VLSI DESIGN LAB. ASSIGNMENT -1

SAURAV GUPTA

ROLL No. – 153070046

Dept.- Electrical Engineering

Specialisation – Microelectronics (TA)

Batch - 2015-17

Teacher concerned – VIRENDRA SINGH

PROBLEM STATEMENT:

Design a signed 8-bit (8 bit inputs and 16-bit output) booth encoded multiplier. Use Structural VHDL for the design. Controller can be defined using behavioural code. You are also required to write test bench to verify your design.

SOL:

Booth's Algorithm:

Step 1: Given two numbers, convert them in binary.

Let X = Multiplicand

Y = Multiplier

Calculate -Y for future use.

Step 2: Set up four Columns and initialize them as follows

U	V	X	X-1
00000000	00000000	<Multiplicand>	0

X-1 stores the previous LSB of X, to start with X-1 should be '0'

Step 3: Analyze the LSB in X and the bit in X-1

if the string is "00" : No Operation

if the string is "01" : Add Y to U

if the string is "10" : Sub Y from U

if the string is "11" : No Operation

Step 4: RSA* UV.

Step 5: Rotate X. Go to step 3 and repeat the process until X has been Rotated to its original position.

* RSA - Right shift arithmetic or Arithmetic shift right(ASR)

Hardware Implementation:

Following Hardware would be required:

1. 8 bit Full Adder -- to add two 8-bit numbers

2. Two's Complement --for subtraction purpose

3. RSA Hardware -- for Right shift arithmetic

4. 16 bit register --to store the product

- **D Flip Flop** --basic component of register

5. 8 bit shift register -- to shift X

- **D Flip Flop** --basic component of register
- **2:1 MUX** -- to choose between load and rotate.

6. D flip flop -- to store X-1

7. Control circuit -- to provide enable signals for each Hardware

8. 8 bit Counter -- to provide stop signal when multiplication is done.

9. Booth's Multiplier -- connect all the Hardware to get the Multiplier

10. Booth's Multiplier test bench --to test the circuit

Circuit Diagram:

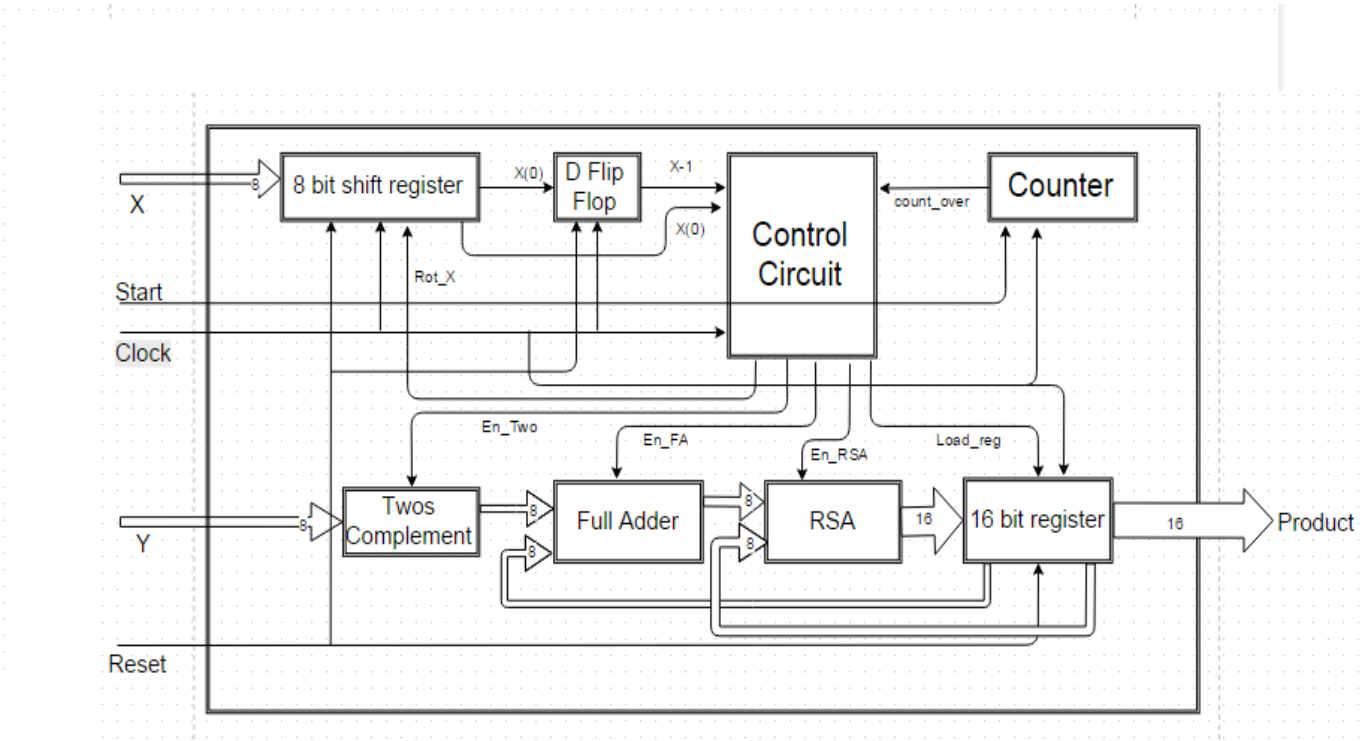

Fig. 1 Booth's Multiplier

Description of individual circuits

1. 8 bit FULL ADDER

Here Full Adder is slightly different from the conventional Full Adder . The operation is as follows

When Enable_FA = 1 : Add X and Y vector

When Enable_FA = 0 : output = Y vector

```
library ieee;
    use ieee.std_logic_1164.all;
    entity FA_8 is
        port(Cin: in std_logic;
            X,Y: in std_logic_vector(7 downto 0);
            Enable_FA: in std_logic;
            sum: out std_logic_vector(7 downto 0);
```

```
        Cout: out std_logic);
end FA_8;

architecture ST of FA_8 is
  begin
       process(X,Y,Enable_FA,Cin)
       variable C:std_logic_vector(8 downto 0);
       begin
            if (Enable_FA='1') then
                 C(0):=Cin;
       for i in 0 to 7 loop
       sum(i) <= (X(i) xor Y(i)) xor C(i);
       C(i+1) := (X(i) and Y(i)) or (C(i) and (X(i) xor Y(i)));
                 end loop;
                 Cout<=C(8);

            else
                 sum<=Y;
            end if;
       end process;
    end ST;
```

2. Two's Complement

When Enable_twos_complement = 1 : output = 2's complement of input

which is calculated by inverting the input and adding 1 to it.

When Enable_twos_complement = 0 : output = input

```
library ieee;
    use ieee.std_logic_1164.all;
    entity twos_complement is
        port(Xin: in std_logic_vector(7 downto 0);
            Xout: out std_logic_vector(7 downto 0);
            Enable_twos_complement: in std_logic
            );
    end twos_complement;

    architecture ST of twos_complement is

     begin
           process(Xin,Enable_twos_complement)
           variable D:std_logic_vector(7 downto 0);
           variable C:std_logic_vector(8 downto 0);
           variable sum: std_logic_vector(7 downto 0);
```

```
        variable X: std_logic_vector(7 downto 0);
        variable Y: std_logic_vector(7 downto 0);

        begin
        if (Enable_twos_complement = '1') then
             X:=not Xin;
             Y:=x"01";
             C(0):='0';

        for i in 0 to 7 loop
        sum(i) := (X(i) xor Y(i)) xor C(i);
        C(i+1) := (X(i) and Y(i)) or (C(i) and (X(i) xor Y(i)));
        end loop;
        D:=sum;
        else
             D:= Xin;
        end if;
             Xout <=D;
        end process;
end ST;
```

3. RSA Hardware

When Enable_RSA =1 : Perform Arithmetic right shift on input

```
library ieee;
    use ieee.std_logic_1164.all;
    entity RSA is
        port(Zin: in std_logic_vector(15 downto 0);
             Enable_RSA: in std_logic;
             Zout: out std_logic_vector(15 downto 0)
             );
    end RSA;

    architecture Behv of RSA is
     begin
          process(Zin,Enable_RSA)
          begin
               if (Enable_RSA ='1') then
                    for i in 0 to 14 loop
                         Zout(i)<=Zin(i+1);
                    end loop;
                    Zout(15)<=Zin(15);
               end if;
          end process;
        end Behv;
```

4. 16 bit register

When reset = 1 clear the register
when Enable_register = 1 : save the input on positive edge of the clock.

```
library ieee;
    use ieee.std_logic_1164.all;
    entity register_16 is
        port(din : in std_logic_vector(15 downto 0);
            clk:in std_logic;
            Enable_register: in std_logic;
            reset: in std_logic;
            dout: buffer std_logic_vector(15 downto 0)
           );
        end register_16;

      architecture ST of register_16 is
            signal temp: std_logic_vector(15 downto 0);
            signal temp_bar: std_logic_vector(15 downto 0);

            component D_ff is
            port(clk: in std_logic;
                 D: in std_logic;
                 reset: in std_logic;
                 Enable_D: in std_logic;
                 Q: out std_logic;
                 QPRIME: out std_logic);
            end component;
        begin
            Gen_Reg:
                 for i in 15 downto 0 generate
                       Dff: D_ff port map(clk=>clk,
D=>din(i),reset=>reset,Enable_D=>Enable_register,Q=>temp(i),QPRIME=>te
mp_bar(i));
                 end generate Gen_Reg;
                 dout<=temp;
            end ST;
```

5. 8 bit shift register

When reset = 1 clear the register
when Load_register = 1 : save the input on positive edge of the clock.
when rotate = 1 : rotate right the input on positive edge of the clock.

```
library ieee;
    use ieee.std_logic_1164.all;

    entity shift_register_8_bit is
        port(clk: in std_logic;
             reset: in std_logic;
             rotate: in std_logic;
```

```
            X: in std_logic_vector(7 downto 0);
            Load_register: in std_logic;
            Y: out std_logic_vector(7 downto 0)
            );
    end shift_register_8_bit;

     architecture ST of shift_register_8_bit is
           signal temp: std_logic_vector(7 downto 0);
               signal temp_bar: std_logic_vector(7 downto 0);
               signal D: std_logic_vector(7 downto 0);
               signal Enable_register: std_logic;

               component D_ff
               port(clk: in std_logic;
                     D: in std_logic;
                     reset: in std_logic;
                     Enable_D: in std_logic;
                     Q: out std_logic;
                     QPRIME: out std_logic);
               end component;

               component mux_21
               port(select_line: in std_logic;
               in1: in std_logic;
               in2: in std_logic;
               Y: out std_logic
               );
               end component;

              begin
                     Enable_register<= Load_register or rotate;
                     Gen_Reg:
                     for i in 7 downto 0 generate
                           mux: mux_21 port
map(select_line=>Load_register, in1=>X(i), in2=>temp((i+1)mod
8),Y=>D(i));
                           Dff : D_ff port map(clk=>clk,
D=>D(i),reset=>reset,Enable_D=>Enable_register,Q=>temp(i),QPRIME=>temp
_bar(i));
                     end generate Gen_Reg;
                     Y<=temp;
    end ST;
```

5.a 2:1 MUX

```
library ieee;
    use ieee.std_logic_1164.all;

    entity mux_21 is
        port(select_line: in std_logic;
```

```
            in1: in std_logic;
            in2: in std_logic;
            Y: out std_logic
            );
    end mux_21;

     architecture DF of mux_21 is

           begin
           process(select_line,in1,in2)
           begin
           Y<=(in1 and select_line) or (in2 and (not select_line));
           end process;
    end DF;
```

6. D flip flop

When reset = 1 clear the register
when Enable_D = 1 : save the input on positive edge of the clock.

```
library ieee;
    use ieee.std_logic_1164.all;
    entity D_ff is
        port(clk: in std_logic;
             D: in std_logic;
             reset: in std_logic;
             Enable_D: in std_logic;
             Q: out std_logic;
             QPRIME: out std_logic);
   end D_ff;

architecture Behv of D_ff is
        begin
           process(clk,D,reset)
           begin
                if reset ='1' then
                     Q<='0';
                     QPRIME<='1';
                else
                     if(Enable_D ='1' and clk='1' and clk'event)then
                          Q<=D;
                          QPRIME<=not D;
                     end if;
                end if;
           end process;
end Behv;
```

7. Control circuit

- for 8 clock cycles Read X(0) and X(-1) on positive edge of the clock

  if the string is "00" : set Enable_RSA =1, Load_Register =1.

  if the string is "01" : set Enable_FA =1, Enable_RSA =1, Load_Register =1.

  if the string is "10" : set Enable_FA =1, Enable_RSA =1, Enable_Twos_complement=1, Load_Register =1.

  if the string is "11" : set Enable_RSA =1, Load_Register =1.

- set Rot_x =1
- After clock cycles set Enable_FA =0, Enable_RSA =0, Enable_Twos_complement=0, Load_Register =0, Rot_x =0.

```
library ieee;
    use ieee.std_logic_1164.all;
     use ieee.numeric_std.all;

    entity control is
        port(Clock: in std_logic;
             in1: in std_logic;
             in2: in std_logic;
             count_over: in std_logic;
             Enable_Twos_complement: out std_logic;
              Enable_FA: out std_logic;
             Enable_RSA: out std_logic;
             Load_Register: out std_logic;
             Rot_x: out std_logic
            );
    end control;

    architecture Behv of control is
     begin
           process(Clock,in1,in2,count_over)
           begin
           if(count_over /='1') then
                if (Clock ='1' and Clock'event) then

                     if(in1 = '0' and in2 = '1') then
                          Enable_FA <='1';
                          Enable_RSA <='1';
                          Enable_Twos_complement<='0';
                          Load_Register<='1';
```

```
                    elsif(in1 = '1' and in2 = '0') then
                          Enable_FA <='1';
                          Enable_RSA <='1';
                          Enable_Twos_complement<='1';
                          Load_Register<='1';
                    else
                          Enable_FA <='0';
                          Enable_RSA <='1';
                          Enable_Twos_complement<='0';
                          Load_Register<='1';
                    end if;
                    Rot_x<='1';
               end if;
          else
          Enable_FA <='0';
          Enable_RSA <='0';
          Enable_Twos_complement<='0';
          Load_Register<='0';
          Rot_x<='0';
          end if;
          end process;

    end Behv;
```

8. 8 bit Counter

```
library ieee;
    use ieee.std_logic_1164.all;
    use ieee.numeric_std.all;


    entity counter_8 is
        port(Clock: in std_logic;
             Enable_counter: in std_logic;
             count_over: out std_logic
            );
    end counter_8;

    architecture Behv of counter_8 is
      signal temp:integer;
      begin
            process(Clock,Enable_counter)
            begin
             if (Clock ='1' and Clock'event) then
                  if Enable_counter='1' then
                        temp<=0;
                  elsif temp=8 then
                        count_over <= '1';
                  else
                        temp <=temp+1;
                  end if;
```

```
        end if;
        end process;

end Behv;
```

9. Booth's Multiplier

Connecting all the Hardware components as shown in Fig. 1

```
library ieee;
    use ieee.std_logic_1164.all;
    entity Booths_mul is
        port(X: in std_logic_vector(7 downto 0);
             Y: in std_logic_vector(7 downto 0);
             start: in std_logic;
             clock: in std_logic;
             reset: in std_logic;
             pdt: buffer std_logic_vector(15 downto 0)
             );
    end Booths_mul;

    architecture ST of Booths_mul is

      signal U: std_logic_vector(15 downto 0);
      signal Z:std_logic_vector(15 downto 0);
      signal Y_prime: std_logic_vector(7 downto 0);
      signal sum_prime: std_logic_vector(7 downto 0);
      signal Enable_twos_complement: std_logic;
      signal Enable_FA: std_logic;
      signal Enable_RSA: std_logic;
      signal Load_Register: std_logic;
      signal Rot_x: std_logic;
      signal Carry: std_logic;
      signal X_prime: std_logic_vector(7 downto 0);
      signal q: std_logic;
      signal q_bar: std_logic;
      signal count_over: std_logic;
      signal Cin: std_logic :='0';

            component control --import control circuit
                 port(Clock: in std_logic;
                 in1: in std_logic;
                 in2: in std_logic;
                 count_over: in std_logic;
                 Enable_Twos_complement: out std_logic;
                 Enable_FA: out std_logic;
                 Enable_RSA: out std_logic;
                 Load_Register: out std_logic;
                 Rot_x: out std_logic
                 );
```

```
end component;

component counter_8 --import 8 bit counter
     port(Clock: in std_logic;
     Enable_counter: in std_logic;
     count_over: out std_logic
     );
end component;

component D_ff --import D flip flop
     port(clk: in std_logic;
     D: in std_logic;
     reset: in std_logic;
     Enable_D: in std_logic;
     Q: out std_logic;
     QPRIME: out std_logic);
end component;

component FA_8 --import 8 bit FA
     port(Cin: in std_logic;
     X,Y: in std_logic_vector(7 downto 0);
     Enable_FA: in std_logic;
     sum: out std_logic_vector(7 downto 0);
     Cout: out std_logic);
end component;

component twos_complement --import twos_complement Hardware
     port(Xin: in std_logic_vector(7 downto 0);
     Xout: out std_logic_vector(7 downto 0);
     Enable_twos_complement: in std_logic
     );
end component;

component RSA  --import 16 bit Right Shift Arithmetic
Hardware
     port(Zin: in std_logic_vector(15 downto 0);
     Enable_RSA: in std_logic;
     Zout: out std_logic_vector(15 downto 0)
     );
end component;

component register_16 --import 16 bit register_16
     port(din : in std_logic_vector(15 downto 0);
     clk: in std_logic;
     Enable_register: in std_logic;
     reset: in std_logic;
     dout: buffer std_logic_vector(15 downto 0)
     );
end component;

component shift_register_8_bit --import 8 bit Right shift
Hardware
```

```
                port(clk: in std_logic;
                reset: in std_logic;
                rotate: in std_logic;
                X: in std_logic_vector(7 downto 0);
                Load_register: in std_logic;
                Y: out std_logic_vector(7 downto 0)
                );
          end component;

     begin
          twos_complement_0 : twos_complement port map(Xin=>Y,
Xout=>Y_prime, Enable_twos_complement=>Enable_twos_complement);

          U<= sum_prime & pdt(7 downto 0);

          FA_1: FA_8 port map(Cin=>Cin, X=>Y_prime, Y=>pdt(15 downto
8), Enable_FA=>Enable_FA, sum=>sum_prime, Cout=>Carry);

          RSA_2: RSA port map(Zin=> U,Enable_RSA=>Enable_RSA , Zout=>
Z);

          register_16bit_3: register_16 port map(din=>Z,clk=>clock,
Enable_register=> Load_Register, reset=>reset , dout=>pdt);

          right_shift_4 : shift_register_8_bit port
map(clk=>clock,reset=> reset ,rotate=> Rot_x, X=>X,
Load_Register=>start, Y=>X_prime);

          D_ff_5: D_ff port map(clk=>clock,
D=>X_prime(0),reset=>reset,Enable_D=>Rot_x, Q=>q ,QPRIME=>q_bar);

          control_6: control port map(Clock=>clock, in1=>X_prime(0),
in2=>q, count_over=>count_over,
Enable_Twos_complement=>Enable_Twos_complement, Enable_FA=>Enable_FA,
Enable_RSA=>Enable_RSA,Load_Register=>Load_Register, Rot_x=>Rot_x);

          counter_7: counter_8 port map(Clock=> clock,
Enable_counter=> start, count_over=>count_over);


    end ST;
```

10. Booth's Multiplier test bench

```
library ieee;
use ieee.std_logic_1164.all;

entity Booths_mul_tb is
 end Booths_mul_tb;

     architecture DF of Booths_mul_tb is
```

```
        --  Declaration of the component that will be instantiated.
        component Booths_mul
             port(X: in std_logic_vector(7 downto 0);
                  Y: in std_logic_vector(7 downto 0);
                  start: in std_logic;
                  clock: in std_logic;
                  reset: in std_logic;
             pdt: buffer std_logic_vector(15 downto 0)
                 );
        end component;

        --  Specifies which entity is bound with the component.
        for DUT: Booths_mul use entity work.Booths_mul;
        signal X: std_logic_vector(7 downto 0) :=x"00"; --initial
values of signal
           signal Y: std_logic_vector(7 downto 0) :=x"00"; --initial
values of signal
           signal start: std_logic:= '0';
           signal clock: std_logic:='0';
           signal reset: std_logic:='0';
           signal pdt: std_logic_vector(15 downto 0);

           begin
                --  Component instantiation.
           DUT: Booths_mul port map (X => X, Y => Y, start =>
start,clock => clock, reset=> reset, pdt => pdt);

           --  This process does the real job.
           process
                begin
                X <= x"36"; Y<=x"53"; -- change this to give input
                     wait for 20 ns;
                     wait;
                end process;

                process
                     begin
                     clock <='1';
                     wait for 20 ns;
                     clock <='0';
                     wait for 20 ns;
                end process;

                process
                begin
                     start<='1';
                     wait for 90 ns;
                     start<='0';
                     wait for 90 ns;
                     wait;
                end process;
```

```
            process
            begin
                  reset<= '1';
                  wait for 50 ns;
                  reset<= '0';
                  wait for 50 ns;
                  wait;
            end process;
end DF;
```