Lab 6: Great Common Divisor

Objective

The purpose of this lab is to design a circuit that calculates the greatest common divisor of two eight-bit numbers. Finite state machine is used to implement the situations that need to repeat itself in this circuit. I have determined 3 states to determine which operation will be done between the input A and input B numbers: A equal to B, A greater than B and A less than B. The circuit choose a way according to these states.

Methodology

Before I started writing the VHDL code, I thought about what the general outline of the system should be. I used the Euclidian algorithm to find the GCD. The logic of this algorithm is to proceed by replacing the two numbers with the mod result of the larger one until their mode is zero. When the mod result is zero, the smaller number gives the answer. Then I designed the diagram shown in Figure 1 and started coding the components.

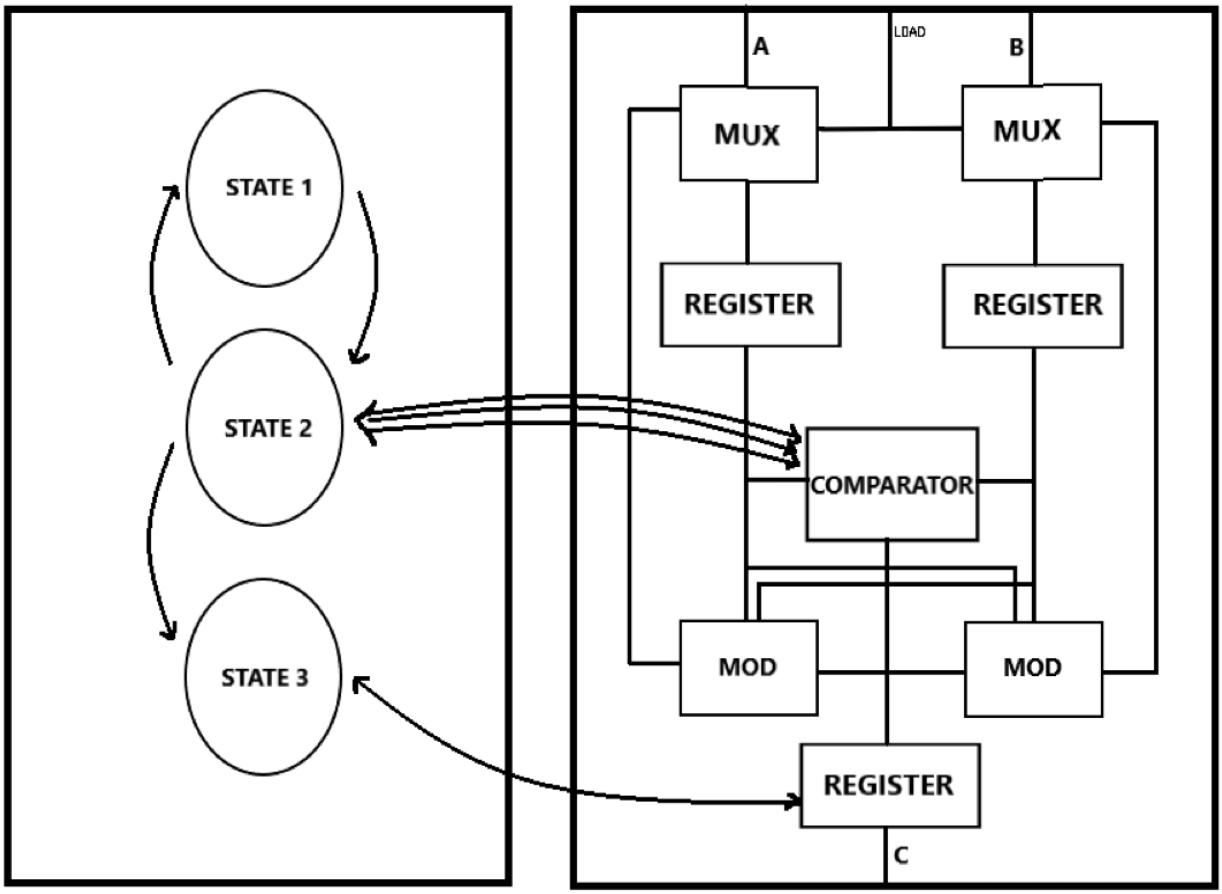

Figure 1 – Design of Great Common Divisor Circuit

First, I gave the load input as select to the multiplexers to receive the external input. When the button is pressed, when the load becomes 1, the circuit takes the input given from the outside. When the button is not pressed, the circuit operates on the numbers it has determined inside. Then the received numbers are recorded in registers. The numbers recorded here are sent to the comparator and mod modules for further processing. In the comparator, the magnitudes of the numbers are compared with each other and results are obtained accordingly. If the number A is larger than B, the output becomes "10" and selects the left multiplexer, then mod mode operation is performed and the new number is recorded in the upper register. In this case, the system switches from state 2 to state 1. If the B number is larger than A, the output becomes "01" and selects the right multiplexer, then mod mode operation is performed and the new number is recorded in the upper register. In this case, the system switches from state 2 to state 1. If the numbers A and B are equal, this indicates that the greatest common divisor is equal to them and the output is "00". This output allows going from state 2 to state 3 and reflects the last register A number to the LEDs without applying any operation as output. I also used Basys3's own 100Mhz clock for all operations. I used the switches on the basys to give the number input in the operations, and the first eight LEDs on the right to see the result.

Results

As a result, when I entered two 8-bit numbers on the card through the switches, I saw that the result was correctly determined on the LEDs. When I pressed the reset button, I reset the result and when I pressed the enable button, the circuit performed the operations again and gave the correct result. In the simulation, I entered two different 8-bit numbers and saw that the result was correct.

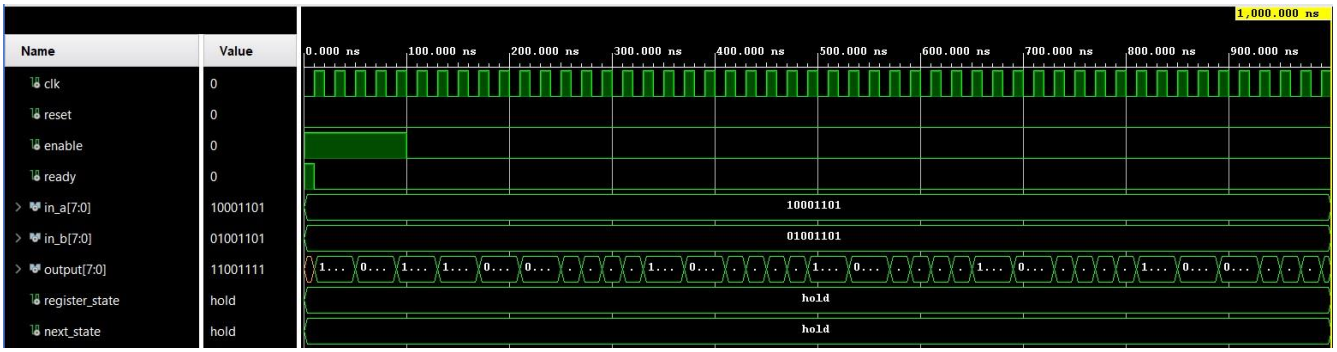

Figure 2 – Simulation Results

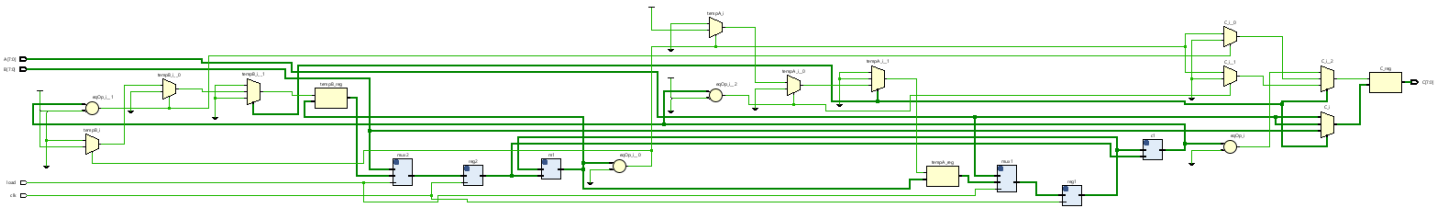

Figure 3 – Elaborated Design

Below are photos of a few situations to show that the code gives the correct results on the card.

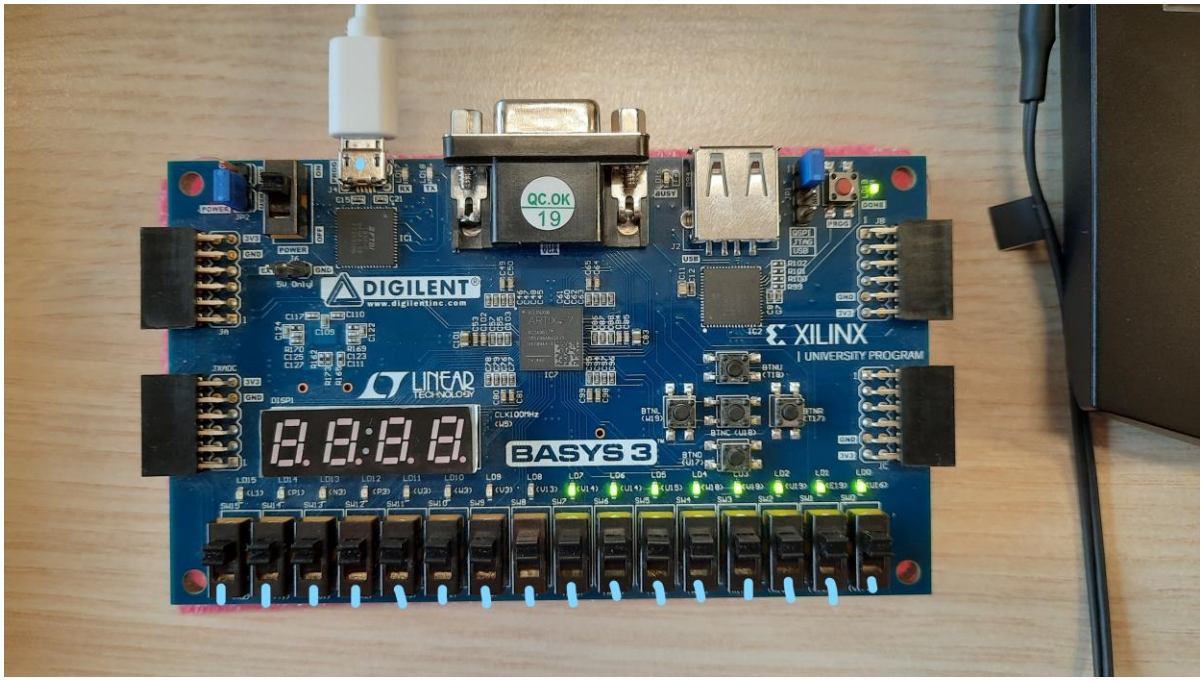

Figure 4 – All switches are ON : A = 255 – B = 255 => C = 255

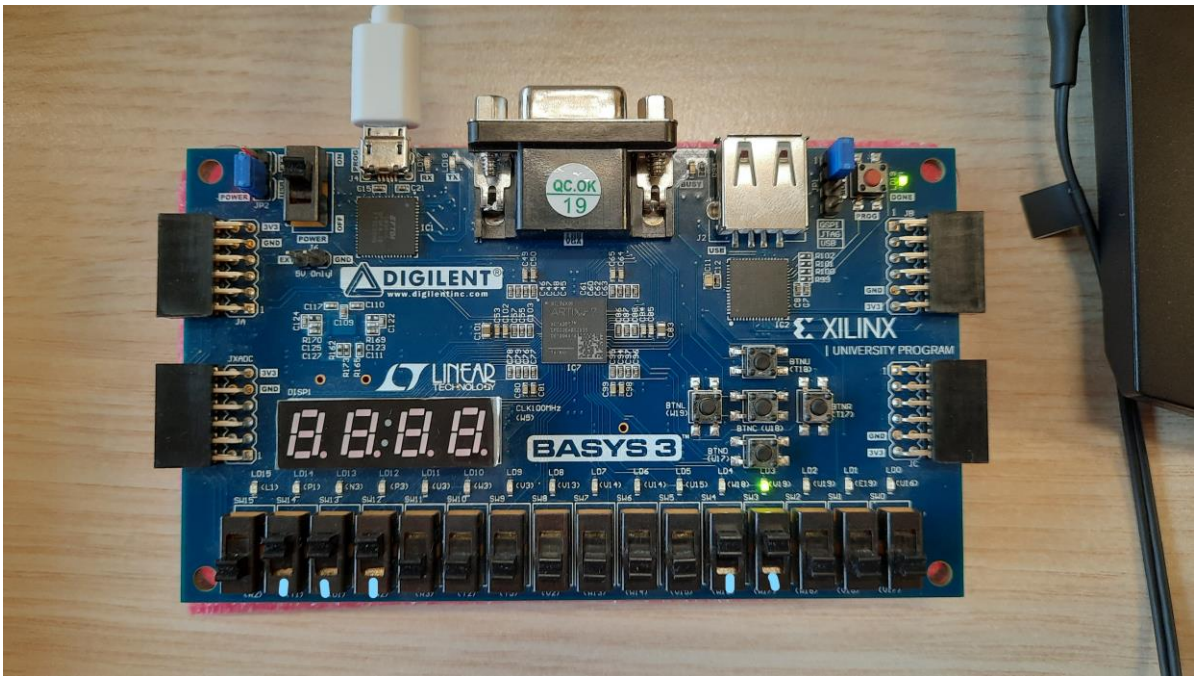

Figure 5 – A = 24 – B = 112 => C = 8

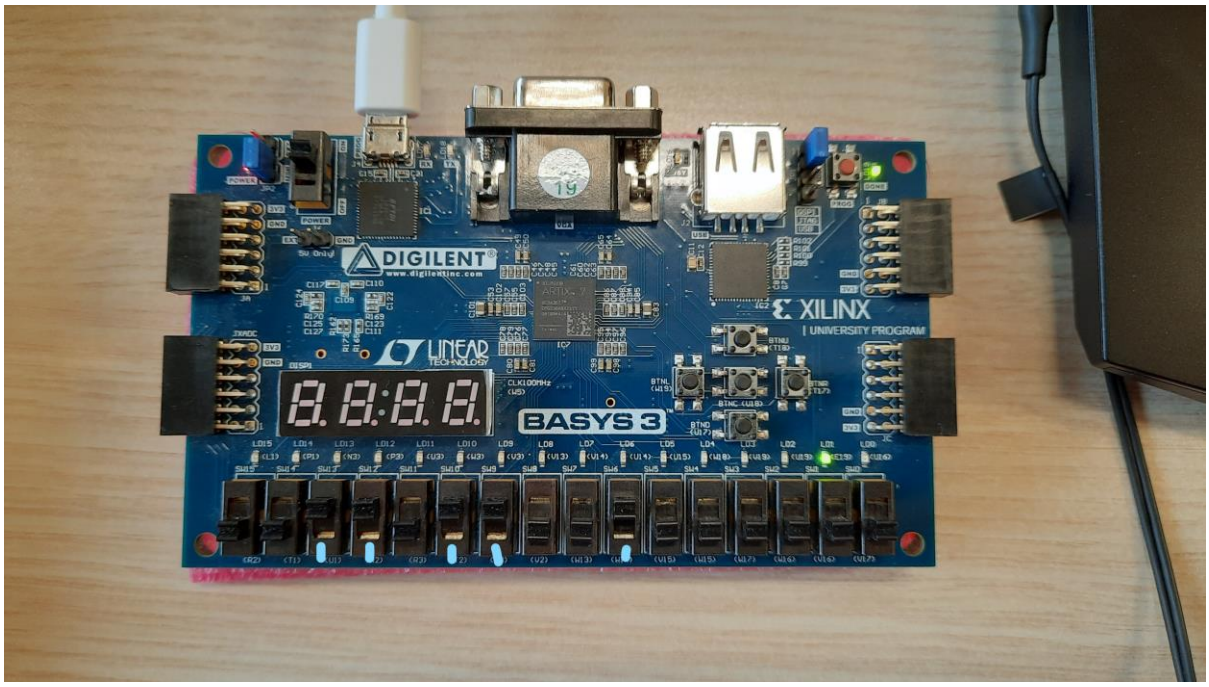

Figure 6 – A = 64 – B = 54 => C = 2

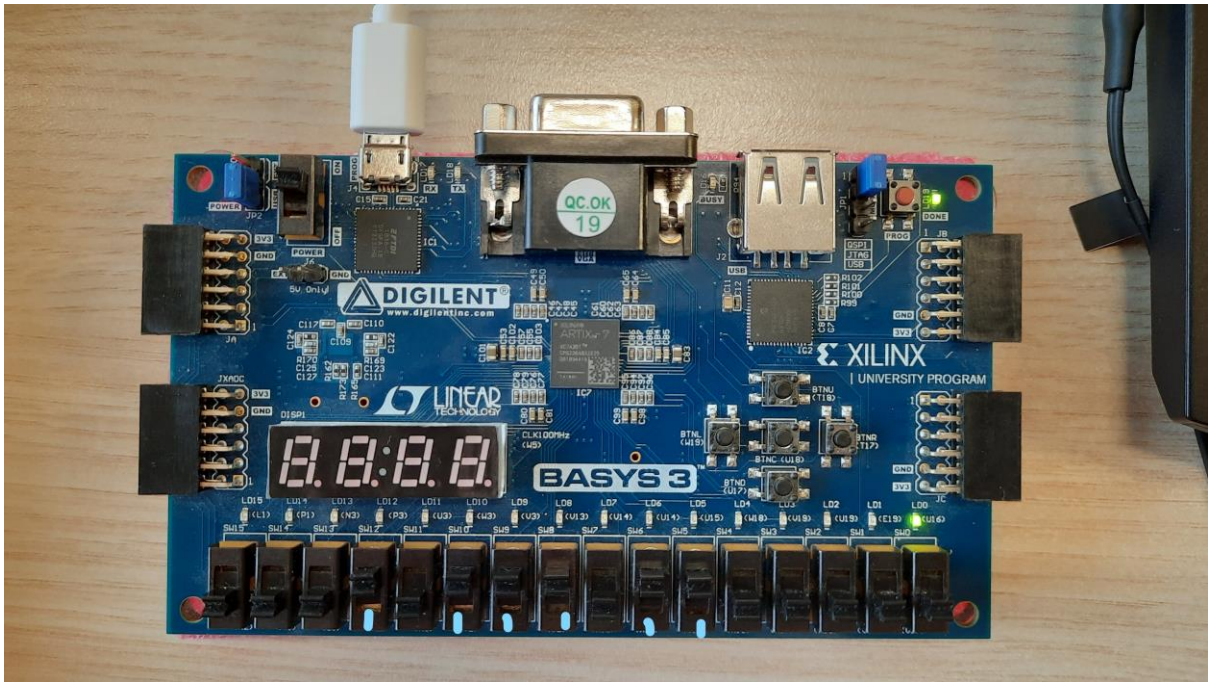

Figure 7 – A = 96 – B = 23 => C = 1

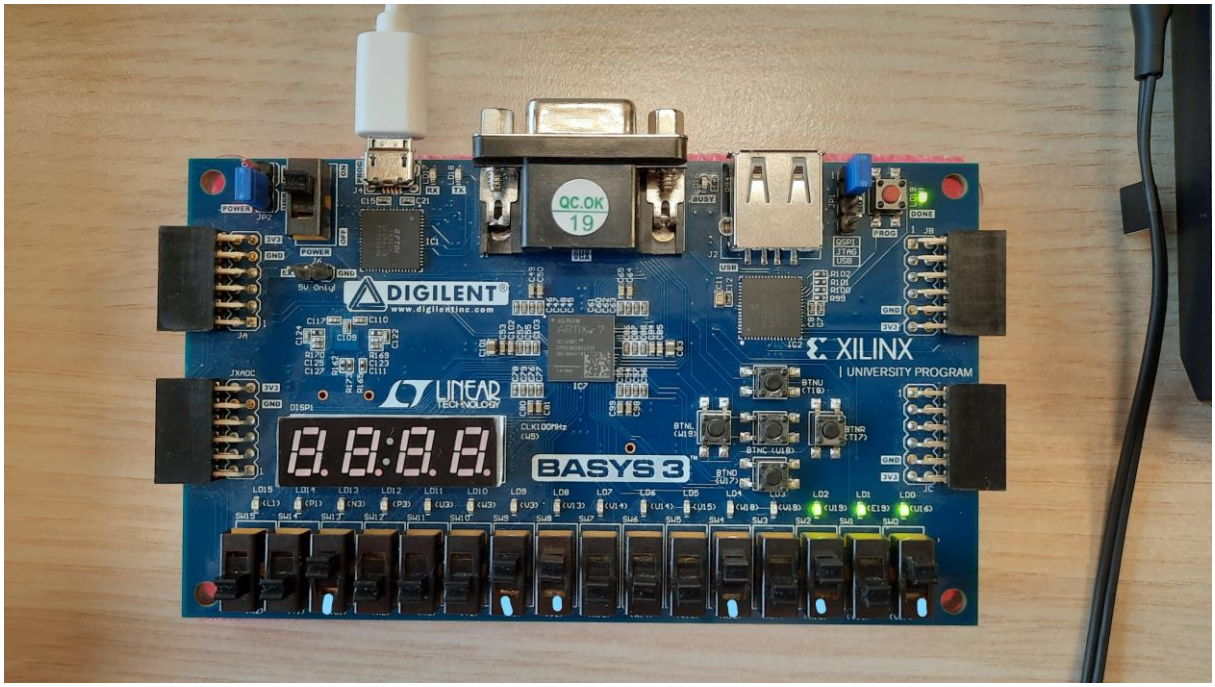

Figure 8 – A = 21 – B = 35 => C = 7

Questions

1. Euclidian algorithm is used to calculate GCD. In short, this algorithm proceeds by taking the mode of the larger of two numbers with respect to the smaller and replacing the larger number with the resulting value.

2.The module is FSM. It is Moore machine. If we were to design this circuit as a combinational, more elements would have to be used. Since the clock time is not important in a combinational circuit, the result is only dependent on the duration of the delays. but since more circuit elements will be used, they may have much larger delays than FSM.

3. Basys 3 has 100MHz inner clock frequency. The length of the operation varies according to the numbers given and a transaction is made at each clock beat. We can extend the processing time by changing the clock frequency with the divider as we did in the previous lab.

Conclusion

As a result, I designed a finite state machine in this lab. I learned how the circuits that loop themselves until they reach the desired state work with the finite state machine. While writing the code, I realized that it is necessary to plan in advance and make a more modular and systematic planning. With this lab, where mathematical algorithms and electronic circuits are used together, I got a better impression of digital design.

Appendices

Top Module

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

entity topmodule is
   Port( A : in std_logic_vector(7 downto 0);
         B : in std_logic_vector(7 downto 0);
         C : out std_logic_vector(7 downto 0);
         load : in std_logic;
         clk : in std_logic);
end topmodule;

architecture Behavioral of topmodule is

type state_type is (AeqB ,AltB ,AgtB);
signal CS, NS: state_type;

signal tempA :std_logic_vector(7 downto 0);
signal tempB :std_logic_vector(7 downto 0);
signal Q1 : std_logic_vector(7 downto 0);
signal Q2 : std_logic_vector(7 downto 0);
signal R1 : std_logic;
signal R2 : std_logic;
signal Cout : std_logic_vector(1 downto 0);
signal modout : std_logic_vector(7 downto 0);
signal regA : std_logic_vector(7 downto 0);
signal regB : std_logic_vector(7 downto 0);

component mux
    Port(A, B: in std_logic_vector(7 downto 0);
        sel : in std_logic;
        f : out std_logic_vector(7 downto 0));
end component;

component comporater
    Port( Q1 : in std_logic_vector(7 downto 0);
          Q2 : in std_logic_vector(7 downto 0);
          R  : out std_logic_vector(1 downto 0));
end component;

component reg
    Port (Xin: in std_logic_vector(7 downto 0);
          Xout: out std_logic_vector(7 downto 0);
          clk: in std_logic);
```

```
end component;

component modulo
   Port (A,B: in std_logic_vector(7 downto 0);
         C: out std_logic_vector(7 downto 0));
end component;

begin

reg1 : reg port map(Xin => regA ,Xout => Q1, clk => clk);
reg2 : reg port map(Xin => regB ,Xout => Q2, clk => clk);
c1 : comporater port map(Q1 => Q1,Q2 =>Q2 ,R => Cout);
m1 : modulo port map(A => Q1, B => Q2, C=> modout);
mux1: mux port map(A => A, B => tempA, sel => load, f => regA);
mux2: mux port map(A => B, B => tempB, sel => load, f => regB);

process(CS, clk, load)begin

case CS is
when AeqB =>
   if Cout = "00" then
      C <= A;
   end if;
when AltB =>
   if Cout = "01" then
      if (modout = "00000000") then
         C <= A;
      else
         tempB <= modout;
      end if;
   end if;
when AgtB =>
   if Cout = "10" then
      if (modout = "00000000") then
         C <= B;
      else
         tempA <= modout;
      end if;
   end if;
end case;
end process;
end Behavioral;
```

Modulo

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

entity modulo is
    Port (A,B: in std_logic_vector(7 downto 0);
        en: in std_logic;
        C: out std_logic_vector(7 downto 0));
end modulo;
architecture Behavioral of modulo is

begin
process (A,B,en)begin
if (en = '1') then
    if (unsigned(A)>0 and unsigned(B)>0) then
        if (unsigned(A) > unsigned(B)) then
            c<=std_logic_vector(unsigned(A) mod unsigned(B));
        elsif (unsigned(A) < unsigned(B)) then
            c<=std_logic_vector(unsigned(B) mod unsigned(A));
        end if;
    end if;
end if;
end process;
end Behavioral;
```

Comparator

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


entity comporater is
    Port( Q1 : in std_logic_vector(7 downto 0);
          Q2 : in std_logic_vector(7 downto 0);
          R  : out std_logic_vector(1 downto 0));

end comporater;

architecture Behavioral of comporater is

begin
process(Q1,Q2)begin
if (Q1 = Q2) then
    R <= "00";
elsif (Q1 < Q2)then
    R <= "01";
elsif (Q1 > Q2)then
    R <= "10";
else
    R <= "11";
end if;
end process;
end Behavioral;
```

Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity reg is
    Port (Xin: in std_logic_vector(7 downto 0);
          Xout: out std_logic_vector(7 downto 0);
          clk: in std_logic);
end reg;
architecture Behavioral of reg is
begin
process(clk,Xin)begin
    if rising_edge(clk)then
        Xout <= Xin;
    end if;
end process;
end Behavioral;
```

Multiplexer

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

entity mux is
  port (A, B: in std_logic_vector(7 downto 0);
        sel : in std_logic;
        f : out std_logic_vector(7 downto 0));
end mux;

architecture behaviour of mux is
begin
  process (A, B, sel)
  begin
    if sel = '1' then
        f <= B;
    else
        f <= A;
    end if;
end process;
end behaviour;
```