Mustafa Bahaeddin Bozan

22102838

EEE-102-001

LAB6: Greatest Common Divisor

A) Introduction

The aim of the 6th lab session is to build a circuit that can compute the greatest common divisor of two numbers by utilising one of the GCD algorithms. Also the circuit should contain an assigned button feature that starts the computation.

B) Questions to be answered

- What is your algorithm to calculate GCD?

I have implemented Euclidean algorithm. It is explained in detail in the Methodology section.

- Is your module a combinational circuit or an FSM? If the latter, is it a Moore machine or a Mealy machine? Would it be cheaper to implement GCD as a combinational circuit or as an FSM? Would it be faster? What are the drawbacks in each case?

Both designs explained in the Methodology section are examples of Moore machines since they tend to change at the rising edges of the clock cycles and the output of the one step becomes the input of the next step. Design of FSMs seems easier to me, but I can think about the cost part that combinational circuits may cost more transistors although in terms of time cost it is cheaper. Besides, Mealy machines are faster, but they are open to get errors; therefore, they are not preferred in this lab session.

- How many clock cycles did it take in your simulation to calculate the GCD? Do you think this can be optimized? How so?

In my implementation and in the best case, where input numbers are equal, it will take one clock cycle to check the equality and set the next state to done state. Then in the second cycle the execution will end. However, if the controls increase, the result comes more clock cycles later. In order to optimize the time cost, we can finish whenever we find the equality of the numbers or choose another algorithm to calculate the GCD. However, since their implementation is harder, I sacrificed from the time it takes to calculate.

C) Methodology

Before building the circuit, one needs to comprehend how Euclidean algorithm works on two numbers. It can simple be told as follows. If both numbers are equal, then the algorithm is done and the GCD is that same number. If they are not equal, then from the larger one the smaller one is subtracted, and the larger number is updated. This operation continues until both numbers become equal to each other. How this operation works can be followed from Table 1.

Num1	Num2	Relation
28	28	Num1 = Num2
GCD = 28		
48	36	Num1 > Num2
48-36=12	36	Num2 > Num1
12	36-12=24	Num2 > Num1
12	24-12=12	Num1 = Num2
GCD = 12		
5	2	Num1 > Num2
5-2=3	2	Num1 > Num2
3-2=1	2	Num2 > Num1
1	2-1=1	Num1 = Num2
GCD = 1		

Table 1. How Euclidean Algorithm works.

There are two simple ways to build a circuit to calculate GCD and both of them will be explained in this section. The first one is easy to understand but it is also time consuming to implement.

The first way uses a datapath that includes 2 2-to-1 multiplexers and 3 registers, a control unit that determines which states is the next and gives feedback to the datapath, and also a top module that ensures the feedback mechanism between the datapath and the control unit. The top module schematic and inside of the datapath can be seen from the Figure 1-2 respectively.

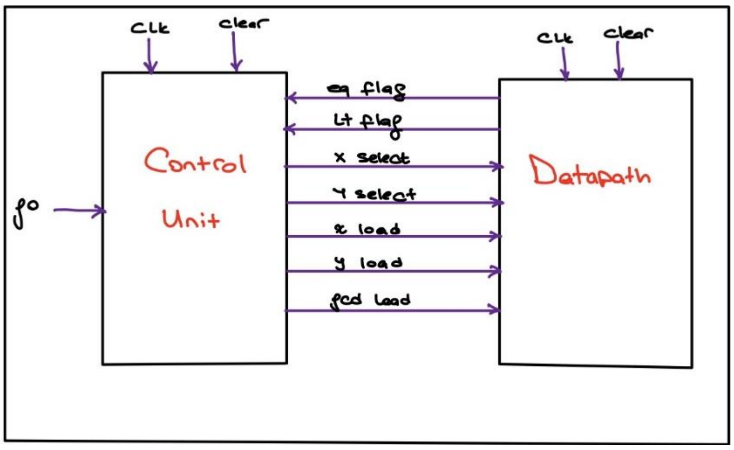

Figure 1. Top module schematic

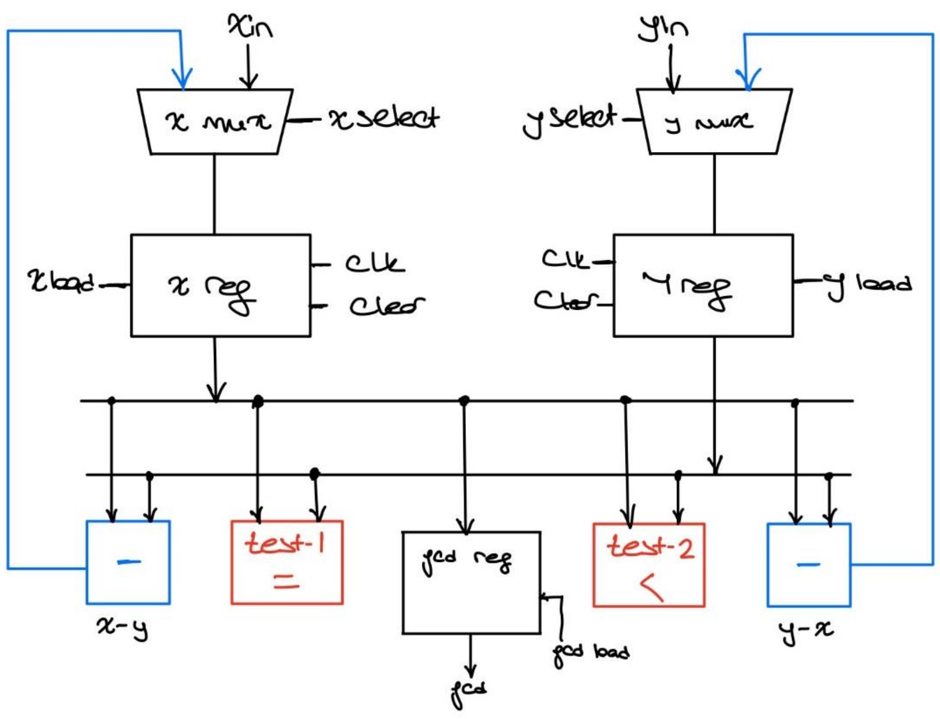

Figure 2. Inside of the datapath.

With the given inputs for numbers and go flag the computation starts. Inside the datapath, input numbers enter to the mux and with the load signals they are loaded to the number signals. Then, the tests are completed. All this process is controlled by the signals between the datapath and the control unit. Also, simple schematic of how this method works can be seen from the Figure 3.

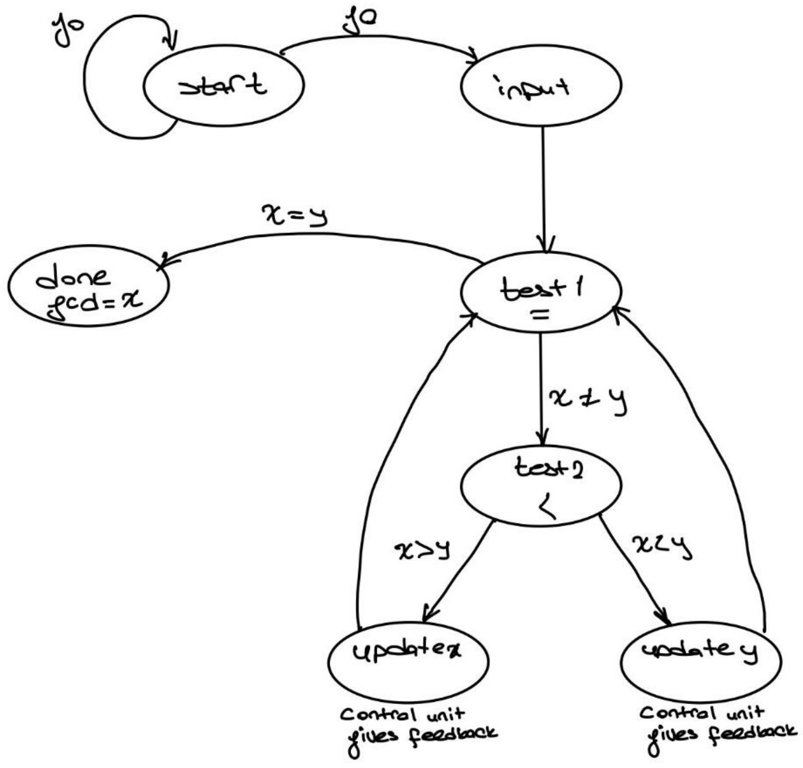

Figure 3. How datapath & control unit method works.

To start designing, if one does not want to use the available libraries, first of all, a full adder and an 8-bit subtractor should be designed. Then, one can move on to building mux and registers. The next step is to design the datapath which uses 8-bit subtractor, mux and register as components. After designing the datapath, one should design control unit which controls the state in which Moore machine is working and to which Moore machine will move on. The last step of this design is creating a top module which ensures the communication between datapath and control unit.

Although the first design is easier to follow there is a second method that can be coded much more easily.

In the second design we start with declaring 3 8-bit vectors for input numbers and GCD output, a clock, and a logic input for starting the operation. Then in the architecture of the code one needs to create 3 8-bit vector signals. Two of them are for changing the values of the input according to the equality and less than tests and the other one is for conveying the last result to GCD output. Also, one needs to create 3 8-bit vector signals to replicating the work of registers. Since we are using states for determining the next step of the operation, one should create 3 logic signals for present, next and final states. Then, one needs to write two different processes in the architecture of the file one of them for conveying the input numbers or updated numbers to the registers in each clock cycle, the other one is for defining what to do in different conditions and defining the states. Although it seems very short code comparing with the first design, this design is hard to follow from the codes.

D) Results

Firstly, I tried the first way I mentioned in the Methodology part. However, I got "0" for the GCD in every testbench simulation and I could not debug the code. Therefore, I implemented the second method. After completing the code file, I coded the testbench and the simulation gave the correct result. The result of the simulation can be seen from the Figure 1.

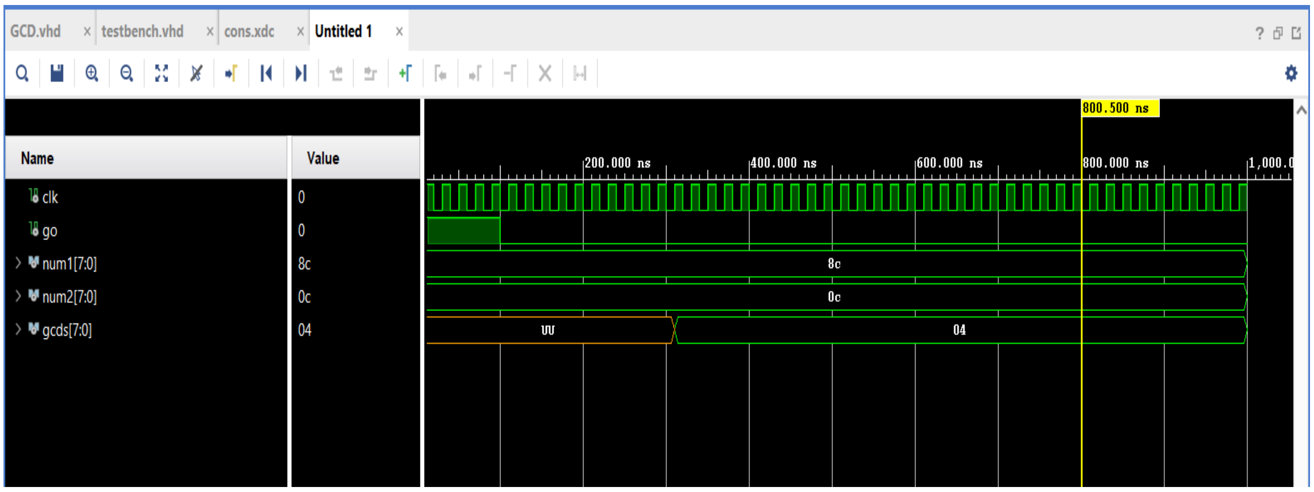

Figure 1. The result of the simulation.

Then, I coded the constrain file and completed the synthesis, implementation and generating the bitstream parts. Before, programming the BASYS3 FPGA board I checked the elaborated design and it can be seen from the Figure 2.

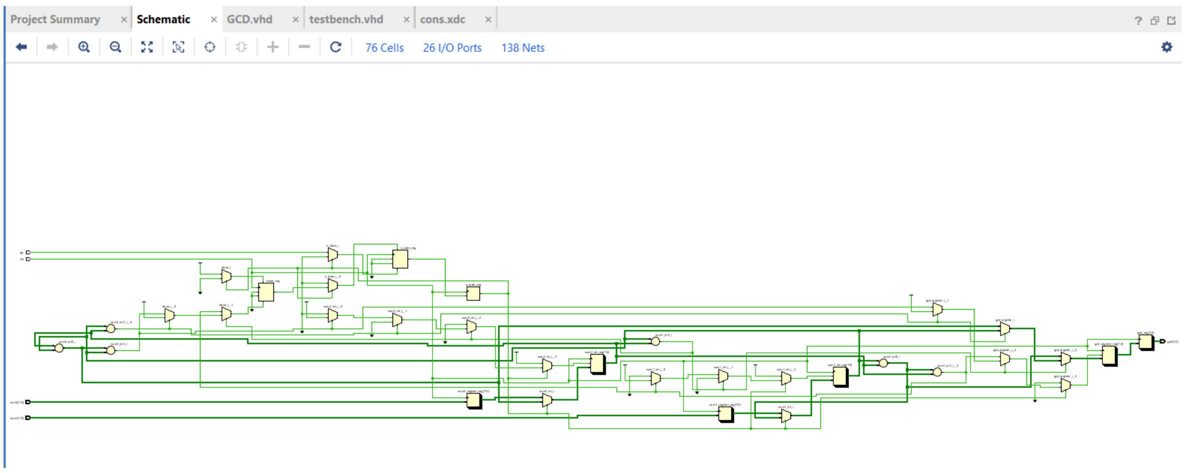

Figure 2. The elaborated design of the designed circuit.

If the first implementation had been done correctly the elaborated design would have been like the Figure 3.

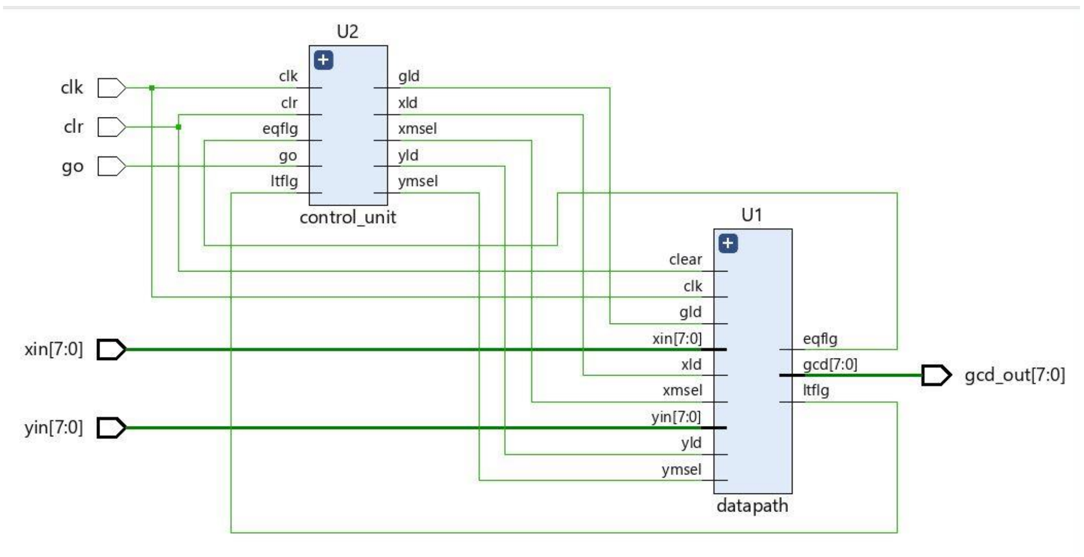

Figure 3. The elaborated design of the first way of implementation.

As it can be seen from the Figure 2 and Figure 3, the first implementation has a compact design, and the second implementation is much more likely to a combinational circuit design although both works as a Moore machine.

After these steps, I moved to program the device and tested the design manually with different inputs. The result can be seen from the Figure 4-8.

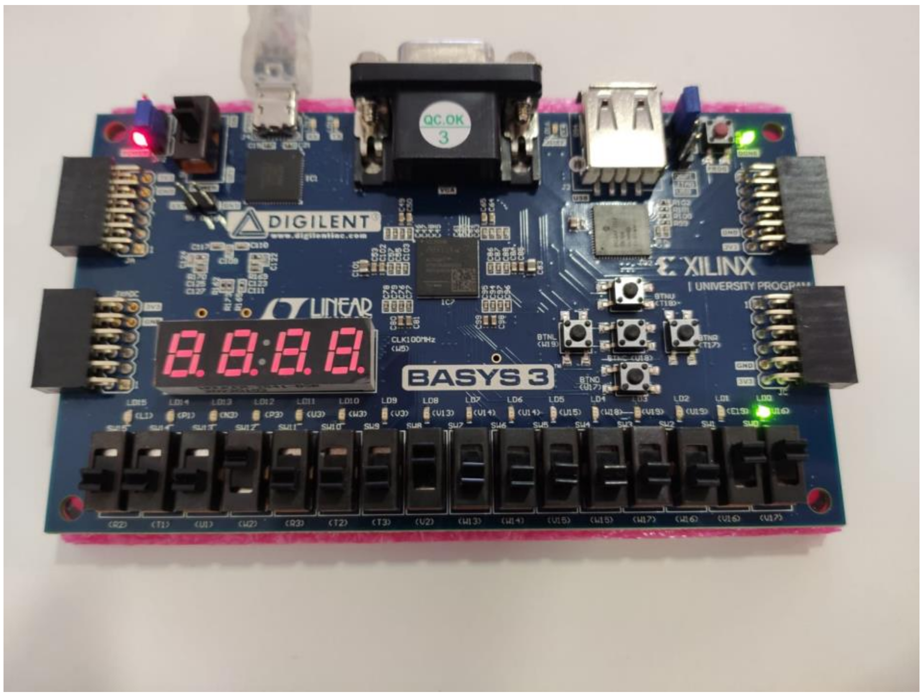

Figure 4. Num1 = “00010001$_2$” (17$_{10}$) Num2 = “00000011$_2$” (3$_{10}$) GCD = “00000001$_2$” (1$_{10}$).

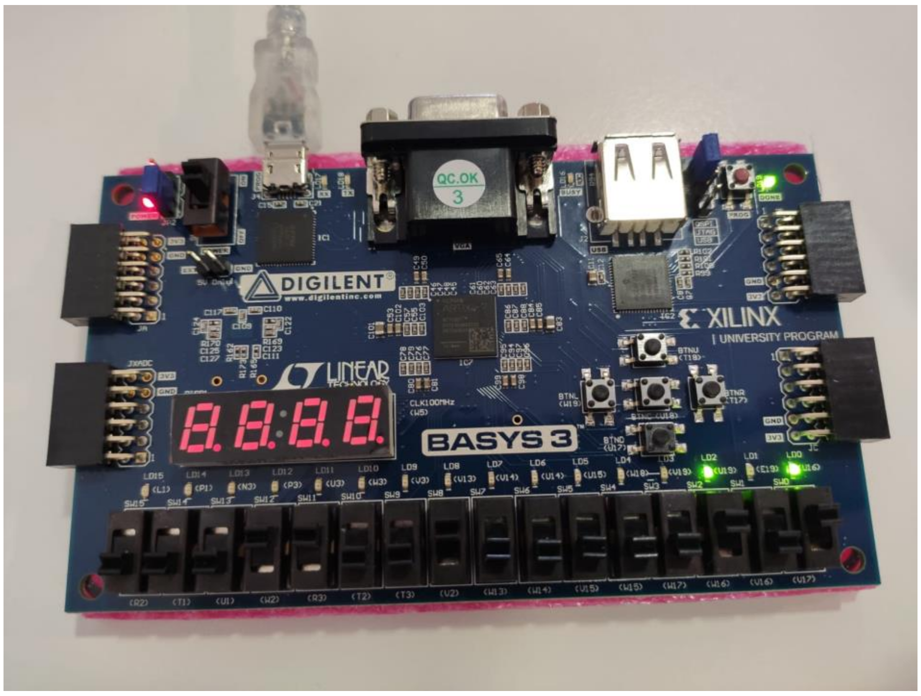

Figure 5. Num1 = “00011001$_2$” (25$_{10}$) Num2 = “00000101$_2$” (5$_{10}$) GCD = “00000101$_2$” (5$_{10}$).

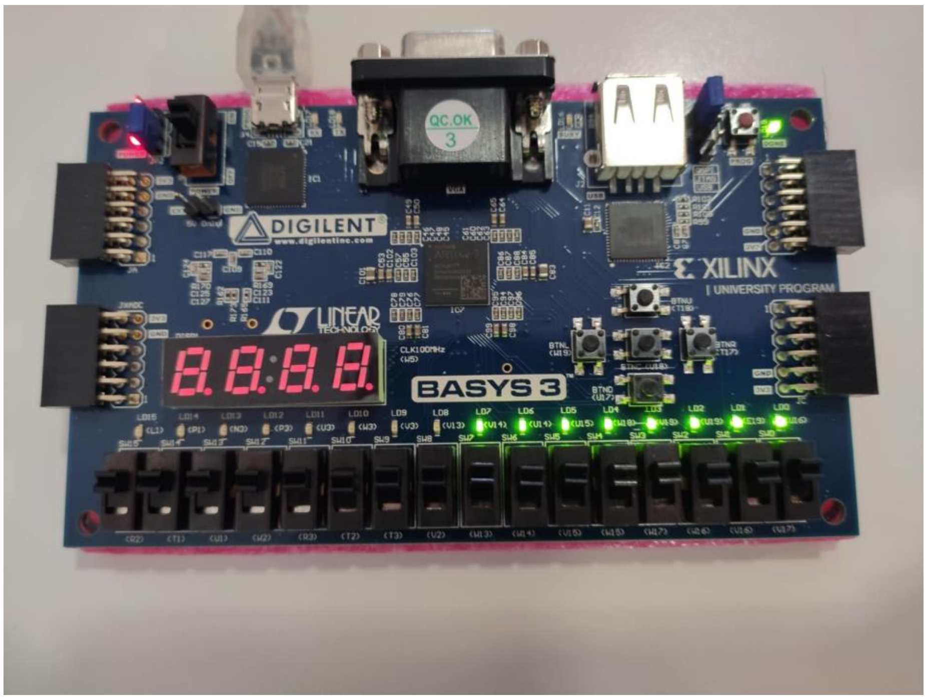
Figure 6. Num1 = “11111111_2” (255_{10}) Num2 = “11111111_2” (255_{10}) GCD = “11111111_2” (255_{10}).

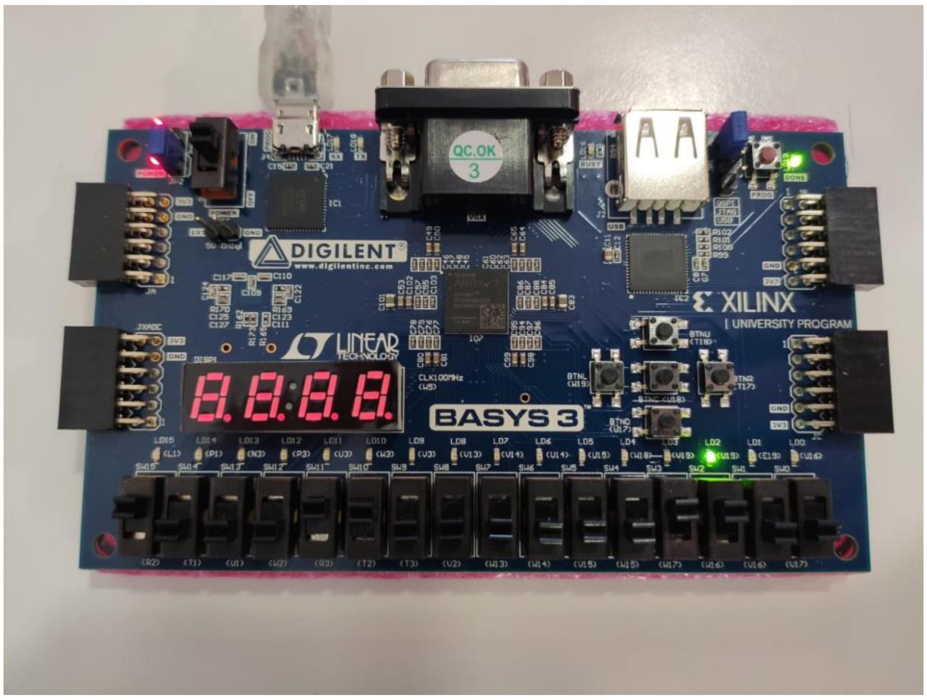
Figure 5. Num1 = “10001100_2” (140_{10}) Num2 = “000001100_2” (12_{10}) GCD = “00000100_2” (4_{10}).

E) Conclusion

The 6th lab session is about to design a circuit that finds the GCD of two numbers given. This task can be completed with different designs. For example, one can use the combinational circuits or utilise the FSMs. These implementations have their own downsides. Although FSM designs are cheaper than the combinational circuits, they are slower. However, implementation of the FSM seemed easier; therefore, I used FMS design in my solution. Also, this design can be classified as a Moore machine since after the given input, present state's output determines the next step's input.

With this lab session I learned that many projects can be completed with many ways since I got many different errors while implementing the design and changed my method. Although I could not debug my code, I understood the importance of debugging skills and I will work on this aspect of the designs by thinking about the possible cause of the error in the following projects.

In conclusion, the 6th lab session is very informative and educative in terms of building a circuit that uses memory and has compact designs. Also, before starting to the project I searched the FSM designs and learned a lot about them. Therefore, this lab session encourages us to do some search about different concepts about digital circuit designs. Besides, this lab session made me start to think about the transistor and time costs of the different implementations of the same project which will be very helpful for the following labs and my term project.

F) Appendix

1. For the method I used

1- Design Code

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.numeric_std.ALL;


entity GCD is

   port(

   num1:   in std_logic_vector(7 downto 0);

   num2:   in std_logic_vector(7 downto 0);

   go:     in std_logic;

   gcd:    out std_logic_vector(7 downto 0);

   clk:    in std_logic);

end GCD;


architecture gcd_arch of GCD is


signal num1_register:   std_logic_vector(7 downto 0);
signal num2_register:    std_logic_vector(7 downto 0);
signal gcd_register:     std_logic_vector(7 downto 0);


signal num1_int:         integer range 0 to 255;
signal num2_int:         integer range 0 to 255;


signal p_state:          std_logic:= '0';
signal  n_state: std_logic:= '0';
signal done:     std_logic:= '0';


begin
Clock: process(clk) begin
        if clk'event and clk= '1' then
                p_state <=  n_state;
                num1_register  <= num1;
                num2_register  <= num2;
                gcd <= gcd_register;
        end if;
end process;


GCDP: process(clk)
variable difference: integer range 0 to 255:=0;
begin
        if rising_edge(clk) then
                if p_state = '0'then
                        num1_int        <= to_integer(unsigned(num1_register));
                        num2_int        <= to_integer(unsigned(num2_register));
                        done <= '0';
                        if go = '1' then
                                n_state <= '1';
                        end if;
                elsif p_state = '1' then
                        if done = '1' then
                                 n_state <= '0';
                        end if;
                        if num1_int > num2_int then
                        difference := num1_int - num2_int;
                        if difference = num2_int then
                                gcd_register <= std_logic_vector(to_unsigned(difference, 8));
                                done <= '1';
                        else
                                num1_int <= difference;
                        end if;
                elsif num1_int < num2_int then
                        difference := num2_int - num1_int;
                        if difference = num1_int then
                                gcd_register <= std_logic_vector(to_unsigned(difference, 8));
                                done <= '1';
                        else
                                num2_int <= difference;
                        end if;
                else
                        gcd_register <= std_logic_vector(to_unsigned(num1_int, 8));
                        done <= '1';
                  end if;
                end if;
        end if;
end process;

end gcd_arch;
```

2- Testbench Code

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

entity testbench is
end testbench;

architecture tb_arch of testbench is

component GCD
   port(
   num1:    in std_logic_vector(7 downto 0);
   num2:   in std_logic_vector(7 downto 0);
          go:                         in std_logic;
   gcd:        out std_logic_vector(7 downto 0);
   clk:          in std_logic);
end component;

signal clk: std_logic;
signal go: std_logic;
signal num1: std_logic_vector (7 downto 0) := "00000000";
signal num2: std_logic_vector (7 downto 0) := "00000000";
signal gcds: std_logic_vector (7 downto 0);

begin
UUT: GCD port map(
clk => clk,
num1 => num1,
num2 => num2,
go => go,
gcd => gcds);
```

```
clk_p: process
begin
clk <= '0';
wait for 10 ns;
clk <= '1';
wait for 10 ns;
end process;

stim_p: process
begin
num1 <= "10001100";
num2 <= "00001100";
go <= '1';
wait for 100 ns;
go <= '0';
wait;
end process;
end tb_arch;
```

3- Constrain

```
set_property PACKAGE_PIN W5 [get_ports clk]

        set_property IOSTANDARD LVCMOS33 [get_ports clk]
        create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]


set_property PACKAGE_PIN V17 [get_ports {num1[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[0]}]
set_property PACKAGE_PIN V16 [get_ports {num1[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[1]}]
set_property PACKAGE_PIN W16 [get_ports {num1[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[2]}]
set_property PACKAGE_PIN W17 [get_ports {num1[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[3]}]
set_property PACKAGE_PIN W15 [get_ports {num1[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[4]}]
set_property PACKAGE_PIN V15 [get_ports {num1[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[5]}]
set_property PACKAGE_PIN W14 [get_ports {num1[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[6]}]
set_property PACKAGE_PIN W13 [get_ports {num1[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num1[7]}]
set_property PACKAGE_PIN V2 [get_ports {num2[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[0]}]
set_property PACKAGE_PIN T3 [get_ports {num2[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[1]}]
set_property PACKAGE_PIN T2 [get_ports {num2[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[2]}]
set_property PACKAGE_PIN R3 [get_ports {num2[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[3]}]
set_property PACKAGE_PIN W2 [get_ports {num2[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[4]}]
set_property PACKAGE_PIN U1 [get_ports {num2[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[5]}]
set_property PACKAGE_PIN T1 [get_ports {num2[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[6]}]
set_property PACKAGE_PIN R2 [get_ports {num2[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {num2[7]}]

set_property PACKAGE_PIN U16 [get_ports {gcd[0]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[0]}]
set_property PACKAGE_PIN E19 [get_ports {gcd[1]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[1]}]
set_property PACKAGE_PIN U19 [get_ports {gcd[2]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[2]}]
set_property PACKAGE_PIN V19 [get_ports {gcd[3]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[3]}]
set_property PACKAGE_PIN W18 [get_ports {gcd[4]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[4]}]
set_property PACKAGE_PIN U15 [get_ports {gcd[5]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[5]}]
set_property PACKAGE_PIN U14 [get_ports {gcd[6]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[6]}]
set_property PACKAGE_PIN V14 [get_ports {gcd[7]}]
        set_property IOSTANDARD LVCMOS33 [get_ports {gcd[7]}]

set_property PACKAGE_PIN U18 [get_ports go]
        set_property IOSTANDARD LVCMOS33 [get_ports go]
```

2. <u>The method I got wrong gcd in testbench</u>

2.1- <u>Full Adder</u>

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

entity full_adder is
   Port ( b1 : in STD_LOGIC;
          b2 : in STD_LOGIC;
          cin : in STD_LOGIC;
          s : out STD_LOGIC;
          cout : out STD_LOGIC);
end full_adder;

architecture fa_arch of full_adder is

begin
```

```
s <= b1 XOR b2 XOR cin;
cout <= (b1 AND b2) OR (cin AND b1) OR (cin OR b2);
end fa_arch;
```

2.2- <u>Subtractor</u>

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

entity substract is
   Port ( num1 : in STD_LOGIC_VECTOR (7 downto 0);
          num2 : in STD_LOGIC_VECTOR (7 downto 0);
          res : out STD_LOGIC_VECTOR (7 downto 0));
end substract;

architecture sub_arch of substract is

signal num2_c : std_logic_vector (7 downto 0);
signal Cin: std_logic := '1';
signal C: std_logic_vector (7 downto 0);

component full_adder
   Port ( b1 : in STD_LOGIC;
          b2 : in STD_LOGIC;
          cin : in STD_LOGIC;
          s : out STD_LOGIC;
          cout : out STD_LOGIC);
end component;
begin
num2_c <= not num2;
s0: full_adder port map (b1 => num1(0), b2 => num2_c(0), cin => Cin, s => res(0), cout
=>C(0));
s1: full_adder port map (b1 => num1(1), b2 => num2_c(1), cin => C(0), s => res(1), cout
=> C(1));
```

```
s2: full_adder port map (b1 => num1(2), b2 => num2_c(2), cin => C(1), s => res(2), cout
=> C(2));

s3: full_adder port map (b1 => num1(3), b2 => num2_c(3), cin => C(2), s => res(3), cout
=> C(3));

s4: full_adder port map (b1 => num1(4), b2 => num2_c(4), cin => C(3), s => res(4), cout
=> C(4));

s5: full_adder port map (b1 => num1(5), b2 => num2_c(5), cin => C(4), s => res(5), cout
=> C(5));

s6: full_adder port map (b1 => num1(6), b2 => num2_c(6), cin => C(5), s => res(6), cout
=> C(6));

s7: full_adder port map (b1 => num1(7), b2 => num2_c(7), cin => C(6), s => res(7), cout
=> C(7));


end sub_arch;
```

2.3- <u>Mux</u>

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.std_logic_unsigned.all;


entity mux is

  Port ( numin : in STD_LOGIC_VECTOR (7 downto 0);

      numup : in STD_LOGIC_VECTOR (7 downto 0);

      sel : in STD_LOGIC;

      num : out STD_LOGIC_VECTOR (7 downto 0));

end mux;


architecture m_arch of mux is

begin

process (numin, numup, sel) begin

if sel = '1' then

  num <= numin;

else

  num <= numup;

end if;
end process;
end m_arch;
```

2.4- Register

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

entity reg is
   Port ( clk : in STD_LOGIC;
          clr : in STD_LOGIC;
          load : in STD_LOGIC;
          data : in STD_LOGIC_VECTOR (7 downto 0);
          q : out STD_LOGIC_VECTOR (7 downto 0));
end reg;

architecture r_arch of reg is
begin

process(clk) begin
   if clr = '1' then
      q <= "00000000";
   elsif rising_edge(clk) then
      if load = '1' then
         q <= data;
      end if;
   end if;
end process;

end r_arch;
```

2.5- Datapath

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity reg is
   Port ( clk : in STD_LOGIC;
          clr : in STD_LOGIC;
          load : in STD_LOGIC;
          data : in STD_LOGIC_VECTOR (7 downto 0);
          q : out STD_LOGIC_VECTOR (7 downto 0));
end reg;

architecture r_arch of reg is
begin

process(clk) begin
   if clr = '1' then
      q <= "00000000";
   elsif rising_edge(clk) then
      if load = '1' then
         q <= data;
      end if;
   end if;
end process;
end r_arch;
```

2.6- <u>Control Unit</u>

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

entity c_unit is
   Port ( clk : in STD_LOGIC;
          clear : in STD_LOGIC;
          goflag : in STD_LOGIC;
```

```
        eqflag : in STD_LOGIC;
        ltflag : in STD_LOGIC;
        xsel : out STD_LOGIC;
        ysel : out STD_LOGIC;
        xload : out STD_LOGIC;
        yload : out STD_LOGIC;
        gcdload : out STD_LOGIC);
end c_unit;

architecture cu_arch of c_unit is

type state_type is (go, test1, test2, up1, up2, done);
signal present_state, next_state: state_type;

begin

process (clk,clear) begin
   if clear = '1' then
      present_state <= go;
   elsif rising_edge(clk) then
      present_state <= next_state;
   end if;
end process;

process (present_state,goflag, eqflag,ltflag) begin
   case present_state is
      when go =>
         if goflag = '1' then
            next_state <= test1;
         else
            next_state <= go;
         end if;
        when test1 =>
            if eqflag = '1' then
                next_state <= done;
            else
                next_state <= test2;
            end if;
        when test2 =>
            if ltflag = '1' then
                next_state <= up1;
            else
                next_state <= up2;
            end if;
        when up1 =>
            next_state <= test1;
        when up2 =>
            next_state <= test1;
        when done =>
            next_state <= done;
        when others =>
            null;
    end case;
end process;

process (present_state) begin
    xload <= '0'; yload <= '0'; gcdload <= '0'; xsel <= '0'; ysel <= '0';
    case present_state is
        when go => xload <= '1'; yload <= '1'; xsel <= '1'; ysel <= '1';
        when up1 => yload <= '1';
        when up2 => xload <= '1';
        when others => gcdload <= '1';
        end case;
end process;
end cu_arch;
```

2.7- <u>Top Module</u>

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

entity top_module is
   Port ( clk : in STD_LOGIC;
          clear : in STD_LOGIC;
          goflag : in STD_LOGIC;
          xin : in STD_LOGIC_VECTOR (7 downto 0);
          yin : in STD_LOGIC_VECTOR (7 downto 0);
          gcd_out : out STD_LOGIC_VECTOR (7 downto 0));
end top_module;

architecture tm_arch of top_module is

component datapath
   Port ( clk : in STD_LOGIC;
          clear : in STD_LOGIC;
          xsel : in STD_LOGIC;
          ysel : in STD_LOGIC;
          xload : in STD_LOGIC;
          yload : in STD_LOGIC;
          gcdload : in STD_LOGIC;
          xin : in STD_LOGIC_VECTOR (7 downto 0);
          yin : in STD_LOGIC_VECTOR (7 downto 0);
          eqflag : out STD_LOGIC;
          ltflag : out STD_LOGIC;
          gcd : out STD_LOGIC_VECTOR (7 downto 0));
end component;

component c_unit
```

```
    Port ( clk : in STD_LOGIC;
           clear : in STD_LOGIC;
           goflag : in STD_LOGIC;
           eqflag : in STD_LOGIC;
           ltflag : in STD_LOGIC;
           xsel : out STD_LOGIC;
           ysel : out STD_LOGIC;
           xload : out STD_LOGIC;
           yload : out STD_LOGIC;
           gcdload : out STD_LOGIC);
end component;

signal eqflag,ltflag,xsel,ysel: std_logic;
signal xload, yload, gcdload: std_logic;

begin

dp: datapath port map(clk => clk, clear => clear, xsel => xsel, ysel => ysel, xload =>
xload, yload => yload, gcdload => gcdload, xin => xin, yin => yin, gcd => gcd_out,
eqflag => eqflag, ltflag => ltflag);
cu: c_unit port map(clk => clk, clear => clear, goflag => goflag, xsel => xsel, ysel =>
ysel, xload => xload, yload => yload, gcdload => gcdload, eqflag => eqflag, ltflag =>
ltflag);
end tm_arch;
```

2.8- Testbench

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

entity testbench is
end testbench;

architecture tb_arch of testbench is

component top_module
```

```
    Port ( clk : in STD_LOGIC;
      clear : in STD_LOGIC;
      goflag : in STD_LOGIC;
      xin : in STD_LOGIC_VECTOR (7 downto 0);
      yin : in STD_LOGIC_VECTOR (7 downto 0);
      gcd_out : out STD_LOGIC_VECTOR (7 downto 0));
end component;

signal clk: std_logic:= '1';
signal clear: std_logic:= '1';
signal goflag: std_logic;
signal x: std_logic_vector (7 downto 0):= "00000000";
signal y: std_logic_vector (7 downto 0):= "00000000";
signal gcd: std_logic_vector (7 downto 0);
signal xsel: std_logic;
signal ysel: std_logic;
signal xload: std_logic;
signal yload: std_logic;
signal gcdload: std_logic;
signal eqflag: std_logic;
signal ltflag: std_logic;



begin

UUT: top_module port map
(
clk => clk,
clear => clear,
goflag => goflag,
xin => x,
yin => y,
gcd_out => gcd);

stim_proc: process
begin
goflag <= '1';
x <= "01010000";
y <= "00100100";
wait for 25 ns;
clear <= '0';
wait;
end process;

end tb_arch;
```