VLSI DESIGN LAB. ASSIGNMENT -2

SAURAV GUPTA

ROLL No. – 153070046

Dept.- Electrical Engineering

Specialisation – Microelectronics (TA)

Batch - 2015-17

Teacher concerned – VIRENDRA SINGH

PROBLEM STATEMENT:

Design a greatest common divisor (GCD) which can compute the GCD of two 8 bit signed numbers (1 sign bit is included). Use two's complement format to represent negative numbers. Partition the design into control (state machine) and datapath (functional units, storage, and steering logic) part. Use VHDL for the describing and simulation of the design. Description of datapath and controller must be clearly shown. Also, write a test bench to verify your design

SOL:

GCD ALGORITHM

Consider two inputs X and Y

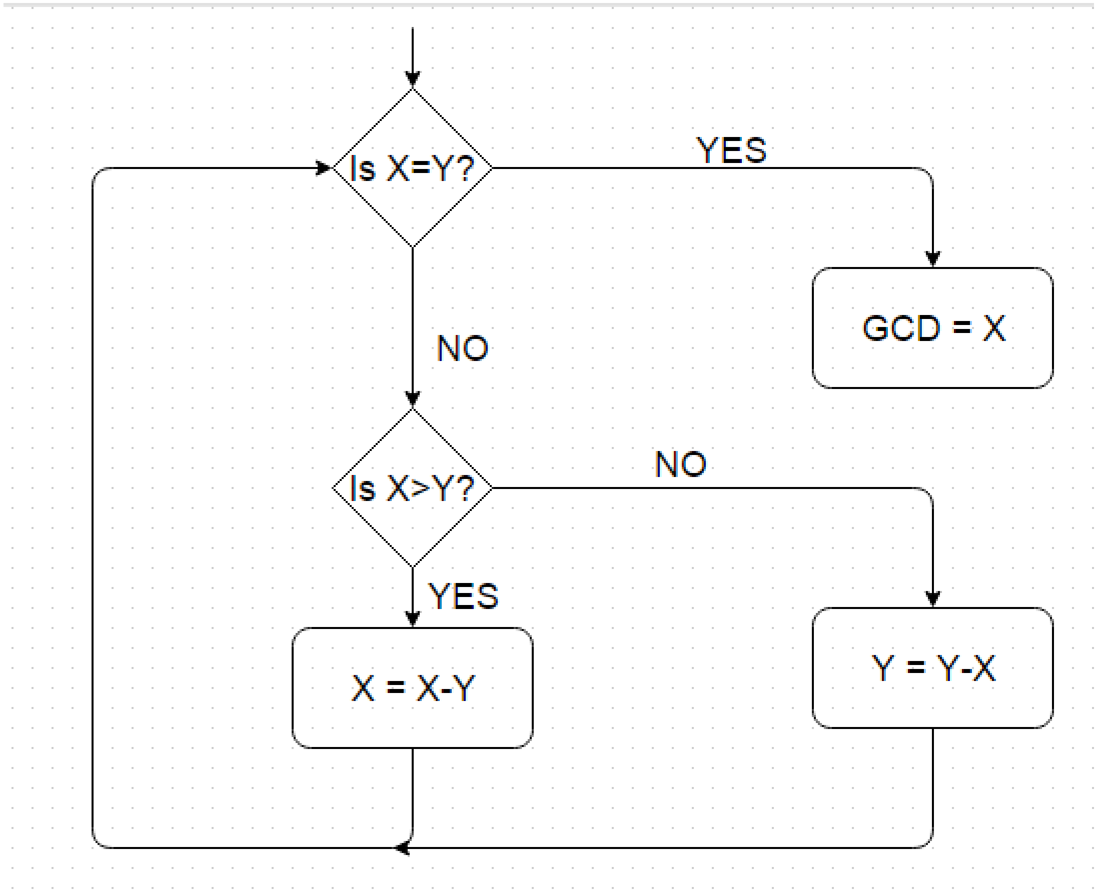


Fig. 1 Flow chart of GCD Algorithm

CIRCUIT DIAGRAM

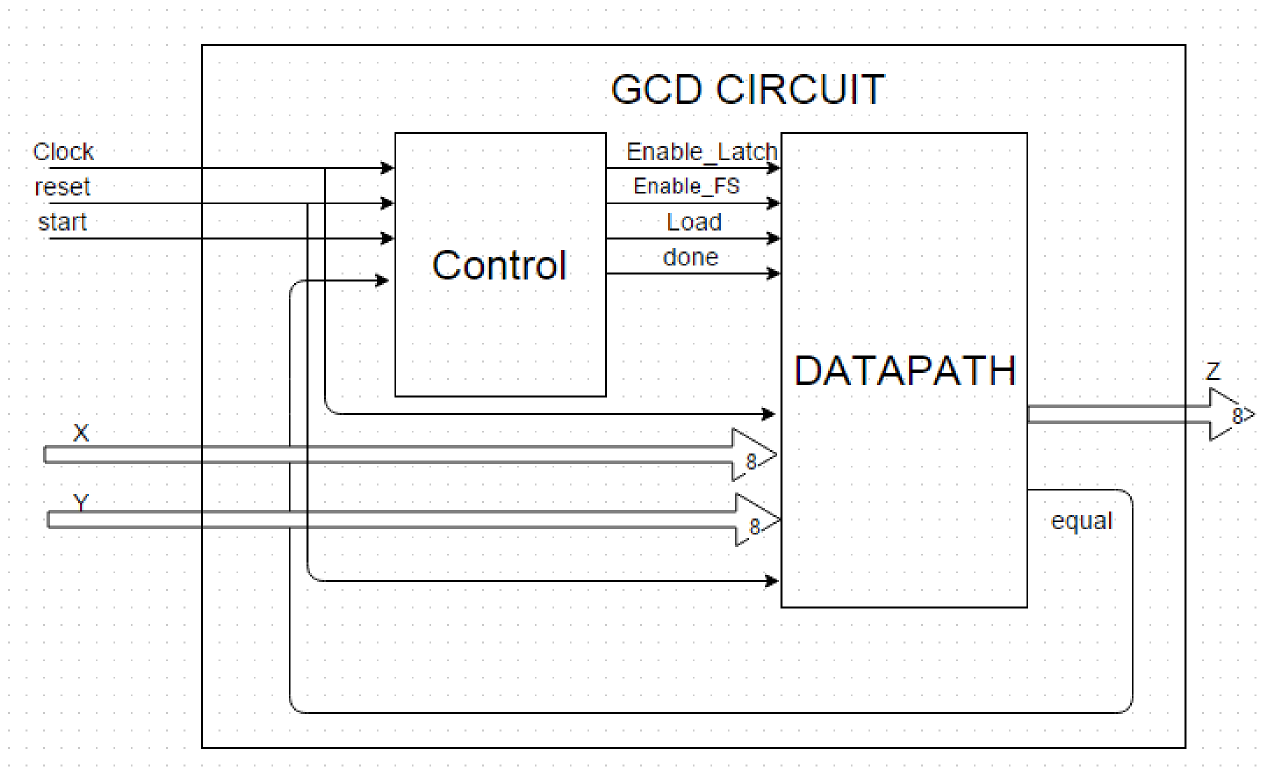


Fig. 2 GCD circuit: control path and data path

Datapath

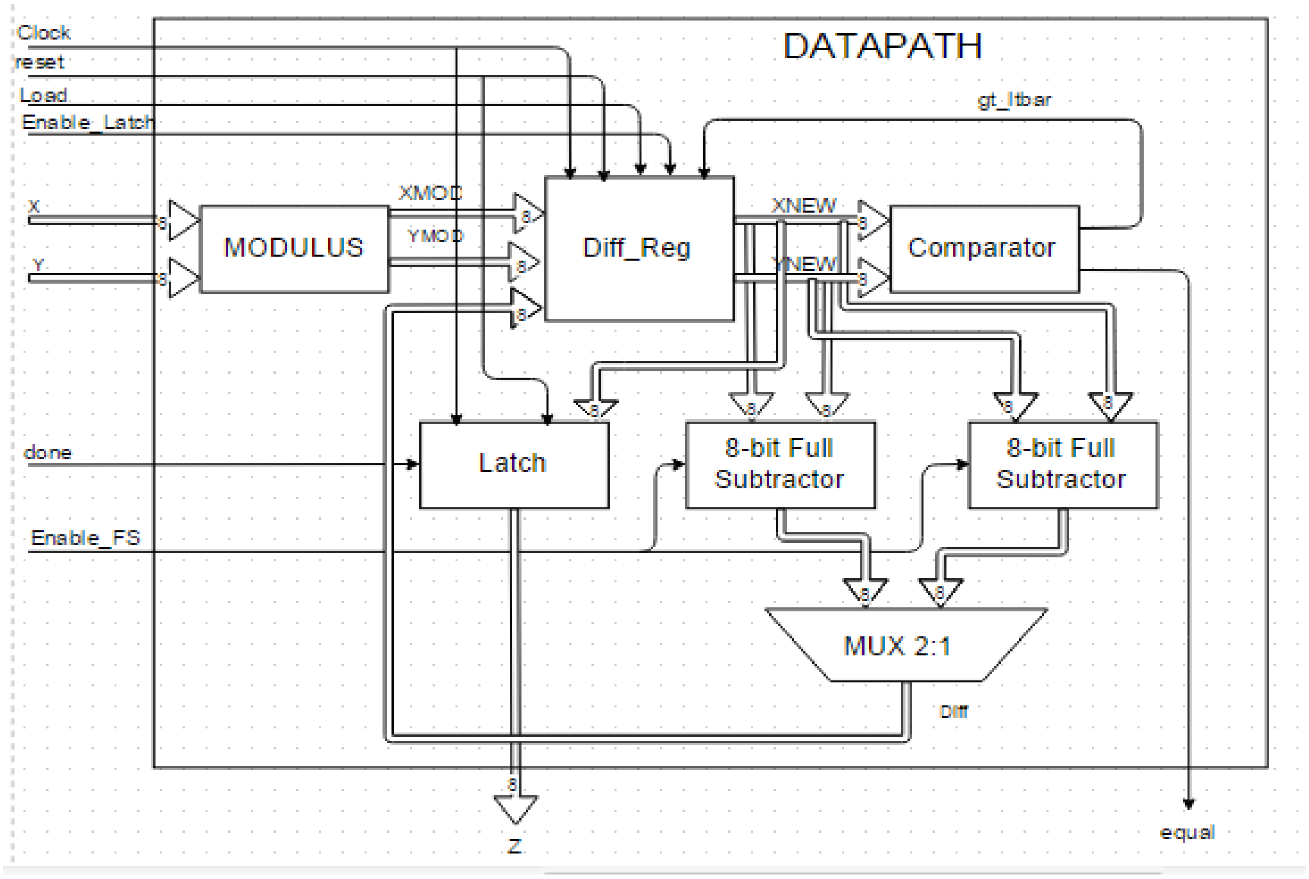


Fig. 3 Datapath- Internal circuitry

```
library ieee;
    use ieee.std_logic_1164.all;
     use ieee.numeric_std.all;
    entity Datapath is
        port(X,Y: in std_logic_vector(7 downto 0);
                clock: in std_logic;
                reset: in std_logic;
                Load: in std_logic;
                Enable_FS: in std_logic;
                Enable_Latch: in std_logic;
                done: in std_logic;
                equal: out std_logic;
                Z: out std_logic_vector(7 downto 0)
                );
    end Datapath;

    architecture ST of Datapath is
     signal Modx: std_logic_vector(7 downto 0);
     signal Mody: std_logic_vector(7 downto 0);
     signal Xnew: std_logic_vector(7 downto 0);
     signal Ynew: std_logic_vector(7 downto 0);
     signal Diff: std_logic_vector(7 downto 0);
```

```vhdl
  signal gt_ltbar: std_logic;
  signal Bin: std_logic:='0';
  signal Borrow: std_logic;
  signal Borrow1: std_logic;
  signal XminusY: std_logic_vector(7 downto 0);
  signal YminusX: std_logic_vector(7 downto 0);

  component Modulus
    port(X,Y: in std_logic_vector(7 downto 0);
        Xout,Yout: out std_logic_vector(7 downto 0)
            );
end component;

  component Diff_reg
       port(X,Y: in std_logic_vector(7 downto 0);
        Diff: in std_logic_vector(7 downto 0);
        gt_ltbar: in std_logic;
        clk: in std_logic;
        Ld_diffbar: in std_logic;
        reset: in std_logic;
        Enable_Latch: in std_logic;
        Xout,Yout: out std_logic_vector(7 downto 0)
        );
  end component;

  component comparator
    port(X,Y: in std_logic_vector(7 downto 0);
        equal: out std_logic;
        gt_ltbar: out std_logic
       );
 end component;

  component FS_8
    port(Bin: in std_logic;
        X,Y: in std_logic_vector(7 downto 0);
        Enable_FS: in std_logic;
        Diff: out std_logic_vector(7 downto 0);
        Bout: out std_logic);
 end component;

  component mux_21
    port(select_line: in std_logic;
        in1: in std_logic_vector(7 downto 0);
        in2: in std_logic_vector(7 downto 0);
        Y: out std_logic_vector(7 downto 0)
        );
  end component;

  component Latchs
    port(X: in std_logic_vector(7 downto 0);
        reset: in std_logic;
        clk: in std_logic;
```

```
            Enable_Latch: in std_logic;
            Z: out std_logic_vector(7 downto 0)
            );
     end component;

     begin
           Modulus_1: Modulus port
map(X=>X,Y=>Y,Xout=>ModX,Yout=>ModY);
           Diff_reg_2: Diff_reg port
map(X=>ModX,Y=>ModY,Diff=>Diff,gt_ltbar=>gt_ltbar,clk=>clock,Ld_diffba
r=>Load,reset=>reset,Enable_Latch=>Enable_Latch,Xout=>Xnew,Yout=>Ynew)
;
           comparator_3: comparator port
map(X=>Xnew,Y=>Ynew,equal=>equal,gt_ltbar=>gt_ltbar);
           FS_8_4: FS_8 port
map(Bin=>Bin,X=>Xnew,Y=>Ynew,Enable_FS=>Enable_FS,Diff=>XminusY,Bout=>
Borrow);
           FS_8_5: FS_8 port
map(Bin=>Bin,X=>Ynew,Y=>Xnew,Enable_FS=>Enable_FS,Diff=>YminusX,Bout=>
Borrow1);
           mux_21_6: mux_21 port
map(select_line=>gt_ltbar,in1=>YminusX,in2=>XminusY,Y=>Diff);
           Latch_7: Latchs port
map(X=>Xnew,reset=>reset,clk=>clock,Enable_Latch=>done,Z=>Z);
    end ST;
```

Control Circuit

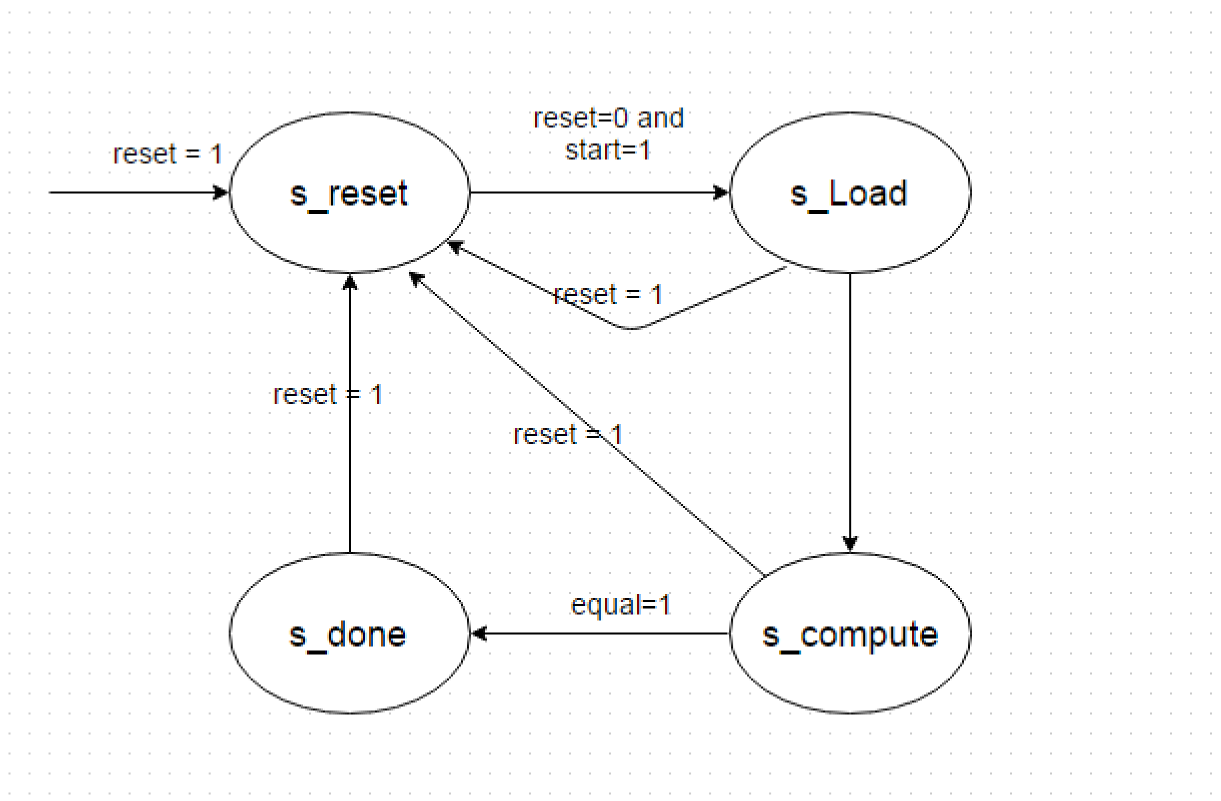


Fig. 4 FSM of Control path

```
library ieee;
use ieee.std_logic_1164.all;
entity  control is
 port(equal : in std_logic;
      clk: in std_logic;
      start: in std_logic;
      reset: in std_logic;
      Enable_FS: out std_logic;
      Enable_Latch: out std_logic;
      Load: out std_logic;
      done: out std_logic
      );
end control;

architecture behave  of control is
type MyState  is (s_reset,s_Load,s_compute,s_done);
   signal state_signal : MyState;

begin  -- behave

  next_state: process (reset,clk,start,equal)
    variable next_state_var : MyState;
  begin  -- process next_state
    next_state_var := state_signal;
```

```
if reset = '1' then
  next_state_var := s_reset;
else
  case state_signal is
    when s_reset =>
      if(reset = '0' and start='1') then
        next_state_var := s_Load;
      end if;
    when s_Load =>
      next_state_var := s_compute;
    when s_compute =>
      if equal = '1' then
        next_state_var := s_done;
      else
        next_state_var := s_compute;
      end if;
    when s_done =>
            if (reset='1') then
        next_state_var := s_reset;
            end if;
    when others => null;
  end case;
end if;

if(clk'event and clk = '1') then
  state_signal <= next_state_var;
end if;

end process next_state;

-- outputs: the outputs of the Control path.
output_process: process(state_signal)
begin  -- process output_process
  -- hot conditions indicated later
  Enable_FS       <= '0';
  Enable_Latch    <= '0';
  Load            <= '0';
   done                  <= '0';

  case state_signal is
    when s_reset=>
      Enable_FS <= '0';
         Enable_Latch <= '0';
         Load <='0';
         done<='0';
    when s_Load=>
      Enable_FS <= '0';
         Enable_Latch <= '0';
         Load <='1';
         done<='0';
    when s_compute =>
```

```
             Enable_FS <= '1';
             Enable_Latch <= '1';
             Load <='0';
             done<='0';
        when s_done =>
             Enable_FS <= '0';
             Enable_Latch <= '0';
             Load <='0';
             done<='1';
        when others =>null;
      end case;
    end process output_process;
  end behave ;
```

Individual Circuits

1. MODULUS

--calculates the mod value of X and Y

```
library ieee;
    use ieee.std_logic_1164.all;
    entity Modulus is
        port(X,Y: in std_logic_vector(7 downto 0);
            Xout,Yout: out std_logic_vector(7 downto 0)
                );
    end Modulus;

    architecture DF of Modulus is
      begin
           process(X,Y)
           variable A: std_logic_vector(7 downto 0);
           variable B: std_logic_vector(7 downto 0);
           variable sum: std_logic_vector(7 downto 0);
           variable C: std_logic_vector(8 downto 0);
           variable A1: std_logic_vector(7 downto 0);
           variable B1: std_logic_vector(7 downto 0);
           variable sum1: std_logic_vector(7 downto 0);
           variable C1: std_logic_vector(8 downto 0);
           begin
                if(X(7)='1') then
                      A:=not X;
                      B:=x"01";
                      C(0):='0';
                      for i in 0 to 7 loop
                      sum(i) := (A(i) xor B(i)) xor C(i);
                      C(i+1) := (A(i) and B(i)) or (C(i) and (A(i) xor
B(i)));
                      end loop;
                      Xout<=sum;
                 else
                      Xout<=X;
                 end if;

                 if(Y(7)='1') then
                      A1:=not Y;
                      B1:=x"01";
                      C1(0):='0';
                      for i in 0 to 7 loop
                      sum1(i) := (A1(i) xor B1(i)) xor C1(i);
                      C1(i+1) := (A1(i) and B1(i)) or (C1(i) and (A1(i)
xor B1(i)));
                      end loop;
                      Yout<=sum1;
                 else
                      Yout<=Y;
                 end if;
            end process;
        end DF;
```

2. Diff_Reg

--updates the value of X and Y depending upon the value of gt_ltbar and Load signal

```
library ieee;
    use ieee.std_logic_1164.all;
    entity Diff_reg is
        port(X,Y: in std_logic_vector(7 downto 0);
                Diff: in std_logic_vector(7 downto 0);
                gt_ltbar: in std_logic;
                clk: in std_logic;
                Ld_diffbar: in std_logic;
                reset: in std_logic;
                Enable_Latch: in std_logic;
            Xout,Yout: out std_logic_vector(7 downto 0)
                );
    end Diff_reg;

    architecture Behv of Diff_reg is
     begin

     process(X,Y,Diff,reset,clk,Enable_Latch,gt_ltbar,Ld_diffbar)
          begin
               if (reset='1') then
                    Xout<=x"00";
                    Yout<=x"00";
               else
```

```
                      if(Ld_diffbar='1') then
                           Xout<=X;
                           Yout<=Y;
                      else
                           if(clk='1' and clk'event and
Enable_Latch='1') then
                                if(gt_ltbar='0') then
                                     Yout<=Diff;
                                else
                                     Xout<=Diff;
                                end if;
                           end if;
                      end if;
                 end if;
            end process;
        end Behv;
```

3. Comparator

--compares X and Y to signify whether X=Y, X>Y or X<Y

```
library ieee;
    use ieee.std_logic_1164.all;
    entity comparator is
        port(X,Y: in std_logic_vector(7 downto 0);
                equal: out std_logic;
            gt_ltbar: out std_logic
                );
    end comparator;

    architecture DF of comparator is
     begin
          process(X,Y)
          begin
               if(X=Y and X/=x"00") then
                    equal<='1';
                    gt_ltbar<='0';
               else
                    if(X>Y) then
                         equal<='0';
                         gt_ltbar<='1';
                    else
                         equal<='0';
                         gt_ltbar<='0';
                    end if;
               end if;

          end process;
        end DF;
```

4. 8 bit Full Subtractor

```
library ieee;
    use ieee.std_logic_1164.all;
    entity FS_8 is
        port(Bin: in std_logic;
            X,Y: in std_logic_vector(7 downto 0);
                Enable_FS: in std_logic;
            Diff: out std_logic_vector(7 downto 0);
                Bout: out std_logic);
    end FS_8;

    architecture DF of FS_8 is
     begin
           process(X,Y,Enable_FS,Bin)
           variable B:std_logic_vector(8 downto 0);
           begin
                if (Enable_FS='1') then
                     B(0):=Bin;
                     for i in 0 to 7 loop
                     Diff(i) <= (X(i) xor Y(i)) xor B(i);
                     B(i+1) := ((not X(i)) and Y(i)) or (B(i) and (not
X(i))) or (B(i) and Y(i));
                     end loop;
                     Bout<=B(8);
                end if;
           end process;
        end DF;
```

5. Latch

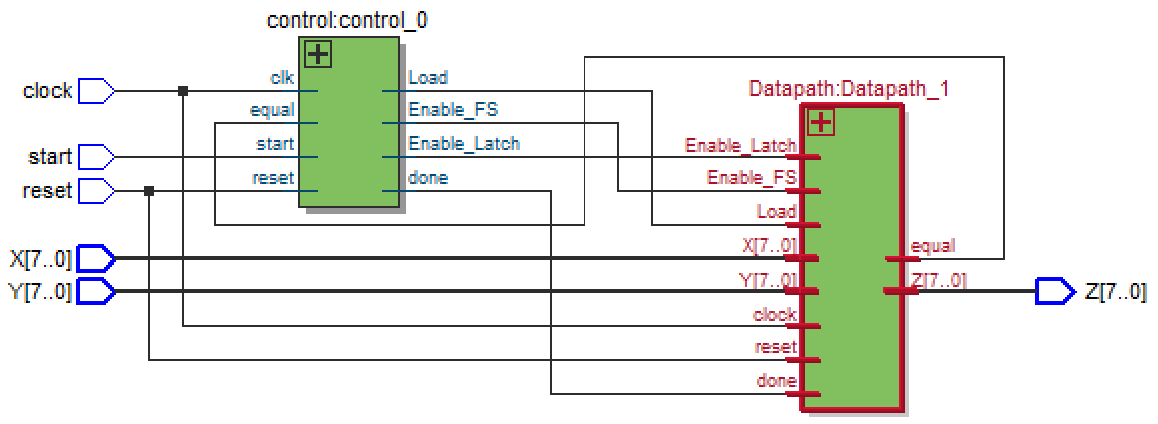
```
library ieee;
    use ieee.std_logic_1164.all;
    entity Latchs is
        port(X: in std_logic_vector(7 downto 0);
                reset: in std_logic;
                clk: in std_logic;
                Enable_Latch: in std_logic;
            Z: out std_logic_vector(7 downto 0)
                );
    end Latchs;

    architecture DF of Latchs is
     begin
           process(X,Enable_Latch,reset,clk)
                begin
                     if(reset='1') then
                          Z<=x"00";
                     else
```

```
                           if(clk='1' and clk'event and
Enable_Latch='1') then
                                Z<=X;
                           end if;
                      end if;
           end process;
    end DF;
```

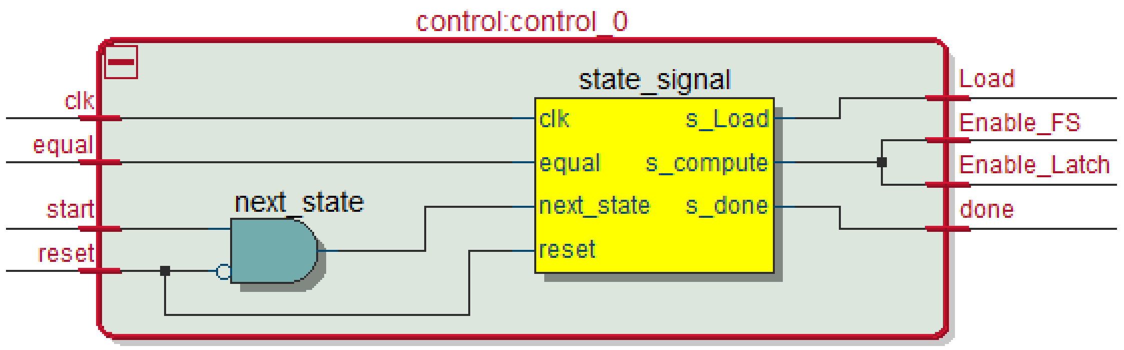
6. MUX 2:1

```
library ieee;
    use ieee.std_logic_1164.all;
    entity mux_21 is
        port(select_line: in std_logic;
                in1: in std_logic_vector(7 downto 0);
                in2: in std_logic_vector(7 downto 0);
            Y: out std_logic_vector(7 downto 0)
                );
    end mux_21;

     architecture Behv of mux_21 is

            begin
                process(select_line,in1,in2)
                begin
                      if(select_line='0') then
                           Y<=in1;
                      else
                           Y<=in2;
                      end if;
                end process;
    end Behv;
```

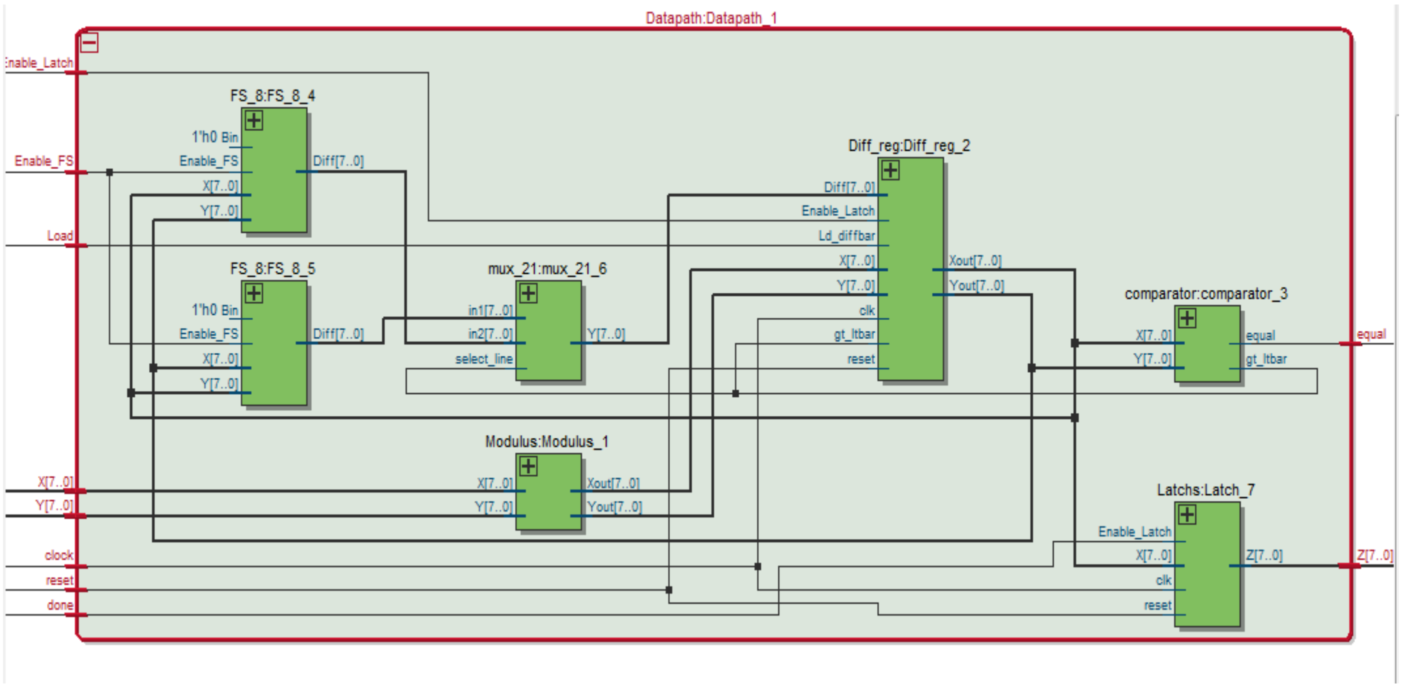
RTL SYNTHESIS
using Quartus tool



Expanding control path



Expanding Data path



Expanding every component we get RTL design