Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

LAB 6: GREATEST COMMON DIVISOR

Purpose

The purpose of the lab is to implement a design that finds the common divisor of two 8-bit numbers using VHDL. The design is then demonstrated on the BASYS3 board.

Methodology

There are many ways to find the greatest common divisor of two numbers. In this design, the most common one is used which is the Euclidean Algorithm. It can be briefly explained like this: if two numbers are equal, then the result is one of them. If not, the smaller number is subtracted from the bigger one until the two results are equal. To implement this algorithm in VHDL, we have to use registers and an FSM. The 8-bit numbers will be entered by the user and the results will be displayed throughout the LEDs.

Design Specifications

Like the previous labs, the design is in a modular fashion. The main module "gcd.vhd" has five inputs and they are:

- clk: The 100MHz internal clock of BASYS3
- reset: Clears the output and the algorithm begins again
- start: Starts the algorithm
- x, y: 8-bit numbers

There are two outputs:

- out_r: LED which lights up when the result is displayed
- gcd_out: The greatest common divisor of x and y

To compare and subtract the numbers, two sub-modules are used: "comparator.vhd" and "subtractor.vhd". These modules were previously used in the Arithmetic Logic Unit lab, and they are adapted for 8-bit numbers. The comparator module takes two 8-bit numbers (x and y) as inputs and returns: "100" if x is smaller than y, "010" if they are equal, and "001" if x is bigger than y. The subtractor module takes two unsigned 8-bit numbers and the result is their subtraction. These modules are not connected to the clock because the results should be given instantly. The modules are adapted from the previous labs with the help of referenced websites.

In the main module, a FSM is used. It is a Moore machine with three states:

- "equal": The state where both numbers are equal. If they are not equal, it moves to the "change" state
- "change": If y is greater than x, the numbers are interchanged using registers and it moves to the "sub" state. If they are equal, it moves to the "equal" state.

- "sub": The numbers are subtracted using the subtraction module then it moves to the "change" state.

When the greatest common divisor calculation is done, the LED corresponding to the output "out_r" will light up. The result "gcd_out" will be displayed through the LEDs with each LED representing one bit of the result.

Results

The RTL schematics for the modules can be seen in Figures 1-3.

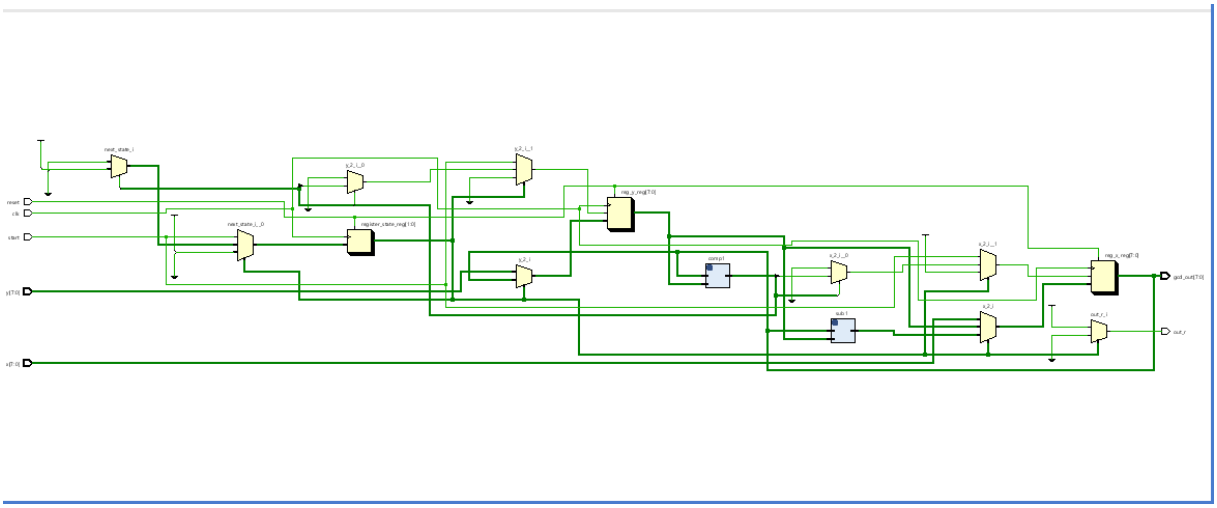

Figure 1: RTL schematic for the main module

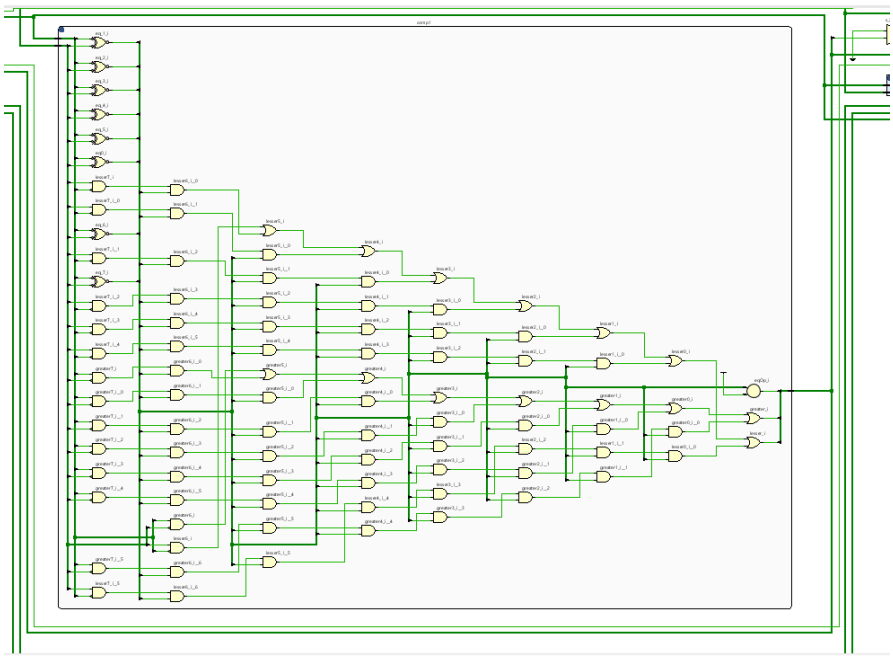

Figure 2: RTL schematic for the comparator module

Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

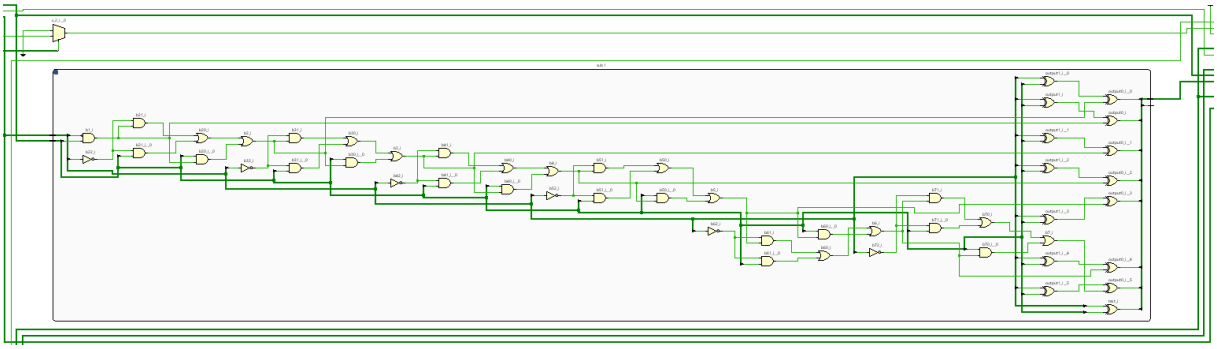

Figure 3: RTL schematic for the subtractor module

After writing the test bench code, the code is simulated with x=”10001100” (140 in decimal) and y=”00001100” (12 in decimal). The result was “00000100” (4 in decimal) as expected. Figure 4 shows the simulation results with the outputs.

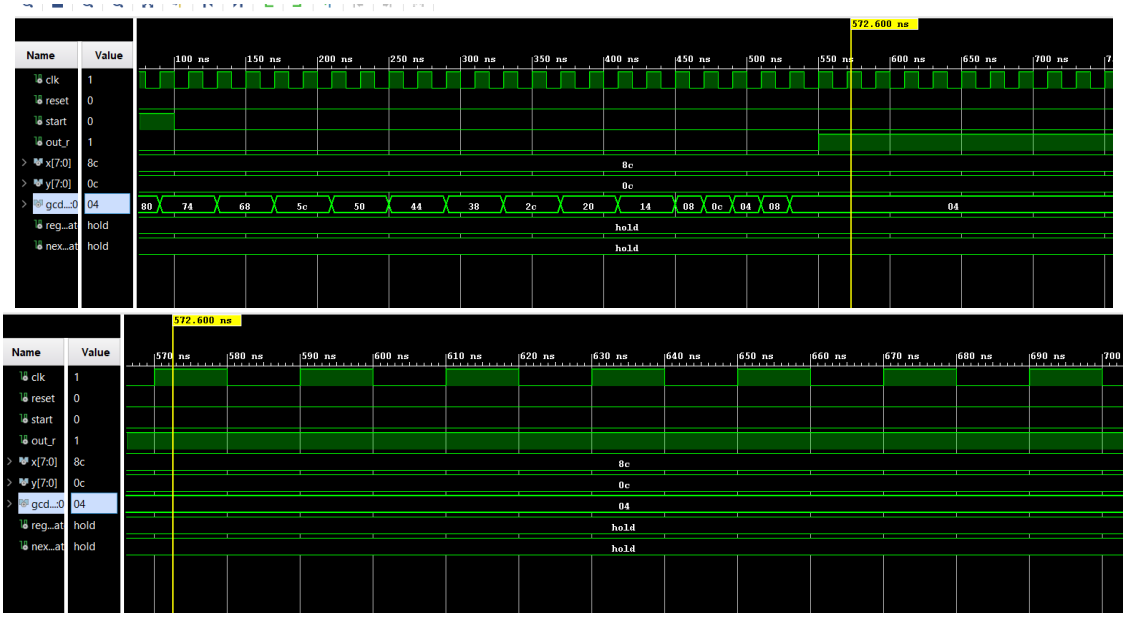

Figure 4: Simulation results of the test bench code

The design is then implemented on the BASYS3 board. The numbers X and Y are input by the user using the eight rightmost switches for X, and the eight leftmost switches for Y. After the start button (indicated in Figure 5) is pressed, the result is ready, indicated by the leftmost LED corresponding to “out_r”. The eight rightmost LEDs display the output “gcd_out”. The output is cleared when “reset” button is pressed. Figures 5-10 show the outputs with the corresponding inputs.

Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

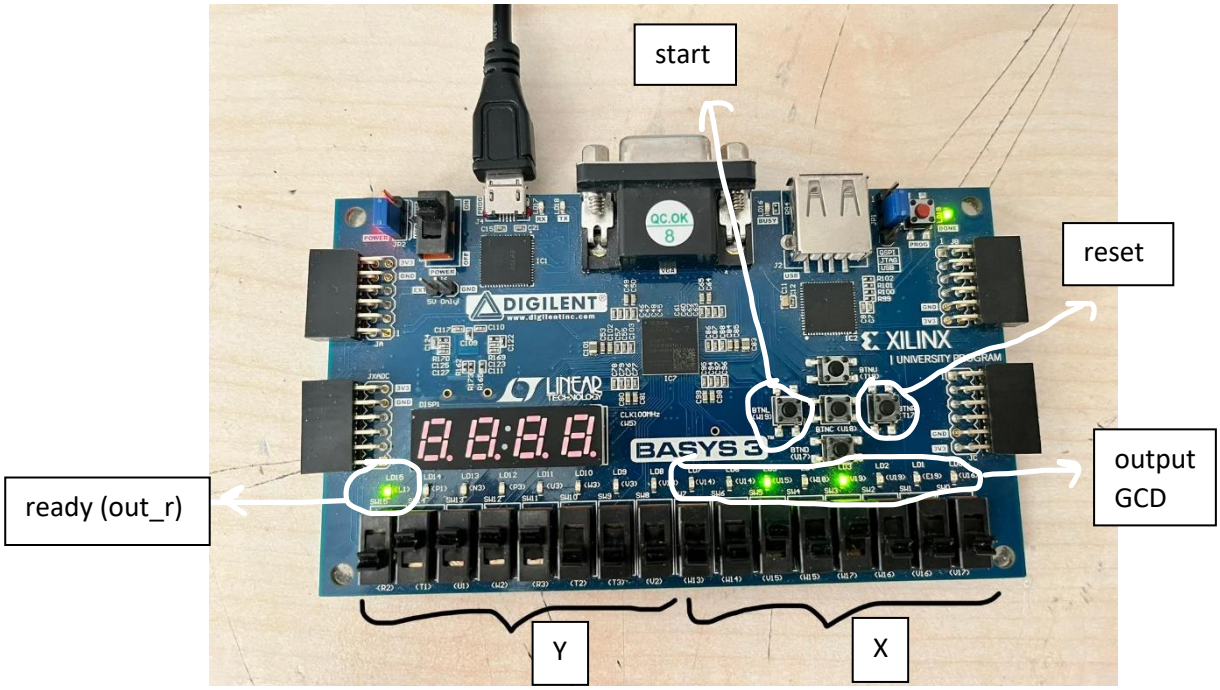

Figure 5: After the result is ready, the output is "00101000" with x=00101000" and y="01111000"

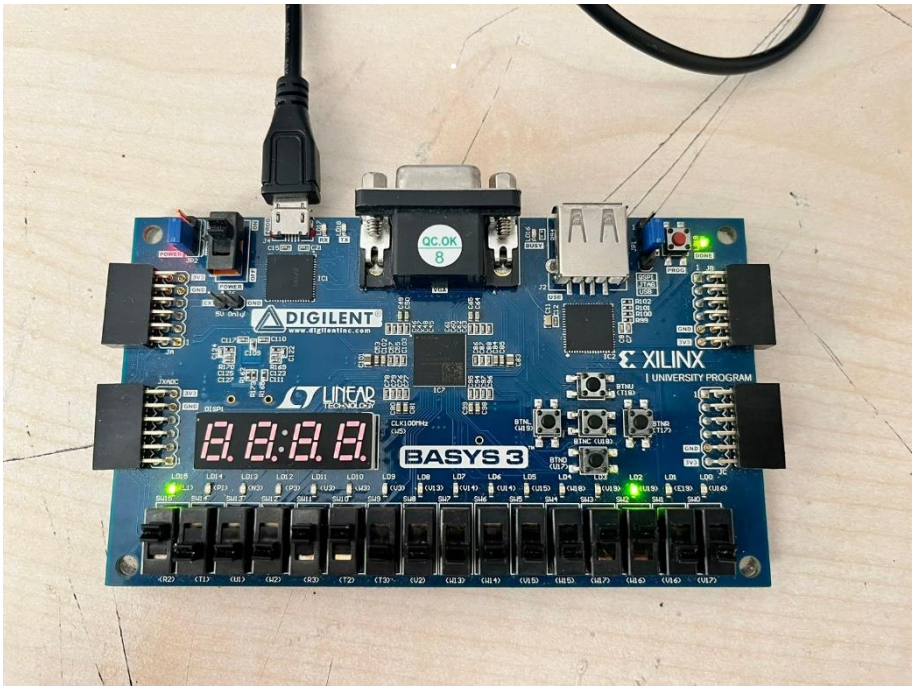

Figure 6: After the result is ready, the output is "00000100" with x=00001100" and y="10001100"

Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

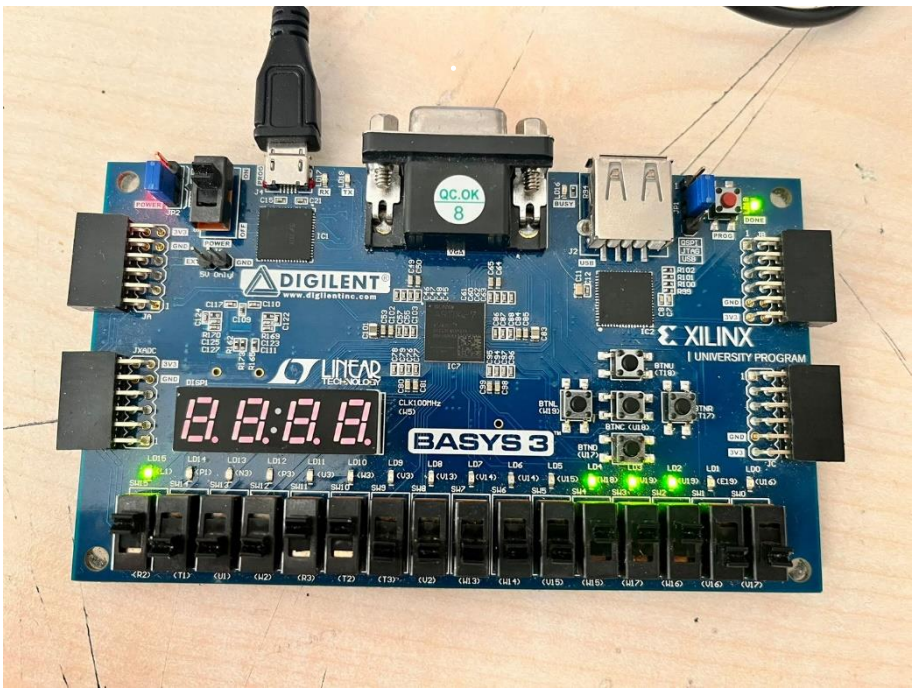

Figure 7: After the result is ready, the output is "00011100" with x=00011100" and y="10001100"

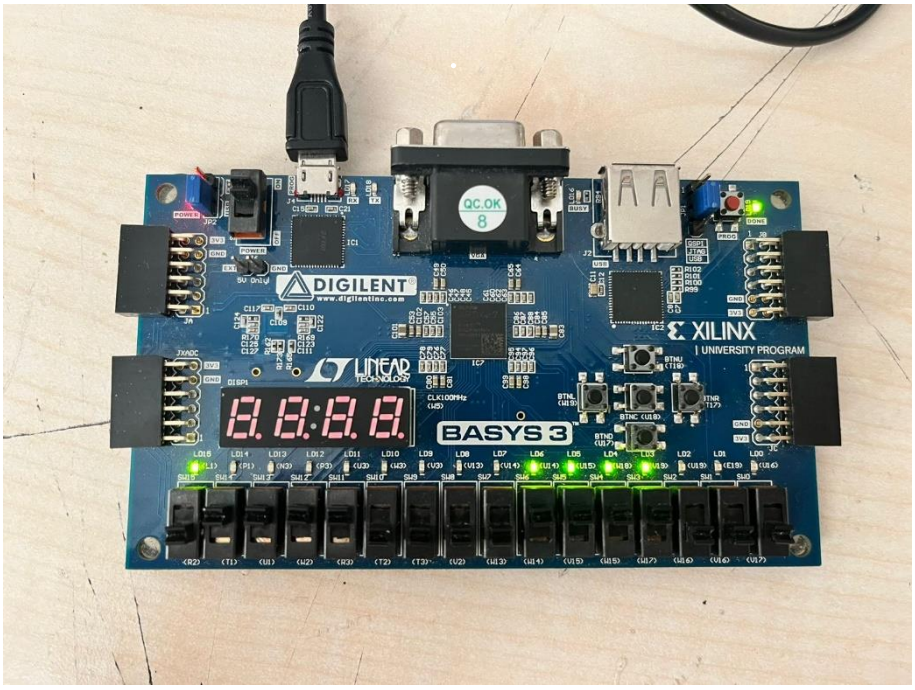

Figure 8: After the result is ready, the output is "01111000" with x=01111000" and y="01111000"

Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

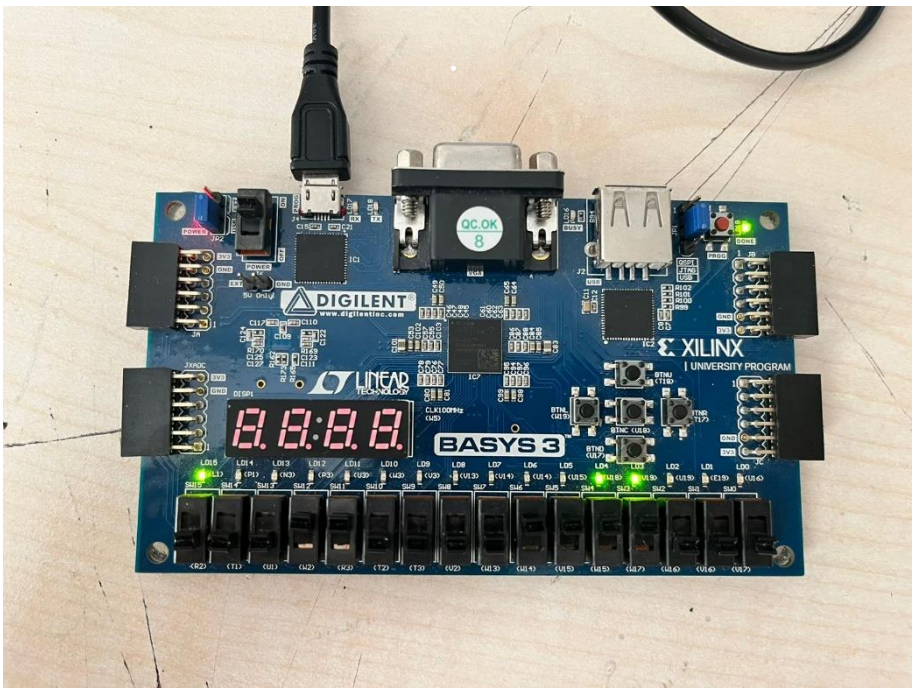

Figure 9: After the result is ready, the output is "00011000" with x=01111000" and y="00011000"

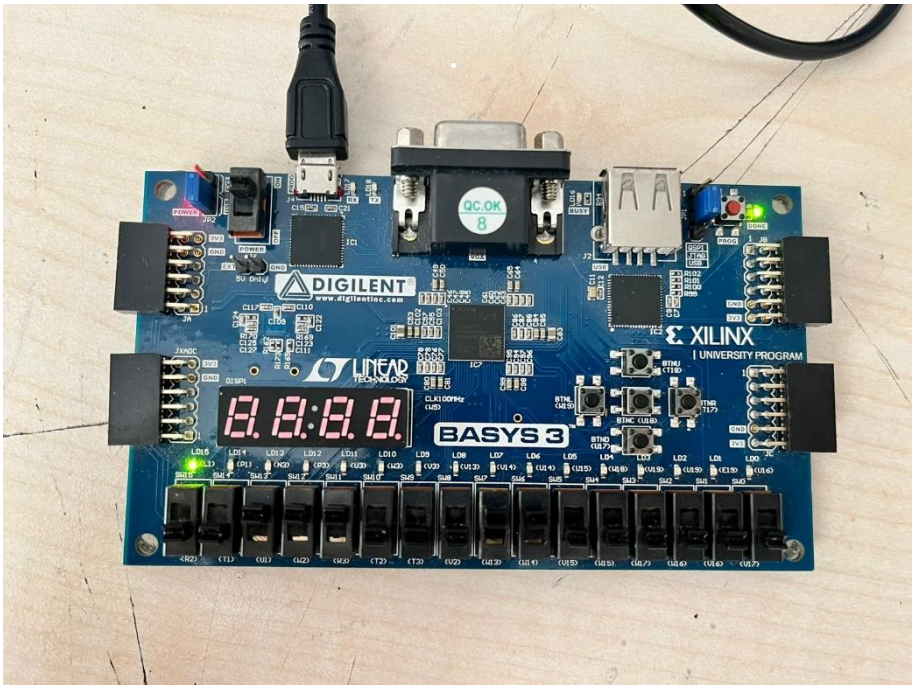

Figure 10: Result after the reset button is pressed

Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

Questions

• *What was your algorithm to calculate the GCD?*

The Euclidian Algorithm is used. If two numbers are equal, then the result is one of them. If not, the smaller number is subtracted from the bigger one until the two results are equal.

• *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?*

The main module is an FSM. It is a Moore machine as the output depends on the states and not on the input. If we used a combinational circuit, it would be faster however we would have to use truth tables to implement it which would be time-consuming as the numbers are 8-bit. In smaller cases, it would be cheaper to implement combinational circuits however this time it might be that it would be more expensive. The sub-modules use combinational logic which is faster and cheaper than FSM. If we used FSM for the sub-modules, it would take a lot of time because it would be clock-driven and we wouldn't get the results instantly like in the combinational circuit case.

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

It took 27 clock cycles to calculate the greatest common divisor of 140 and 12, which is 4. This number can be optimized by using fewer clock-dependent operations in the design. It would also be faster if we used a combinational circuit but it would take a lot of time to implement it.

Conclusion

The purpose of this lab was to design a greatest common divisor calculator and implement it on the BASYS3 board. The simulation results and the implementation on the board all turned out as expected. I can say I learned a lot about how to use registers and how to implement an FSM using VHDL in this lab as the design was essentially an FSM.

References

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

https://www.engineersgarage.com/vhdl-tutorial-22-designing-a-1-bit-an-8-bit-comparator-by-using-vhdl/

https://www.fpga4student.com/2016/11/verilog-code-for-8-bit-74f521-identity.html

http://www.csit-sun.pub.ro/courses/Masterat/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/data/docs/xst/hdlcode13.html

Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

Appendices

Design Codes

Note: As told in the lab prompt, the codes for sub-modules “comparator.vhd” and “subtractor.vhd” aren’t included because they were previously used in the other labs and they are adapted using the referenced websites.

gcd.vhd

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

entity gcd is
 port(
 clk, reset: in std_logic;
 start: in std_logic;
 x, y: in std_logic_vector(7 downto 0);
 out_r: out std_logic;
 gcd_out: out std_logic_vector(7 downto 0)
);
end gcd ;

architecture arch of gcd is

component comparator
 port (x, y: in unsigned(7 downto 0);
 output: out std_logic_vector(2 downto 0));
end component;

component subtractor
 port (x, y: in std_logic_vector ( 7 downto 0);
```

```
 output : out std_logic_vector ( 7 downto 0));
end component;

 type state_type is (equal, change, sub);
 signal register_state, next_state : state_type;

 signal reg_x, reg_y, x_2, y_2: unsigned(7 downto 0);
 signal x_comp,y_comp : unsigned(7 downto 0);

 signal y_sub, x_sub, out_sub: std_logic_vector(7 downto 0);
 signal out_comp: std_logic_vector(2 downto 0);

begin

comp1: comparator
port map(x => x_comp, y => y_comp, output => out_comp);

sub1: subtractor
port map(x => x_sub, y => y_sub, output => out_sub);

process(clk,reset)
    begin
    if reset='1' then
        register_state <= equal;
        reg_x <= (others=>'0');
        reg_y <= (others=>'0');
    elsif (rising_edge(clk)) then
        register_state <= next_state;
        reg_x <= x_2;
```

```
        reg_y <= y_2;
    end if;
end process;

process(register_state,reg_x,reg_y,start,x,y)
    begin
    x_2 <= reg_x;
    y_2 <= reg_y;
    x_comp <= reg_x;
    y_comp <= reg_y;
    x_sub <= std_logic_vector(reg_x);
    y_sub <= std_logic_vector(reg_y);

    case register_state is

        when equal =>
        if start = '1' then
            x_2 <= unsigned(x);
            y_2 <= unsigned(y);
            next_state <= change;
        else
            next_state <= equal;
        end if;

        when change =>
            if out_comp(1)='1' then
                next_state <= equal;
            else
                if out_comp(2)='1' then
```

```
                x_2 <= reg_y;
                y_2 <= reg_x;
            end if;
            next_state <= sub;
        end if;

    when sub =>
        x_2 <= unsigned(out_sub);
        next_state <= change;
    end case;
end process;

out_r <= '1' when register_state = equal else '0';
gcd_out <= std_logic_vector(reg_x);
----------------------------------------------------------------
end arch;
```

gcd_tb.vhd

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

entity gcd_tb is
end gcd_tb;

architecture arch of gcd_tb is

component gcd
 port(
 clk, reset: in std_logic;
```

Yiğit Narter
22102718
Section 01
Lab Report 6
28.11.2022

```
 start: in std_logic;
 x, y: in std_logic_vector(7 downto 0);
 out_r: out std_logic;
 gcd_out: out std_logic_vector(7 downto 0));
end component;

 signal clk, reset,start,out_r: std_logic;
 signal x, y, gcd_out : std_logic_vector (7 downto 0);
 type state_type is (equal, change, sub);
 signal register_state, next_state : state_type;
begin

 dut: gcd
 port map(clk,reset,start,x,y,out_r,gcd_out);

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

 sim: process begin
 start <= '1';
 x <= "01111000";
 y <= "00101000";
 reset <= '0';
 wait for 100ns;
 start <= '0';
```

```
 wait;
 end process;
end arch;

configuration arch of gcd_tb is
 for arch
 end for;
end arch;
```