Lab 6: Greatest Common Divisor

Date: 28.11.2022

Name: Orkun İbrahim Kök

Section: EE102-01

Purpose

The aim of this lab is calculating the greatest common divisor of two 8-bit unsigned binary numbers using the Basys3 FPGA. Also, because there are plenty of options while calculating the greatest common divisor of two numbers, choosing one of them and implementing it on VHDL is the another purpose of the experiment.

Design Specifications

The design that I created consists of 5 inputs. They are clock, reset, first input, second input, and enable. When the reset button is pressed, it resets the memory of the Basys3, when enable button is pressed, the greatest common divisor will show up on the LEDs on the Basys3. The output is the binary number which corresponds to the greatest common divisor of the two binary number inputs. The working principle of the greatest common divisor can be explained as follows:

1)Subtracts the smaller number from the bigger number

2)Continues step 1 with replacing bigger number with smaller number and smaller number with difference that is found at the previous step.

3)Whenever the subtrahend value is equal to the difference value, it shows that this number is the greatest common divisor of the two numbers that is inputted earlier.

My module is a FSM. A Moore machine is used in this experiment. It consists of three states which are: Hold, Change, and Set. Combinational circuit will be more rapid and cheaper compared to the FSM's, but there is a drawback of combinational circuits, they have no memory unit, therefore it would be more difficult to calculate greatest common divisor of two unsigned binary numbers compared to FSM's. On the other hand, because Moore machine is a clock dependent operation, it is slower than the combinational circuits. It took 296 clock ticks to reach to the answer. This can be optimized by using a combinational circuit because it is not time dependent.

Methodology

I searched on internet about how to find greatest common divisor of two numbers and found "Euclidian algorithm with subtraction only" method. Figure 1 shows an example about how Euclidian algorithm is used while calculating the greatest common divisor of two numbers. So, I implemented this algorithm to VHDL. While implementing it, I only used two separate untis which are 8-bit comparator and 8-bit subtractor. I looked at the codes that I wrote while creating an ALU, and changed them a little in order to fit in this experiment's requirements.

Let it be required to find the GCD(108, 72):

1. 108 - 72 = 36
2. 72 - 36 = 36
3. 36 - 36 = 0
4. GCD(108, 72) = 36

GCD(44, 60):

1. 60 - 44 = 16
2. 44 - 16 = 28
3. 28 - 16 = 12
4. 16 - 12 = 4
5. 12 - 4 = 8
6. 8 - 4 = 4
7. 4 - 4 = 0
8. GCD(44, 60) = 4

Figure 1: Logic behind the Euclidian algorithm

Results

The schematic of the logic design can be seen in Figure 2.a, 2.b, 2c. Figure 2.a show the main module schematic, Figure 2.b shows the inside of the 8-bit comparator unit schematic and lastly, Figure 2.c shows the inside of the 8-bit subtractor unit schematic.

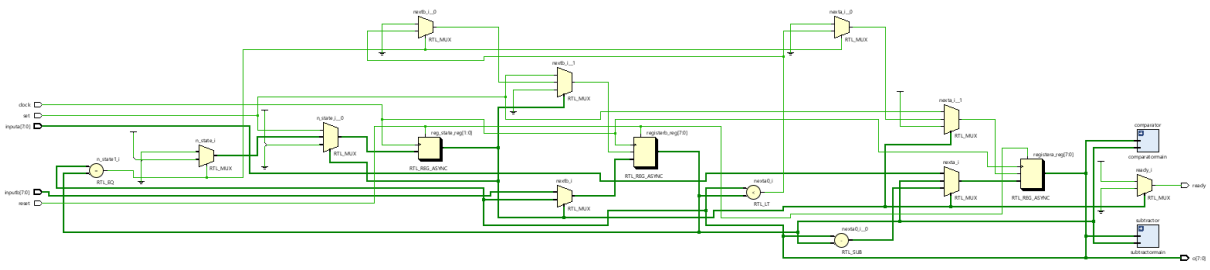

Figure 2.a: Main module schematic

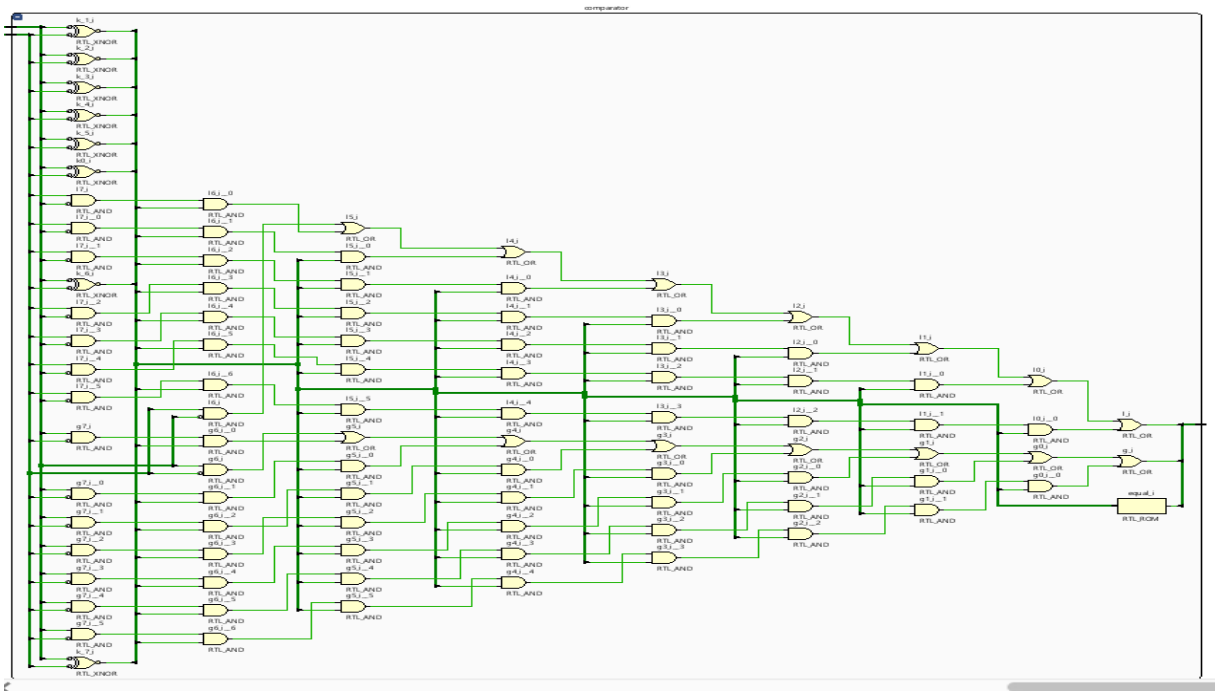

Figure 2.b: 8-bit comparator schematic

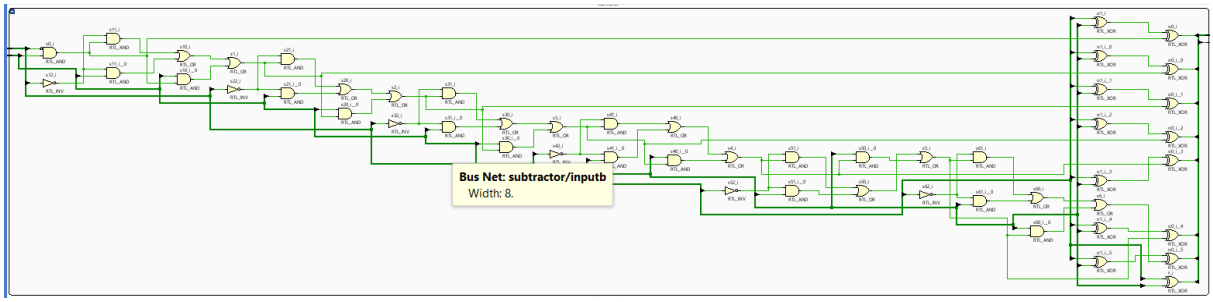

Figure 2.c: 8-bit subtractor schematic

As wanted in the lab introduction, a testbench is created with the input combinations 140 and 12. The corresponding 8-bit binary numbers are “10001100” and “00001100”. As seen in the Figure 3.c, the output values is 4, whose corresponding binary number equivalent is “100”. Also this input combination can be seen in Figure 4.

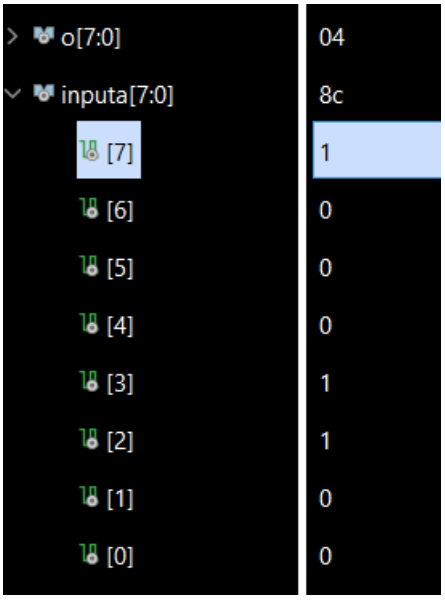

Figure 3.a: First input combination (140)

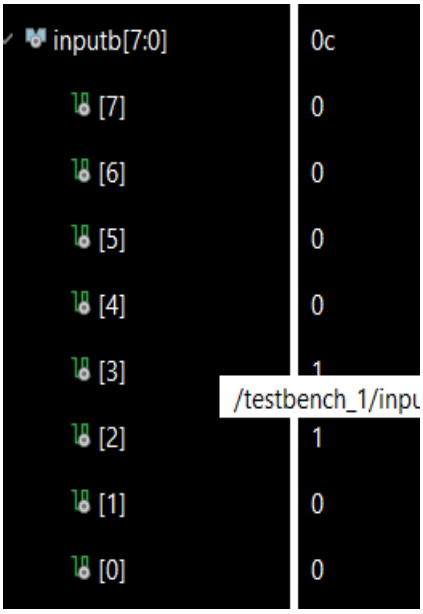

Figure 3.b: Second input combination (12)

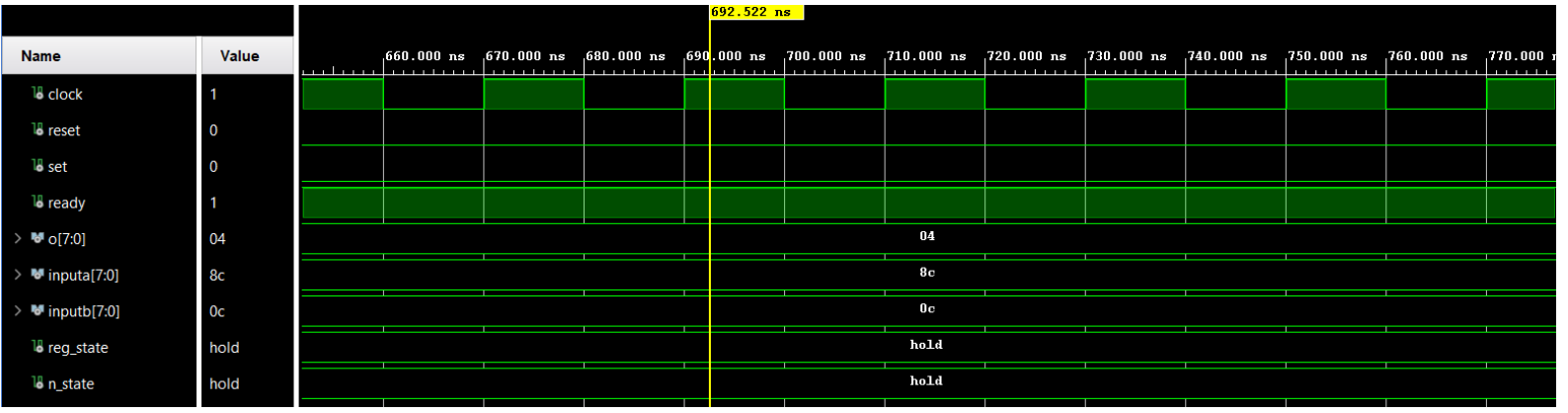

Figure 3.c: Time diagram of the testbench with input combinations
inputa = 140 and inputb = 12

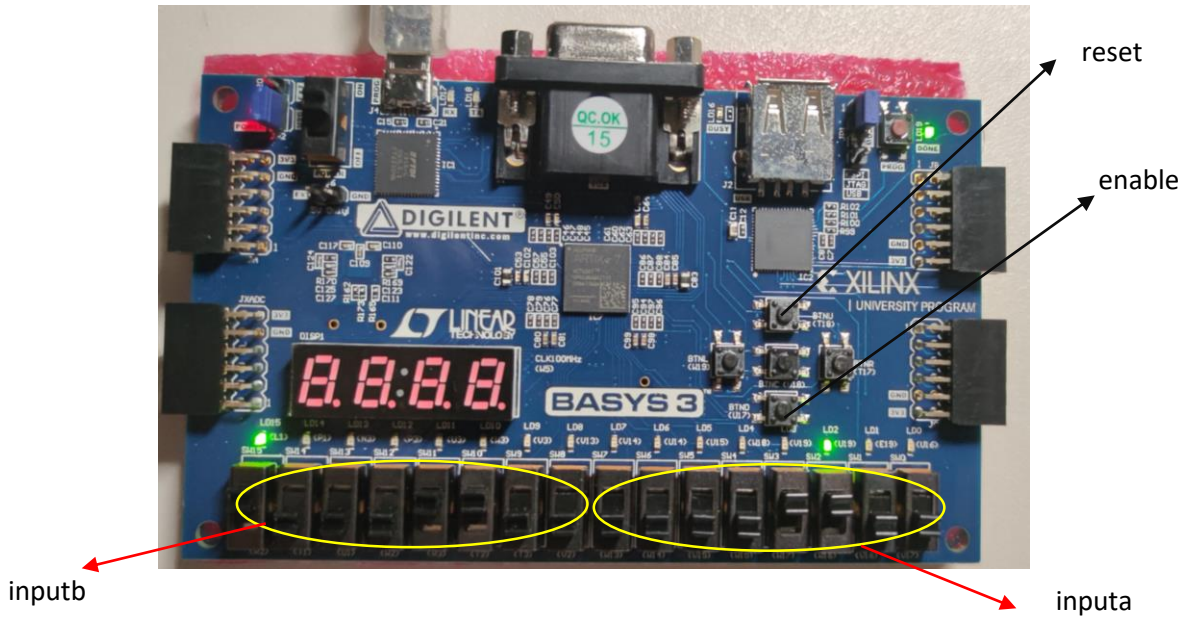

Figure 4: Testbench input configuration

Inputa = “00001100” (12)

Inputb = “10001100” (140)

Output = “100” (4)

Figure 5, 6, and 7 shows some examples with random input value combinations.

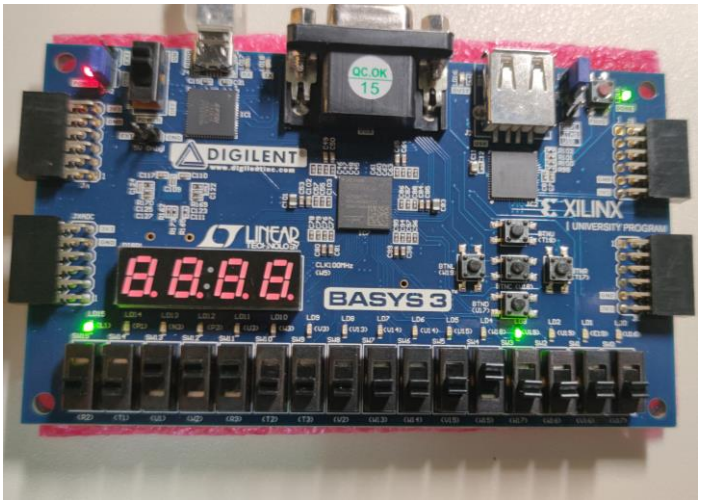

Figure 5: inputa = 56 & inputb = 16

Inputa = “00010000” (16)

Inputb = “00111000” (56)

Output = “1000” (8)

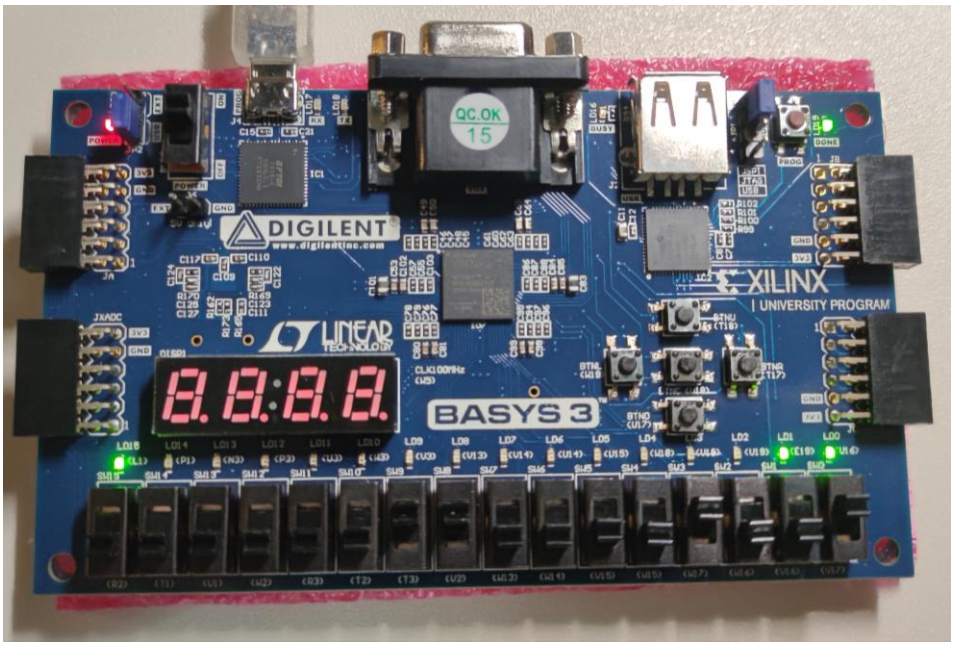

Figure 6: inputa = 9 & inputb = 3

Inputa = “00001001” (9)

Inputb = “00000011” (3)

Output = “011” (3)

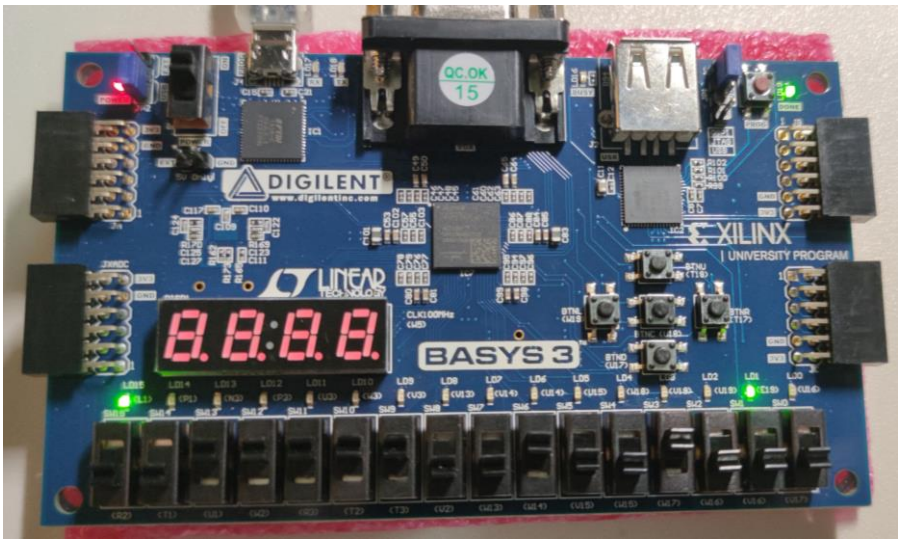

Figure 7: inputa = 62 & inputb = 72

Inputa = “01001000” (72)

Inputb = “00111110” (62)

Output = “010” (2)

Conclusion

The purpose of the lab is creating a circuit that calculates the greatest common divisor of two unsigned binary numbers. I used the Euclidian algorithm with only subtraction method to calculate the greatest common divisor. The result on my Basys3 FPGA are correct and the outputs values are as expected. This lab made me understand the FSM logic and learn how to do the implementation of it to VHDL. Also, while answerieng the questions in the lab report, I understood the difference between FSMs and combinational logic circuits.

References

https://scienceland.info/en/algebra8/euclid-algorithm

http://quitoart.blogspot.com/2017/11/vhdl-greatest-common-divisor-gcd.html

Appendix

gcd_module.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity gcd_module is
  port(
    clock, reset: in std_logic;
    set: in std_logic;
    inputa, inputb: in std_logic_vector(7 downto 0);
    ready: out std_logic;
    o: out std_logic_vector(7 downto 0)
  );
end gcd_module ;

architecture Behavioral of gcd_module is
  type s_type is (hold, change, sub);
  signal reg_state, n_state: s_type;
```

```
  signal registera, nexta, registerb, nextb: unsigned(7 downto 0);
  signal comparatora, comparatorb : unsigned(7 downto 0);
  signal subtractorb, subtractora, subtractorout: std_logic_vector(7 downto 0);
  signal comparatorout: std_logic_vector(2 downto 0);
begin
  comparator: entity work.comparatormain(behavioralc)
      port map(inputa => comparatora, inputb=> comparatorb, o=> comparatorout);
   subtractor: entity work.subtractormain(behaviorals)
      port map(inputa => subtractora, inputb => subtractorb, o => subtractorout);
  process(clock,reset)
  begin
    if reset='1' then
      reg_state <= hold;
      registera <= (others=>'0');
      registerb <= (others=>'0');
    elsif (rising_edge(clock)) then
      reg_state <= n_state;
      registera <= nexta;
      registerb <= nextb;
    end if;
  end process;
  -- next-state logic & data path functional units/routing
  process(reg_state, registera, registerb, set, inputa, inputb)
  begin
    nexta <= registera;
    nextb <= registerb;
    comparatora <= registera;
    comparatorb <= registerb;
    subtractora <= std_logic_vector(registera);
    subtractorb <= std_logic_vector(registerb);
    case reg_state is
```

```
        when hold =>
          if set='1' then
            nexta <= unsigned(inputa);
            nextb <= unsigned(inputb);
            n_state <= change;
          else
            n_state <= hold;
          end if;
        when change =>
          if (registera=registerb) then
            n_state <= hold;
          else
            if (registera < registerb) then
              nexta <= registerb;
              nextb <= registera;
            end if;
            n_state <= sub;
          end if;
        when sub =>
          nexta <= registera - registerb;
          n_state <= change;
        end case;
    end process;
    ready <= '1' when reg_state=hold else '0';
    o <= std_logic_vector(registera);
end Behavioral;
```

comparatormain.vhd

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

```
entity comparatormain is
 port (inputa, inputb: in unsigned(7 downto 0);
 o: out std_logic_vector(2 downto 0)
 );
end entity;
architecture behavioralc of comparatormain is
 signal k: std_logic_vector(7 downto 0);
 signal greater1, greater2: std_logic;
 signal res: std_logic_vector(2 downto 0);
 signal equal, g, l: std_logic;
begin
k(0) <= (not inputa(0)) xnor (not inputb(0));
 k(1) <= (not inputa(1)) xnor (not inputb(1));
 k(2) <= (not inputa(2)) xnor (not inputb(2));
 k(3) <= (not inputa(3)) xnor (not inputb(3));
 k(4) <= (not inputa(4)) xnor (not inputb(4));
 k(5) <= (not inputa(5)) xnor (not inputb(5));
 k(6) <= (not inputa(6)) xnor (not inputb(6));
 k(7) <= (not inputa(7)) xnor (not inputb(7));

 equal<= '1' when k = x"FF" else '0';
g <= (inputa(7) and not(inputb(7)))or
          (inputa(6) and not(inputb(6)) and k(7))or
          (inputa(5) and not(inputb(5)) and k(7) and k(6))or
          (inputa(4) and not(inputb(4)) and k(7) and k(6) and k(5))or
          (inputa(3) and not(inputb(3)) and k(7) and k(6) and k(5) and k(4))or
          (inputa(2) and not(inputb(2)) and k(7) and k(6) and k(5) and k(4) and k(3))or
          (inputa(1) and not(inputb(1)) and k(7) and k(6) and k(5) and k(4) and k(3) and
k(2))or
       (inputa(0) and not(inputb(0)) and k(7) and k(6) and k(5) and k(4) and k(3) and
k(2) and k(1));
```

```
l <= (inputb(7) and not(inputa(7)))or
        (inputb(6) and not(inputa(6)) and k(7))or
        (inputb(5) and not(inputa(5)) and k(7) and k(6))or
        (inputb(4) and not(inputa(4)) and k(7) and k(6) and k(5))or
        (inputb(3) and not(inputa(3)) and k(7) and k(6) and k(5) and k(4))or
        (inputb(2) and not(inputa(2)) and k(7) and k(6) and k(5) and k(4) and k(3))or
        (inputb(1) and not(inputa(1)) and k(7) and k(6) and k(5) and k(4) and k(3) and
k(2))or
 (inputb(0) and not(inputa(0)) and k(7) and k(6) and k(5) and k(4) and k(3) and
k(2) and k(1));
res(2) <= l;
 res(0) <= g;
 res(1) <= equal;
 o <= res;
end behavioralc;
```

subtractormain.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity subtractormain is
    port (inputa, inputb: IN STD_LOGIC_VECTOR ( 7 DOWNTO 0);
        o : OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0);
        t: out std_logic
        );
end subtractormain;
architecture behaviorals of subtractormain is
 signal s0,s1,s2,s3,s4,s5,s6,s7: std_logic;
begin
 t <= inputa(0) XOR inputb(0) xor '0';
 o(0) <= inputa(0) XOR inputb(0) xor '0';
 s0 <= (not(inputa(0)) and '0') or (not(inputa(0)) and inputb(0)) or (inputb(0) and '0');
```

```
 o(1) <= inputa(1) XOR inputb(1) xor s0;
 s1 <= (not(inputa(1)) and s0) or (not(inputa(1)) and inputb(1)) or (inputb(1) and s0) ;
 o(2) <= inputa(2) XOR inputb(2) xor s1;
 s2 <= (not(inputa(2)) and s1) or (not(inputa(2)) and inputb(2)) or (inputb(2) and s1);
 o(3) <= inputa(3) XOR inputb(3) xor s2;
 s3 <= (not(inputa(3)) and s2) or (not(inputa(3)) and inputb(3)) or (inputb(3) and s2);
 o(4) <= inputa(4) XOR inputb(4) xor s3;
 s4 <= (not(inputa(4)) and s3) or (not(inputa(4)) and inputb(4)) or (inputb(4) and s3);
 o(5) <= inputa(5) XOR inputb(5) xor s4;
 s5 <= (not(inputa(5)) and s4) or (not(inputa(5))     and inputb(5)) or (inputb(5) and s4);
 o(6) <= inputa(6) XOR inputb(6) xor s5;
 s6 <= (not(inputa(6)) and s5) or (not(inputa(6)) and inputb(6)) or (inputb(6) and s5);
 o(7) <= inputa(7) XOR inputb(7) xor s6;
end behaviorals;
```

testbench_1.vhd

```
entity testbench_1 is
end testbench_1;

architecture Behavioral of testbench_1 is
   signal clock, reset, set, ready: std_logic;
   signal o, inputa, inputb : std_logic_vector (7 downto 0);

   type s_type is (hold, change, sub);
   signal reg_state, n_state : s_  type;
begin
     dut: entity work.gcd_module(Behavioral)
       port map(clock, reset, set, inputa, inputb, ready, o);
    clock_process: process begin
       clock <= '0';
       wait for 10ns;
```

```
        clock <= '1';
        wait for 10ns;
    end process;
    stim_process     : process begin
        set <= '1';
        inputa <= "10001100";
        inputb <= "00001100";
        reset <= '0';
        wait for 100ns;
        set     <= '0';
        wait;
    end process;
end Behavioral;
```