Lab 6: Greatest Common Divisor (GCD)

Date: 13.11.2023

Name: Şevval ERBAY

Section: 2

Purpose:

The purpose of this lab is to design a circuit that calculates the greatest common divisor (GCD) of two 8- bit numbers by using VHDL.

Methodology:

Someone can find the greatest common divisor of two numbers in many ways. In this design, the Euclidean algorithm is implemented. The algorithm works like this: suppose you have two numbers **a** and **b**;

- if **a** = **b** then **GCD** = **a** = **b**,
- if **a** > **b** then **GCD** is found by subtracting **b** from **a** until **a** becomes equal to **b**
- if **b** > **a** then **GCD** is found by subtracting **a** from **b** until **b** becomes equal to **a**

In order to implement this algorithm, registers are used to create a finite-state machine (FSM). This FSM is specifically a Moore machine with 3 different states. In a Moore machine, the output depends solely on the chosen states, not on the inputs (a Moore machine gets help from the internal clock). This calculation could be made with a combinational circuit too. If this design was a combinational circuit, it would give the result faster. Also, in small computations a combinational circuit is cheaper to construct. All in all, when implementing this GCD, a combinational circuit would be faster and cheaper. On the other hand, as the number of the bits gets larger, a FSM would be more optimal since a combinational circuit has to compute the corresponding truth tables (also the cost and computation time would go up).

The 8-bit numbers are indicated by the switches and the GCD is shown by using the 8 right-most LEDs. The RTL schematic of gcd_top.vhd is shown below. We should also note that this is a modular design which uses two sub-modules: subtractor.vhd and comparator.vhd. The states of the FSM are:

- *equal*: This is the state where **a** = **b**. If this condition is not satisfied, the next state, *change*, occurs.
- *change*: This is the state where **b** > **a**. Then, **a** and **b** becomes interchanged by the help of registers and the new values goes into the next state, which is *subtract*.
- *subtract*: This is the state where **a** > **b**. subtractor.vhd module becomes active and **b** is subtracted from **a**. Then the new values get reevaluated and the process continues.

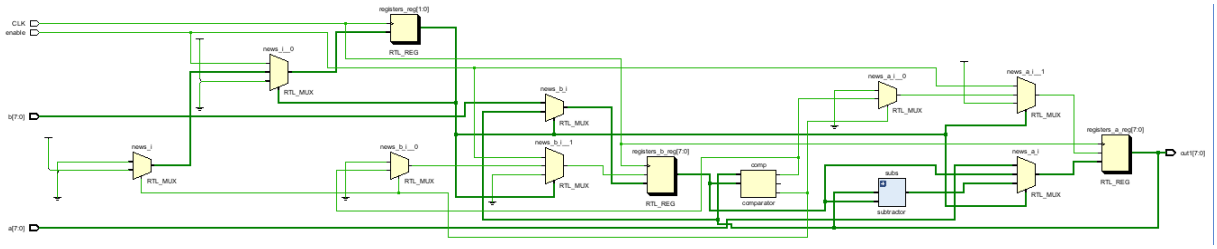


Figure.1 RTL Schematic of gcd_top.vhd

A subtractor is implemented by using "*ieee.std_logic_unsigned*" package. The RTL schematic of subtractor.vhd is shown below. This module subtracts **b** from **a**.

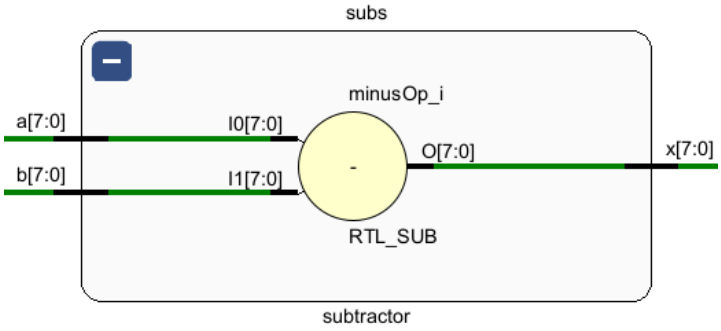


Figure.2 RTL Schematic of subtractor.vhd

The comparator module gives 3 different outputs depending on the result: **a** = **b**, **a** > **b**, **b** > **a**.

Results:

A waveform is observed by a test bench (testbench.vhd). The values are

a = 10001100 (140)

b = 00001100 (12)

out1 = 00000100 (4)

The center button (U18) is used to enable the GCD calculating process. It took 32 clock cycles for my simulation to calculate the GCD of 140 and 12, which is 4. This number can be reduced by implementing this design as a combinational circuit or by lessening the number of clock-driven operations in our FSM.

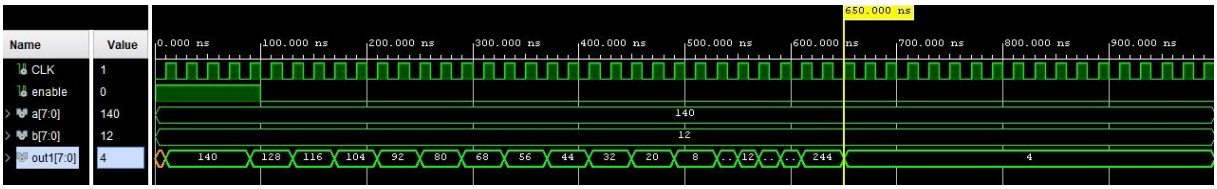
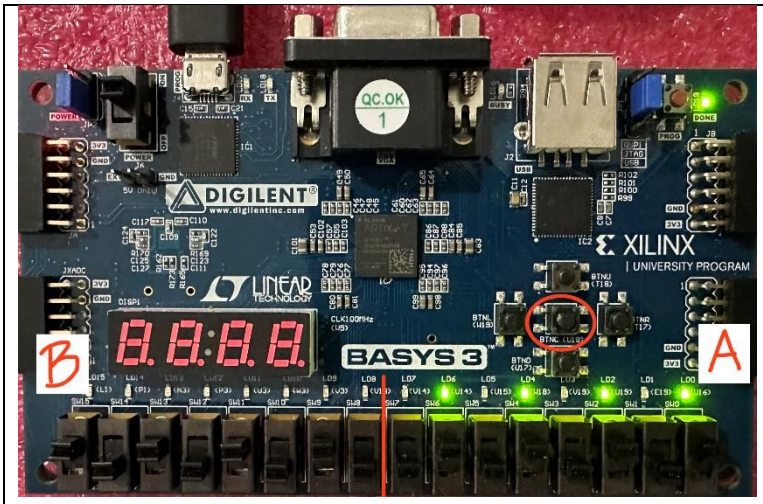
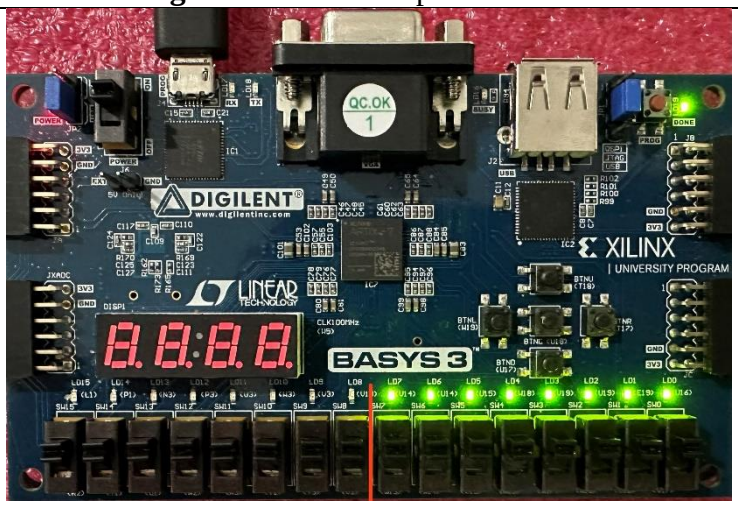
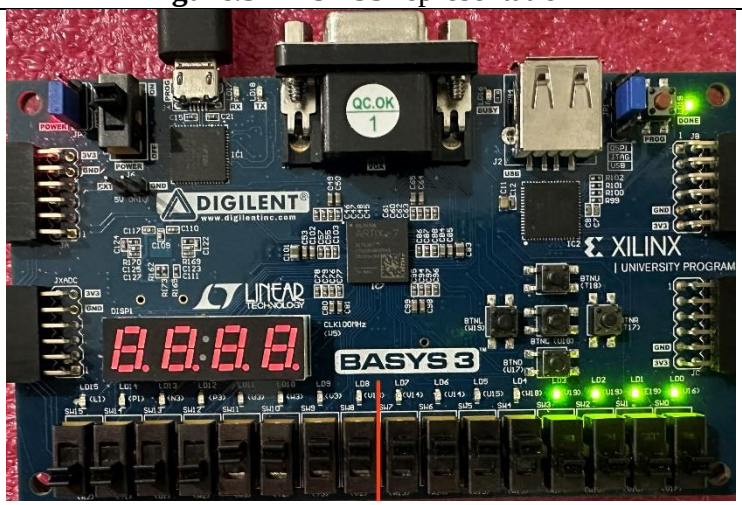
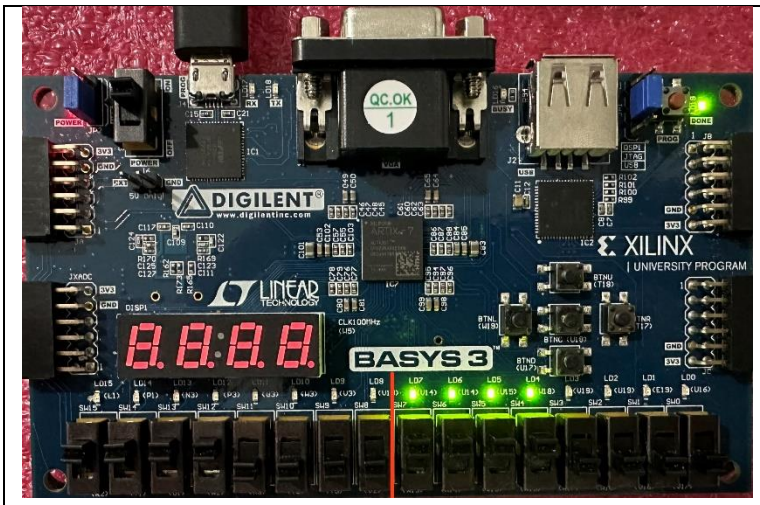
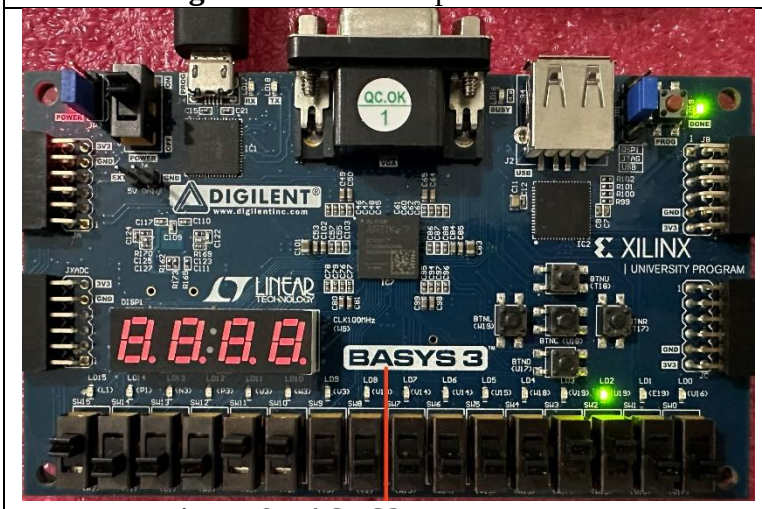


Figure.3 Waveform, generated by the test bench

The waveform and the BASYS3 representations are in parallel. Below that are some BASYS3 representations.

 **Figure.4** BASYS3 representation 1	**a** : 01010101 (85) **b** : 10101010 (170) **out1**: 01010101 (85)
 **Figure.5** BASYS3 representation 2	**a** : 11111111 (255) **b** : 11111111 (255) **out1**: 11111111 (255)
 **Figure.6** BASYS3 representation 3	**a** : 11110000 (240) **b** : 00001111 (15) **out1**: 00001111 (15)

 **Figure.7** BASYS3 representation 4	**a** : 11110000 (240) **b** : 11110000 (240) **out1**: 11110000 (240)
 **Figure.8** BASYS3 representation 5	**a** : 00001100 (12) **b** : 10001100 (140) **out1**: 00000100 (4)

Conclusion:

The purpose of this lab was to teach us how to make a circuit that calculates the greatest common divisor (GCD) of two 8- bit numbers by using VHDL. I learned how to properly write a register and write a FSM by using VHDL. All in all, the experiment was successful at the end.

References:

https://studylib.net/doc/8958892/vhdl-model-of-comparator--8-bits--

APPENDIX

gcd_top.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
entity gcd_top is
    Port (CLK , enable : in STD_LOGIC;
          a , b : in STD_LOGIC_VECTOR(7 downto 0);
          out1 : out STD_LOGIC_VECTOR(7 downto 0));
end gcd_top;
architecture Behavioral of gcd_top is
type state_type is (equal, change, subtract);
signal registers, news : state_type;
signal registers_a, registers_b, news_a, news_b: std_logic_vector(7 downto 0);
signal comp_a,comp_b : std_logic_vector(7 downto 0);
signal subs_b, subs_a, subs_out: std_logic_vector(7 downto 0);
signal comp_out: std_logic_vector(2 downto 0);
component subtractor is
    port(a, b : in std_logic_vector(7 downto 0);
         x : out std_logic_vector(7 downto 0));
end component;
component comparator is
    port ( a , b : in std_logic_vector(7 downto 0);
           x , y , z : out std_logic);
end component;
begin
comp : comparator port map(a => comp_a, b=> comp_b, x=> comp_out(0), y=> comp_out(1),
z=> comp_out(2));
subs : subtractor port map(a => subs_a, b => subs_b, x=> subs_out);
process(CLK)
```

```
    begin
    if rising_edge(CLK) then
        registers <= news;
        registers_a <= news_a;
        registers_b <= news_b;
    end if;
end process;
process(registers,registers_a,registers_b,enable,a,b)
    begin
    news_a <= registers_a;
    news_b <= registers_b;
    comp_a <= registers_a;
    comp_b <= registers_b;
    subs_a <= registers_a;
    subs_b <= registers_b;
case registers is
    when equal =>
        if enable = '1' then
            news_a <= a;
            news_b <= b;
            news <= change;
        else
            news <= equal;
        end if;
    when change =>
        if (comp_out(2) = '1') then
            news <= equal;
        else
            if(comp_out(0) = '1') then
                news_a <= registers_b;
```

```
            news_b <= registers_a;
          end if;
          news <= subtract;
        end if;
    when subtract =>
        news_a <= subs_out;
        news <= change;
end case;
end process;
out1 <= registers_a;
end Behavioral;
```

subtractor.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
entity subtractor is
  port(a, b : in std_logic_vector(7 downto 0);
       x : out std_logic_vector(7 downto 0));
end subtractor;
architecture Behavioral of subtractor is
begin
    x <= a-b;
end Behavioral;
```

comparator.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
entity comparator is
    port ( a , b : in std_logic_vector(7 downto 0);
          x , y , z : out std_logic);
end comparator;
architecture Behavioral of comparator is
begin
z <= '1' when (a = b) else '0';
x <= '1' when (a < b) else '0';
y <= '1' when (a > b) else '0';
end Behavioral;
```

const.xdc

```
set_property PACKAGE_PIN W5 [get_ports CLK]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property PACKAGE_PIN V17 [get_ports {a[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property PACKAGE_PIN V16 [get_ports {a[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
set_property PACKAGE_PIN W16 [get_ports {a[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]
set_property PACKAGE_PIN W17 [get_ports {a[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]
set_property PACKAGE_PIN W15 [get_ports {a[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[4]}]
set_property PACKAGE_PIN V15 [get_ports {a[5]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[5]}]
set_property PACKAGE_PIN W14 [get_ports {a[6]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[6]}]
set_property PACKAGE_PIN W13 [get_ports {a[7]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {a[7]}]
set_property PACKAGE_PIN V2 [get_ports {b[0]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property PACKAGE_PIN T3 [get_ports {b[1]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property PACKAGE_PIN T2 [get_ports {b[2]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property PACKAGE_PIN R3 [get_ports {b[3]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property PACKAGE_PIN W2 [get_ports {b[4]}]
   set_property IOSTANDARD LVCMOS33 [get_ports {b[4]}]
set_property PACKAGE_PIN U1 [get_ports {b[5]}]
```

```
    set_property IOSTANDARD LVCMOS33 [get_ports {b[5]}]
set_property PACKAGE_PIN T1 [get_ports {b[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {b[6]}]
set_property PACKAGE_PIN R2 [get_ports {b[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {b[7]}]
set_property PACKAGE_PIN U16 [get_ports {out1[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[0]}]
set_property PACKAGE_PIN E19 [get_ports {out1[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[1]}]
set_property PACKAGE_PIN U19 [get_ports {out1[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[2]}]
set_property PACKAGE_PIN V19 [get_ports {out1[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[3]}]
set_property PACKAGE_PIN W18 [get_ports {out1[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[4]}]
set_property PACKAGE_PIN U15 [get_ports {out1[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[5]}]
set_property PACKAGE_PIN U14 [get_ports {out1[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[6]}]
set_property PACKAGE_PIN V14 [get_ports {out1[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {out1[7]}]
set_property PACKAGE_PIN U18 [get_ports enable]
    set_property IOSTANDARD LVCMOS33 [get_ports enable]
```

testbench.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
entity gcd_top_tb is
end gcd_top_tb;
architecture Behavioral of gcd_top_tb is
    signal CLK, enable : std_logic;
    signal a, b, out1 : std_logic_vector(7 downto 0);
begin
dut: entity work.gcd_top(Behavioral)
port map(CLK , enable, a , b , out1);
process
begin
CLK <= '0';
wait for 10ns;
CLK<= '1';
wait for 10ns;
end process;
process
begin
enable <= '1';
a <= "10001100";
b <= "00001100";
wait for 100ns;
enable <= '0';
wait;
end process;
end Behavioral;
```