Datapath Design

Laboratory 1

CS2022 Computer Architecture

Alexandru Sulea
D Stream
#12315152
19 Februrary 2015

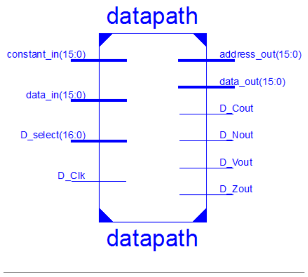
Contents

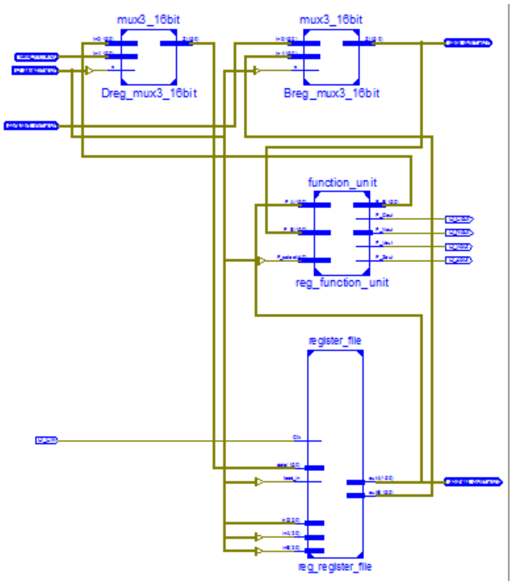
List of Tables

1 CODE

1.1 DATA PATH

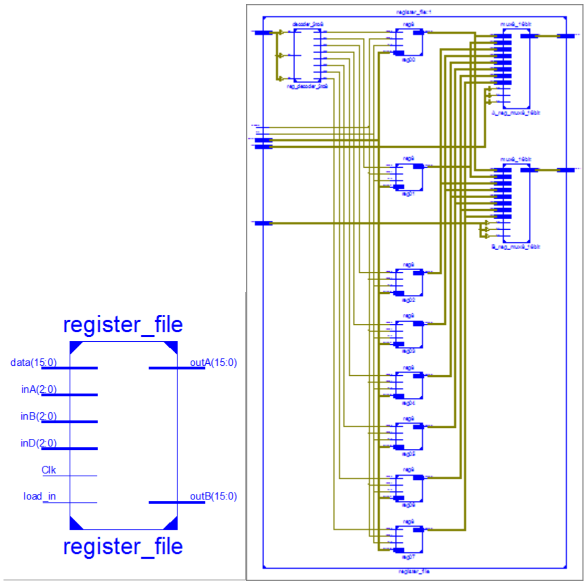
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity datapath is
Port ( data_in : in STD_LOGIC_VECTOR(15 downto 0);
           constant_in : in STD_LOGIC_VECTOR(15 downto 0);
          D_select : in STD_LOGIC_VECTOR(16 downto 0);
          D_Clk : in STD_LOGIC;
          D_Vout : out STD_LOGIC;
          D_Cout : out STD_LOGIC;
          D_Nout : out STD_LOGIC;
          D_Zout : out STD_LOGIC;
          address_out : out STD_LOGIC_VECTOR(15 downto 0);
          data_out : out STD_LOGIC_VECTOR(15 downto 0));

end datapath;

architecture Behavioral of datapath is
component mux3_16bit
    Port ( s : in STD_LOGIC;
          In0 : in STD_LOGIC_VECTOR(15 downto 0);
          In1 : in STD_LOGIC_VECTOR(15 downto 0);
          Z : out STD_LOGIC_VECTOR(15 downto 0));
end Component;

Component function_unit
Port ( F_A : in STD_LOGIC_VECTOR(15 downto 0);
          F_B : in STD_LOGIC_VECTOR(15 downto 0);
          F_select : in STD_LOGIC_VECTOR(4 downto 0);
          F_Vout : out STD_LOGIC;
          F_Cout : out STD_LOGIC;
          F_Nout : out STD_LOGIC;
          F_Zout : out STD_LOGIC;
          F_F : out STD_LOGIC_VECTOR(15 downto 0));
end Component;

Component register_file
    Port ( inA : in STD_LOGIC_VECTOR(2 downto 0);
          inB : in STD_LOGIC_VECTOR(2 downto 0);
          inD : in STD_LOGIC_VECTOR(2 downto 0);
          Clk : in STD_LOGIC;
         load_in : in STD_LOGIC;
          data : in STD_LOGIC_VECTOR(15 downto 0);
           outA : out  STD_LOGIC_VECTOR(15 downto 0);
           outB : out  STD_LOGIC_VECTOR(15 downto 0));
end Component ;

signal src_muxD, src_muxB, src_regA, src_regB, src_out : STD_LOGIC_VECTOR(15 downto 0);

begin
Dreg_mux3_16bit : mux3_16bit PORT MAP ( s => D_select(1),
          In0 => src_out,
          In1 => data_in,
          Z => src_muxD);

Breg_mux3_16bit : mux3_16bit PORT MAP ( s => D_select(7),
          In0 => constant_in,
          In1 => src_regB,
          Z => src_muxB);

reg_function_unit: function_unit PORT MAP ( F_A => src_regA,
          F_B => src_muxB,
          F_select => D_select(6 downto 2),
          F_Vout => D_Vout,
          F_Cout => D_Cout,
          F_Nout => D_Nout,
          F_Zout => D_Zout,
          F_F => src_out);

reg_register_file : register_file PORT MAP( inA => D_select(13 downto 11),
          inB => D_select(10 downto 8),
          inD => D_select(16 downto 14),
          Clk => D_Clk,
         load_in => D_select(0),
          data => src_muxD,
           outA => src_regA,
           outB => src_regB);

data_out <= src_muxB;
address_out <= src_regA;
end Behavioral;
```





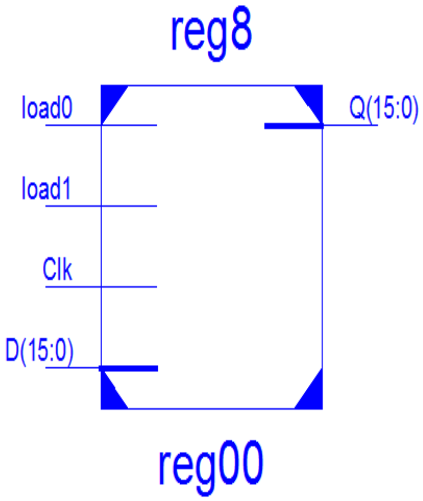
1.2 REGISTER FILE

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


entity register_file is
    Port (
          inA : in STD_LOGIC_VECTOR(2 downto 0);
          inB : in STD_LOGIC_VECTOR(2 downto 0);
          inD : in STD_LOGIC_VECTOR(2 downto 0);
          Clk : in STD_LOGIC;
         load_in : in STD_LOGIC;
          data : in STD_LOGIC_VECTOR(15 downto 0);
           outA : out  STD_LOGIC_VECTOR(15 downto 0);
           outB : out  STD_LOGIC_VECTOR(15 downto 0)
         );
end register_file;

architecture Behavioral of register_file is

   Component decoder_3to8
   Port ( A0 : in STD_LOGIC;
         A1 : in STD_LOGIC;
         A2 : in STD_LOGIC;
         Q0 : out STD_LOGIC;
         Q1 : out STD_LOGIC;
         Q2 : out STD_LOGIC;
         Q3 : out STD_LOGIC;
         Q4 : out STD_LOGIC;
         Q5 : out STD_LOGIC;
         Q6 : out STD_LOGIC;
         Q7 : out STD_LOGIC);
   End Component;

   Component mux3_16bit
   Port ( s : in STD_LOGIC;
         In0 : in STD_LOGIC_VECTOR(15 downto 0);
         In1 : in STD_LOGIC_VECTOR(15 downto 0);
         Z : out STD_LOGIC_VECTOR(15 downto 0));
   End Component;

   Component reg8
   Port ( load0 : in STD_LOGIC;
           load1 : in STD_LOGIC;
         Clk : in STD_LOGIC;
         D : in STD_LOGIC_VECTOR(15 downto 0);
         Q : out STD_LOGIC_VECTOR(15 downto 0));
   End Component;

   Component mux8_16bit
   Port ( S0 : in STD_LOGIC;
          S1 : in STD_LOGIC;
          S2 : in STD_LOGIC;
          In0 : in STD_LOGIC_VECTOR(15 downto 0);
          In1 : in STD_LOGIC_VECTOR(15 downto 0);
          In2 : in STD_LOGIC_VECTOR(15 downto 0);
          In3 : in STD_LOGIC_VECTOR(15 downto 0);
          In4 : in STD_LOGIC_VECTOR(15 downto 0);
          In5 : in STD_LOGIC_VECTOR(15 downto 0);
          In6 : in STD_LOGIC_VECTOR(15 downto 0);
          In7 : in STD_LOGIC_VECTOR(15 downto 0);
          Z : out STD_LOGIC_VECTOR(15 downto 0));
   End Component;

   signal load_reg0, load_reg1, load_reg2, load_reg3, load_reg4,
           load_reg5, load_reg6, load_reg7 : STD_LOGIC;
   signal reg0_q, reg1_q, reg2_q, reg3_q, reg4_q, reg5_q, reg6_q,
           reg7_q, d_mux, src_reg, src_A, src_B : STD_LOGIC_VECTOR(15 downto 0);

begin
   reg_decoder_3to8 : decoder_3to8 PORT MAP(
      A0 => inD(0),
      A1 => inD(1),
      A2 => inD(2),
      Q0 => load_reg0,
      Q1 => load_reg1,
      Q2 => load_reg2,
      Q3 => load_reg3,
      Q4 => load_reg4,
      Q5 => load_reg5,
      Q6 => load_reg6,
      Q7 => load_reg7
      );

   A_reg_mux8_16bit : mux8_16bit PORT MAP(
      S0 => inA(0),
      S1 => inA(1),
      S2 => inA(2),
      In0 => reg0_q,
      In1 => reg1_q,
      In2 => reg2_q,
      In3 => reg3_q,
      In4 => reg4_q,
      In5 => reg5_q,
      In6 => reg6_q,
      In7 => reg7_q,
      Z => src_A
      );

   B_reg_mux8_16bit : mux8_16bit PORT MAP(
      S0 => inB(0),
      S1 => inB(1),
      S2 => inB(2),
      In0 => reg0_q,
      In1 => reg1_q,
      In2 => reg2_q,
      In3 => reg3_q,
      In4 => reg4_q,
      In5 => reg5_q,
      In6 => reg6_q,
      In7 => reg7_q,
      Z => src_B
      );

   reg00 : reg8 PORT MAP(
      load0 => load_reg0,
      load1 => load_in,
      Clk =>  Clk,
      D =>  data,
      Q => reg0_q);

   reg01 : reg8 PORT MAP(
      load0 => load_reg1,
      load1 => load_in,
      Clk =>  Clk,
      D =>  data,
      Q => reg1_q);

   reg02 : reg8 PORT MAP(
      load0 => load_reg2,
      load1 => load_in,
      Clk =>  Clk,
      D =>  data,
      Q => reg2_q);

   reg03 : reg8 PORT MAP(
      load0 => load_reg3,
      load1 => load_in,
      Clk =>  Clk,
      D =>  data,
      Q => reg3_q);

   reg04 : reg8 PORT MAP(
      load0 => load_reg4,
      load1 => load_in,
      Clk =>  Clk,
      D =>  data,
      Q => reg4_q);

   reg05 : reg8 PORT MAP(
      load0 => load_reg5,
      load1 => load_in,
      Clk =>  Clk,
      D =>  data,
      Q => reg5_q);

   reg06 : reg8 PORT MAP(
      load0 => load_reg6,
      load1 => load_in,
      Clk =>  Clk,
      D =>  data,
      Q => reg6_q);

   reg07 : reg8 PORT MAP(
      load0 => load_reg7,
      load1 => load_in,
      Clk =>  Clk,
      D => data,
      Q => reg7_q);

   outA <= src_A;
   outB <= src_B;


 end Behavioral;
```



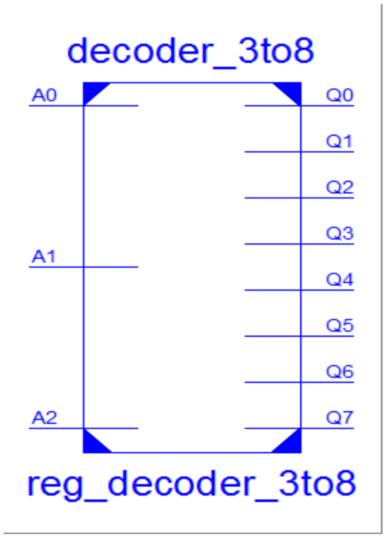
1.2.1 REG8

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity reg8 is
    Port ( load0 : in STD_LOGIC;
           load1 : in STD_LOGIC;
           Clk : in STD_LOGIC;
           D : in STD_LOGIC_VECTOR(15 downto 0);
           Q : out STD_LOGIC_VECTOR(15 downto 0));
end reg8;

architecture Behavioral of reg8 is

begin
process(Clk)
begin
      if(rising_edge(Clk)) then
         if(load0 ='1') and (load1 ='1') then
            Q<= D after 5ns;
         end if;
      end if;
   end process;
end Behavioral;
```



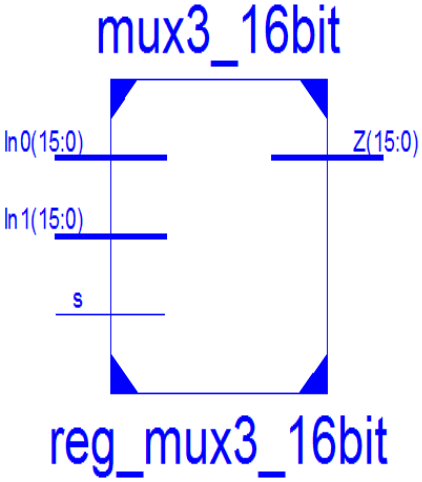
1.2.2 DECODER

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


entity decoder_3to8 is
    Port ( A0 : in STD_LOGIC;
           A1 : in STD_LOGIC;
           A2 : in STD_LOGIC;
           Q0 : out STD_LOGIC;
           Q1 : out STD_LOGIC;
           Q2 : out STD_LOGIC;
           Q3 : out STD_LOGIC;
           Q4 : out STD_LOGIC;
           Q5 : out STD_LOGIC;
           Q6 : out STD_LOGIC;
           Q7 : out STD_LOGIC);
end decoder_3to8;

architecture Behavioral of decoder_3to8 is

begin
   Q0 <= (( NOT A0) AND (NOT A1) AND (NOT A2)) AFTER 5ns;
   Q1 <= (( NOT A0) AND (NOT A1) AND A2) AFTER 5ns;
   Q2 <= (( NOT A0) AND A1 AND (NOT A2)) AFTER 5ns;
   Q3 <= (( NOT A0) AND A1 AND A2) AFTER 5ns;
   Q4 <= (A0 AND (NOT A1) AND (NOT A2)) AFTER 5ns;
   Q5 <= (A0 AND (NOT A1) AND A2) AFTER 5ns;
   Q6 <= (A0 AND A1 AND (NOT A2)) AFTER 5ns;
   Q7 <= (A0 AND A1 AND A2) AFTER 5ns;

end Behavioral;
```



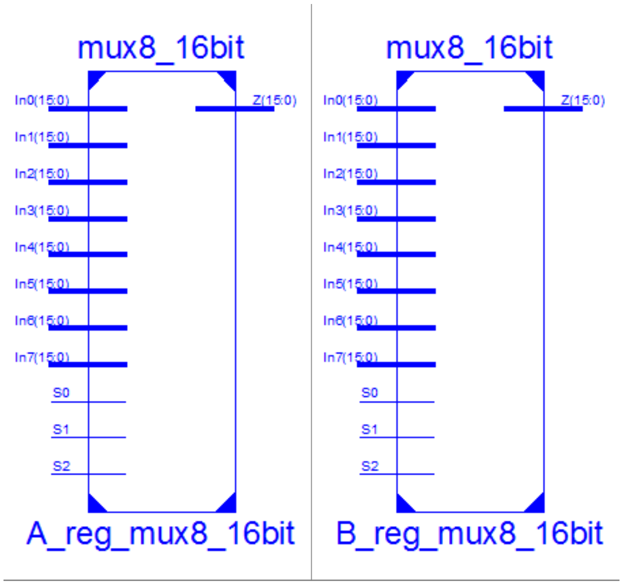
1.2.3 MUX3 TO 16BIT

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


entity mux3_16bit is
    Port ( s : in STD_LOGIC;
           In0 : in STD_LOGIC_VECTOR(15 downto 0);
           In1 : in STD_LOGIC_VECTOR(15 downto 0);
           Z : out STD_LOGIC_VECTOR(15 downto 0));
end mux3_16bit;

architecture Behavioral of mux3_16bit is

begin
   Z <= In0 after 5ns when s = '0' else
         In1 after 5ns when s = '1' else
         x"0000" after 5ns;

end Behavioral;
```



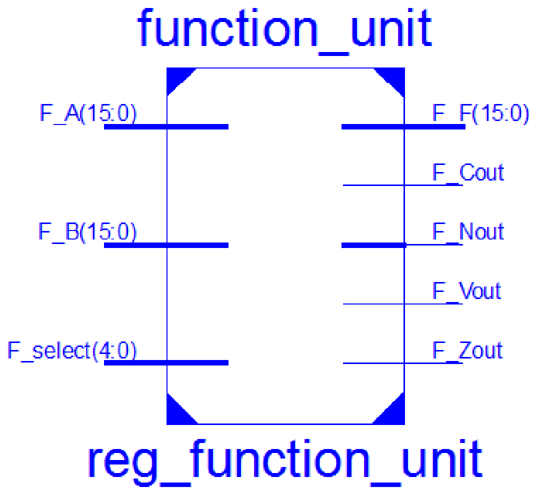
1.2.4 MUX8 TO 16BIT

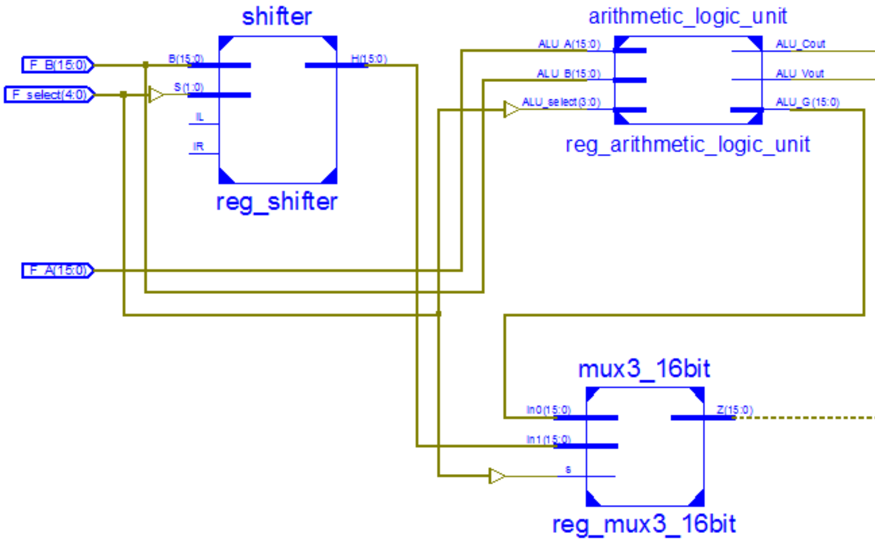
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity mux8_16bit is
   Port ( S0 : in STD_LOGIC;
          S1 : in STD_LOGIC;
          S2 : in STD_LOGIC;
          In0 : in STD_LOGIC_VECTOR(15 downto 0);
          In1 : in STD_LOGIC_VECTOR(15 downto 0);
          In2 : in STD_LOGIC_VECTOR(15 downto 0);
          In3 : in STD_LOGIC_VECTOR(15 downto 0);
          In4 : in STD_LOGIC_VECTOR(15 downto 0);
          In5 : in STD_LOGIC_VECTOR(15 downto 0);
          In6 : in STD_LOGIC_VECTOR(15 downto 0);
          In7 : in STD_LOGIC_VECTOR(15 downto 0);
          Z : out STD_LOGIC_VECTOR(15 downto 0));
end mux8_16bit;

architecture Behavioral of mux8_16bit is

begin
  Z <= In0 after 5ns when S0 = '0' and S1 = '0' and S2 = '0' else
        In1 after 5ns when S0 = '0' and S1 = '0' and S2 = '1' else
        In2 after 5ns when S0 = '0' and S1 = '1' and S2 = '0' else
        In3 after 5ns when S0 = '0' and S1 = '1' and S2 = '1' else
        In4 after 5ns when S0 = '1' and S1 = '0' and S2 = '0' else
        In5 after 5ns when S0 = '1' and S1 = '0' and S2 = '1' else
        In6 after 5ns when S0 = '1' and S1 = '1' and S2 = '0' else
        In7 after 5ns when S0 = '1' and S1 = '1' and S2 = '1' else
        x"0000" after 5ns;
end Behavioral;
```


mux8_16bit
In0(15:0)
In1(15:0)
In2(15:0)
In3(15:0)
In4(15:0)
In5(15:0)
In6(15:0)
In7(15:0)
S0
S1
S2
Z(15:0)
A_reg_mux8_16bit
mux8_16bit
In0(15:0)
In1(15:0)
In2(15:0)
In3(15:0)
In4(15:0)
In5(15:0)
In6(15:0)
In7(15:0)
S0
S1
S2
Z(15:0)
B_reg_mux8_16bit

1.3 FUNCTION UNIT

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


entity function_unit is
Port ( F_A : in STD_LOGIC_VECTOR(15 downto 0);
          F_B : in STD_LOGIC_VECTOR(15 downto 0);
          F_select : in STD_LOGIC_VECTOR(4 downto 0);
          F_Vout : out STD_LOGIC;
          F_Cout : out STD_LOGIC;
          F_Nout : out STD_LOGIC;
          F_Zout : out STD_LOGIC;
          F_F : out STD_LOGIC_VECTOR(15 downto 0));
end function_unit;

architecture Behavioral of function_unit is

Component arithmetic_logic_unit
   Port ( ALU_A : in STD_LOGIC_VECTOR(15 downto 0);
          ALU_B : in STD_LOGIC_VECTOR(15 downto 0);
          ALU_select : in STD_LOGIC_VECTOR(3 downto 0);
          ALU_Cout : out STD_LOGIC;
          ALU_Vout : out STD_LOGIC;
          ALU_G : out STD_LOGIC_VECTOR(15 downto 0));
end Component;


Component shifter Port ( B : in STD_LOGIC_VECTOR(15 downto 0);
          S : in STD_LOGIC_VECTOR(1 downto 0);
          IR : in STD_LOGIC;
          IL : in STD_LOGIC;
          H : out STD_LOGIC_VECTOR(15 downto 0));
end Component;

Component mux3_16bit Port ( s : in STD_LOGIC;
          In0 : in STD_LOGIC_VECTOR(15 downto 0);
          In1 : in STD_LOGIC_VECTOR(15 downto 0);
          Z : out STD_LOGIC_VECTOR(15 downto 0));
end Component;

signal src_ALU, src_shift, src_mux : STD_LOGIC_VECTOR(15 downto 0);

begin

reg_arithmetic_logic_unit : arithmetic_logic_unit
   Port MAP ( ALU_A => F_A,
          ALU_B => F_B,
          ALU_select => F_select(3 downto 0),
          ALU_Cout => F_Cout,
          ALU_Vout => F_Vout,
          ALU_G => src_ALU);

reg_shifter : shifter Port MAP ( B => F_B,
          S => F_select(3 downto 2),
          IR => '0',
          IL => '0',
          H => src_shift);


reg_mux3_16bit : mux3_16bit Port MAP ( s => F_select(4),
          In0 => src_ALU,
          In1 => src_shift,
          Z  => src_mux);

F_F <= src_mux;
F_Nout <= src_mux(15);
F_Zout <= ((src_mux(15)) or (src_mux(14)) or (src_mux(13)) or
           (src_mux(12)) or (src_mux(11)) or (src_mux(10)) or
           (src_mux(9)) or (src_mux(8)) or (src_mux(7)) or
           (src_mux(6)) or (src_mux(5)) or (src_mux(4)) or
           (src_mux(3)) or (src_mux(2)) or
           (src_mux(1)) or (src_mux(0)));



end Behavioral;
```



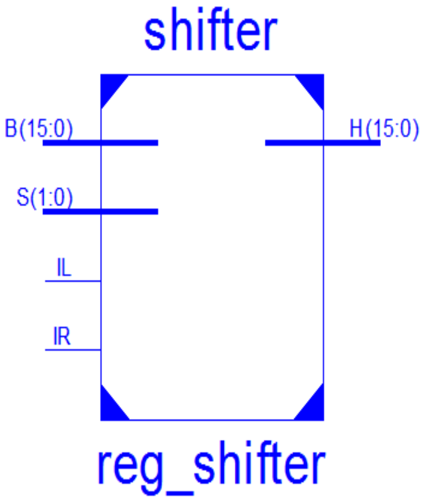


1.4 SHIFTER

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


entity shifter is
   Port ( B : in STD_LOGIC_VECTOR(15 downto 0);
          S : in STD_LOGIC_VECTOR(1 downto 0);
          IR : in STD_LOGIC;
          IL : in STD_LOGIC;
          H : out STD_LOGIC_VECTOR(15 downto 0));
end shifter;

architecture Behavioral of shifter is

component mux3_1bit Port ( In0 : in STD_LOGIC;
          In1 : in STD_LOGIC;
          In2 : in STD_LOGIC;
          S0 : in STD_LOGIC;
          S1 : in STD_LOGIC;
          Z : out STD_LOGIC);
end component;

begin



mux3_1bit00 : mux3_1bit PORT MAP ( In0 =>B(0) ,
          In1 => B(1),
          In2 => IL,
          S0 => S(0),
          S1 => S(1),
          Z  => H(0));

mux3_1bit01 : mux3_1bit PORT MAP ( In0 =>B(1) ,
          In1 => B(2),
          In2 => B(0),
          S0 => S(0),
          S1 => S(1),
          Z  => H(1));

mux3_1bit02 : mux3_1bit PORT MAP ( In0 =>B(2) ,
          In1 => B(3),
          In2 => B(1),
          S0 => S(0),
          S1 => S(1),
          Z  => H(2));

mux3_1bit03 : mux3_1bit PORT MAP ( In0 =>B(3) ,
          In1 => B(4),
          In2 => B(2),
          S0 => S(0),
          S1 => S(1),
          Z  => H(3));
```

```
mux3_1bit04 : mux3_1bit PORT MAP ( In0 =>B(4) ,
          In1 => B(5),
          In2 => B(3),
          S0 => S(0),
          S1 => S(1),
          Z  => H(4));

mux3_1bit05 : mux3_1bit PORT MAP ( In0 =>B(5) ,
          In1 => B(6),
          In2 => B(4),
          S0 => S(0),
          S1 => S(1),
          Z  => H(5));

mux3_1bit06 : mux3_1bit PORT MAP ( In0 =>B(6) ,
          In1 => B(7),
          In2 => B(5),
          S0 => S(0),
          S1 => S(1),
          Z  => H(6));

mux3_1bit07 : mux3_1bit PORT MAP ( In0 =>B(7) ,
          In1 => B(8),
          In2 => B(6),
          S0 => S(0),
          S1 => S(1),
          Z  => H(7));

mux3_1bit08 : mux3_1bit PORT MAP ( In0 =>B(8) ,
          In1 => B(9),
          In2 => B(7),
          S0 => S(0),
          S1 => S(1),
          Z  => H(8));

mux3_1bit09 : mux3_1bit PORT MAP ( In0 =>B(9) ,
          In1 => B(10),
          In2 => B(8),
          S0 => S(0),
          S1 => S(1),
          Z  => H(9));

mux3_1bit10 : mux3_1bit PORT MAP ( In0 =>B(10) ,
          In1 => B(11),
          In2 => B(9),
          S0 => S(0),
          S1 => S(1),
          Z  => H(10));

mux3_1bit11 : mux3_1bit PORT MAP ( In0 =>B(11) ,
          In1 => B(12),
          In2 => B(10),
          S0 => S(0),
          S1 => S(1),
          Z  => H(11));
```

```
mux3_1bit12 : mux3_1bit PORT MAP ( In0 =>B(12) ,
          In1 => B(13),
          In2 => B(11),
          S0 => S(0),
          S1 => S(1),
          Z  => H(12));

mux3_1bit13 : mux3_1bit PORT MAP ( In0 =>B(13) ,
          In1 => B(14),
          In2 => B(12),
          S0 => S(0),
          S1 => S(1),
          Z  => H(13));

mux3_1bit14 : mux3_1bit PORT MAP ( In0 =>B(14) ,
          In1 => B(15),
          In2 => B(13),
          S0 => S(0),
          S1 => S(1),
          Z  => H(14));

mux3_1bit15 : mux3_1bit PORT MAP ( In0 =>B(15) ,
          In1 => IR,
          In2 => B(14),
          S0 => S(0),
          S1 => S(1),
          Z  => H(15));
end Behavioral;
```



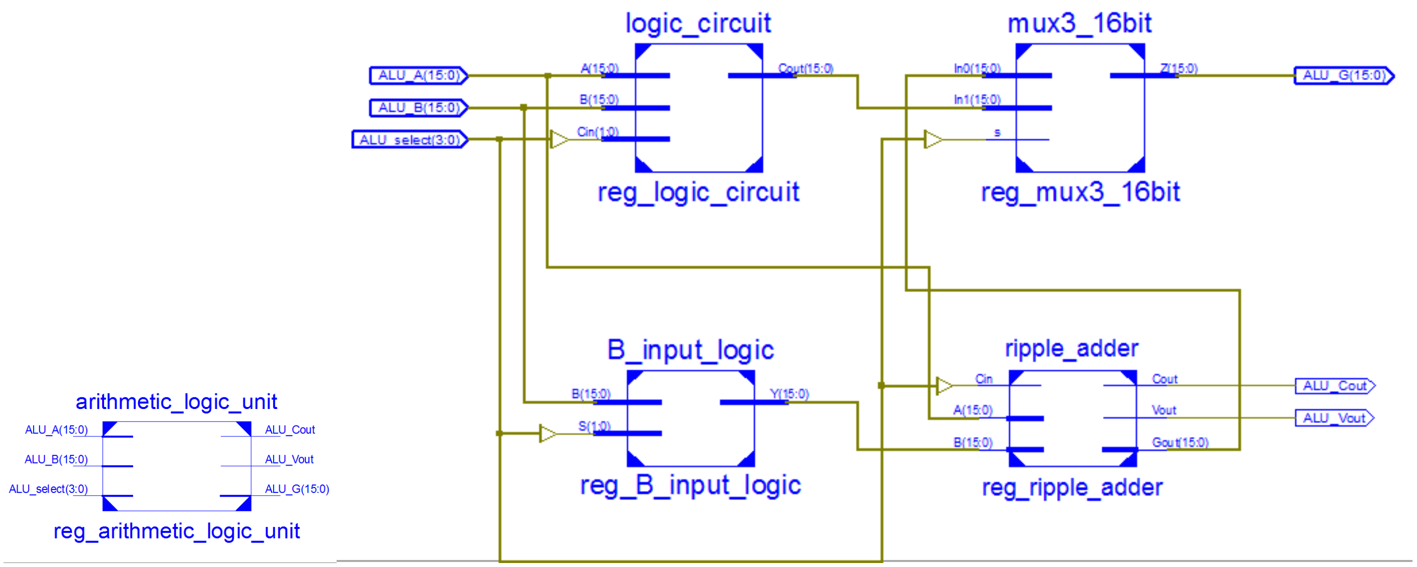
1.4.1 MUX3 TO 1BIT

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


entity mux3_1bit is
   Port ( In0 : in STD_LOGIC;
          In1 : in STD_LOGIC;
          In2 : in STD_LOGIC;
          S0 : in STD_LOGIC;
          S1 : in STD_LOGIC;
          Z : out STD_LOGIC);
end mux3_1bit;

architecture Behavioral of mux3_1bit is

begin
Z <= In0 after 1ns when S0 = '0' and S1='0' else
     In1 after 1ns when S0 ='0' and S1= '1' else
     In2 after 1ns when S0 ='0' and S1 = '0' else
     '0' after 1ns;

end Behavioral;
```

1.5 ALU

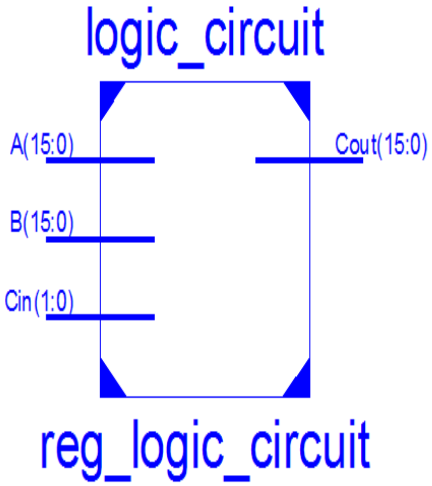
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity arithmetic_logic_unit is
    Port ( ALU_A : in STD_LOGIC_VECTOR(15 downto 0);
          ALU_B : in STD_LOGIC_VECTOR(15 downto 0);
          ALU_select : in STD_LOGIC_VECTOR(3 downto 0);
          ALU_Cout : out STD_LOGIC;
          ALU_Vout : out STD_LOGIC;
          ALU_G : out STD_LOGIC_VECTOR(15 downto 0));
end arithmetic_logic_unit;

architecture Behavioral of arithmetic_logic_unit is

component ripple_adder Port ( A : in STD_LOGIC_VECTOR(15 downto 0);
          B : in STD_LOGIC_VECTOR(15 downto 0);
          Cin : in STD_LOGIC;
          Cout : out STD_LOGIC;
          Gout : out STD_LOGIC_VECTOR(15 downto 0);
          Vout : out STD_LOGIC);
End Component;

component mux3_16bit Port ( s : in STD_LOGIC;
          In0 : in STD_LOGIC_VECTOR(15 downto 0);
          In1 : in STD_LOGIC_VECTOR(15 downto 0);
          Z : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

component B_input_logic Port ( B : in STD_LOGIC_VECTOR(15 downto 0);
          S : in STD_LOGIC_VECTOR(1 downto 0);
          Y : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

Component logic_circuit Port ( A : in STD_LOGIC_VECTOR(15 downto 0);
          B : in STD_LOGIC_VECTOR(15 downto 0);
          Cin : in STD_LOGIC_VECTOR(1 downto 0);
          Cout : out STD_LOGIC_VECTOR(15 downto 0));
End Component;

signal src_logic , src_B_input_logic, src_ripple : STD_LOGIC_VECTOR(15 downto 0);

begin


reg_ripple_adder : ripple_adder PORT MAP ( A => ALU_A,
          B => src_logic,
          Cin => ALU_select(0),
          Cout => ALU_Cout,
          Gout => src_ripple,
          Vout => ALU_Vout);
```

```
reg_mux3_16bit :mux3_16bit PORT MAP ( s => ALU_select(3),
          In0 => src_ripple,
          In1 => src_B_input_logic,
          Z =>ALU_G);


reg_B_input_logic : B_input_logic PORT MAP ( B => ALU_B,
          S => ALU_SELECT(2 downto 1),
          Y => src_logic);

reg_logic_circuit : logic_circuit PORT MAP( A => ALU_A ,
          B => ALU_B,
          Cin => ALU_select(2 downto 1),
          Cout => src_B_input_logic);



end Behavioral;
```



LOGIC

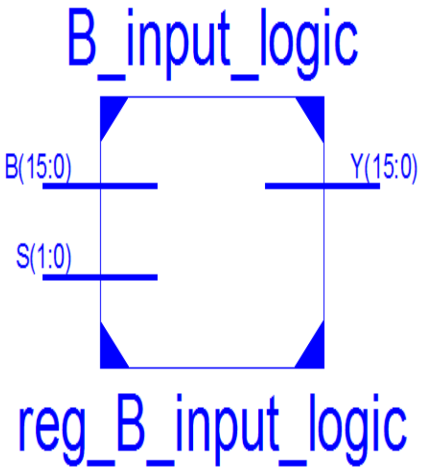
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity logic_circuit is
    Port ( A : in STD_LOGIC_VECTOR(15 downto 0);
           B : in STD_LOGIC_VECTOR(15 downto 0);
           Cin : in STD_LOGIC_VECTOR(1 downto 0);
           Cout : out STD_LOGIC_VECTOR(15 downto 0));
end logic_circuit;

architecture Behavioral of logic_circuit is

begin
Cout <= (A and B) after 1ns when Cin = "00" else
        (A or B) after 1ns when Cin = "01" else
        (A xor B) after 1ns when Cin = "10" else
        (not (A)) after 1ns;
end Behavioral;
```



1.5.1 B LOGIC

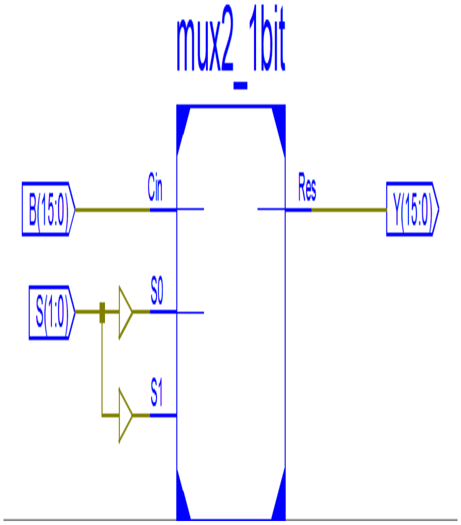
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity B_input_logic is
    Port ( B : in STD_LOGIC_VECTOR(15 downto 0);
           S : in STD_LOGIC_VECTOR(1 downto 0);
           Y : out STD_LOGIC_VECTOR(15 downto 0));
end B_input_logic;

architecture Behavioral of B_input_logic is

component mux2_1bit Port ( S0 : in STD_LOGIC;
                           S1 : in STD_LOGIC;
                           Cin : in STD_LOGIC;
                           Res : out STD_LOGIC);
End Component;

begin
mux2_1bit00 : mux2_1bit PORT MAP( S0 => S(0),
                                  S1 => S(1),
                                  Cin =>B(0),
                                  Res => Y(0));

mux2_1bit01 : mux2_1bit PORT MAP( S0 => S(0),
                                  S1 => S(1),
                                  Cin =>B(1),
                                  Res => Y(1));

mux2_1bit02 : mux2_1bit PORT MAP( S0 => S(0),
                                  S1 => S(1),
                                  Cin =>B(2),
                                  Res => Y(2));

mux2_1bit03 : mux2_1bit PORT MAP( S0 => S(0),
                                  S1 => S(1),
                                  Cin =>B(3),
                                  Res => Y(3));

mux2_1bit04 : mux2_1bit PORT MAP( S0 => S(0),
                                  S1 => S(1),
                                  Cin =>B(4),
                                  Res => Y(4));

mux2_1bit05 : mux2_1bit PORT MAP( S0 => S(0),
                                  S1 => S(1),
                                  Cin =>B(5),
                                  Res => Y(5));

mux2_1bit06 : mux2_1bit PORT MAP( S0 => S(0),
                                  S1 => S(1),
                                  Cin =>B(6),
                                  Res => Y(6));
```

```
mux2_1bit07 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(7),
                                Res => Y(7));

mux2_1bit08 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(8),
                                Res => Y(8));

mux2_1bit09 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(9),
                                Res => Y(9));

mux2_1bit10 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(10),
                                Res => Y(10));

mux2_1bit11 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(11),
                                Res => Y(11));

mux2_1bit12 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(12),
                                Res => Y(12));

mux2_1bit13 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(13),
                                Res => Y(13));

mux2_1bit14 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(14),
                                Res => Y(14));

mux2_1bit15 : mux2_1bit PORT MAP( S0 => S(0),
                                S1 => S(1),
                                Cin =>B(15),
                                Res => Y(15));

end Behavioral;
```


B_input_logic
B(15:0)
S(1:0)
Y(15:0)
reg_B_input_logic

1.5.2 MUX2 TO 1BIT

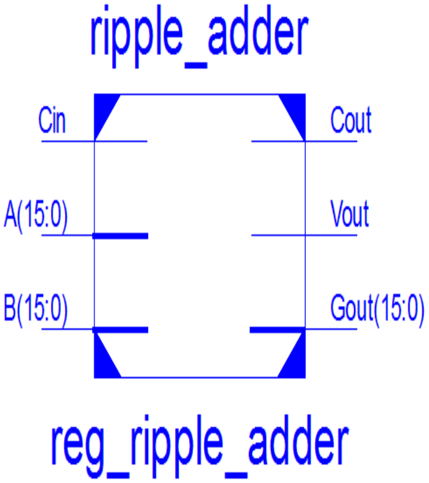
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity mux2_1bit is
    Port ( S0 : in STD_LOGIC;
          S1 : in STD_LOGIC;
          Cin : in STD_LOGIC;
          Res : out STD_LOGIC);
end mux2_1bit;

architecture Behavioral of mux2_1bit is

begin
Res <= S0 after 1ns when Cin = ’1’ else
        S1 after 1ns when Cin = ’1’ else
        ’0’ after 1ns;

end Behavioral;
```



1.5.3 RIPPLE

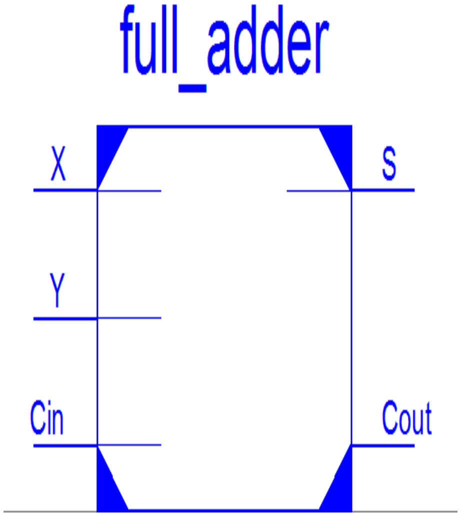
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity ripple_adder is
    Port ( A : in STD_LOGIC_VECTOR(15 downto 0);
           B : in STD_LOGIC_VECTOR(15 downto 0);
           Cin : in STD_LOGIC;
           Cout : out STD_LOGIC;
           Gout : out STD_LOGIC_VECTOR(15 downto 0);
           Vout : out STD_LOGIC);
end ripple_adder;

architecture Behavioral of ripple_adder is

Component full_adder
PORT(X: in STD_LOGIC;
      Y: in STD_LOGIC;
      S: out STD_LOGIC;
      Cin: in STD_LOGIC;
      Cout: out STD_LOGIC);
End Component;

signal src_sig0, src_sig1, src_sig2, src_sig3, src_sig4,
 src_sig5, src_sig6, src_sig7, src_sig8, src_sig9, src_sig10,
 src_sig11, src_sig12, src_sig13, src_sig14, src_sig15,
 src_out: STD_LOGIC;
begin
full_adder00 : full_adder PORT MAP(X => A(0),
                                Y => B(0),
                                S => Gout(0),
                                Cin => Cin,
                                Cout =>src_sig0 );

full_adder01 : full_adder PORT MAP(X => A(1),
                                Y => B(1),
                                S => Gout(1),
                                Cin => Cin,
                                Cout =>src_sig1);

full_adder02 : full_adder PORT MAP(X => A(2),
                                Y => B(2),
                                S => Gout(2),
                                Cin => Cin,
                                Cout =>src_sig2);

full_adder03 : full_adder PORT MAP(X => A(3),
                                Y => B(3),
                                S => Gout(3),
                                Cin => Cin,
                                Cout =>src_sig3);
```

```
full_adder04 : full_adder PORT MAP(X => A(4),
                                 Y => B(4),
                                 S => Gout(4),
                                 Cin => Cin,
                                 Cout =>src_sig4);

full_adder05 : full_adder PORT MAP(X => A(5),
                                 Y => B(5),
                                 S => Gout(5),
                                 Cin => Cin,
                                 Cout =>src_sig5);

full_adder06 : full_adder PORT MAP(X => A(6),
                                 Y => B(6),
                                 S => Gout(6),
                                 Cin => Cin,
                                 Cout =>src_sig6);

full_adder07 : full_adder PORT MAP(X => A(7),
                                 Y => B(7),
                                 S => Gout(7),
                                 Cin => Cin,
                                 Cout =>src_sig7);

full_adder08 : full_adder PORT MAP(X => A(8),
                                 Y => B(8),
                                 S => Gout(8),
                                 Cin => Cin,
                                 Cout =>src_sig8);

full_adder09 : full_adder PORT MAP(X => A(9),
                                 Y => B(9),
                                 S => Gout(9),
                                 Cin => Cin,
                                 Cout =>src_sig9);

full_adder10 : full_adder PORT MAP(X => A(10),
                                 Y => B(10),
                                 S => Gout(10),
                                 Cin => Cin,
                                 Cout =>src_sig10);

full_adder11 : full_adder PORT MAP(X => A(11),
                                 Y => B(11),
                                 S => Gout(11),
                                 Cin => Cin,
                                 Cout =>src_sig11);

full_adder12 : full_adder PORT MAP(X => A(12),
                                 Y => B(12),
                                 S => Gout(12),
                                 Cin => Cin,
                                 Cout =>src_sig12);

full_adder13 : full_adder PORT MAP(X => A(13),
                                 Y => B(13),
                                 S => Gout(13),
                                 Cin => Cin,
                                 Cout =>src_sig13);

full_adder14 : full_adder PORT MAP(X => A(14),
                                 Y => B(14),
                                 S => Gout(14),
                                 Cin => Cin,
                                 Cout =>src_sig14);

full_adder15 : full_adder PORT MAP(X => A(15),
                                 Y => B(15),
                                 S => Gout(15),
                                 Cin => Cin,
                                 Cout =>src_sig15);



end Behavioral;
```



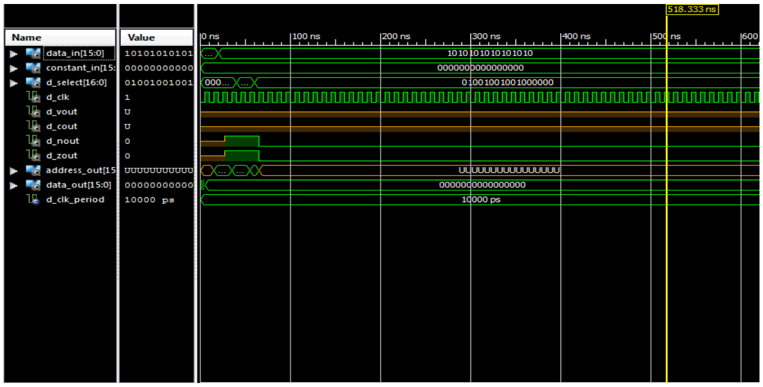
1.5.4 FULL ADDER

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


entity full_adder is
PORT(X: in STD_LOGIC;
     Y: in STD_LOGIC;
     S: out STD_LOGIC;
     Cin: in STD_LOGIC;
     Cout: out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is

   signal S1, S2, S3: STD_LOGIC;
begin
   S1 <= (X xor Y) after 1ns;
   S2 <= (Cin and S1) after 1ns;
   S3 <= (X and Y) after 1ns;
   S <= (S1 xor Cin) after 1ns;
   Cout <= (S2 or S3) after 1ns;

end Behavioral;
```



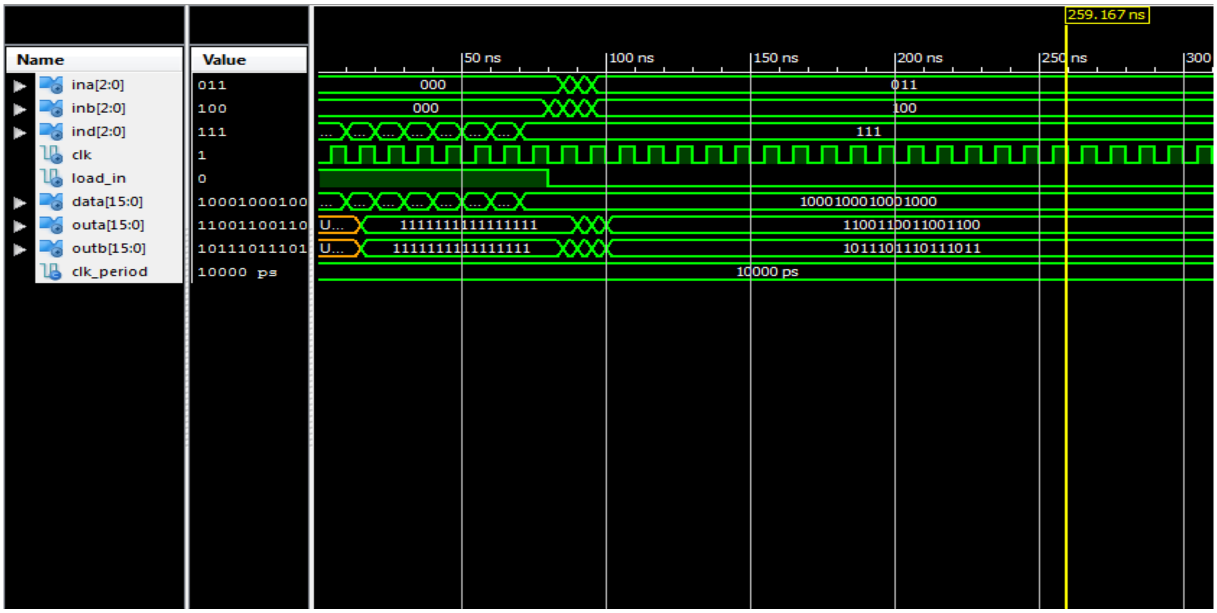
2 TESTBENCHES

2.1 DATA PATH



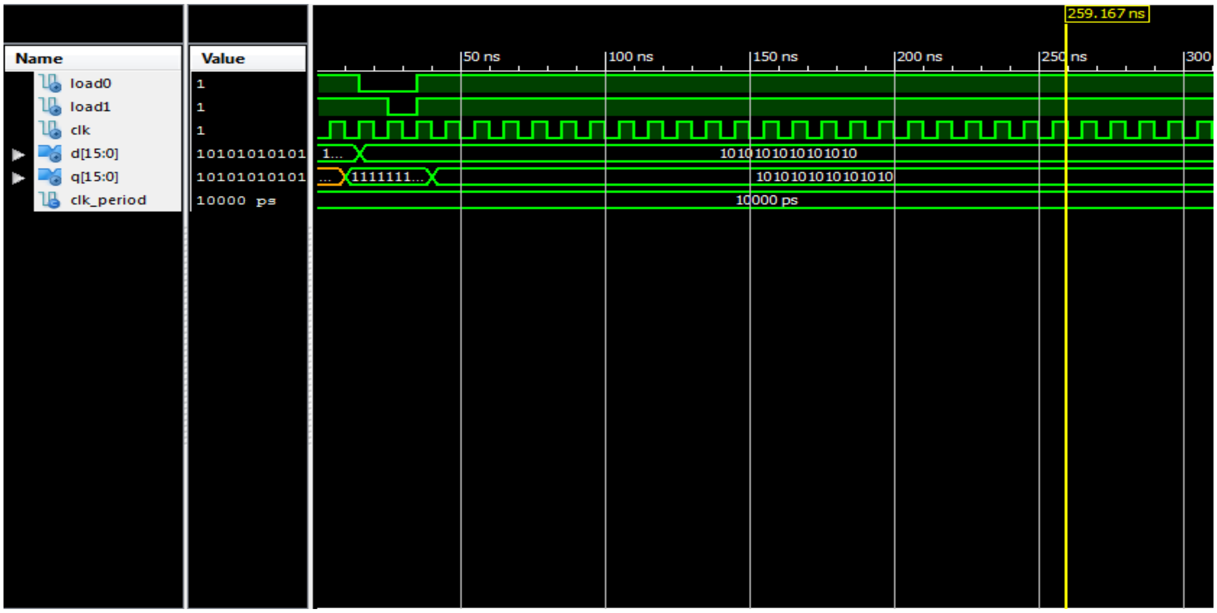
Name	Value
data_in[15:0]	10101010101
constant_in[15:	00000000000
d_select[16:0]	01001001001
d_clk	1
d_vout	U
d_cout	U
d_nout	0
d_zout	0
address_out[15	UUUUUUUUUUU
data_out[15:0]	00000000000
d_clk_period	10000 ps

2.2 REGISTER FILE



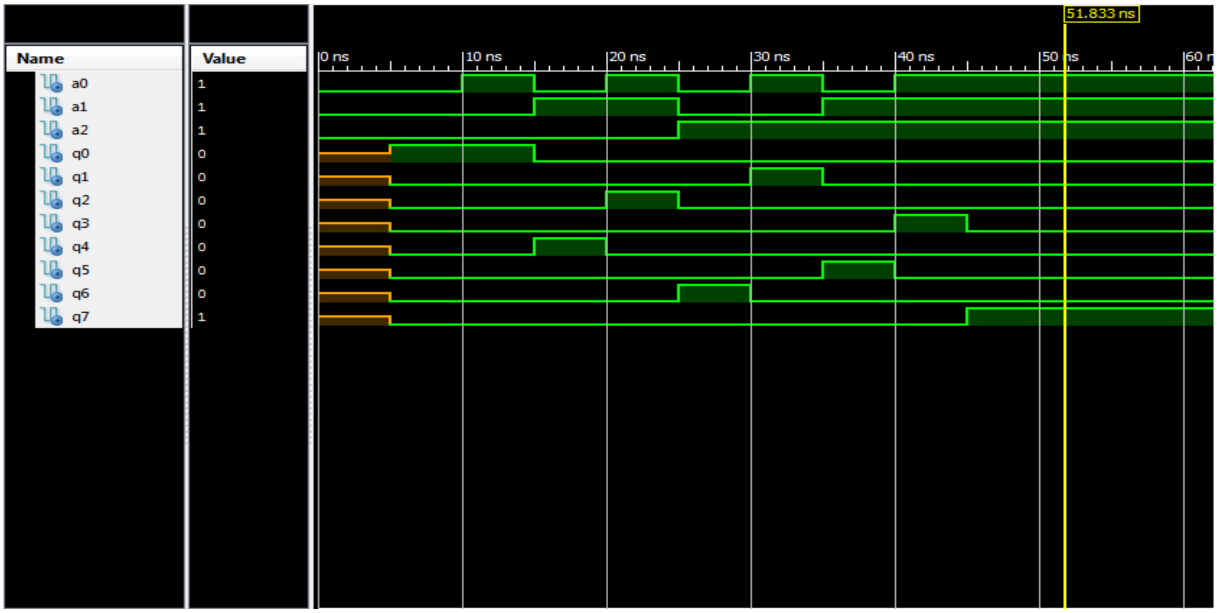
Name	Value
ina[2:0]	011
inb[2:0]	100
ind[2:0]	111
clk	1
load_in	0
data[15:0]	10001000100
outa[15:0]	11001100110
outb[15:0]	10111011101
clk_period	10000 ps

2.2.1 REG8



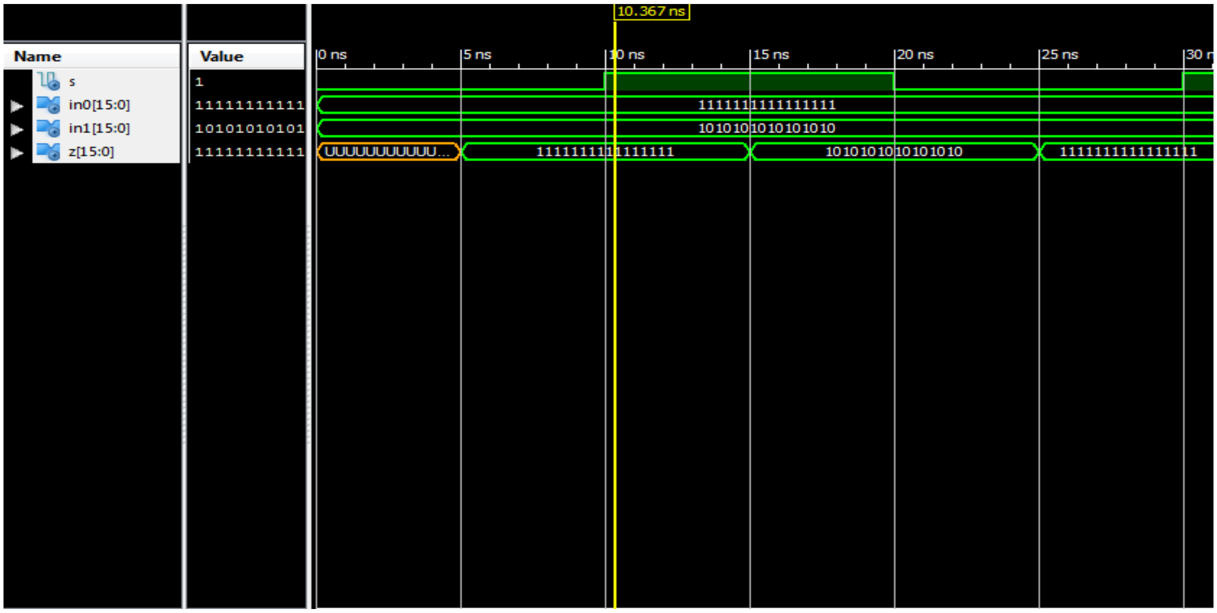
Name	Value
load0	1
load1	1
clk	1
d[15:0]	10101010101
q[15:0]	10101010101
clk_period	10000 ps

2.2.2 DECODER

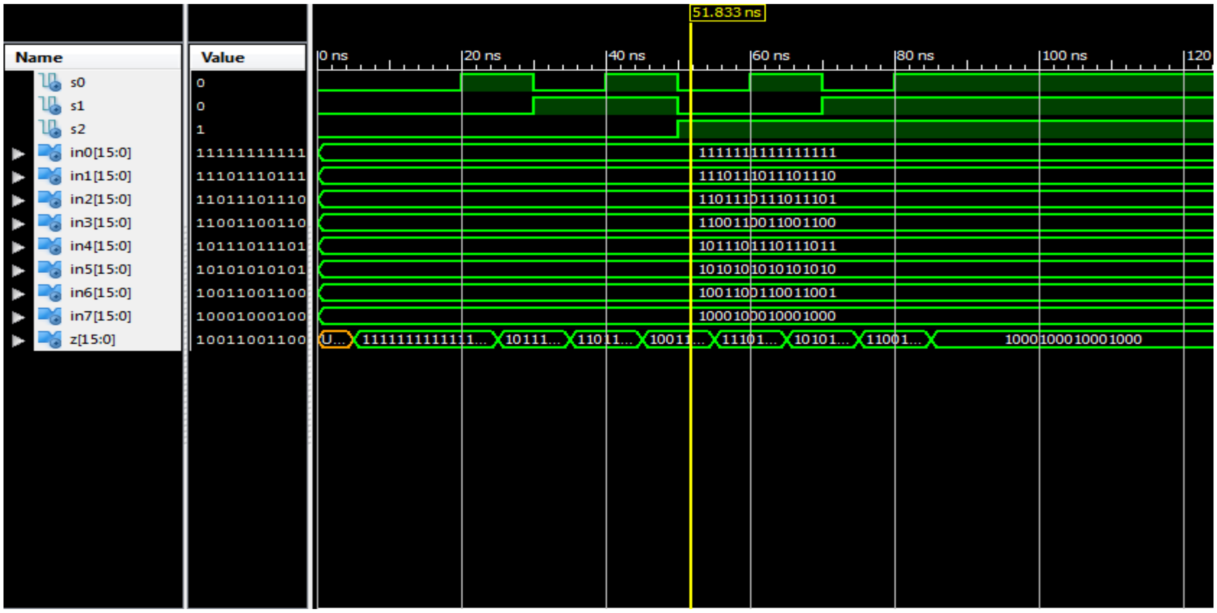


Name	Value
a0	1
a1	1
a2	1
q0	0
q1	0
q2	0
q3	0
q4	0
q5	0
q6	0
q7	1

2.2.3 MUX3 TO 16BIT

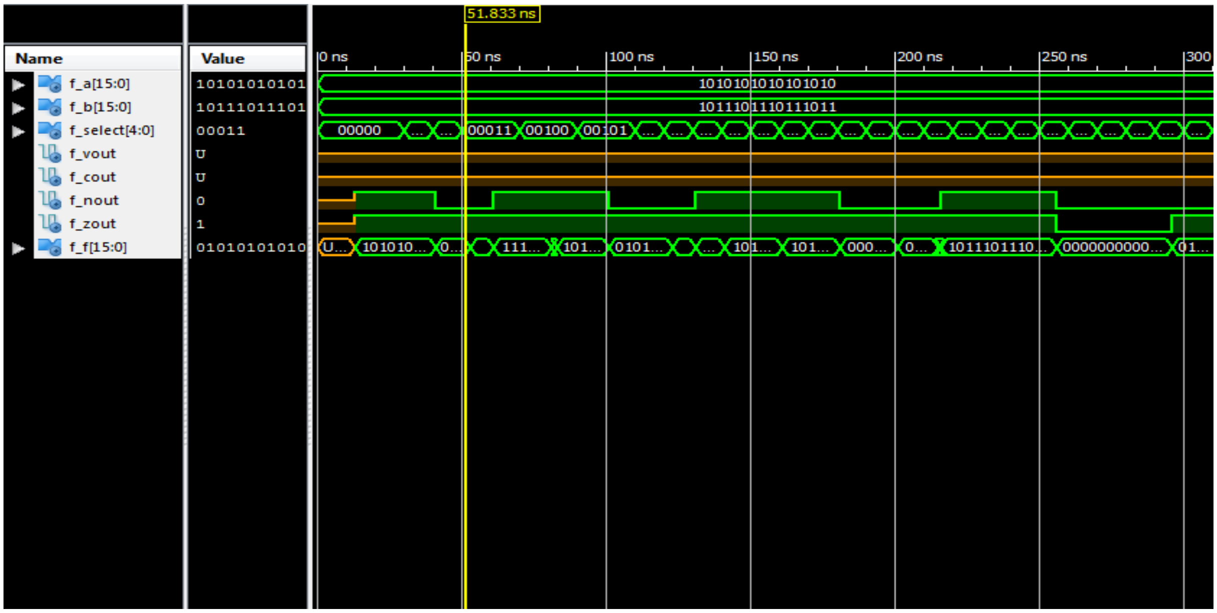


2.2.4 MUX8 TO 16BIT



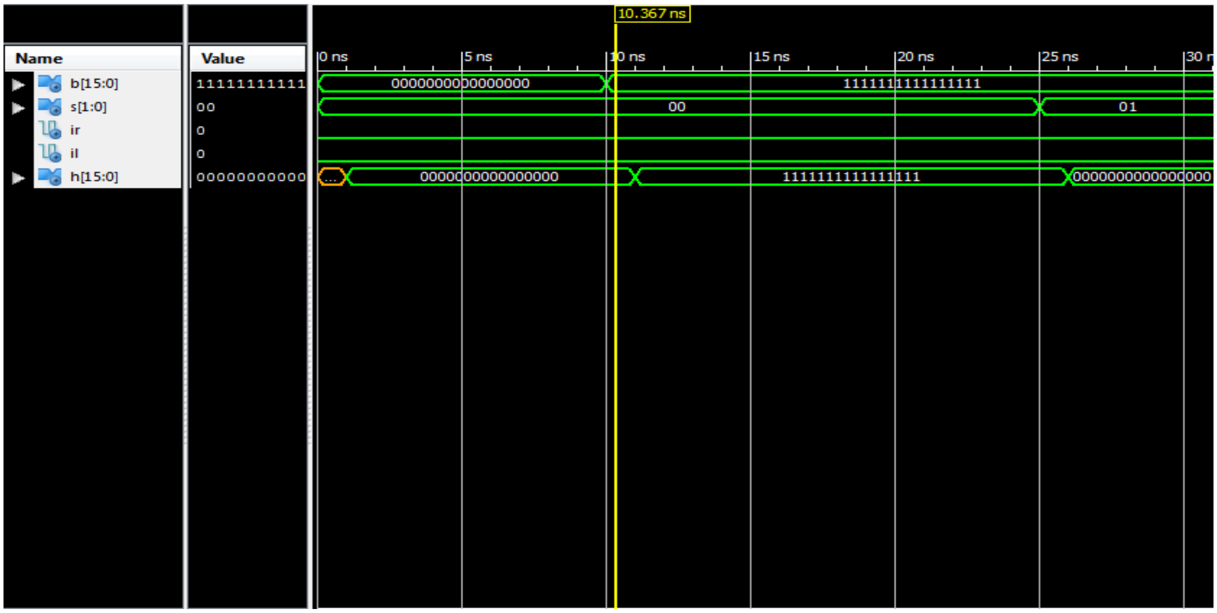
Name	Value
s0	0
s1	0
s2	1
in0[15:0]	11111111111
in1[15:0]	11101110111
in2[15:0]	11011101110
in3[15:0]	11001100110
in4[15:0]	10111011101
in5[15:0]	10101010101
in6[15:0]	10011001100
in7[15:0]	10001000100
z[15:0]	10011001100

2.3 FUNCTION UNIT

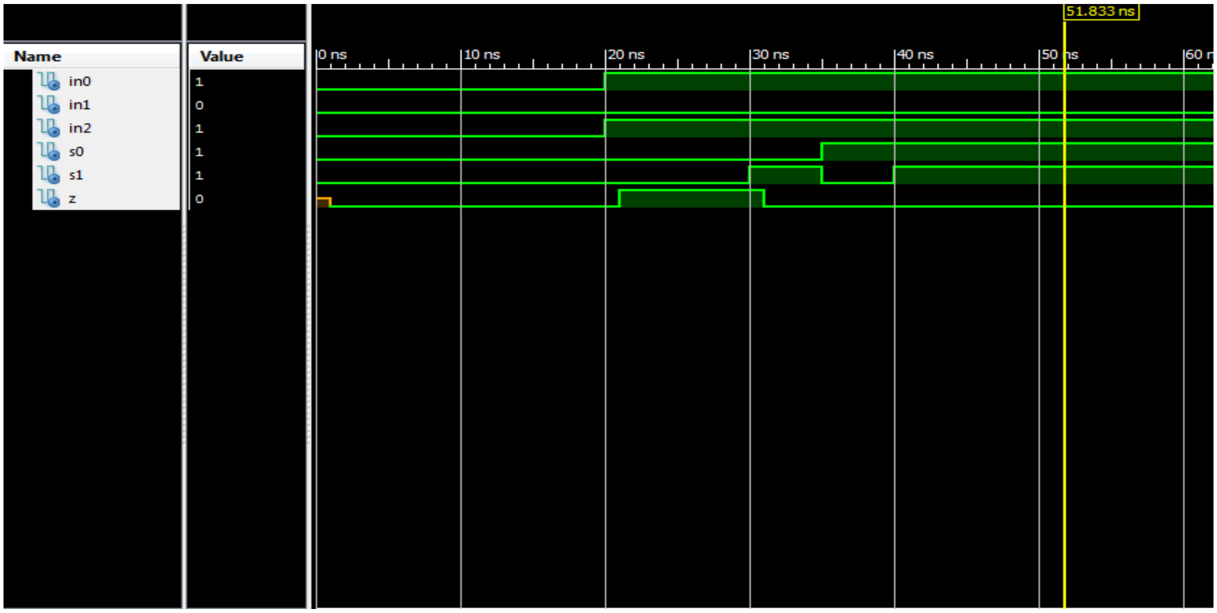


Name	Value
f_a[15:0]	10101010101
f_b[15:0]	10111011101
f_select[4:0]	00011
f_vout	U
f_cout	U
f_nout	0
f_zout	1
f_f[15:0]	01010101010

2.4 SHIFTER

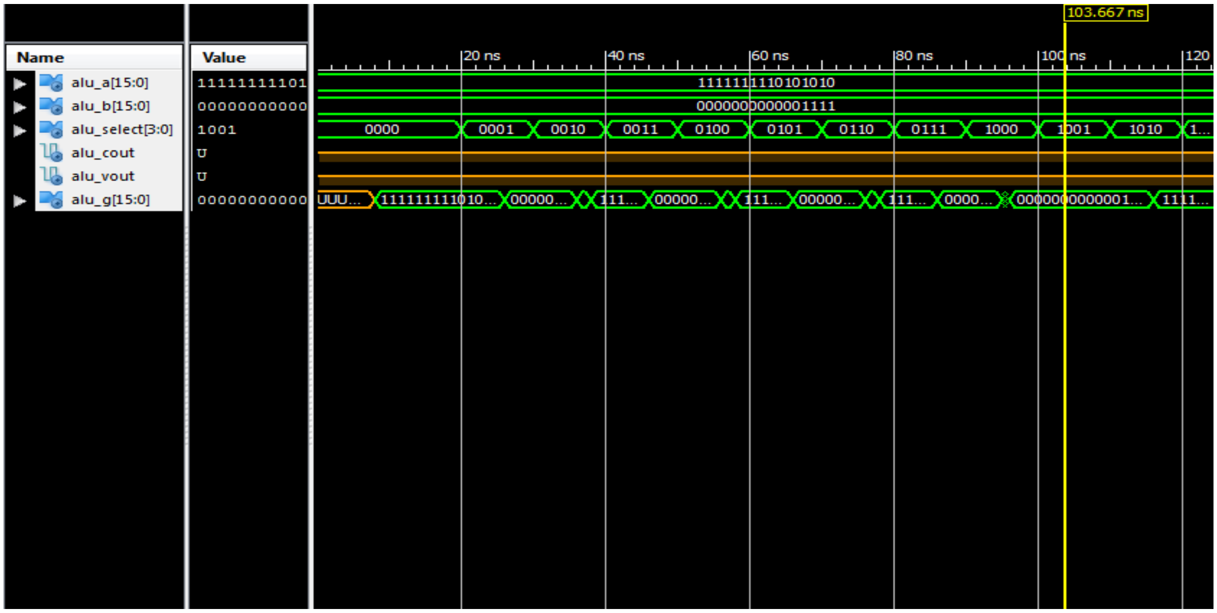

10.367 ns
Name
Value
b[15:0]
11111111111
s[1:0]
00
ir
0
il
0
h[15:0]
00000000000
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 n
0000000000000000
1111111111111111
00
01
...
0000000000000000
1111111111111111
0000000000000000

2.4.1 MUX3 TO 1BIT



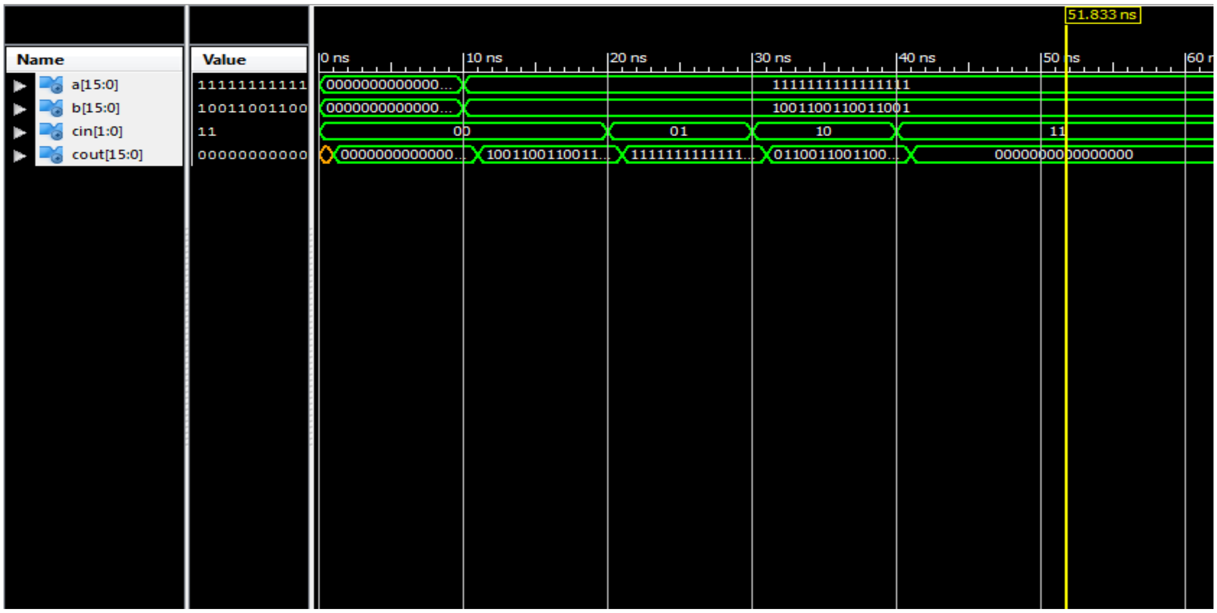
Name	Value
in0	1
in1	0
in2	1
s0	1
s1	1
z	0

2.5 ALU

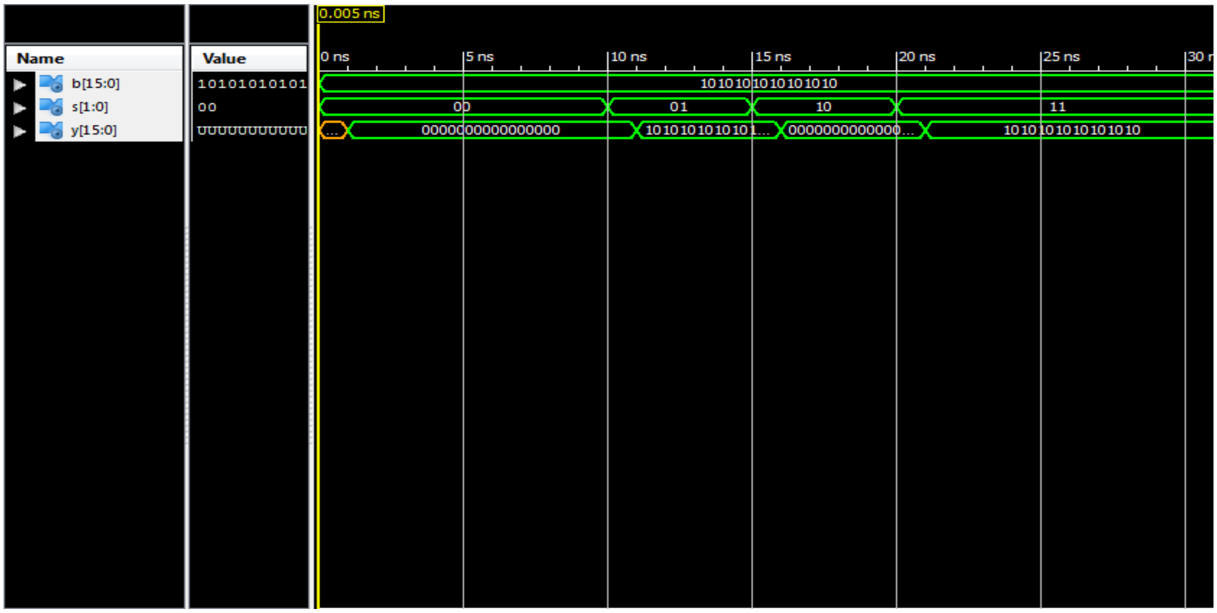


Name	Value
alu_a[15:0]	11111111101
alu_b[15:0]	00000000000
alu_select[3:0]	1001
alu_cout	U
alu_vout	U
alu_g[15:0]	00000000000

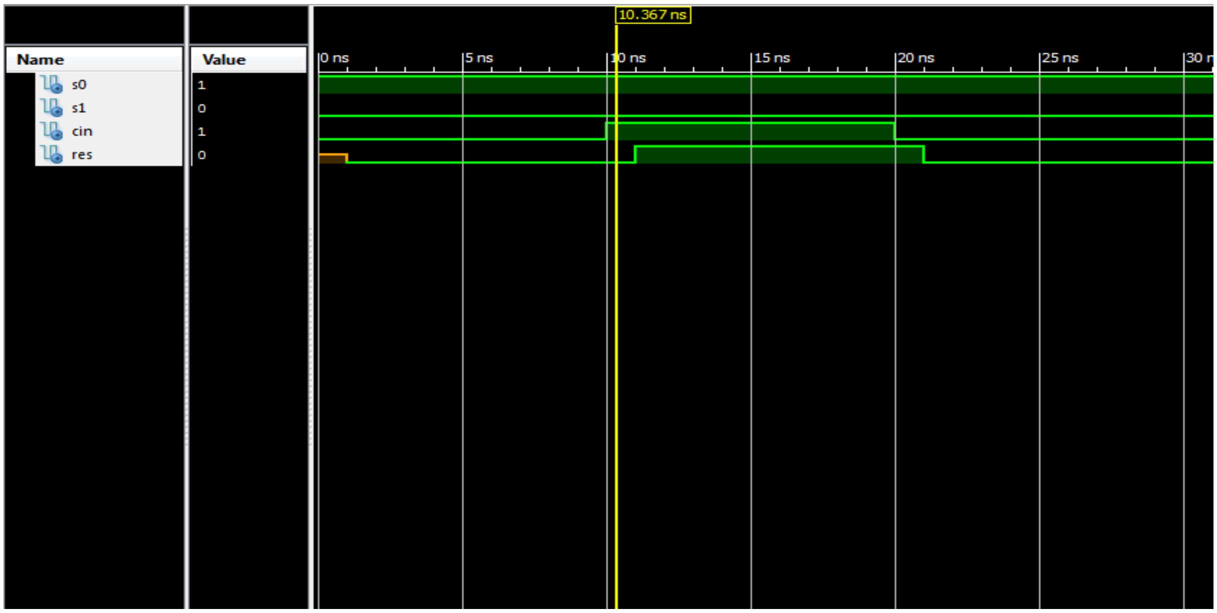
2.5.1 LOGIC


51.833 ns
Name
Value
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 n
a[15:0]
11111111111
0000000000000...
1111111111111111
b[15:0]
10011001100
0000000000000...
1001100110011001
cin[1:0]
11
00
01
10
11
cout[15:0]
00000000000
0000000000000...
1001100110011...
1111111111111...
0110011001100...
0000000000000000

2.5.2 B LOGIC

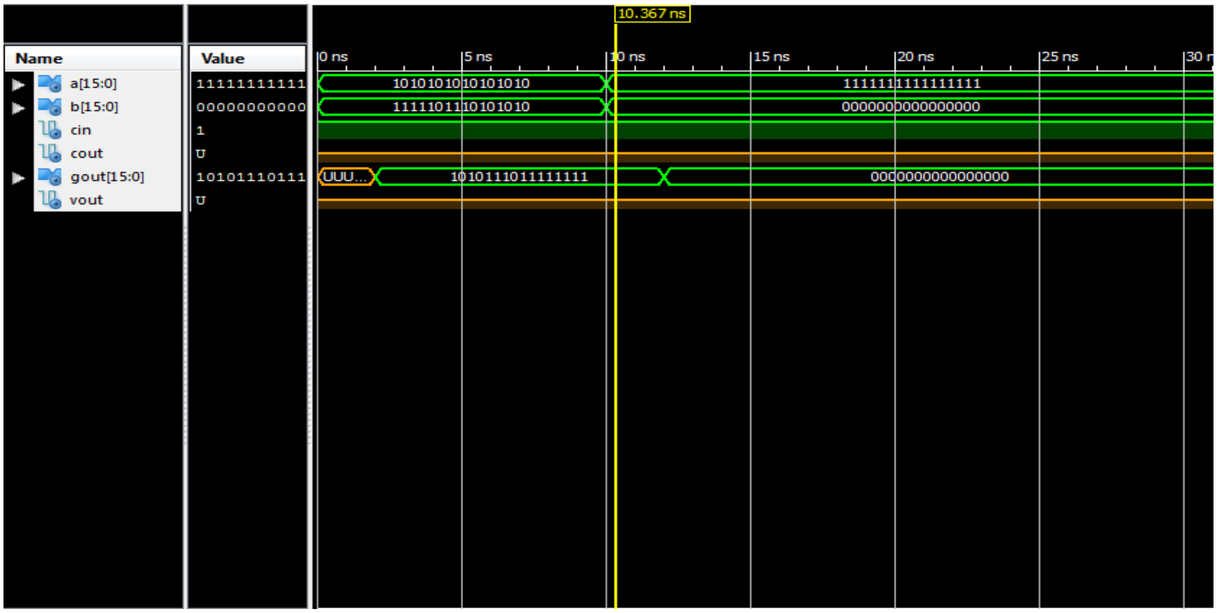

0.005 ns
Name
Value
b[15:0]
s[1:0]
y[15:0]
10101010101
00
UUUUUUUUUUU
0 ns
5 ns
10 ns
15 ns
20 ns
25 ns
30 r
1010101010101010
00
01
10
11
...
0000000000000000
101010101010101...
0000000000000...
1010101010101010

2.5.3 MUX2 TO 1BIT



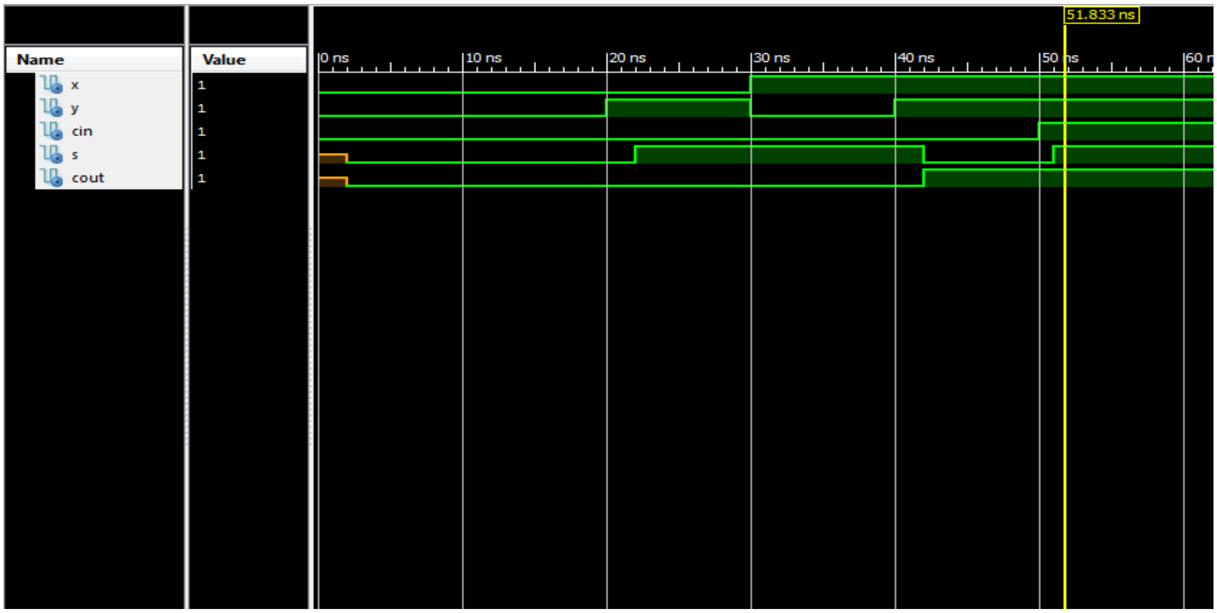
Name	Value
s0	1
s1	0
cin	1
res	0

2.5.4 RIPPLE



Name	Value
a[15:0]	11111111111
b[15:0]	00000000000
cin	1
cout	U
gout[15:0]	10101110111
vout	U

2.5.5 FULL ADDER



51.833 ns
Name
Value
x
y
cin
s
cout
1
1
1
1
1
0 ns
10 ns
20 ns
30 ns
40 ns
50 ns
60 n