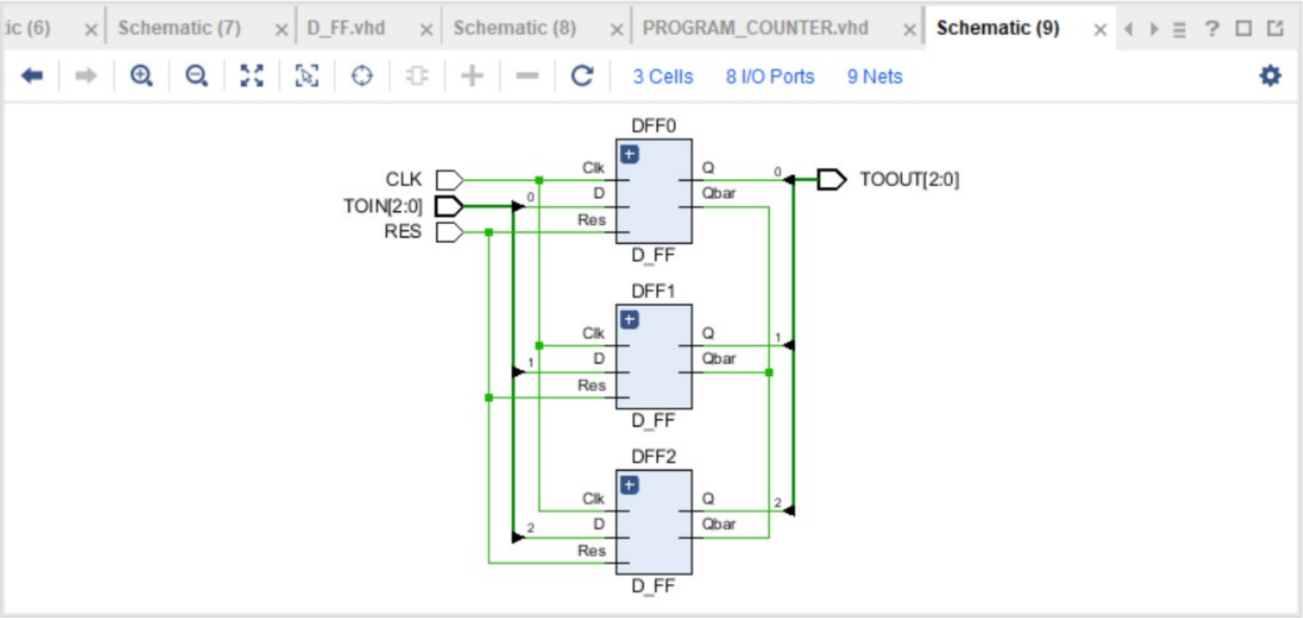
# Program Counter

### Code

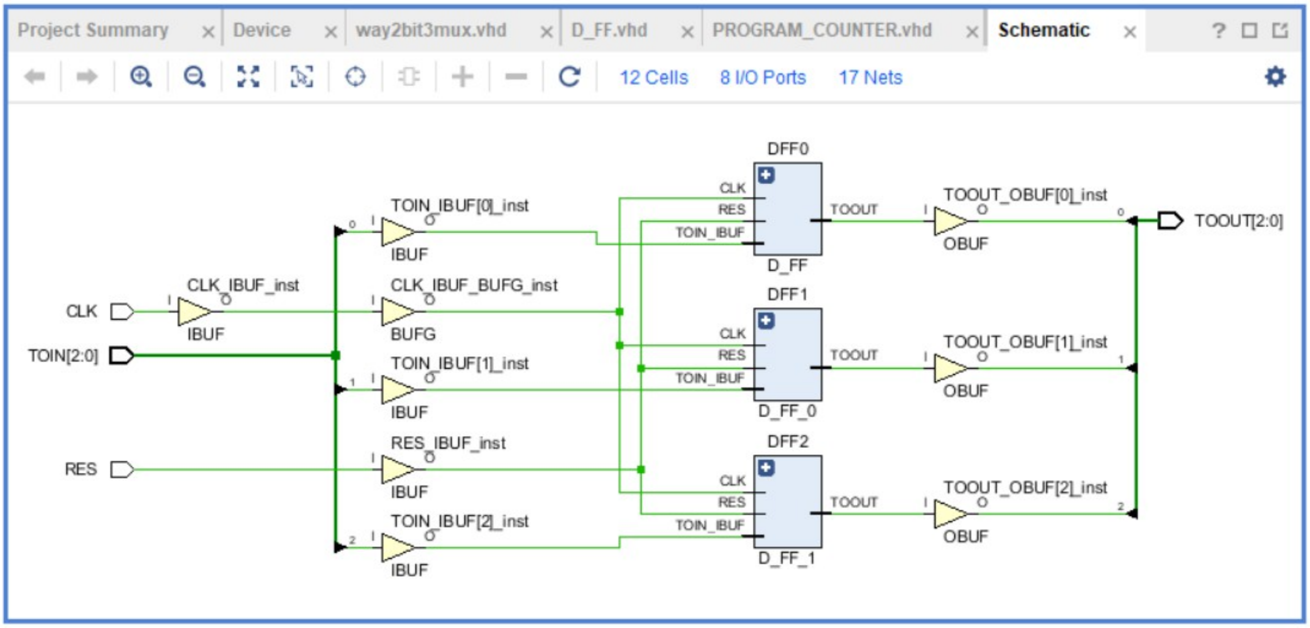
```
entity PROGRAM_COUNTER is
    Port ( TOIN : in STD_LOGIC_VECTOR (2 downto 0);
           TOOUT : out STD_LOGIC_VECTOR (2 downto 0);
           CLK : in STD_LOGIC;
           RES : in STD_LOGIC);
end PROGRAM_COUNTER;

architecture Behavioral of PROGRAM_COUNTER is


component D_FF
```

```
    Port (  D : in STD_LOGIC;
          Res : in STD_LOGIC;
          Clk : in STD_LOGIC;
          Q : out STD_LOGIC;
          Qbar : out STD_LOGIC);
end component;

signal ha0_s :std_logic;
begin

DFF0 : D_FF PORT MAP (
         D => toin(0),
         Res =>res,
         Clk =>clk,
         Q =>toout(0),
         Qbar =>ha0_s);
DFF1 : D_FF PORT MAP (
         D => toin(1),
         Res =>res,
         Clk =>clk,
         Q =>toout(1),
         Qbar =>ha0_s);
DFF2 : D_FF PORT MAP (
         D => toin(2),
         Res =>res,
         Clk =>clk,
         Q =>toout(2),
         Qbar =>ha0_s);


end Behavioral;
```

**RTL**



## Synthesized Design

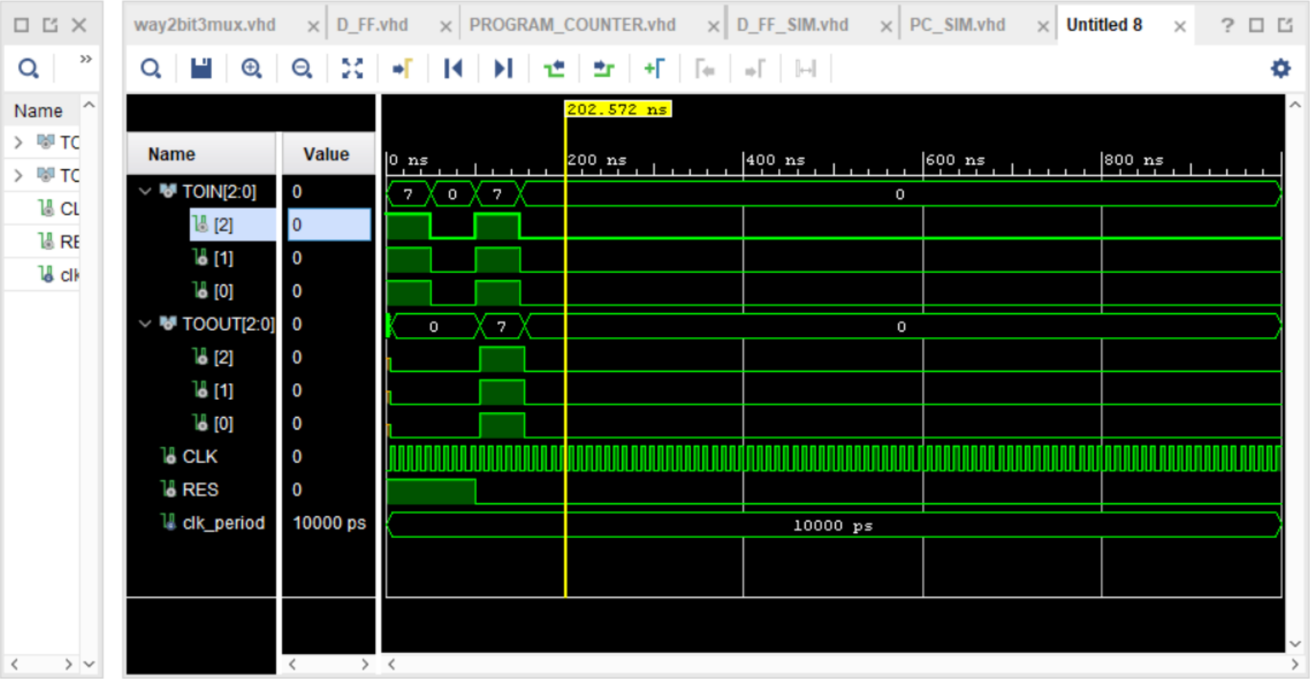


## Simulation_Code

```
entity PC_SIM is
--  Port ( );
end PC_SIM ;

architecture Behavioral of PC_SIM is
```

```
component PROGRAM_COUNTER
Port ( TOIN : in STD_LOGIC_VECTOR (2 downto 0);
          TOOUT : out STD_LOGIC_VECTOR (2 downto 0);
          CLK : in STD_LOGIC;
          RES : in STD_LOGIC);
end component;

signal TOIN : STD_LOGIC_VECTOR (2 downto 0);
signal TOOUT : STD_LOGIC_VECTOR (2 downto 0);
signal CLK : STD_LOGIC;
signal RES :STD_LOGIC;

constant clk_period : time := 10 ns;

begin
uut: PROGRAM_COUNTER port map(
   TOOUT => TOOUT,
   TOIN => TOIN,
   CLK=>CLK,
   RES=>RES);
clk_process :process
begin
 Clk <='0';
 wait for clk_period/2;
 Clk <='1';
 wait for clk_period/2;
 end process;
stim_proc : process
   begin
    Res <='1';
    TOIN <="111";
    wait for 50 ns;
    Res <='1';
    TOIN <="000";
    wait for 50 ns;
    Res <='0';
    TOIN <="111";
    wait for 50 ns;
    Res <='0';
    TOIN <="000";
    wait;
  end process;

end Behavioral;
```

## Behavioral Simulation



| Name | Value |
|---|---|
| TOIN[2:0] | 0 |
| [2] | 0 |
| [1] | 0 |
| [0] | 0 |
| TOOUT[2:0] | 0 |
| [2] | 0 |
| [1] | 0 |
| [0] | 0 |
| CLK | 0 |
| RES | 0 |
| clk_period | 10000 ps |

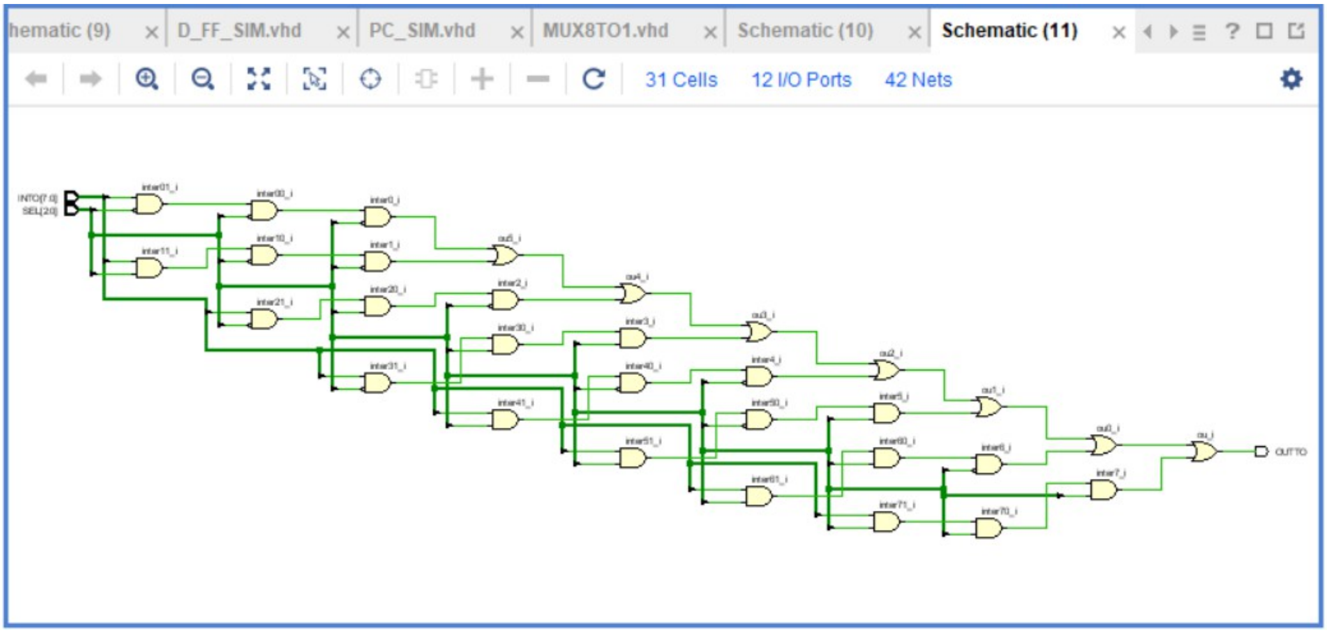
# 8 to 1 MUX

**Code**

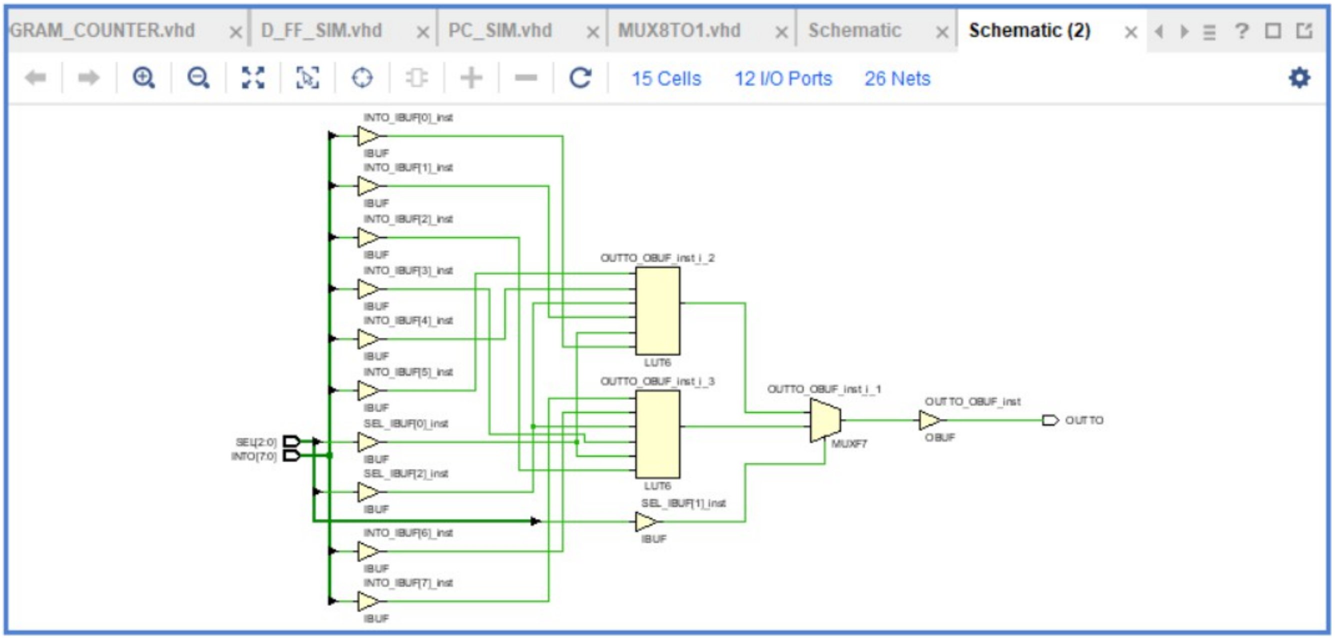
```
entity MUX8TO1 is
    Port ( INTO : in STD_LOGIC_VECTOR (7 downto 0);
           SEL : in STD_LOGIC_VECTOR (2 downto 0);
           OUTTO : out STD_LOGIC);
end MUX8TO1;

architecture Behavioral of MUX8TO1 is

signal ou,sel0,sel0b ,sel1,sel1b,sel2,sel2b,inter0,inter1,inter2,inter3,inter4,inter5,inter6,inter7
:std_logic;

begin

SEL0<=sel(0);
sel0b<=not sel(0);
sel1<=sel(1);
sel1b<=not sel(1);
sel2 <= sel(2);
sel2b<=not sel(2);

inter0<=into(0) and sel1b and sel2b and sel0b;
inter1<=into(1) and sel0 and sel1b and sel2b;
inter2<=into(2) and sel2b and sel1 and sel0b;
inter3<=into(3) and sel2b and sel1 and sel0;
```

```
inter4<=into(4) and sel2 and sel1b and sel0b;
inter5<=into(5) and sel2 and sel1b and sel0;
inter6<=into(6) and sel2 and sel1 and sel0b;
inter7<=into(7) and sel2 and sel1 and sel0;

ou<=inter0 or inter1 or inter2 or inter3 or inter4 or inter5 or inter6 or inter7;
outto<=ou;
end Behavioral;
```

## RTL



## Synthesized Design

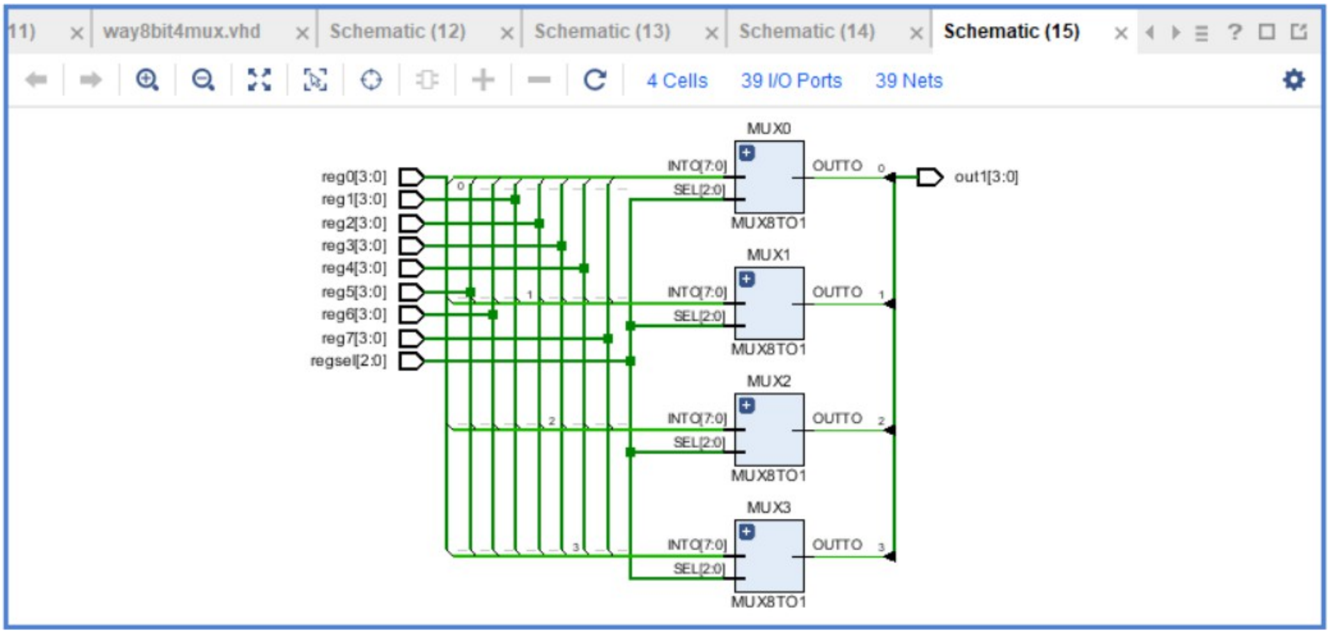


# 8 WAY 4 BIT MUX

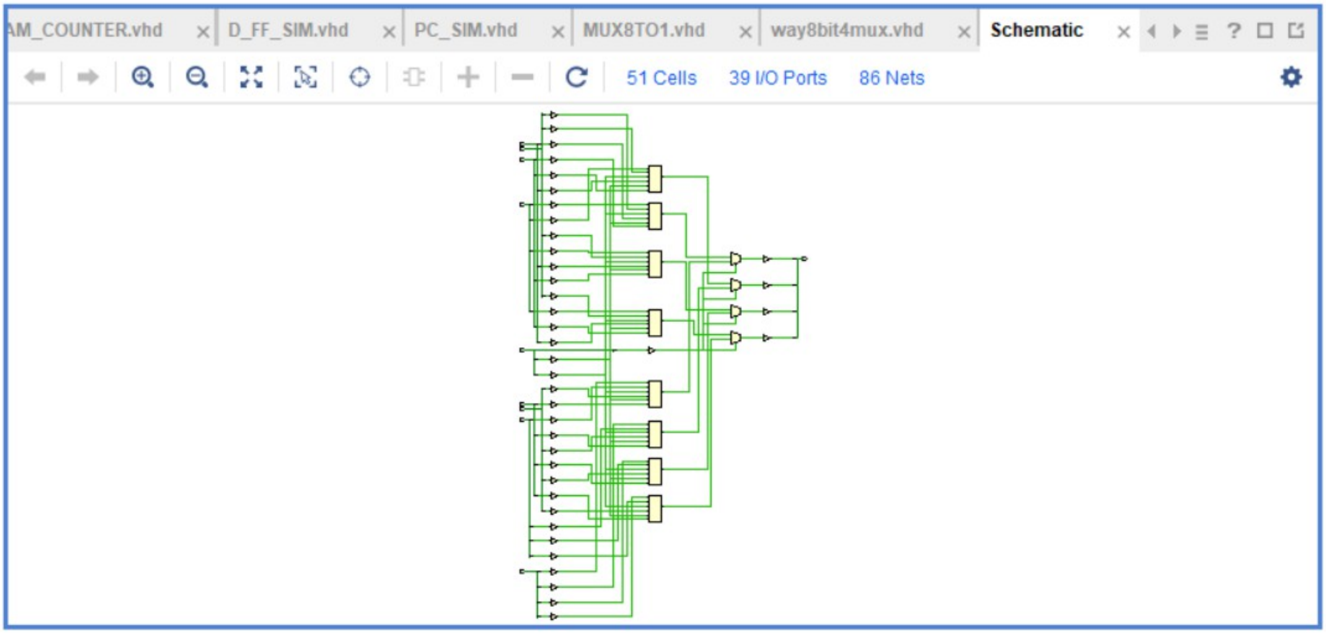
## Code

```
entity way8bit4mux is
   Port ( regsel : in STD_LOGIC_VECTOR (2 downto 0);
          reg0 : in STD_LOGIC_VECTOR (3 downto 0);
          reg1 : in STD_LOGIC_VECTOR (3 downto 0);
          reg2 : in STD_LOGIC_VECTOR (3 downto 0);
          reg3 : in STD_LOGIC_VECTOR (3 downto 0);
          reg4 : in STD_LOGIC_VECTOR (3 downto 0);
          reg5 : in STD_LOGIC_VECTOR (3 downto 0);
          reg6 : in STD_LOGIC_VECTOR (3 downto 0);
          reg7 : in STD_LOGIC_VECTOR (3 downto 0);
          out1 : out STD_LOGIC_VECTOR (3 downto 0));
end way8bit4mux;

architecture Behavioral of way8bit4mux is

component mux8to1
   Port ( INTO : in STD_LOGIC_VECTOR (7 downto 0);
          SEL : in STD_LOGIC_VECTOR (2 downto 0);
          OUTTO : out STD_LOGIC);
end component;

signal ha0_s, ha0_c, hal_c :std_logic;
begin

MUX0 : mux8to1 PORT MAP (
   sel => regsel,
   into(0)=> reg0(0),
   into(1)=> reg1(0),
   into(2)=> reg2(0),
   into(3)=> reg3(0),
   into(4)=> reg4(0),
   into(5)=> reg5(0),
   into(6)=> reg6(0),
   into(7)=> reg7(0),
   outto=>out1(0));

MUX1 : mux8to1 PORT MAP (
   sel => regsel,
   into(0)=> reg0(1),
   into(1)=> reg1(1),
   into(2)=> reg2(1),
   into(3)=> reg3(1),
   into(4)=> reg4(1),
   into(5)=> reg5(1),
   into(6)=> reg6(1),
   into(7)=> reg7(1),
   outto=>out1(1));

MUX2 : mux8to1 PORT MAP (
   sel => regsel,
   into(0)=> reg0(2),
   into(1)=> reg1(2),
   into(2)=> reg2(2),
   into(3)=> reg3(2),
   into(4)=> reg4(2),
   into(5)=> reg5(2),
   into(6)=> reg6(2),
   into(7)=> reg7(2),
   outto=>out1(2));

MUX3 : mux8to1 PORT MAP (
   sel => regsel,
   into(0)=> reg0(3),
   into(1)=> reg1(3),
   into(2)=> reg2(3),
   into(3)=> reg3(3),
   into(4)=> reg4(3),
   into(5)=> reg5(3),
   into(6)=> reg6(3),
   into(7)=> reg7(3),
   outto=>out1(3));
end Behavioral;
```

**RTL**



## Synthesized Design



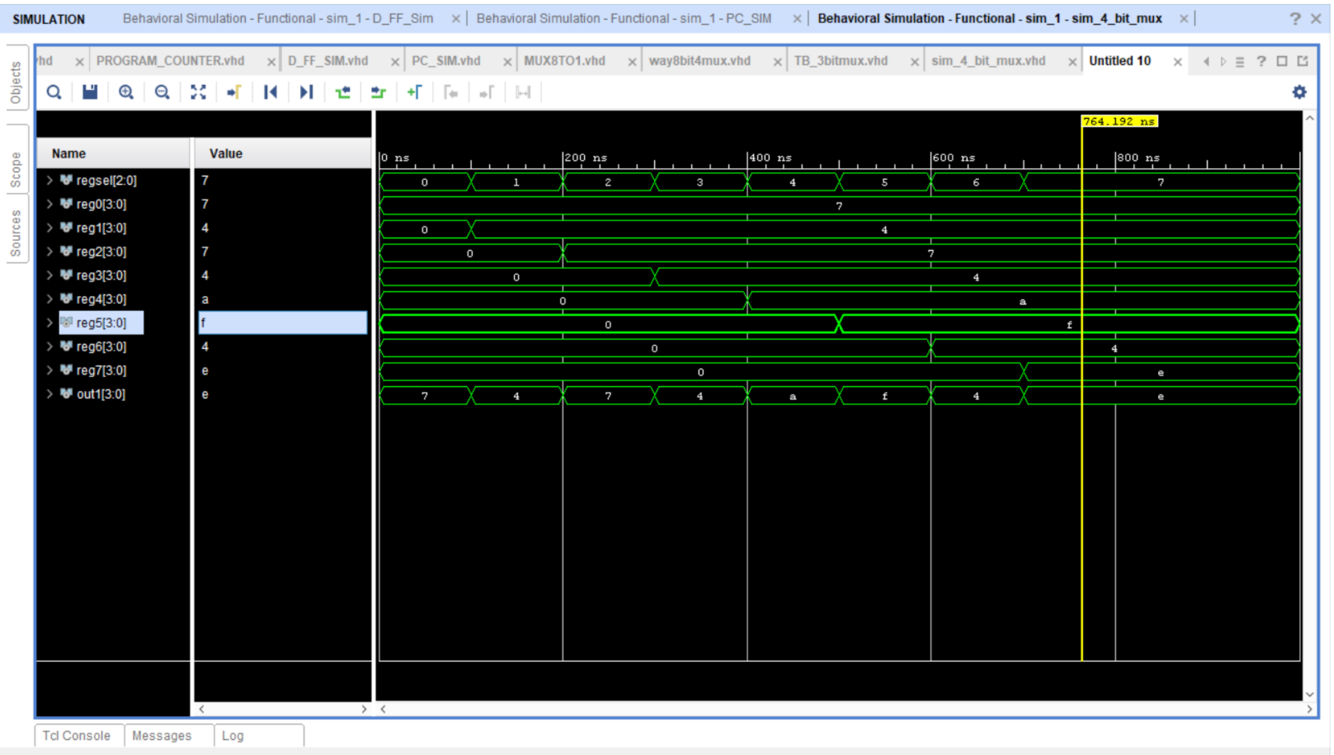
## Simulation_Code

```
entity sim_4_bit_mux is
--  Port ( );
end sim_4_bit_mux;

architecture Behavioral of sim_4_bit_mux is
```

```
 component way8bit4mux
   Port (   regsel : in STD_LOGIC_VECTOR (2 downto 0);
          reg0 : in STD_LOGIC_VECTOR (3 downto 0);
          reg1 : in STD_LOGIC_VECTOR (3 downto 0);
          reg2 : in STD_LOGIC_VECTOR (3 downto 0);
          reg3 : in STD_LOGIC_VECTOR (3 downto 0);
          reg4 : in STD_LOGIC_VECTOR (3 downto 0);
          reg5 : in STD_LOGIC_VECTOR (3 downto 0);
          reg6 : in STD_LOGIC_VECTOR (3 downto 0);
          reg7 : in STD_LOGIC_VECTOR (3 downto 0);
          out1 : out STD_LOGIC_VECTOR (3 downto 0));

end component;

          signal regsel :  STD_LOGIC_VECTOR (2 downto 0);
          signal reg0 :  STD_LOGIC_VECTOR (3 downto 0);
          signal reg1 :  STD_LOGIC_VECTOR (3 downto 0);
          signal reg2 :  STD_LOGIC_VECTOR (3 downto 0);
          signal reg3 :  STD_LOGIC_VECTOR (3 downto 0);
          signal reg4 :  STD_LOGIC_VECTOR (3 downto 0);
          signal reg5 :  STD_LOGIC_VECTOR (3 downto 0);
          signal reg6 :  STD_LOGIC_VECTOR (3 downto 0);
          signal reg7 :  STD_LOGIC_VECTOR (3 downto 0);
          signal out1 :  STD_LOGIC_VECTOR (3 downto 0);


begin

uut : way8bit4mux port map(
          regsel =>regsel,
          reg0 =>reg0,
          reg1 =>reg1,
          reg2 =>reg2,
          reg3 =>reg3,
          reg4 =>reg4,
          reg5 =>reg5,
          reg6 =>reg6,
          reg7 =>reg7,
          out1 =>out1);

stiM_PROC: PROCESS
BEGIN

reg0 <= "0000";
reg1 <= "0000";
reg2 <= "0000";
reg3 <= "0000";
reg4 <= "0000";
reg5 <= "0000";
```

```
reg6 <= "0000";
reg7 <= "0000";


regsel <= "000";
reg0 <= "0111";
wait for 100 ns;

regsel <= "001";
reg1 <= "0100";
wait for 100 ns;

regsel <= "010";
reg2 <= "0111";
wait for 100 ns;

regsel <= "011";
reg3 <= "0100";
wait for 100 ns;

regsel <= "100";
reg4 <= "1010";
wait for 100 ns;

regsel <= "101";
reg5 <= "1111";
wait for 100 ns;

regsel <= "110";
reg6 <= "0100";
wait for 100 ns;

regsel <= "111";
reg7 <= "1110";
wait;

end process;
end;
```

## Behavioral Simulation



# Program ROM

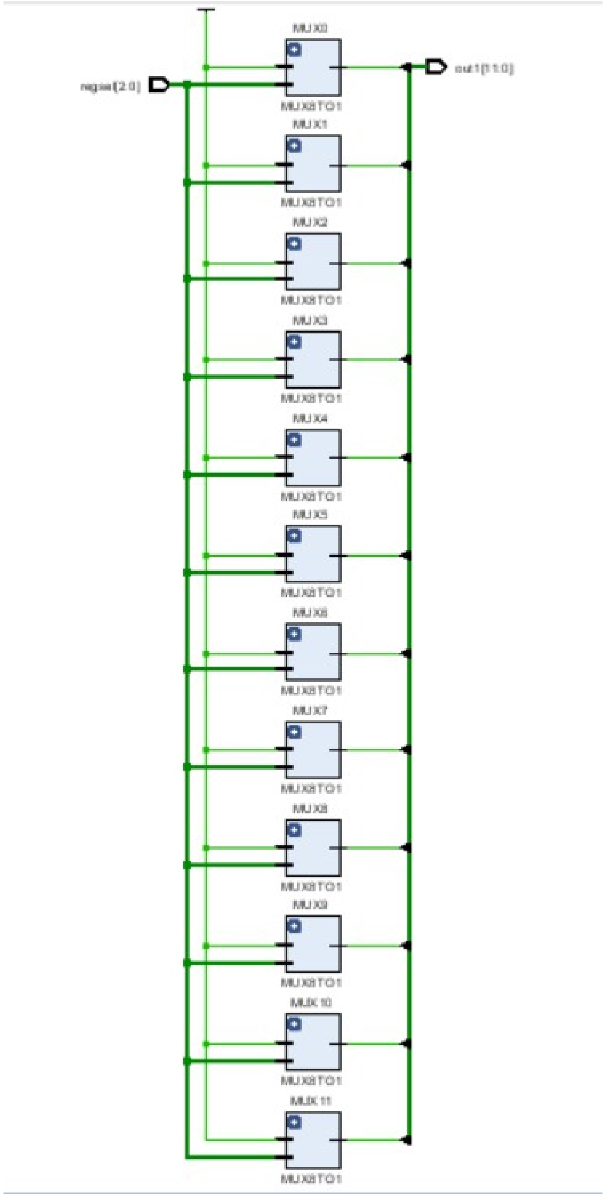
## Code

```
entity program_rom is
   Port ( regsel : in STD_LOGIC_VECTOR (2 downto 0);
          out1 : out STD_LOGIC_VECTOR (11 downto 0));
end program_rom;

architecture Behavioral of program_rom is

component mux8to1
   Port ( INTO : in STD_LOGIC_VECTOR (7 downto 0);
          SEL : in STD_LOGIC_VECTOR (2 downto 0);
          OUTTO : out STD_LOGIC);
end component;

type rom_type is array (0 to 7) of std_logic_vector (11 downto 0);
signal ROM : rom_type :=(
             "111111111111",----gfedcba---0
             "111111111111",---1
             "111111111111",---2
             "111111111111",---3
             "111111111111",--4
             "111111111111",---5
             "111111111111",---6
             "111111111111"---7
```

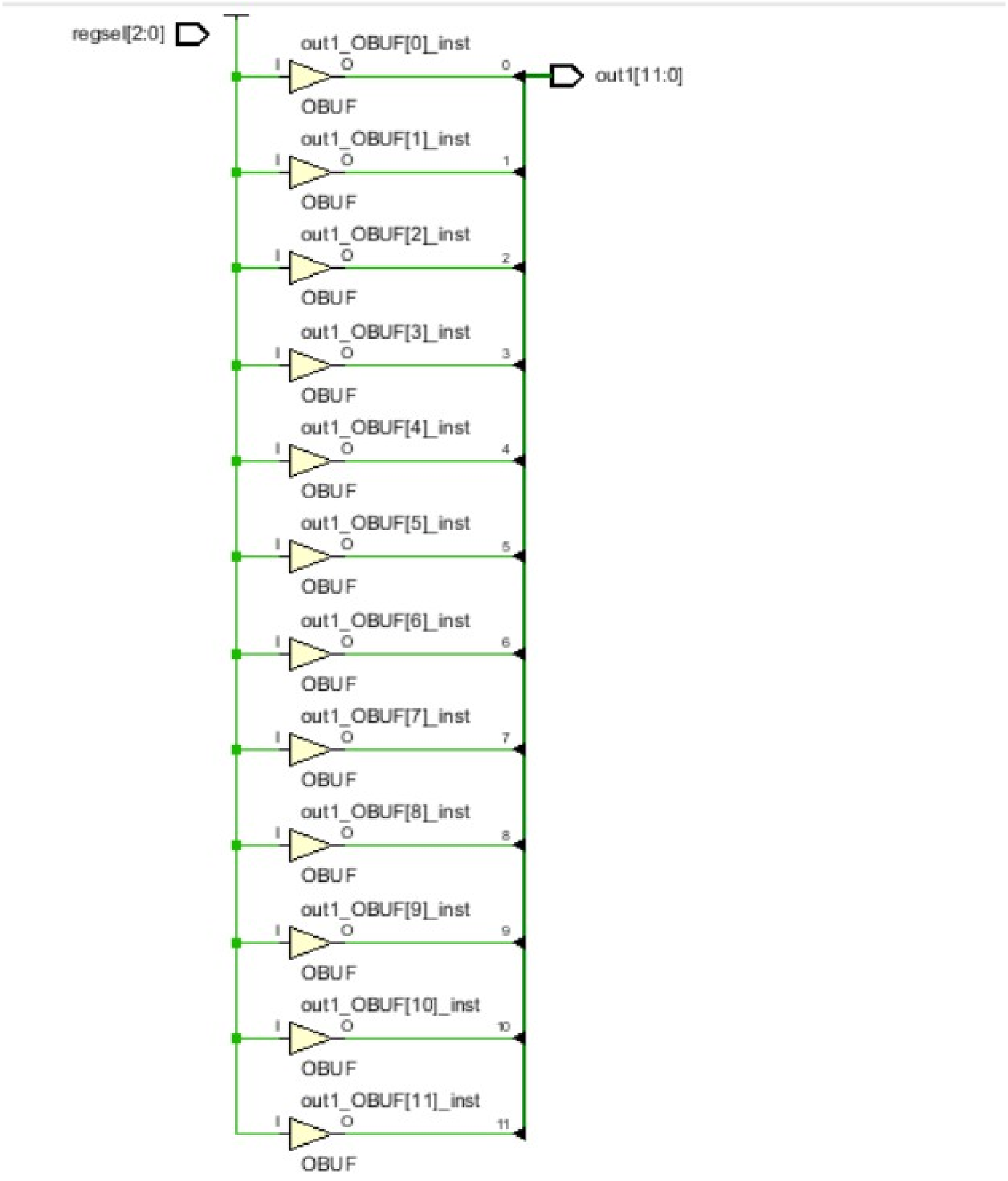
```
            );

begin

MUX0 : mux8to1 PORT MAP (
    sel => regsel,
    into(0)=> rom(0)(0),
    into(1)=> rom(1)(0),
    into(2)=> rom(2)(0),
    into(3)=> rom(3)(0),
    into(4)=> rom(4)(0),
    into(5)=> rom(5)(0),
    into(6)=> rom(6)(0),
    into(7)=> rom(7)(0),
    outto=>out1(0));

MUX1 : mux8to1 PORT MAP (
    sel => regsel,
    into(0)=> rom(0)(1),
    into(1)=> rom(1)(1),
    into(2)=> rom(2)(1),
    into(3)=> rom(3)(1),
    into(4)=> rom(4)(1),
    into(5)=> rom(5)(1),
    into(6)=> rom(6)(1),
    into(7)=> rom(7)(1),
    outto=>out1(1));
MUX2 : mux8to1 PORT MAP (
    sel => regsel,
    into(0)=> rom(0)(2),
    into(1)=> rom(1)(2),
    into(2)=> rom(2)(2),
    into(3)=> rom(3)(2),
    into(4)=> rom(4)(2),
    into(5)=> rom(5)(2),
    into(6)=> rom(6)(2),
    into(7)=> rom(7)(2),
    outto=>out1(2));
MUX3 : mux8to1 PORT MAP (
        sel => regsel,
        into(0)=> rom(0)(3),
        into(1)=> rom(1)(3),
        into(2)=> rom(2)(3),
        into(3)=> rom(3)(3),
        into(4)=> rom(4)(3),
        into(5)=> rom(5)(3),
        into(6)=> rom(6)(3),
        into(7)=> rom(7)(3),
        outto=>out1(3));
```

```
MUX4 : mux8to1 PORT MAP (
        sel => regsel,
        into(0)=> rom(0)(4),
        into(1)=> rom(1)(4),
        into(2)=> rom(2)(4),
        into(3)=> rom(3)(4),
        into(4)=> rom(4)(4),
        into(5)=> rom(5)(4),
        into(6)=> rom(6)(4),
        into(7)=> rom(7)(4),
        outto=>out1(4));
MUX5 : mux8to1 PORT MAP (
        sel => regsel,
        into(0)=> rom(0)(5),
        into(1)=> rom(1)(5),
        into(2)=> rom(2)(5),
        into(3)=> rom(3)(5),
        into(4)=> rom(4)(5),
        into(5)=> rom(5)(5),
        into(6)=> rom(6)(5),
        into(7)=> rom(7)(5),
        outto=>out1(5));
MUX6 : mux8to1 PORT MAP (
        sel => regsel,
        into(0)=> rom(0)(6),
        into(1)=> rom(1)(6),
        into(2)=> rom(2)(6),
        into(3)=> rom(3)(6),
        into(4)=> rom(4)(6),
        into(5)=> rom(5)(6),
        into(6)=> rom(6)(6),
        into(7)=> rom(7)(6),
        outto=>out1(6));
MUX7 : mux8to1 PORT MAP (
        sel => regsel,
        into(0)=> rom(0)(7),
        into(1)=> rom(1)(7),
        into(2)=> rom(2)(7),
        into(3)=> rom(3)(7),
        into(4)=> rom(4)(7),
        into(5)=> rom(5)(7),
        into(6)=> rom(6)(7),
        into(7)=> rom(7)(7),
        outto=>out1(7));
MUX8 : mux8to1 PORT MAP (
            sel => regsel,
            into(0)=> rom(0)(8),
            into(1)=> rom(1)(8),
            into(2)=> rom(2)(8),
            into(3)=> rom(3)(8),
            into(4)=> rom(4)(8),
            into(5)=> rom(5)(8),
            into(6)=> rom(6)(8),
            into(7)=> rom(7)(8),
            outto=>out1(8));
MUX9 : mux8to1 PORT MAP (
            sel => regsel,
            into(0)=> rom(0)(9),
            into(1)=> rom(1)(9),
            into(2)=> rom(2)(9),
            into(3)=> rom(3)(9),
            into(4)=> rom(4)(9),
            into(5)=> rom(5)(9),
            into(6)=> rom(6)(9),
            into(7)=> rom(7)(9),
            outto=>out1(9));
MUX10 : mux8to1 PORT MAP (
            sel => regsel,
            into(0)=> rom(0)(10),
            into(1)=> rom(1)(10),
            into(2)=> rom(2)(10),
            into(3)=> rom(3)(10),
            into(4)=> rom(4)(10),
            into(5)=> rom(5)(10),
            into(6)=> rom(6)(10),
            into(7)=> rom(7)(10),
            outto=>out1(10));
MUX11 : mux8to1 PORT MAP (
            sel => regsel,
            into(0)=> rom(0)(11),
            into(1)=> rom(1)(11),
            into(2)=> rom(2)(11),
            into(3)=> rom(3)(11),
            into(4)=> rom(4)(11),
            into(5)=> rom(5)(11),
            into(6)=> rom(6)(11),
            into(7)=> rom(7)(11),
            outto=>out1(11));
end Behavioral;
```

**RTL**



## **Synthesized Design**

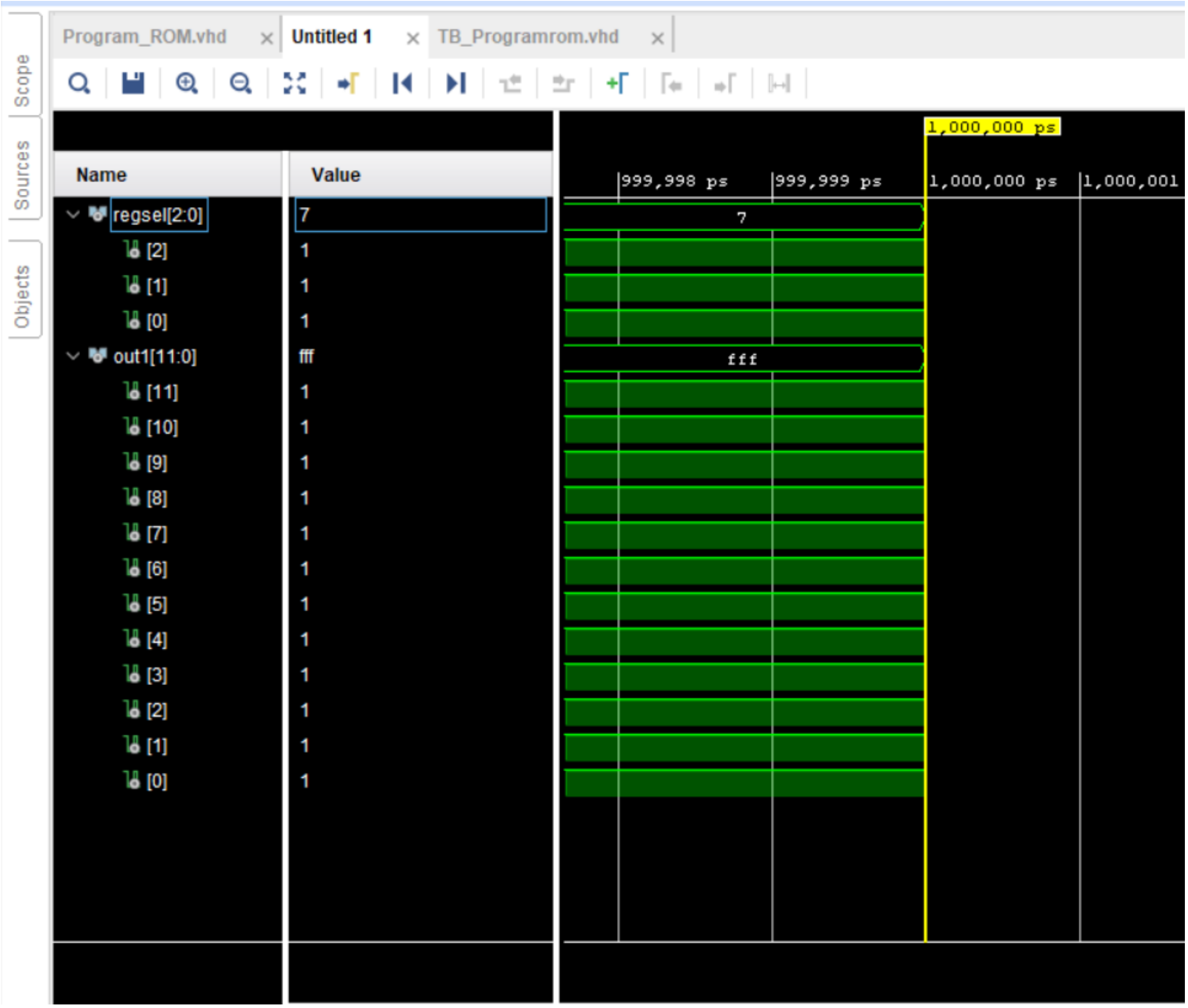


## **Simulation_Code**

```
entity TB_Programrom is
--  Port ( );
end TB_Programrom;

architecture Behavioral of TB_Programrom is
 component program_rom
   Port (   regsel : in STD_LOGIC_VECTOR (2 downto 0);
            out1  : out STD_LOGIC_VECTOR (11 downto 0));

end component;

            signal regsel :  STD_LOGIC_VECTOR (2 downto 0);
            signal out1 :  STD_LOGIC_VECTOR (11   downto 0);
```

```
begin

uut : program_rom port map(
          regsel =>regsel,
          out1 =>out1);

stiM_PROC: PROCESS
BEGIN


regsel <= "110";
wait for 100 ns;

regsel <= "111";
wait;

end process;
end;
```

**Behavioral Simulation**



| Name | Value |
|---|---|
| regsel[2:0] | 7 |
| [2] | 1 |
| [1] | 1 |
| [0] | 1 |
| out1[11:0] | fff |
| [11] | 1 |
| [10] | 1 |
| [9] | 1 |
| [8] | 1 |
| [7] | 1 |
| [6] | 1 |
| [5] | 1 |
| [4] | 1 |
| [3] | 1 |
| [2] | 1 |
| [1] | 1 |
| [0] | 1 |

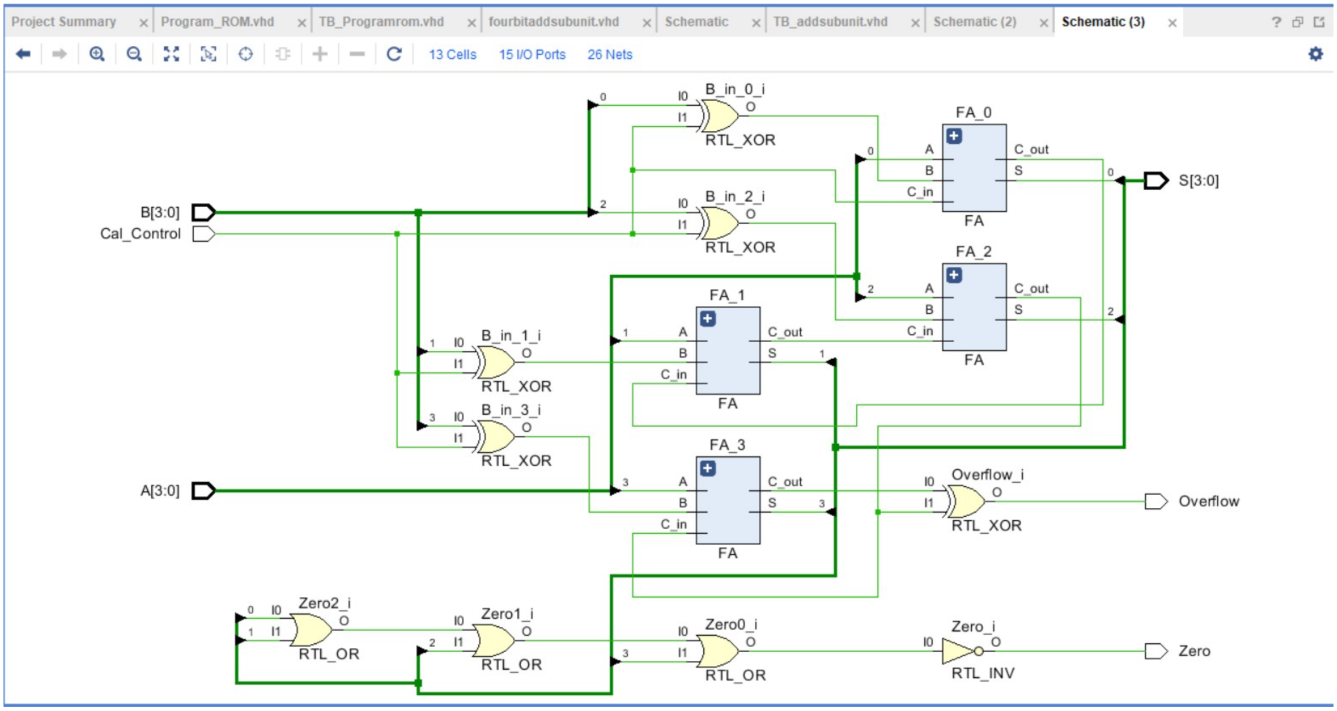
# 4-Bit_Add/Sub_Unit

**Code**

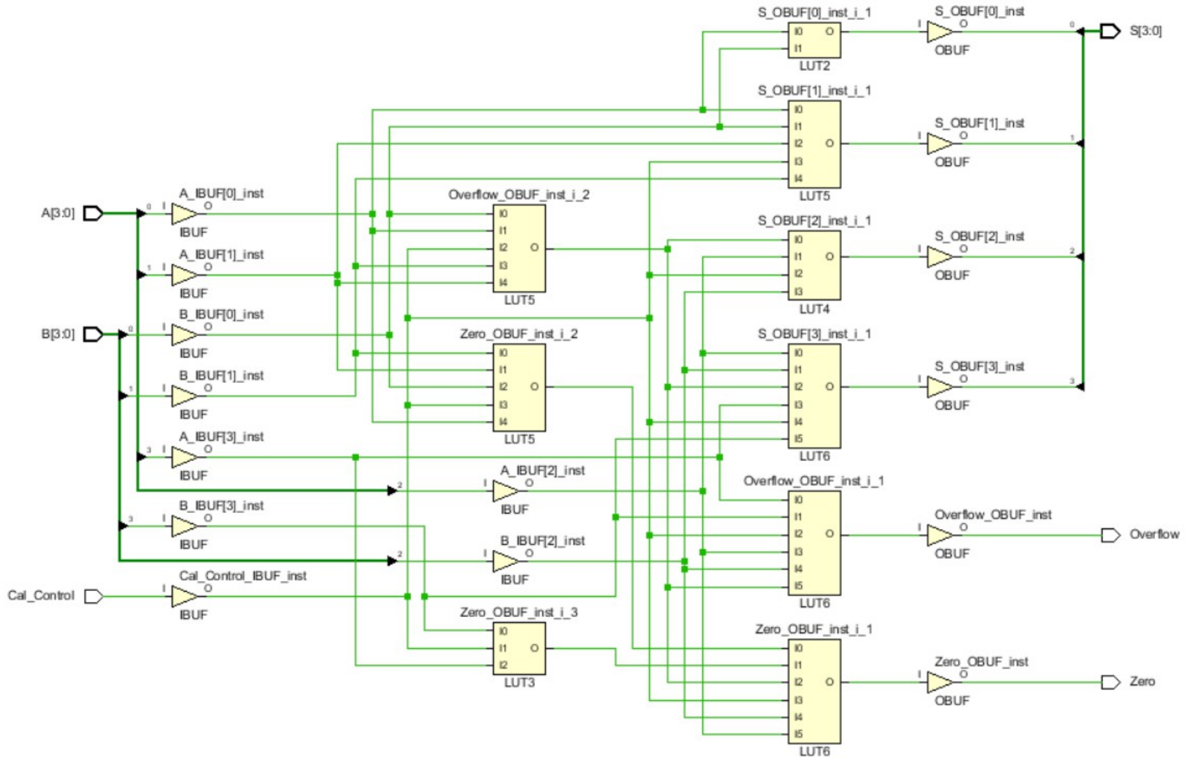
```
entity fourbitaddsubunit is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          Cal_Control : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          S : out STD_LOGIC_VECTOR (3 downto 0));
end fourbitaddsubunit;

architecture Behavioral of fourbitaddsubunit is

component FA
   port (
        A: in std_logic;
        B: in std_logic;
        C_in: in std_logic;
        S: out std_logic;
        C_out: out std_logic);
end component;

SIGNAL B_in, S_out : std_logic_vector(3 downto 0);
SIGNAL FA0_C, FA1_C,FA2_C,FA3_C : std_logic;

begin

B_in(0) <= B(0) XOR Cal_Control;
B_in(1) <= B(1) XOR Cal_Control;
B_in(2) <= B(2) XOR Cal_Control;
B_in(3) <= B(3) XOR Cal_Control;

FA_0 : FA
port map (
   A => A(0),
   B => B_in(0),
   C_in => Cal_Control,
   S => S_out(0),
   C_Out => FA0_C);

FA_1 : FA
port map (
   A => A(1),
   B => B_in(1),
   C_in => FA0_C,
   S => S_out(1),
   C_Out => FA1_C);

FA_2 : FA
port map (
   A => A(2),
   B => B_in(2),
   C_in => FA1_C,
   S => S_out(2),
   C_Out => FA2_C);

FA_3 : FA
port map (
   A => A(3),
   B => B_in(3),
   C_in => FA2_C,
   S => S_out(3),
   C_Out => FA3_C);
```

```
Overflow <= (FA3_C XOR FA2_C);

S(0) <= S_out(0);
S(1) <= S_out(1);
S(2) <= S_out(2);
S(3) <= S_out(3);

Zero <= not (S_out(0) or S_out(1) or S_out(2) or S_out(3));

end Behavioral;
```

## RTL



## Synthesized Design



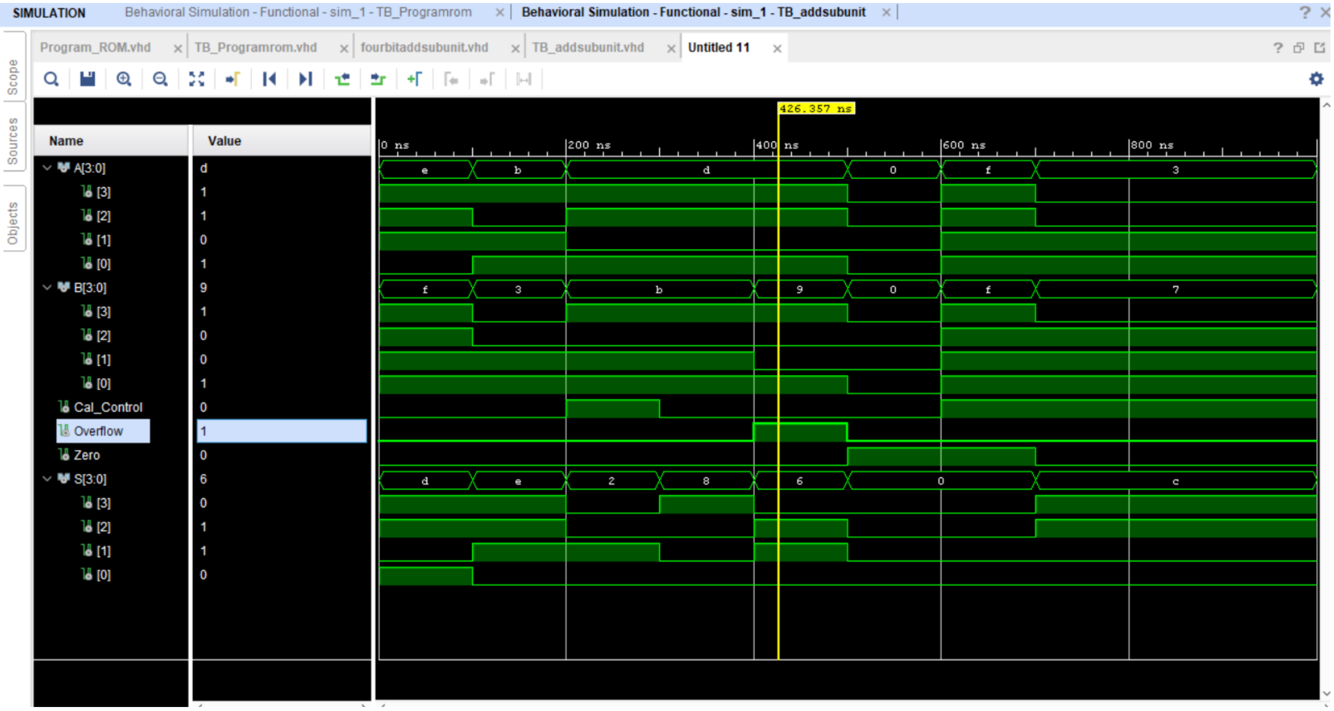
**Simulation_Code**

```
entity TB_addsubunit is
--  Port ( );
end TB_addsubunit;

architecture Behavioral of TB_addsubunit is
component fourbitaddsubunit is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          Cal_Control : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          S : out STD_LOGIC_VECTOR (3 downto 0));
end component;

          signal A : STD_LOGIC_VECTOR (3 downto 0);
          signal B : STD_LOGIC_VECTOR (3 downto 0);
          signal Cal_Control : STD_LOGIC;
          signal Overflow : STD_LOGIC;
          signal Zero : STD_LOGIC;
          signal S : STD_LOGIC_VECTOR (3 downto 0);


begin
```

```
uut : fourbitaddsubunit port map(
        A =>A,
        B=>B,
        Cal_Control => Cal_Control,
        Overflow => Overflow,
        zero => zero,
        s=>s);

stiM_PROC: PROCESS
BEGIN


A <= "1110";
B <= "1111";
Cal_control <='0';
wait for 100 ns;

A <= "1011";
B <= "0011";
Cal_control <='0';
wait for 100 ns;

A <= "1101";
B <= "1011";
Cal_control <='1';
wait for 100 ns;
A <= "1101";
B <= "1011";
Cal_control <='0';
wait for 100 ns;
A <= "1101";
B <= "1001";
Cal_control <='0';
wait for 100 ns;

A <= "0000";
B <= "0000";
Cal_control <='0';
wait for 100 ns;
A <= "1111";
B <= "1111";
Cal_control <='1';
wait for 100 ns;
A <= "0011";
B <= "0111";
Cal_control <='1';
wait;

end process;
```

```
end;
```

### Behavioral Simulation



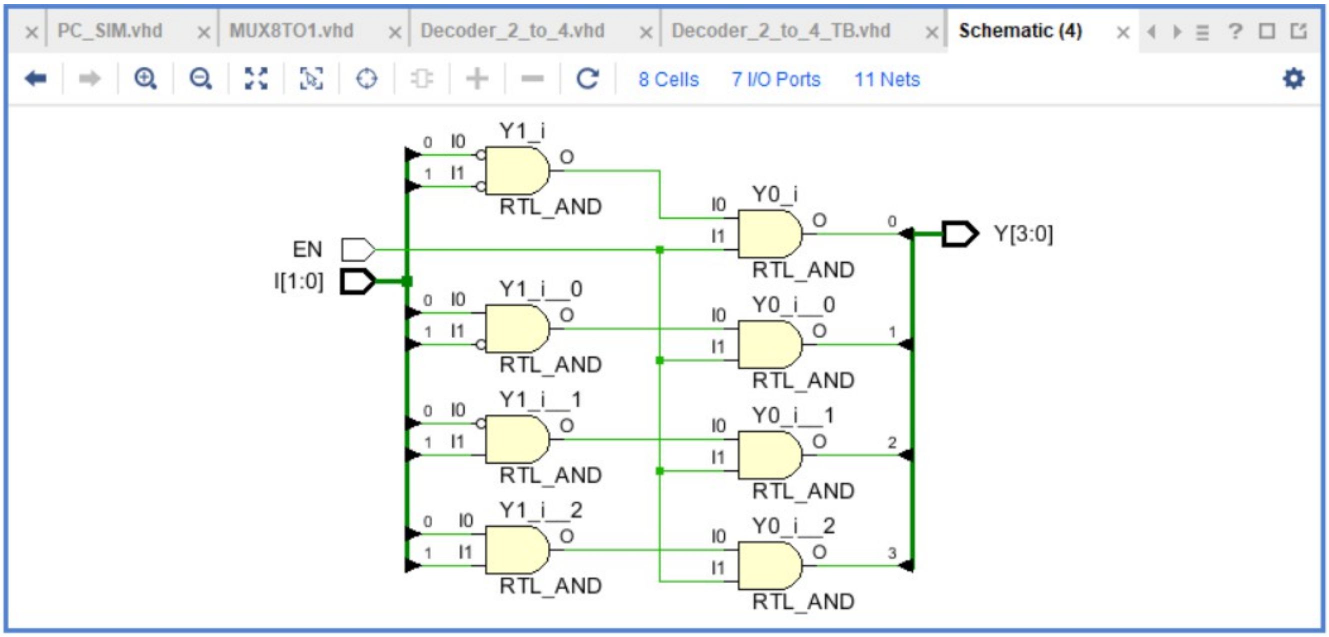
| Name | Value |
|---|---|
| A[3:0] | d |
| [3] | 1 |
| [2] | 1 |
| [1] | 0 |
| [0] | 1 |
| B[3:0] | 9 |
| [3] | 1 |
| [2] | 0 |
| [1] | 0 |
| [0] | 1 |
| Cal_Control | 0 |
| Overflow | 1 |
| Zero | 0 |
| S[3:0] | 6 |
| [3] | 0 |
| [2] | 1 |
| [1] | 1 |
| [0] | 0 |

# Decoder 2 to 4

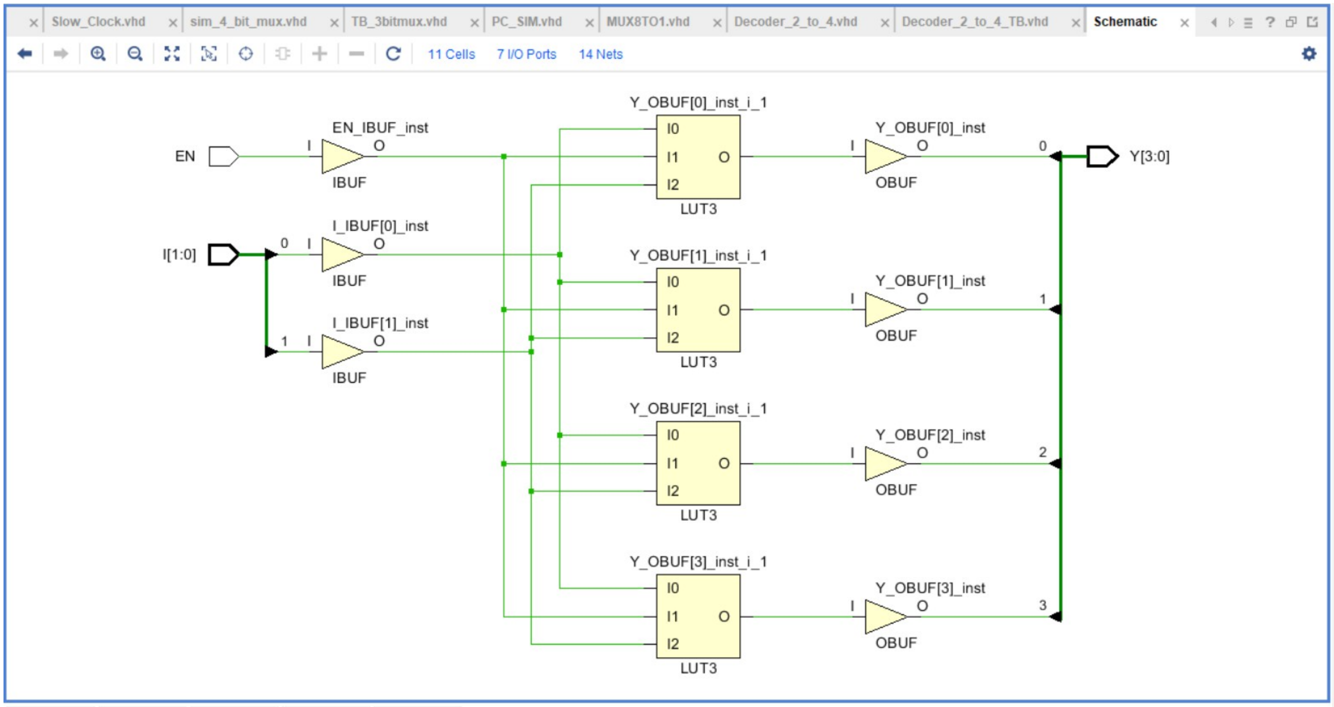
### Code

```
entity Decoder_2_to_4 is
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           Y : out STD_LOGIC_VECTOR (3 downto 0);
           EN : in STD_LOGIC);
end Decoder_2_to_4;

architecture Behavioral of Decoder_2_to_4 is
signal I0_n , I1_n : std_logic;
begin

I0_n <= not(I(0));
I1_n <= not(I(1));

Y(0) <= (I0_n and I1_n and EN);
Y(1) <= (I(0) and I1_n and EN);
Y(2) <= (I0_n and I(1) and EN);
Y(3) <= (I(0) and I(1) and EN);

end Behavioral;
```

## RTL



## Synthesized Design

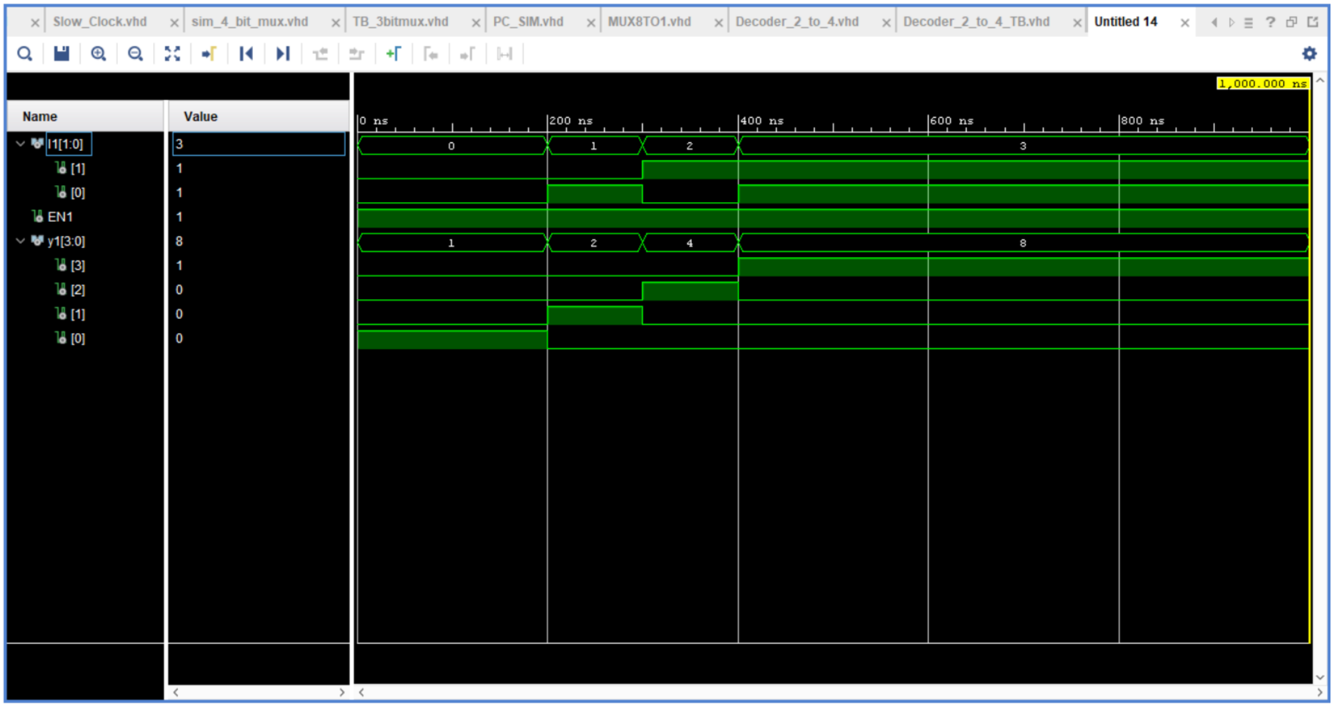


## Simulation_Code

```
entity Decoder_2_to_4_TB is
--  Port ( );
end Decoder_2_to_4_TB;
```

```
architecture Behavioral of Decoder_2_to_4_TB is

component Decoder_2_to_4
   port ( I : in STD_logic_vector (1 downto 0);
        EN: in std_logic;
        y: out std_logic_vector (3 downto 0));
end component;

signal I1 :STD_logic_vector (1 downto 0):= (others => '0');
signal EN1 :STD_logic:= '0';

signal y1 :STD_logic_vector (3 downto 0);
begin

uut: Decoder_2_to_4 port map(
   I => I1,
   EN => EN1,
   Y => Y1
   );

process
   begin
         EN1 <= '1';

         wait for 100 ns;
         I1 <= "00";
         wait for 100 ns;
         I1 <= "01";
         wait for 100 ns;
         I1 <= "10";
         wait for 100 ns;
         I1 <= "11";
         wait;
         end process;
end Behavioral;
```

**Behavioral Simulation**



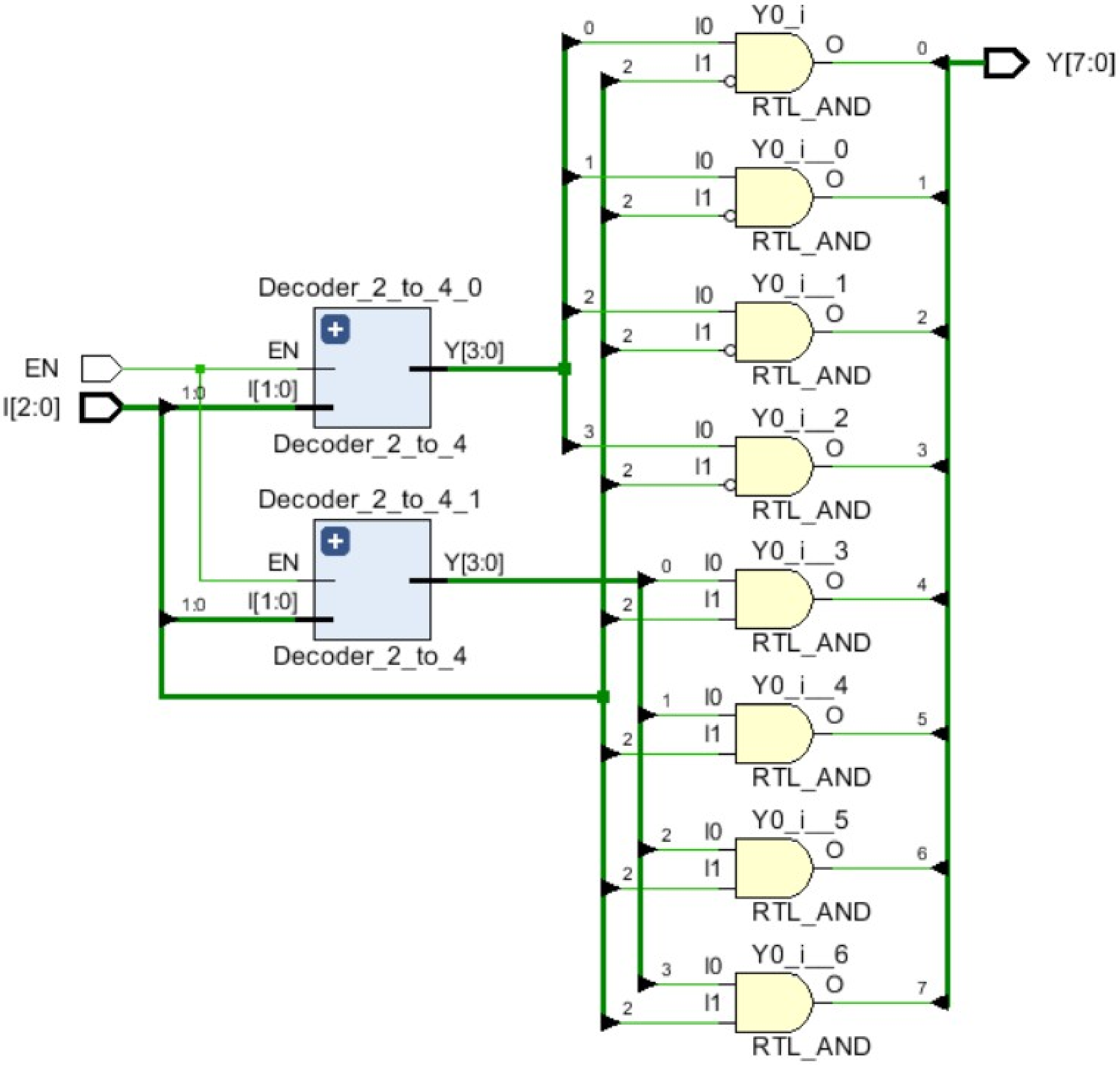
# Decoder 3 to 8

## Code

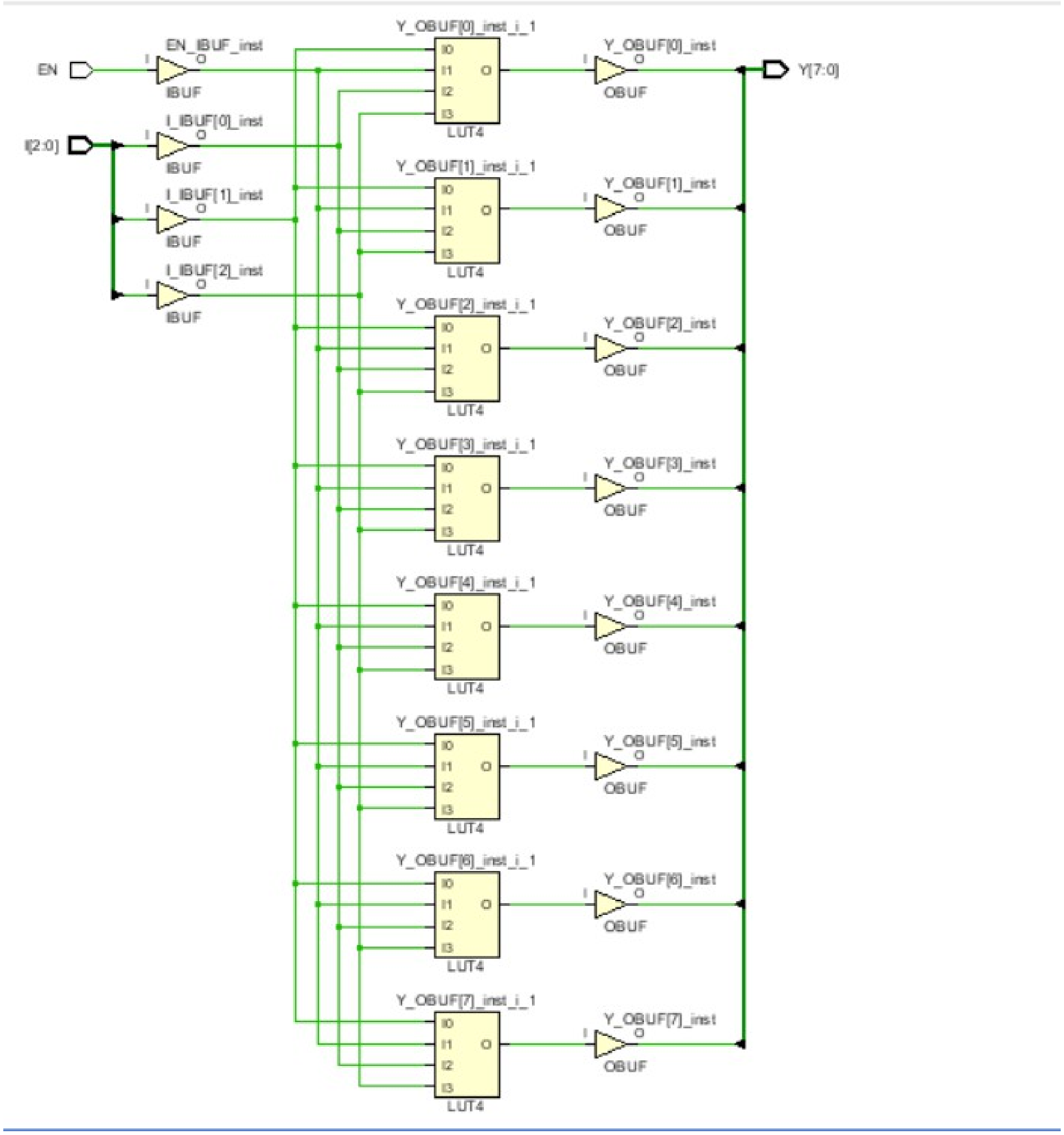
```
entity Decoder_3_to_8 is
    Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (7 downto 0));
end Decoder_3_to_8;

architecture Behavioral of Decoder_3_to_8 is

component Decoder_2_to_4
    Port ( I : in STD_LOGIC_VECTOR (1 downto 0);
           EN : in STD_LOGIC;
           Y : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal Y0, Y1 : std_logic_vector ( 3 downto 0);

begin

Decoder_2_to_4_0:Decoder_2_to_4 port map (
                        I (1 downto 0) => I (1 downto 0),
                        EN => EN,
                        Y (3 downto 0) => Y0 (3 downto 0));
```

```
Decoder_2_to_4_1:Decoder_2_to_4 port map (
                    I (1 downto 0) => I (1 downto 0),
                    EN => EN,
                    Y (3 downto 0) => Y1 (3 downto 0));

Y(0) <= Y0(0) and (not I(2));
Y(1) <= Y0(1) and (not I(2));
Y(2) <= Y0(2) and (not I(2));
Y(3) <= Y0(3) and (not I(2));

Y(4) <= Y1(0) and (I(2));
Y(5) <= Y1(1) and (I(2));
Y(6) <= Y1(2) and (I(2));
Y(7) <= Y1(3) and (I(2));


end Behavioral;
```

**RTL**


EN
I[2:0]
Decoder_2_to_4_0
EN
I[1:0]
Y[3:0]
Decoder_2_to_4
1:0
Decoder_2_to_4_1
EN
I[1:0]
Y[3:0]
Decoder_2_to_4
1:0
Y0_i
Y0_i__0
Y0_i__1
Y0_i__2
Y0_i__3
Y0_i__4
Y0_i__5
Y0_i__6
I0
I1
O
RTL_AND
Y[7:0]
0
1
2
3
4
5
6
7

## Synthesized Design

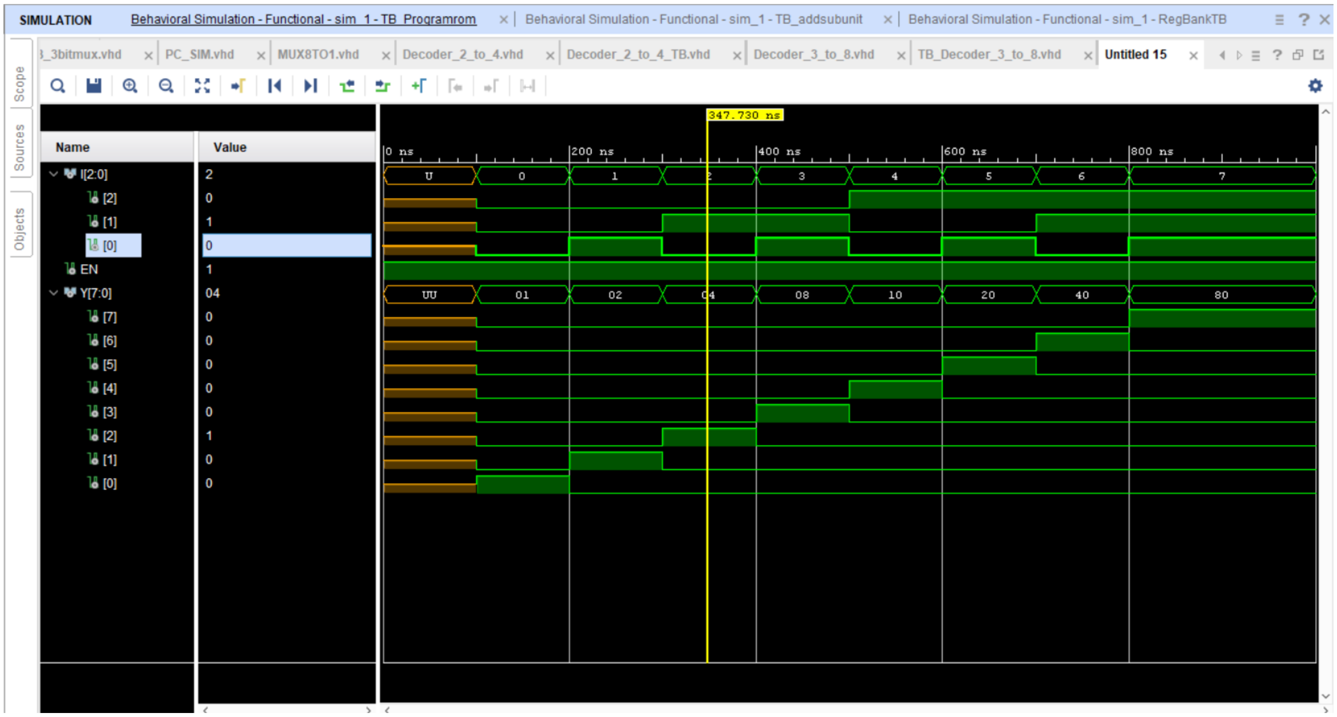


## Simulation_Code

```
entity TB_Decoder_3_to_8 is
--  Port ( );
end TB_Decoder_3_to_8;

architecture Behavioral of TB_Decoder_3_to_8 is

component Decoder_3_to_8
Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
       EN : in STD_LOGIC;
       Y : out STD_LOGIC_VECTOR (7 downto 0));
```

```
end component;

signal I : std_logic_vector (2 downto 0);
signal EN : std_logic;
signal Y : std_logic_vector (7 downto 0);

begin

UUT : Decoder_3_to_8 port map (
            I => I,
            EN => EN,
            Y => Y);

sium : process

    begin EN <= '1';
    wait for 100 ns;
    I <= "000";
    wait for 100 ns;
    I <= "001";
    wait for 100 ns;
    I <= "010";
    wait for 100 ns;
    I <= "011";
    wait for 100 ns;
    I <= "100";
    wait for 100 ns;
    I <= "101";
    wait for 100 ns;
    I <= "110";
    wait for 100 ns;
    I <= "111";
    wait;
    end process;
end Behavioral;
```

**Behavioral Simulation**

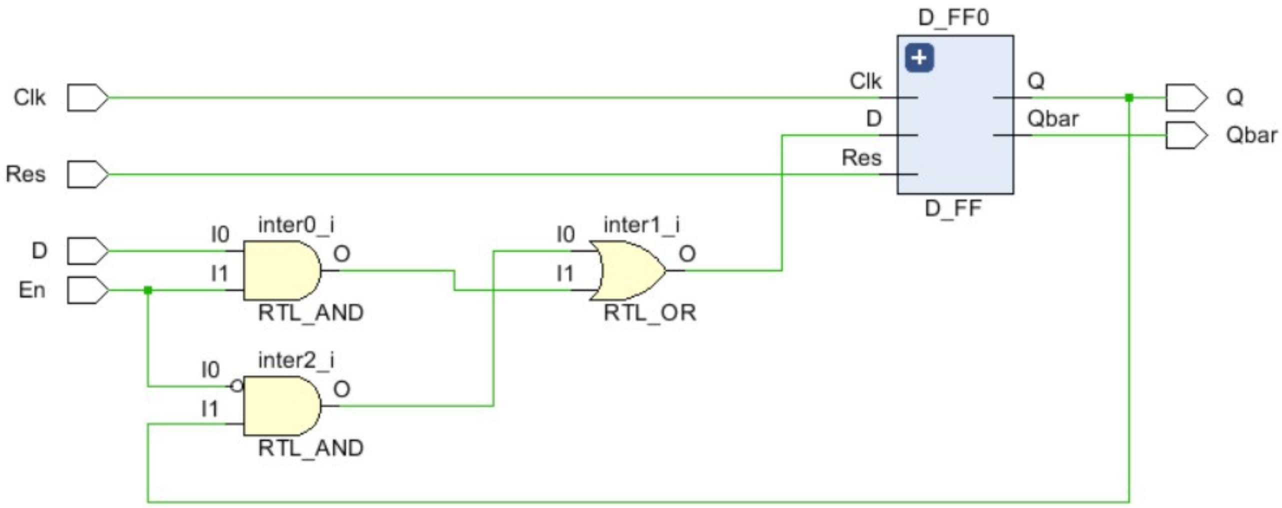


# D_FF_WITH_EN

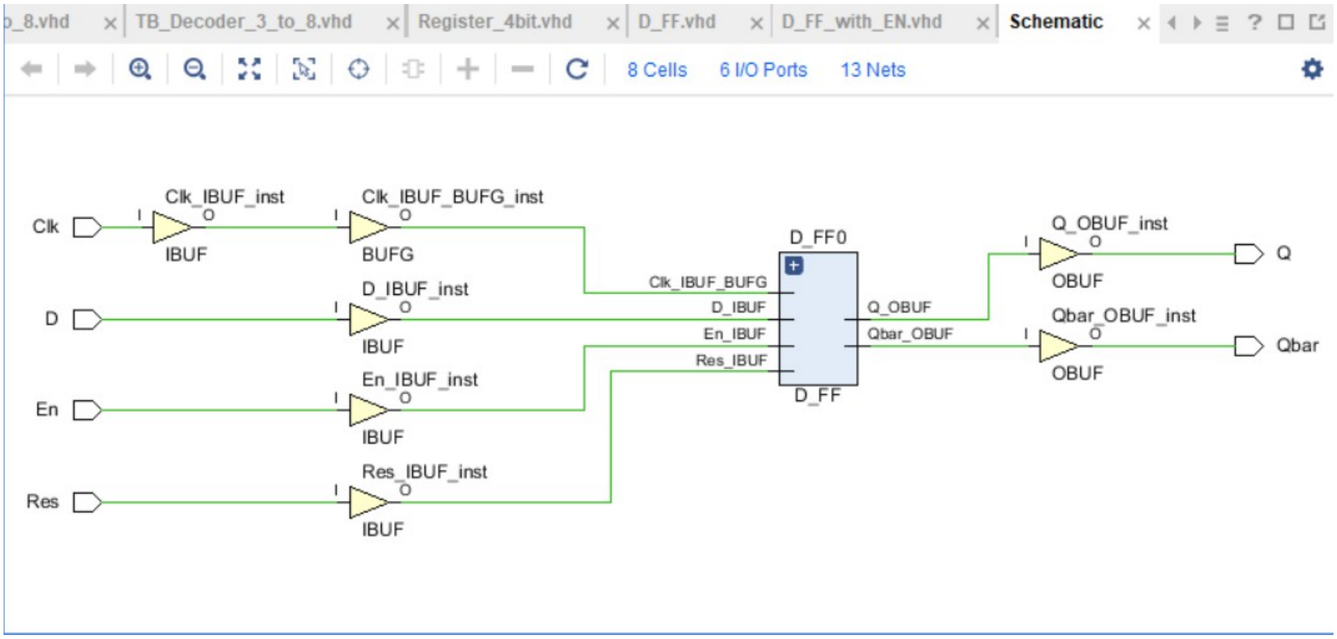
## Code

```
entity D_FF_WITH_EN is
    Port ( D : in STD_LOGIC;
           En : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
end D_FF_WITH_EN;

architecture Behavioral of D_FF_WITH_EN is

Component D_FF Port ( D : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
end component;

signal res0,clk0,d0,en0,not_en0,q0,qbar0,inter0,inter1,inter2:std_logic;

begin

D_FF0 : D_FF Port map
 ( D => inter1,
        Res =>res0,
        Clk =>clk0,
        Q =>q0,
        Qbar =>qbar0);

d0 <= d;
en0 <=en;
res0<=res;
clk0<=clk;

q<=q0;
qbar<=qbar0;

inter0<= d0 and en0;

not_en0<=not en0;
inter2<=not_en0 and q0;

inter1<=inter2 or inter0;

end Behavioral;
```

**RTL**



**Synthesized Design**



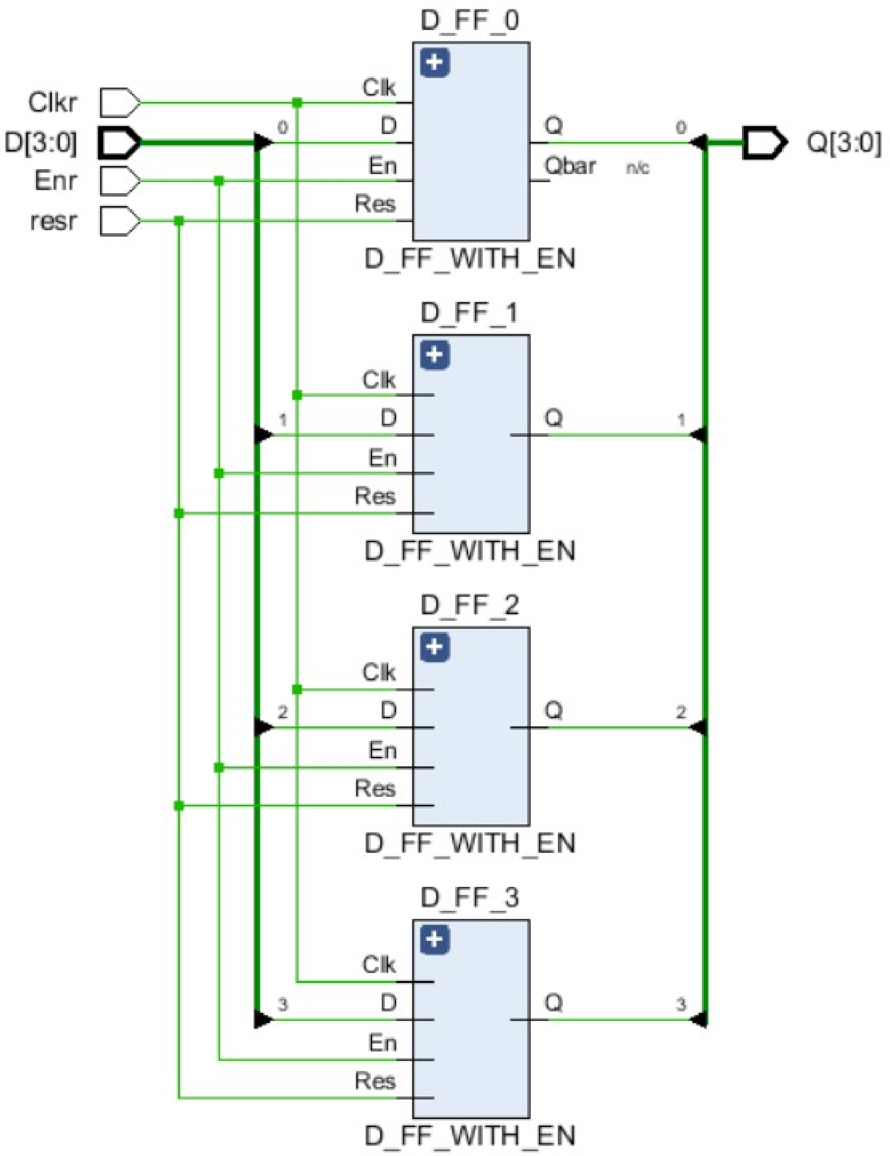
# Register_4bit

**Code**

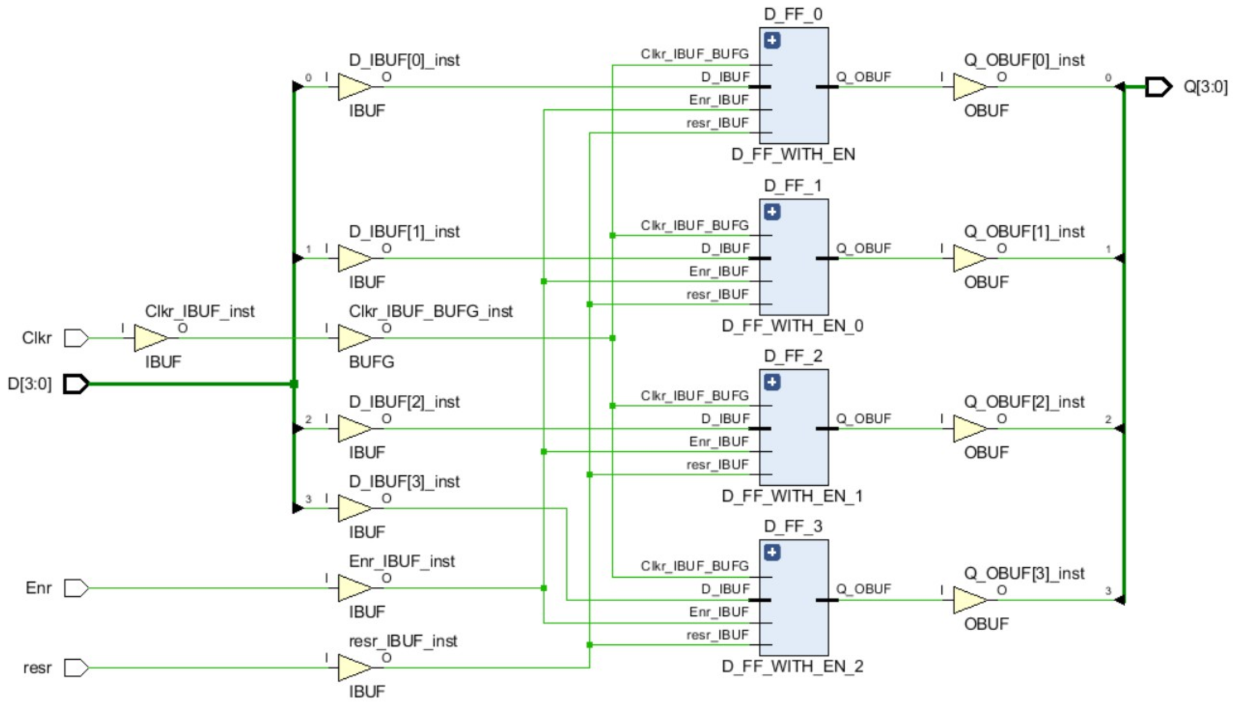
```
entity Register_4bit is

    Port ( RegIn : in STD_LOGIC_VECTOR (3 downto 0);
           RegOut : out STD_LOGIC_VECTOR (3 downto 0);
           Clock : in STD_LOGIC;
           RegEnable : in STD_LOGIC;
           RegReset : in STD_LOGIC);
end Register_4bit;

architecture Behavioral of Register_4bit is

component D_FF_WITH_EN Port ( D : in STD_LOGIC;
           En : in STD_LOGIC;
           Res : in STD_LOGIC;
           Clk : in STD_LOGIC;
           Q : out STD_LOGIC;
           Qbar : out STD_LOGIC);
end component;

signal cout0,cout1,cout2,cout3,cin:std_logic;

begin
```

```
D_FF_0:  D_FF_WITH_EN port map(
   d=> RegIn(0),
   en =>RegEnable,
   res=>RegReset,
   clk=>Clock,
   q=>RegOut(0)
   );
D_FF_1:  D_FF_WITH_EN port map(
   d=> RegIn(1),
   en =>RegEnable,
   res=>RegReset,
   clk=>Clock,
   q=>RegOut(1)
   );
 D_FF_2:  D_FF_WITH_EN port map(
   d=> RegIn(2),
   en =>RegEnable,
   res=>RegReset,
   clk=>Clock,
   q=>RegOut(2)
   );
D_FF_3:  D_FF_WITH_EN port map(
   d=> RegIn(3),
   en =>RegEnable,
   res=>RegReset,
   clk=>Clock,
   q=>RegOut(3)
   );


end Behavioral;
```

## RTL



## Synthesized Design



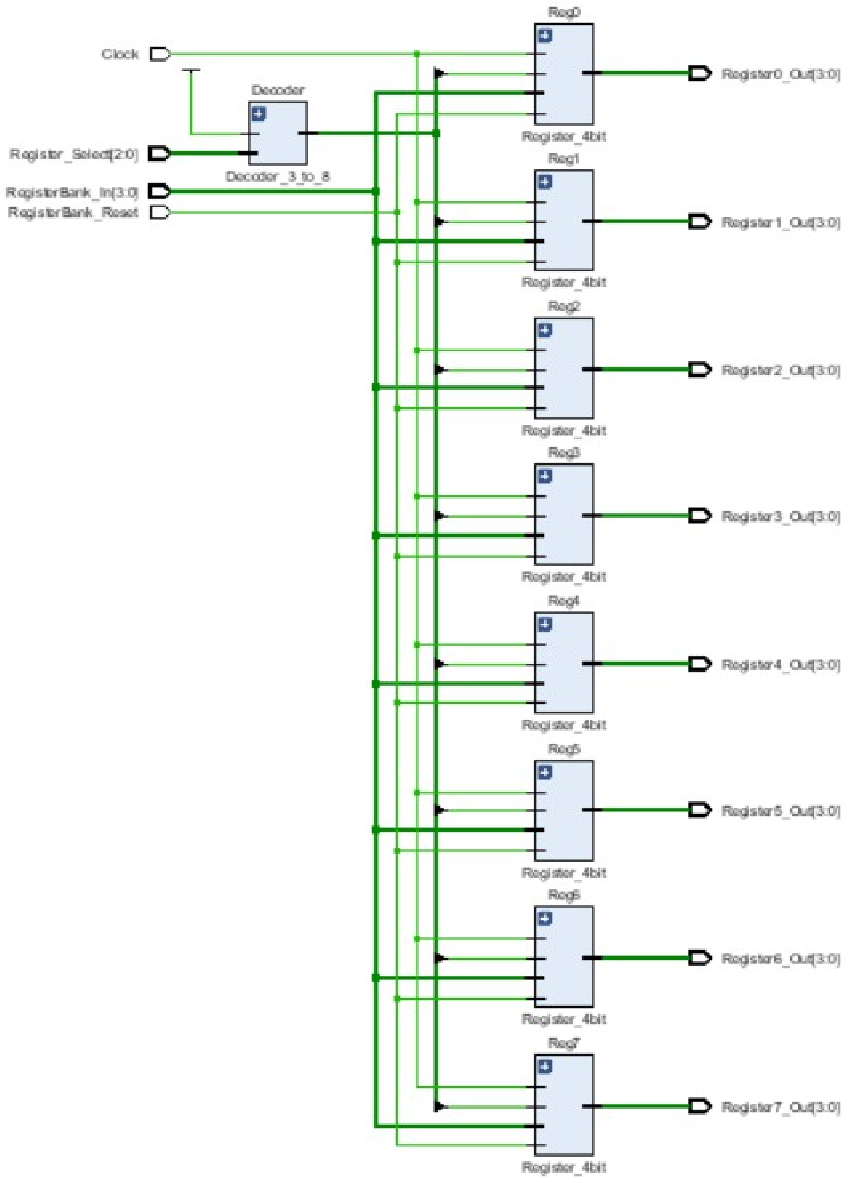
# Register Bank

## Code

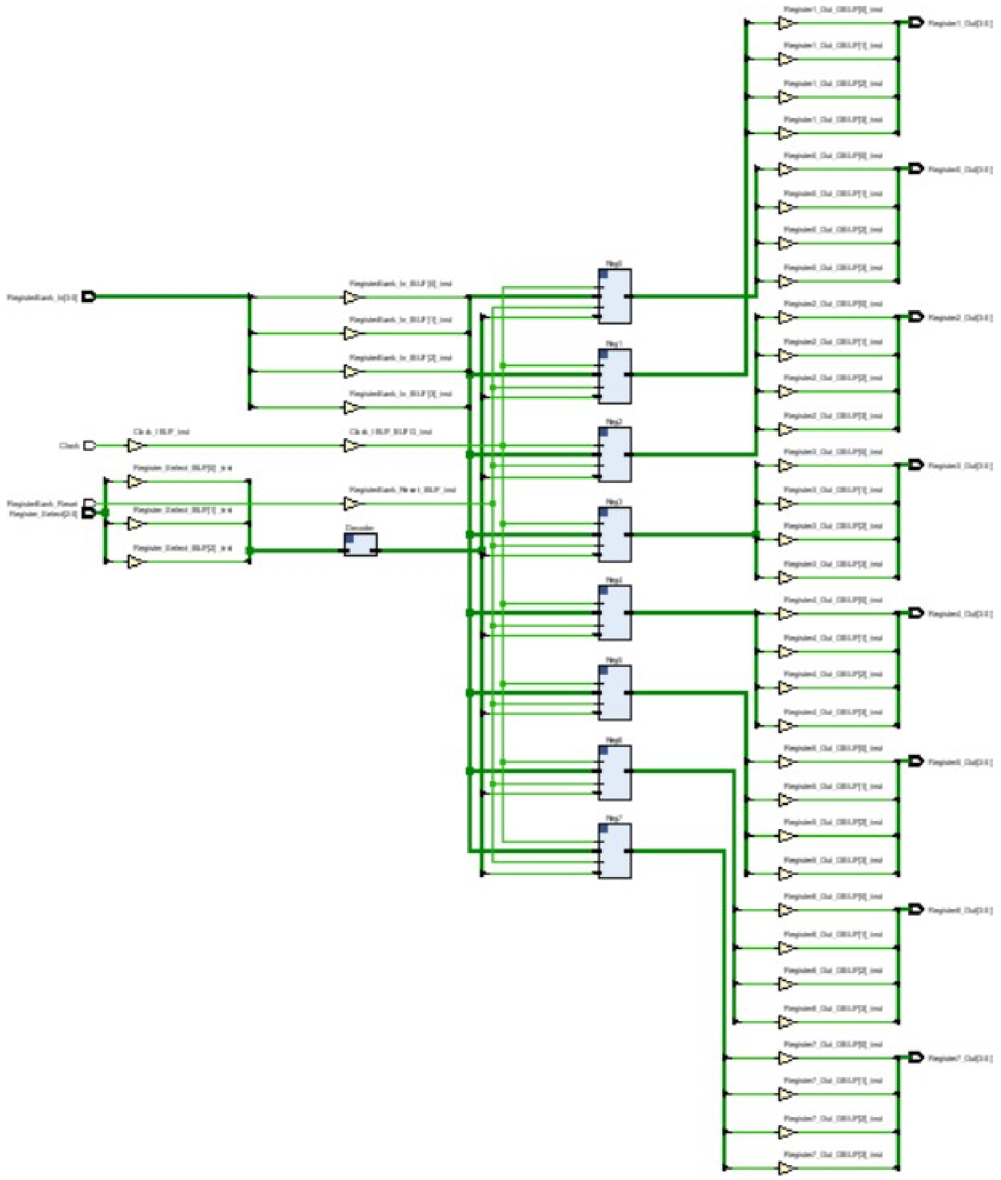
```
entity RegisterBank is
    Port ( RegisterBank_In : in STD_LOGIC_VECTOR (3 downto 0) ;
           RegisterBank_Reset : in STD_LOGIC;
           Register_Select : in STD_LOGIC_VECTOR (2 downto 0);
           Clock : in STD_LOGIC;
           Register0_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register1_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register2_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register3_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register4_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register5_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register6_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register7_Out : out STD_LOGIC_VECTOR (3 downto 0));
end RegisterBank;

architecture Behavioral of RegisterBank is

component Register_4bit is
    Port ( RegIn : in STD_LOGIC_VECTOR (3 downto 0);
           RegOut : out STD_LOGIC_VECTOR (3 downto 0);
           Clock : in STD_LOGIC;
           RegEnable : in STD_LOGIC;
           RegReset : in STD_LOGIC);
end component;
```

```
component Decoder_3_to_8 is
   Port ( I : in STD_LOGIC_VECTOR (2 downto 0);
          EN : in STD_LOGIC;
          Y : out STD_LOGIC_VECTOR (7 downto 0));
end component;

Signal Active: std_logic;
Signal RegisterSelecter: std_logic_vector (7 downto 0);

begin

Active <= '1';

Decoder : Decoder_3_to_8
port map(
   I => Register_Select,
   EN => Active,
   Y => RegisterSelecter);

Reg0 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register0_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(0),
   Clock => Clock);

Reg1 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register1_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(1),
   Clock => Clock);

Reg2 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register2_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(2),
   Clock => Clock);

Reg3 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register3_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(3),
   Clock => Clock);

Reg4 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register4_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(4),
   Clock => Clock);

Reg5 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register5_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(5),
   Clock => Clock);

Reg6 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register6_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(6),
   Clock => Clock);

Reg7 : Register_4bit
port map(
   RegIn => RegisterBank_In,
   RegOut => Register7_Out,
   RegReset => RegisterBank_Reset,
   RegEnable => RegisterSelecter(7),
   Clock => Clock);

end Behavioral;
```

## RTL



## Synthesized Design



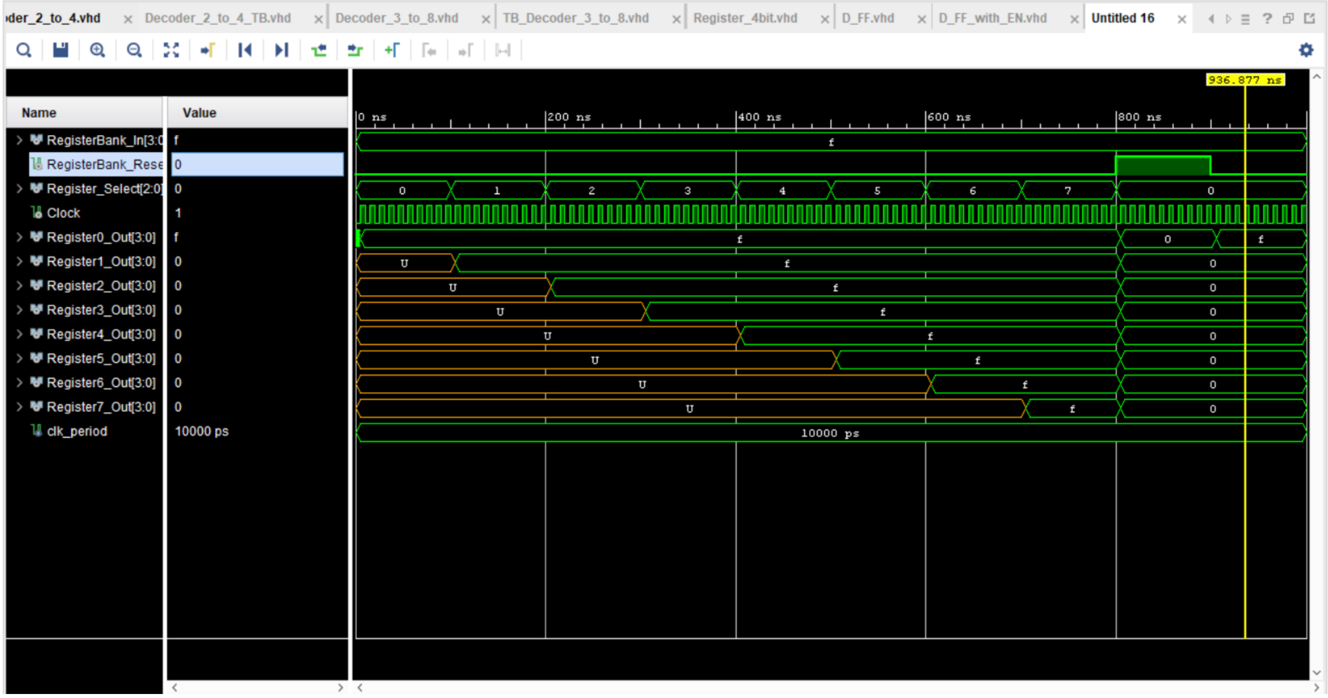
**Simulation_Code**

```
entity RegBankTB is
--  Port ( );
end RegBankTB;

architecture Behavioral of RegBankTB is
component RegisterBank is
    Port ( RegisterBank_In : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterBank_Reset : in STD_LOGIC;
           Register_Select : in STD_LOGIC_VECTOR (2 downto 0);
           Clock : in STD_LOGIC;
           Register0_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register1_Out : out STD_LOGIC_VECTOR (3 downto 0);
           Register2_Out : out STD_LOGIC_VECTOR (3 downto 0);
```

```
        Register3_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register4_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register5_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register6_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register7_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;

        signal RegisterBank_In : STD_LOGIC_VECTOR (3 downto 0);
        signal RegisterBank_Reset : STD_LOGIC;
        signal Register_Select :  STD_LOGIC_VECTOR (2 downto 0);
        signal Clock : STD_LOGIC;
        signal Register0_Out :  STD_LOGIC_VECTOR (3 downto 0);
        signal Register1_Out :  STD_LOGIC_VECTOR (3 downto 0);
        signal Register2_Out :  STD_LOGIC_VECTOR (3 downto 0);
        signal Register3_Out :  STD_LOGIC_VECTOR (3 downto 0);
        signal Register4_Out :  STD_LOGIC_VECTOR (3 downto 0);
        signal Register5_Out :  STD_LOGIC_VECTOR (3 downto 0);
        signal Register6_Out :  STD_LOGIC_VECTOR (3 downto 0);
        signal Register7_Out :  STD_LOGIC_VECTOR (3 downto 0);
        constant clk_period : time := 10 ns;

begin


uut : registerbank port map(
        RegisterBank_In => RegisterBank_In,
        RegisterBank_Reset =>RegisterBank_Reset,
        Register_Select=> Register_Select,
        Clock =>Clock,
        Register0_Out=> Register0_Out,
        Register1_Out=> Register1_Out,
        Register2_Out=> Register2_Out,
        Register3_Out=> Register3_Out,
        Register4_Out=> Register4_Out,
        Register5_Out=> Register5_Out,
        Register6_Out=> Register6_Out,
        Register7_Out=> Register7_Out
        );
clk_process :process
        begin
         Clock <='0';
         wait for clk_period/2;
         Clock <='1';
         wait for clk_period/2;
         end process;
stiM_PROC: PROCESS
BEGIN

RegisterBank_In<="1111";
```

```
RegisterBank_Reset<='0';
Register_Select<="000";
wait for 100 ns;
RegisterBank_In<="1111";
RegisterBank_Reset<='0';
Register_Select<="001";
wait for 100 ns;
RegisterBank_In<="1111";
RegisterBank_Reset<='0';
Register_Select<="010";
wait for 100 ns;
RegisterBank_In<="1111";
RegisterBank_Reset<='0';
Register_Select<="011";
wait for 100 ns;
RegisterBank_In<="1111";
RegisterBank_Reset<='0';
Register_Select<="100";
wait for 100 ns;
RegisterBank_In<="1111";
RegisterBank_Reset<='0';
Register_Select<="101";
wait for 100 ns;
RegisterBank_In<="1111";
RegisterBank_Reset<='0';
Register_Select<="110";
wait for 100 ns;
RegisterBank_In<="1111";
RegisterBank_Reset<='0';
Register_Select<="111";
wait for 100 ns;


RegisterBank_In<="1111";
RegisterBank_Reset<='1';
Register_Select<="000";
wait for 100 ns;



end process;
end;
```

## Behavioral Simulation



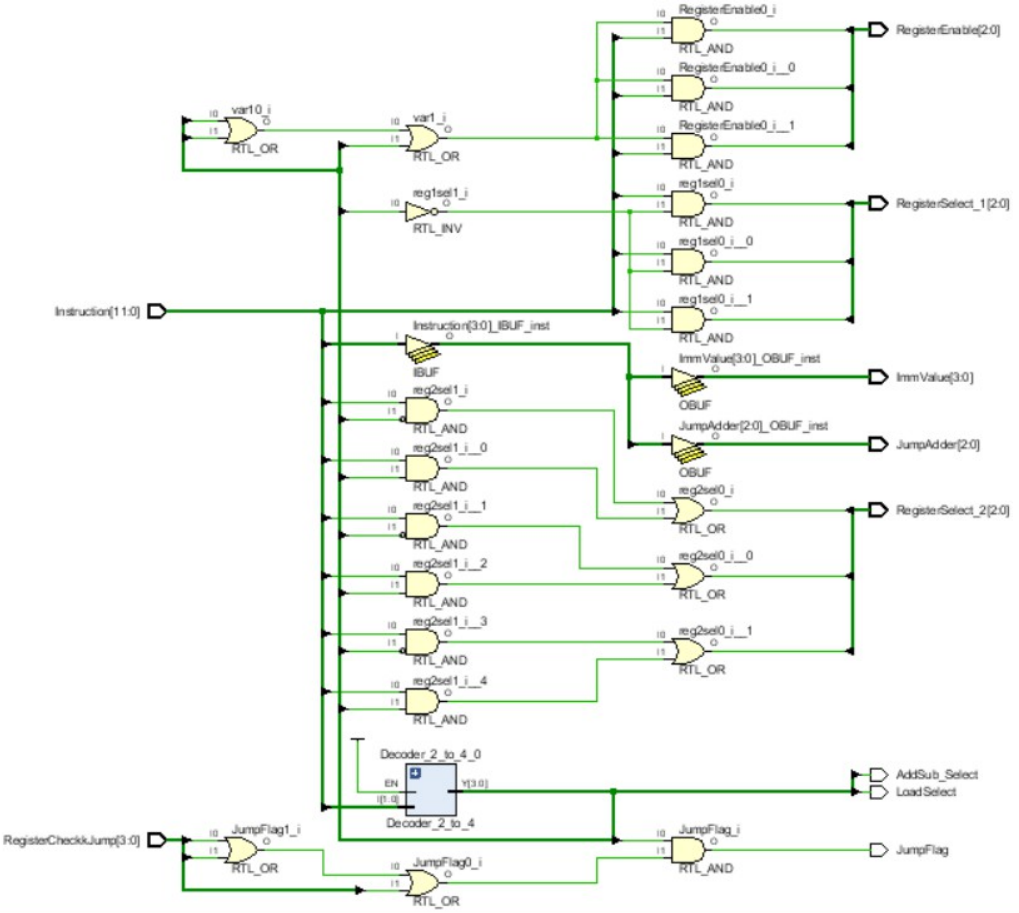
-

# Instruction Decoder

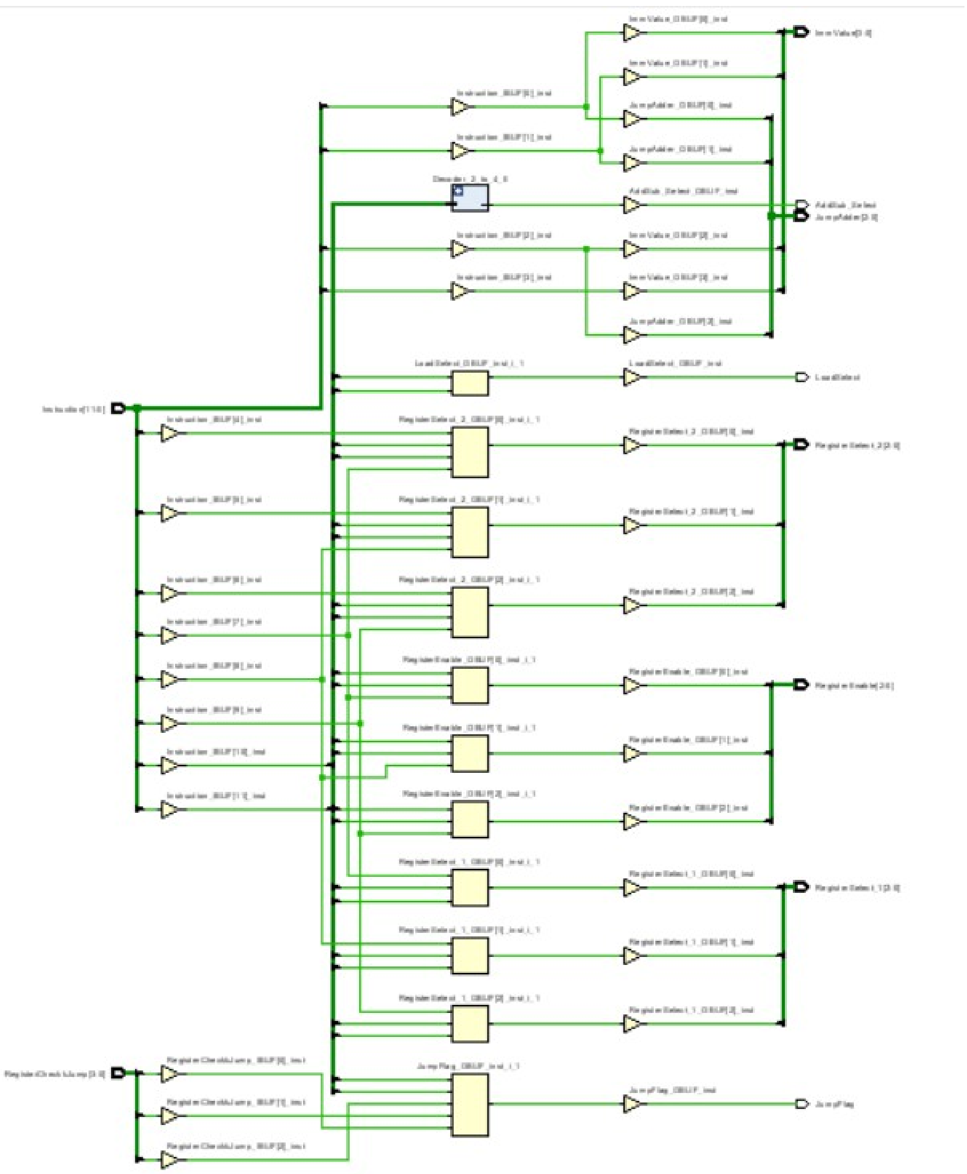
## Code

```
entity Instruction_Decoder is
    Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
           RegisterCheckkJump : in STD_LOGIC_VECTOR (3 downto 0);
           RegisterEnable : out STD_LOGIC_VECTOR (2 downto 0);
           LoadSelect : out STD_LOGIC;
           ImmValue : out STD_LOGIC_VECTOR (3 downto 0);
           RegisterSelect_1 : out STD_LOGIC_VECTOR (2 downto 0);
           RegisterSelect_2 : out STD_LOGIC_VECTOR (2 downto 0);
           AddSub_Select : out STD_LOGIC;
           JumpFlag : out STD_LOGIC;
           JumpAdder : out STD_LOGIC_VECTOR (2 downto 0));
end Instruction_Decoder;

 architecture Behavioral of Instruction_Decoder is

component Decoder_2_to_4
    port(
        I: in std_logic_vector(1 downto 0);
        En: in std_logic;
        Y: out std_logic_vector(3 downto 0));
    end component;

Signal add, neg, mov, jzr : std_logic;
Signal var1 : std_logic;
signal reg1sel,reg2sel : STD_LOGIC_VECTOR (2 downto 0);

begin

Decoder_2_to_4_0 : Decoder_2_to_4
   port map (
   I(0) => Instruction(10),
   I(1) => Instruction(11),
   EN => '1',
   Y(0)=> add,
   Y(1) => neg,
   Y(2) => mov,
   Y(3) => jzr);

LoadSelect <= mov;
AddSub_Select <= neg;
JumpFlag <=jzr and  ((RegisterCheckkJump(0) or RegisterCheckkJump(1) or
RegisterCheckkJump(2)));
JumpAdder <= Instruction (2 downto 0);

var1 <= add or neg or mov;
RegisterEnable(0) <= var1 and Instruction(7);
RegisterEnable(1) <= var1 and Instruction(8);
RegisterEnable(2) <= var1 and Instruction(9);
ImmValue <= Instruction (3 downto 0);

reg1sel(0) <=((Instruction (7))and(not neg));
reg1sel(1) <=((Instruction (8))and(not neg));
reg1sel(2) <=((Instruction (9))and(not neg));

reg2sel(2) <=((Instruction (6))and(not neg)) or ((Instruction (9))and(neg));
reg2sel(1) <=((Instruction (5))and(not neg)) or ((Instruction (8))and(neg));
reg2sel(0) <=((Instruction (4))and(not neg)) or ((Instruction (7))and(neg));

RegisterSelect_1 <= reg1sel;
RegisterSelect_2 <= reg2sel;

end Behavioral;
```

**RTL**



## Synthesized Design



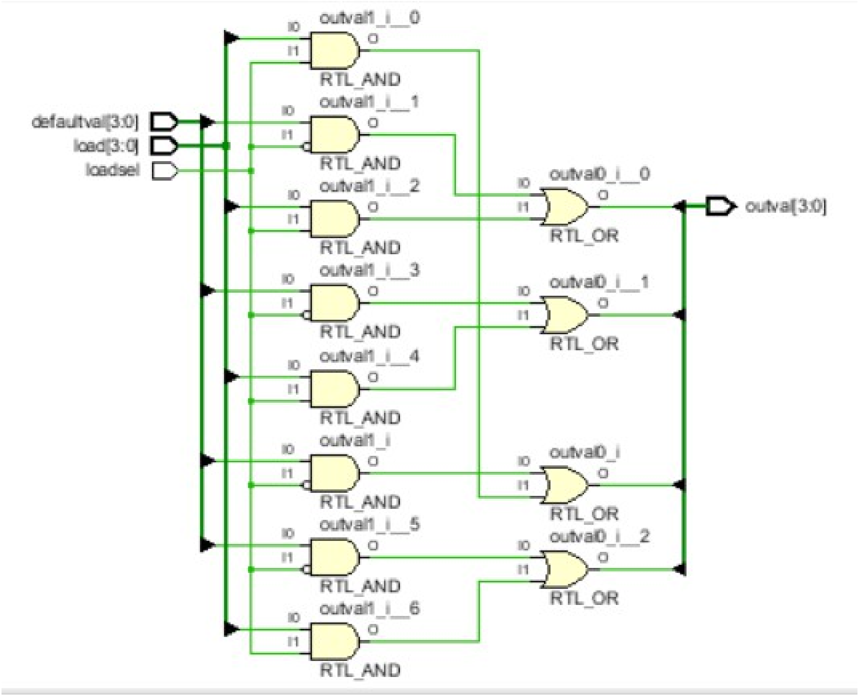
# way2bit4mux

**Code**

```
entity way2bit4mux is
    Port ( loadsel : STD_LOGIC;
           load : in STD_LOGIC_VECTOR (3 downto 0);
           defaultval : in STD_LOGIC_VECTOR (3 downto 0);
           outval : out STD_LOGIC_VECTOR (3 downto 0)
           );
end way2bit4mux;

architecture Behavioral of way2bit4mux is
```

```
begin

outval(0) <= (defaultval(0)and (not loadsel)) or (load(0) and loadsel);
outval(1) <= (defaultval(1)and (not loadsel)) or (load(1) and loadsel);
outval(2) <= (defaultval(2)and (not loadsel)) or (load(2) and loadsel);
outval(3) <= (defaultval(3)and (not loadsel)) or (load(3) and loadsel);


end Behavioral;
```

### RTL



# Micro Processor

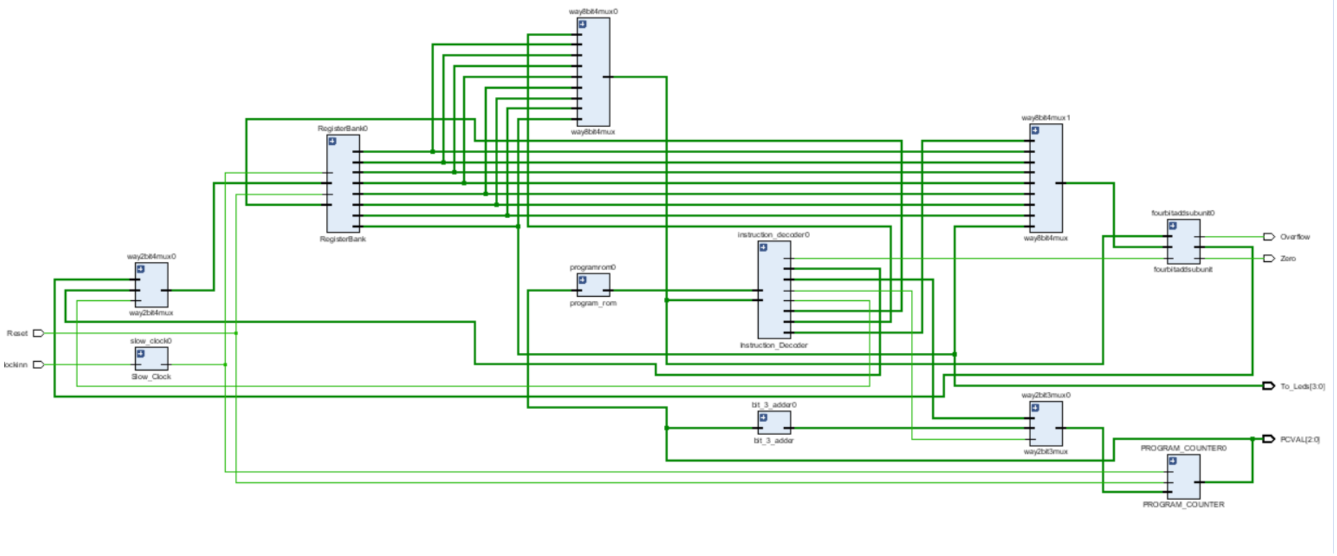
### Code

```
entity Micro_Processor is
   Port ( Clockinn : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          To_Leds : out STD_LOGIC_VECTOR (3 downto 0);
          PCVAL : out STD_LOGIC_VECTOR (2 downto 0));

end Micro_Processor;
```

```
architecture Behavioral of Micro_Processor is

component PROGRAM_COUNTER
   Port ( TOIN : in STD_LOGIC_VECTOR (2 downto 0);
        TOOUT : out STD_LOGIC_VECTOR (2 downto 0);
        CLK : in STD_LOGIC;
        RES : in STD_LOGIC);
end component;
component way2bit3mux
   Port ( jumpflag : in STD_LOGIC;
        addresstojump : in STD_LOGIC_VECTOR (2 downto 0);
        default1 : in STD_LOGIC_VECTOR (2 downto 0);
        out1 : out STD_LOGIC_VECTOR (2 downto 0));
end component;
component bit_3_adder
   Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
        S : out STD_LOGIC_VECTOR (2 downto 0)
        );
end component;
component program_rom is
   Port ( regsel : in STD_LOGIC_VECTOR (2 downto 0);
        out1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component Instruction_Decoder is
   Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
        RegisterCheckkJump : in STD_LOGIC_VECTOR (3 downto 0);
        RegisterEnable : out STD_LOGIC_VECTOR (2 downto 0);
        LoadSelect : out STD_LOGIC;
        ImmValue : out STD_LOGIC_VECTOR (3 downto 0);
        RegisterSelect_1 : out STD_LOGIC_VECTOR (2 downto 0);
        RegisterSelect_2 : out STD_LOGIC_VECTOR (2 downto 0);
        AddSub_Select : out STD_LOGIC;
        JumpFlag : out STD_LOGIC;
        JumpAdder : out STD_LOGIC_VECTOR (2 downto 0));
end component;

component way2bit4mux is
   Port ( loadsel : STD_LOGIC;
        load : in STD_LOGIC_VECTOR (3 downto 0);
        defaultval : in STD_LOGIC_VECTOR (3 downto 0);
        outval : out STD_LOGIC_VECTOR (3 downto 0)
        );
end component;
component Slow_Clock is
   Port ( Clk_in : in STD_LOGIC;
        Clk_out : out STD_LOGIC);
end component;
component RegisterBank is
```

```
   Port ( RegisterBank_In : in STD_LOGIC_VECTOR (3 downto 0) ;
          RegisterBank_Reset : in STD_LOGIC;
          Register_Select : in STD_LOGIC_VECTOR (2 downto 0);
          Clock : in STD_LOGIC;
          Register0_Out : out STD_LOGIC_VECTOR (3 downto 0);
          Register1_Out : out STD_LOGIC_VECTOR (3 downto 0);
          Register2_Out : out STD_LOGIC_VECTOR (3 downto 0);
          Register3_Out : out STD_LOGIC_VECTOR (3 downto 0);
          Register4_Out : out STD_LOGIC_VECTOR (3 downto 0);
          Register5_Out : out STD_LOGIC_VECTOR (3 downto 0);
          Register6_Out : out STD_LOGIC_VECTOR (3 downto 0);
          Register7_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component way8bit4mux is
   Port ( regsel : in STD_LOGIC_VECTOR (2 downto 0);
          reg0 : in STD_LOGIC_VECTOR (3 downto 0);
          reg1 : in STD_LOGIC_VECTOR (3 downto 0);
          reg2 : in STD_LOGIC_VECTOR (3 downto 0);
          reg3 : in STD_LOGIC_VECTOR (3 downto 0);
          reg4 : in STD_LOGIC_VECTOR (3 downto 0);
          reg5 : in STD_LOGIC_VECTOR (3 downto 0);
          reg6 : in STD_LOGIC_VECTOR (3 downto 0);
          reg7 : in STD_LOGIC_VECTOR (3 downto 0);
          out1 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component fourbitaddsubunit is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
          B : in STD_LOGIC_VECTOR (3 downto 0);
          Cal_Control : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          S : out STD_LOGIC_VECTOR (3 downto 0));
end component;


signal signal6,newsig,signalid2,signalid6,cclk,rres: STD_LOGIC;
signal signal1,signal2,signal3,signal4,signal5,signal32,signalid1,signalid4,signalid5 :
STD_LOGIC_VECTOR (2 downto 0);
signal signal7 :  STD_LOGIC_VECTOR (11 downto 0);
signal signalid3,signal8,signal9,signalid7 :  STD_LOGIC_VECTOR (3 downto 0);
signal rbout0,rbout1,rbout2,rbout3,rbout4,rbout5,rbout6,rbout7,signal10,signal11 :
STD_LOGIC_VECTOR (3 downto 0);

begin



PROGRAM_COUNTER0 : PROGRAM_COUNTER PORT MAP (
        TOIN =>signal1  ,
        TOOUT => signal32,
        CLK => cclk,
        res =>rres );
way2bit3mux0 : way2bit3mux Port map (
        jumpflag =>signal6,
        addresstojump =>signal5,
        default1 =>signal4,
        out1 =>signal1);
bit_3_adder0 : bit_3_adder Port map (
        A =>signal2,
        S =>signal4);
programrom0 : program_rom  Port map (
        regsel =>signal3,
        out1 =>signal7 );
instruction_decoder0 : Instruction_Decoder Port map (
        Instruction =>signal7,
        RegisterCheckkJump =>signalid7,
        RegisterEnable =>signalid1,
        LoadSelect =>signalid2,
        ImmValue =>signalid3,
        RegisterSelect_1=>signalid4,
        RegisterSelect_2 =>signalid5,
        AddSub_Select =>signalid6,
        JumpFlag =>signal6,
        JumpAdder =>signal5);

way2bit4mux0 : way2bit4mux Port map (
        loadsel =>signalid2,
        load =>signalid3,
        defaultval =>signal9,
        outval =>signal8
        );
slow_clock0 : Slow_Clock Port map (
        Clk_in =>Clockinn,
        Clk_out=>cclk);

RegisterBank0 : RegisterBank Port map (
         RegisterBank_In=>signal8,
         RegisterBank_Reset =>rres,
         Register_Select =>signalid1,
         Clock =>cclk,
         Register0_Out =>rbout0,
         Register1_Out =>rbout1,
         Register2_Out =>rbout2,
         Register3_Out =>rbout3,
         Register4_Out =>rbout4,
         Register5_Out =>rbout5,
         Register6_Out =>rbout6,
         Register7_Out =>rbout7);

way8bit4mux0 : way8bit4mux Port map (
         regsel =>signalid4,
         reg0 =>rbout0,
         reg1 =>rbout1,
         reg2 =>rbout2,
         reg3 =>rbout3,
         reg4 =>rbout4,
         reg5 =>rbout5,
         reg6 =>rbout6,
         reg7 =>rbout7,
         out1 =>signal10);
way8bit4mux1 : way8bit4mux Port map (
         regsel =>signalid5,
         reg0 =>rbout0,
         reg1 =>rbout1,
         reg2 =>rbout2,
         reg3 =>rbout3,
         reg4 =>rbout4,
         reg5 =>rbout5,
         reg6 =>rbout6,
         reg7 =>rbout7,
         out1 =>signal11);

fourbitaddsubunit0 : fourbitaddsubunit Port map (
         A =>signal10,
         B =>signal11,
         Cal_Control =>signalid6,
         Overflow =>overflow,
         Zero =>zero,
         S =>signal9);
signalid7<=signal10;
To_Leds(0)<=rbout7(0);
To_Leds(1)<=rbout7(1);
To_Leds(2)<=rbout7(2);
To_Leds(3)<=rbout7(3);
rres <=reset;
signal3 <= signal32;
signal2 <= signal32;
PCVAL <=signal32;
--signal1<=firstin;
end Behavioral;
```

## RTL

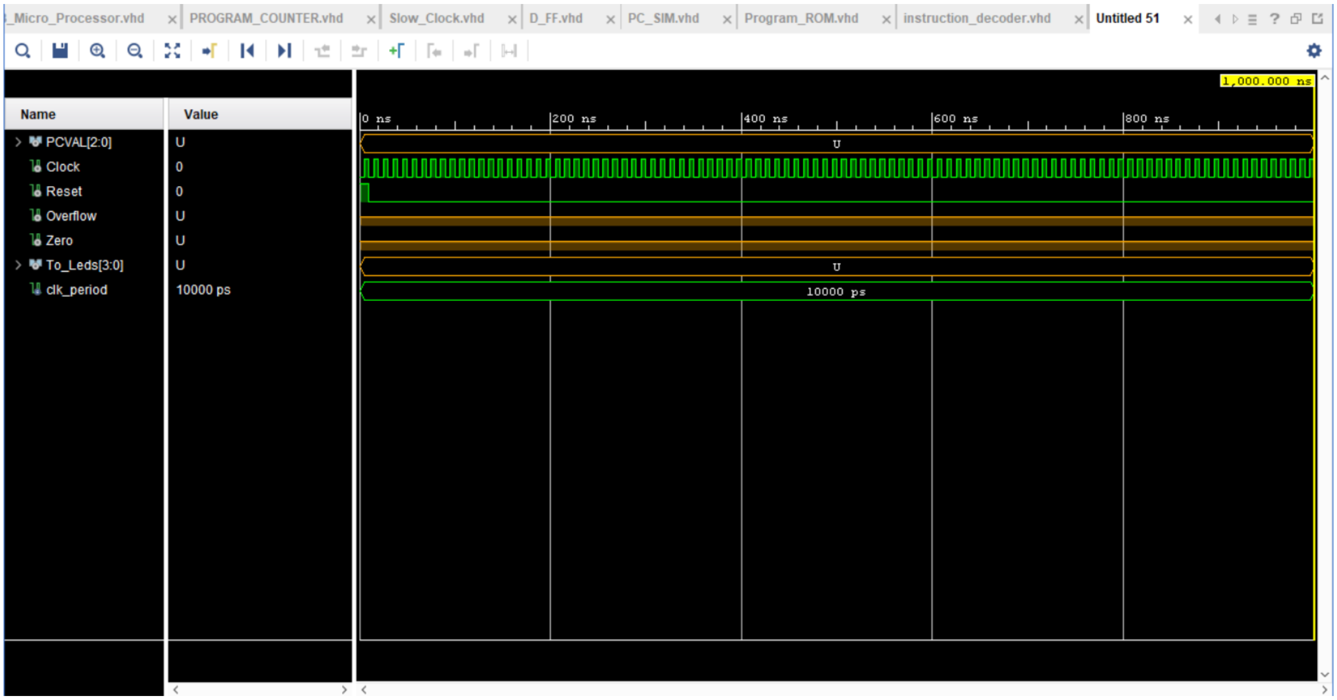


**Simulation_Code**

```
entity TB_Micro_Processor is
--  Port ( );
end TB_Micro_Processor;

architecture Behavioral of TB_Micro_Processor is
component Micro_Processor is
   Port ( Clockinn : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          PCVAL : out STD_LOGIC_VECTOR (2 downto 0);
          To_Leds : out STD_LOGIC_VECTOR (3 downto 0));
end component;
          signal PCVAL : STD_LOGIC_VECTOR (2 downto 0);
          signal Clock : STD_LOGIC;
          signal Reset :  STD_LOGIC;
          signal Overflow :  STD_LOGIC;
          signal Zero :  STD_LOGIC;
          signal To_Leds :  STD_LOGIC_VECTOR (3 downto 0);
          constant clk_period : time := 10 ns;

begin


uut : Micro_Processor Port map (
          Clockinn=>Clock,
          Reset=>Reset,
          Overflow=>Overflow,
          Zero=>Zero,
          PCVAL=>PCVAL,
          To_Leds=>To_Leds);

counte_process :process
          begin
           clock <='0';
           wait for clk_period/2;
           clock <='1';
           wait for clk_period/2;
           end process;
stiM_PROC :process
           begin

           reset<='1';
           wait for 10ns;
           reset<='0';
           wait;




end process;
end;
```

*** When Simulation is done using **Slowclock** Some OUTPUT Values seems to be set to to "U'***

## Behavioral Simulation



| Name | Value |
|---|---|
| PCVAL[2:0] | U |
| Clock | 0 |
| Reset | 0 |
| Overflow | U |
| Zero | U |
| To_Leds[3:0] | U |
| clk_period | 10000 ps |

*** Hence for testing we use the processor without slow clock***

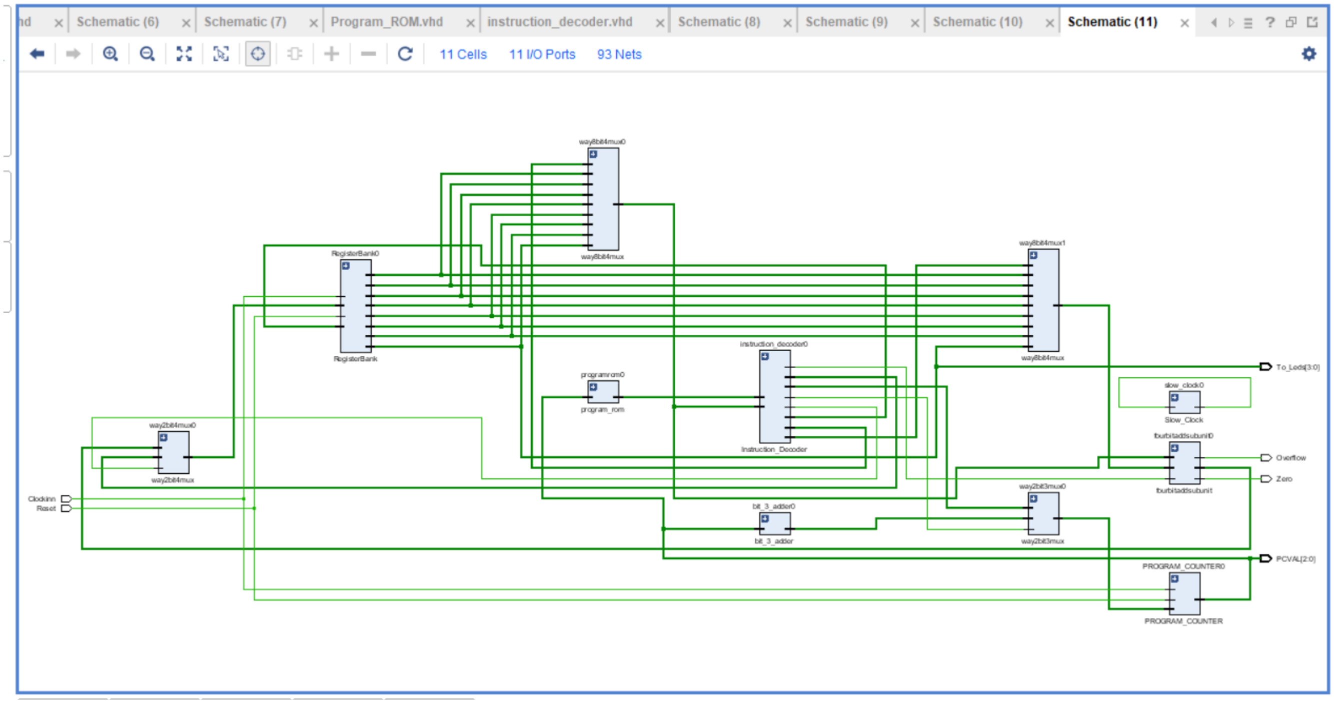
**Code**

```
entity Micro_Processor is
   Port ( Clockinn : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          To_Leds : out STD_LOGIC_VECTOR (3 downto 0);
          PCVAL : out STD_LOGIC_VECTOR (2 downto 0));

end Micro_Processor;

architecture Behavioral of Micro_Processor is

component PROGRAM_COUNTER
   Port ( TOIN : in STD_LOGIC_VECTOR (2 downto 0);
          TOOUT : out STD_LOGIC_VECTOR (2 downto 0);
          CLK : in STD_LOGIC;
          RES : in STD_LOGIC);
end component;
component way2bit3mux
   Port ( jumpflag : in STD_LOGIC;
          addresstojump : in STD_LOGIC_VECTOR (2 downto 0);
          default1 : in STD_LOGIC_VECTOR (2 downto 0);
          out1 : out STD_LOGIC_VECTOR (2 downto 0));
end component;
component bit_3_adder
   Port ( A : in STD_LOGIC_VECTOR (2 downto 0);
          S : out STD_LOGIC_VECTOR (2 downto 0)
          );
end component;
component program_rom is
   Port ( regsel : in STD_LOGIC_VECTOR (2 downto 0);
          out1 : out STD_LOGIC_VECTOR (11 downto 0));
end component;
component Instruction_Decoder is
   Port ( Instruction : in STD_LOGIC_VECTOR (11 downto 0);
          RegisterCheckkJump : in STD_LOGIC_VECTOR (3 downto 0);
          RegisterEnable : out STD_LOGIC_VECTOR (2 downto 0);
          LoadSelect : out STD_LOGIC;
          ImmValue : out STD_LOGIC_VECTOR (3 downto 0);
          RegisterSelect_1 : out STD_LOGIC_VECTOR (2 downto 0);
          RegisterSelect_2 : out STD_LOGIC_VECTOR (2 downto 0);
          AddSub_Select : out STD_LOGIC;
          JumpFlag : out STD_LOGIC;
          JumpAdder : out STD_LOGIC_VECTOR (2 downto 0));
```

```
end component;

component way2bit4mux is
   Port ( loadsel : STD_LOGIC;
        load : in STD_LOGIC_VECTOR (3 downto 0);
        defaultval : in STD_LOGIC_VECTOR (3 downto 0);
        outval : out STD_LOGIC_VECTOR (3 downto 0)
        );
end component;
component Slow_Clock is
   Port ( Clk_in : in STD_LOGIC;
        Clk_out : out STD_LOGIC);
end component;
component RegisterBank is
   Port ( RegisterBank_In : in STD_LOGIC_VECTOR (3 downto 0) ;
        RegisterBank_Reset : in STD_LOGIC;
        Register_Select : in STD_LOGIC_VECTOR (2 downto 0);
        Clock : in STD_LOGIC;
        Register0_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register1_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register2_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register3_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register4_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register5_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register6_Out : out STD_LOGIC_VECTOR (3 downto 0);
        Register7_Out : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component way8bit4mux is
   Port ( regsel : in STD_LOGIC_VECTOR (2 downto 0);
        reg0 : in STD_LOGIC_VECTOR (3 downto 0);
        reg1 : in STD_LOGIC_VECTOR (3 downto 0);
        reg2 : in STD_LOGIC_VECTOR (3 downto 0);
        reg3 : in STD_LOGIC_VECTOR (3 downto 0);
        reg4 : in STD_LOGIC_VECTOR (3 downto 0);
        reg5 : in STD_LOGIC_VECTOR (3 downto 0);
        reg6 : in STD_LOGIC_VECTOR (3 downto 0);
        reg7 : in STD_LOGIC_VECTOR (3 downto 0);
        out1 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component fourbitaddsubunit is
   Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
        B : in STD_LOGIC_VECTOR (3 downto 0);
        Cal_Control : in STD_LOGIC;
        Overflow : out STD_LOGIC;
        Zero : out STD_LOGIC;
        S : out STD_LOGIC_VECTOR (3 downto 0));
end component;
```

```
signal signal6,isolateclock,signalid2,signalid6,cclk,rres: STD_LOGIC;
signal signal1,signal2,signal3,signal4,signal5,signal32,signalid1,signalid4,signalid5 :
STD_LOGIC_VECTOR (2 downto 0);
signal signal7 :  STD_LOGIC_VECTOR (11 downto 0);
signal signalid3,signal8,signal9,signalid7 :  STD_LOGIC_VECTOR (3 downto 0);
signal rbout0,rbout1,rbout2,rbout3,rbout4,rbout5,rbout6,rbout7,signal10,signal11 :
STD_LOGIC_VECTOR (3 downto 0);

begin



PROGRAM_COUNTER0 : PROGRAM_COUNTER PORT MAP (
        TOIN =>signal1  ,
        TOOUT => signal32,
        CLK => cclk,
        res =>rres );
way2bit3mux0 : way2bit3mux Port map (
        jumpflag =>signal6,
        addresstojump =>signal5,
        default1 =>signal4,
        out1 =>signal1);
bit_3_adder0 : bit_3_adder Port map (
        A =>signal2,
        S =>signal4);
programrom0 : program_rom  Port map (
        regsel =>signal3,
        out1 =>signal7 );
instruction_decoder0 : Instruction_Decoder Port map (
        Instruction =>signal7,
        RegisterCheckkJump =>signalid7,
        RegisterEnable =>signalid1,
        LoadSelect =>signalid2,
        ImmValue =>signalid3,
        RegisterSelect_1=>signalid4,
        RegisterSelect_2 =>signalid5,
        AddSub_Select =>signalid6,
        JumpFlag =>signal6,
        JumpAdder =>signal5);

way2bit4mux0 : way2bit4mux Port map (
        loadsel =>signalid2,
        load =>signalid3,
        defaultval =>signal9,
        outval =>signal8
        );
slow_clock0 : Slow_Clock Port map (
        Clk_in =>isolateclock,
        Clk_out=>isolateclock);

RegisterBank0 : RegisterBank Port map (
         RegisterBank_In=>signal8,
         RegisterBank_Reset =>rres,
         Register_Select =>signalid1,
         Clock =>cclk,
         Register0_Out =>rbout0,
         Register1_Out =>rbout1,
         Register2_Out =>rbout2,
         Register3_Out =>rbout3,
         Register4_Out =>rbout4,
         Register5_Out =>rbout5,
         Register6_Out =>rbout6,
         Register7_Out =>rbout7);

way8bit4mux0 : way8bit4mux Port map (
         regsel =>signalid4,
         reg0 =>rbout0,
         reg1 =>rbout1,
         reg2 =>rbout2,
         reg3 =>rbout3,
         reg4 =>rbout4,
         reg5 =>rbout5,
         reg6 =>rbout6,
         reg7 =>rbout7,
         out1 =>signal10);
way8bit4mux1 : way8bit4mux Port map (
         regsel =>signalid5,
         reg0 =>rbout0,
         reg1 =>rbout1,
         reg2 =>rbout2,
         reg3 =>rbout3,
         reg4 =>rbout4,
         reg5 =>rbout5,
         reg6 =>rbout6,
         reg7 =>rbout7,
         out1 =>signal11);

fourbitaddsubunit0 : fourbitaddsubunit Port map (
         A =>signal10,
         B =>signal11,
         Cal_Control =>signalid6,
         Overflow =>overflow,
         Zero =>zero,
         S =>signal9);
signalid7<=signal10;
To_Leds(0)<=rbout7(0);
To_Leds(1)<=rbout7(1);
To_Leds(2)<=rbout7(2);
To_Leds(3)<=rbout7(3);
rres <=reset;
Cclk<=Clockinn;
signal3 <= signal32;
signal2 <= signal32;
PCVAL <=signal32;
--signal1<=firstin;
end Behavioral;
```

## RTL



## Simulation_Code

```
entity TB_Micro_Processor is
--  Port ( );
end TB_Micro_Processor;

architecture Behavioral of TB_Micro_Processor is
component Micro_Processor is
   Port ( Clockinn : in STD_LOGIC;
          Reset : in STD_LOGIC;
          Overflow : out STD_LOGIC;
          Zero : out STD_LOGIC;
          PCVAL : out STD_LOGIC_VECTOR (2 downto 0);
          To_Leds : out STD_LOGIC_VECTOR (3 downto 0));
end component;
```

```
        signal PCVAL : STD_LOGIC_VECTOR (2 downto 0);
        signal Clock : STD_LOGIC;
        signal Reset :  STD_LOGIC;
        signal Overflow :  STD_LOGIC;
        signal Zero :  STD_LOGIC;
        signal To_Leds :  STD_LOGIC_VECTOR (3 downto 0);
        constant clk_period : time := 10 ns;

begin


uut : Micro_Processor Port map (
        Clockinn=>Clock,
        Reset=>Reset,
        Overflow=>Overflow,
        Zero=>Zero,
        PCVAL=>PCVAL,
        To_Leds=>To_Leds);

counte_process :process
        begin
         clock <='0';
         wait for clk_period/2;
         clock <='1';
         wait for clk_period/2;
         end process;
stiM_PROC :process
         begin

         reset<='1';
         wait for 10ns;
         reset<='0';
         wait;






end process;
end;
```

## Behavioral Simulation

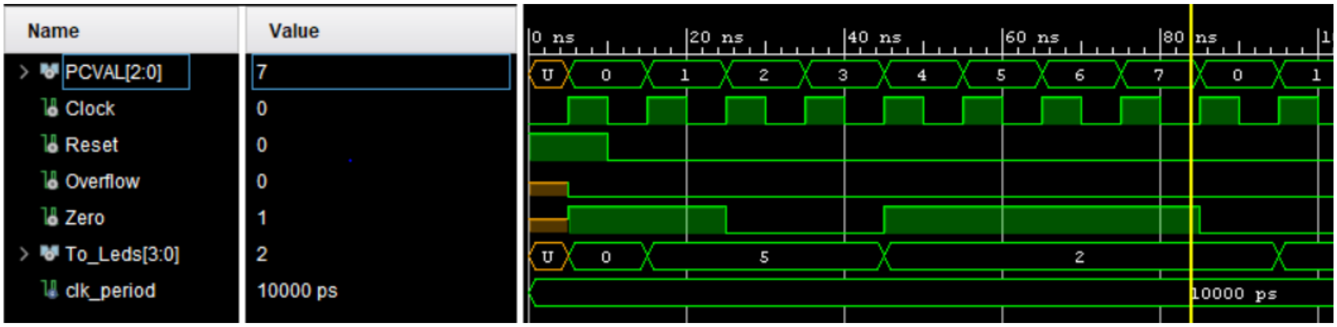
## :: TESTING 1 ::

## ProgramROM

```
signal ROM : rom_type :=(
            "101110000101",---  r7 <= 5
```

```
              "101100000011",---  r6 <= 3
              "011100000000",---  r6 <=-r6
              "001111100000",---  r7 <= r7+r6
              "100000000000",--   r0<=0
              "100000000000",---  r0<=0
              "100000000000",---  r0<=0
              "100000000000" ---  r0<=0
              );
```

## OUTPUT



| Name | Value |
|---|---|
| PCVAL[2:0] | 7 |
| Clock | 0 |
| Reset | 0 |
| Overflow | 0 |
| Zero | 1 |
| To_Leds[3:0] | 2 |
| clk_period | 10000 ps |

## :: TESTING 2 ::

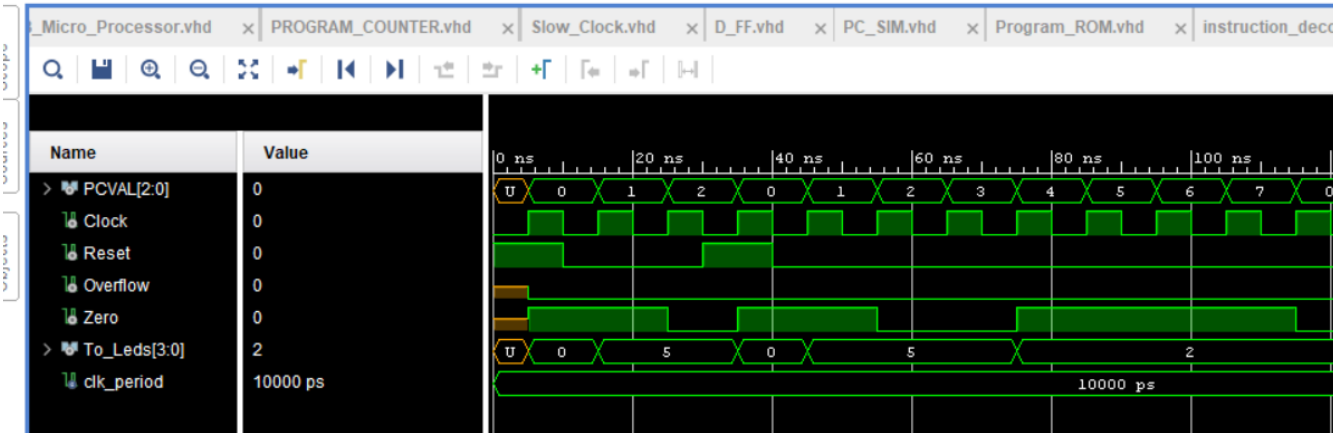
## ProgramROM

```
signal ROM : rom_type :=(
              "101110000101",---  r7 <= 5
              "101100000011",---  r6 <= 3
              "011100000000",---  r6 <=-r6
              "001111100000",---  r7 <= r7+r6
              "100000000000",--   r0<=0
              "100000000000",---  r0<=0
              "100000000000",---  r0<=0
              "100000000000" ---  r0<=0
              );
```

## TB_CODE

```
reset<='1';
        wait for 10ns;
        reset<='0';
        wait for 20ns;
        reset<='1';
        wait for 10ns;
        reset<='0';
        wait;
```

## OUTPUT



| Name | Value |
|---|---|
| PCVAL[2:0] | 0 |
| Clock | 0 |
| Reset | 0 |
| Overflow | 0 |
| Zero | 0 |
| To_Leds[3:0] | 2 |
| clk_period | 10000 ps |

## :: TESTING 3 ::

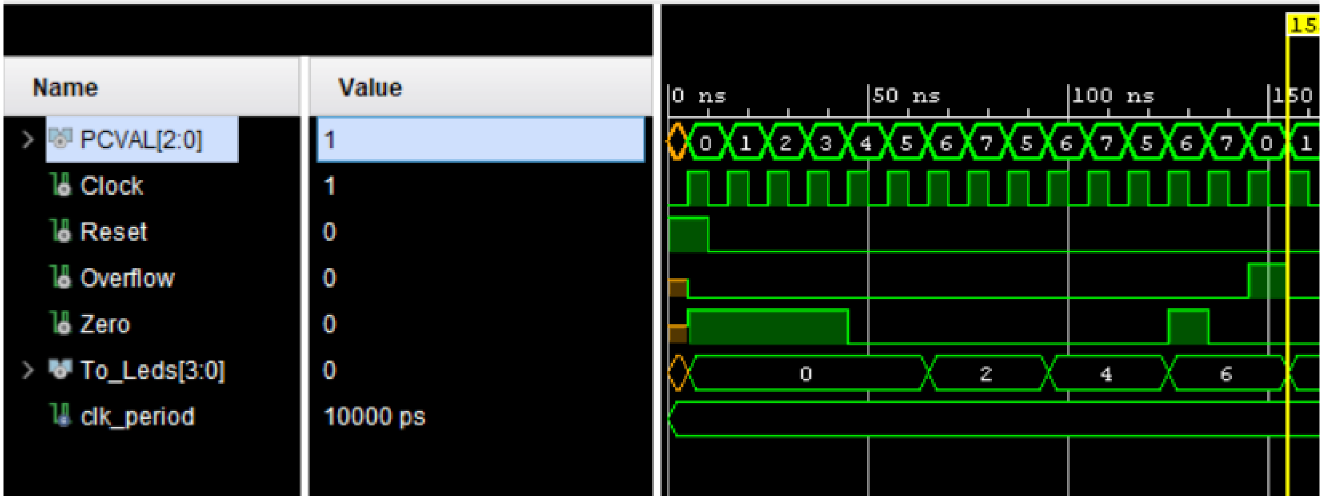
## ProgramROM

```
signal ROM : rom_type :=(
                "101110000000",--- r7 <= 0
                "101000000010",--- r4 <= 2
                "101100000011",--- r6 <= 3
                "101010000001",--- r5 <= 1
                "011010000000",--- r5 <= -r5
                "001111000000",--- r7 <= r7+r4
                "001101010000",--- r6 <= r6+r5
                "111100000101" --- PC <= 5
                );
```

TB_CODE

```
reset<='1';
 wait for 10ns;
 reset<='0';
```

## OUTPUT



| Name | Value |
|---|---|
| PCVAL[2:0] | 1 |
| Clock | 1 |
| Reset | 0 |
| Overflow | 0 |
| Zero | 0 |
| To_Leds[3:0] | 0 |
| clk_period | 10000 ps |