Lab6 Report: Greatest Common Divider

Defne Yaz Kılıç
22102167
Section 2
EEE, Bilkent University
(Dated: 28.11.22)

I. PURPOSE OF THE EXPERIMENT

The aim of this experiment was to design a greatest common divisor calculating circuit. It takes two 8 bit numbers as inputs and calculates their gcd and show the output on LEDs.In order to create such circuit it is needed to understand registers and their functionality.

II. Q&A

a. What was your algorithm to calculate the GCD? There are multiple methods to calculate the GCD. The one that is used in this design is called as Euclidean Algorithm:

- Comparing numbers.
- If two numbers are equal i.e A=B, the GCD is one of these numbers
- If one of them is greater i.e A>B subtraction of smaller input from the larger input until they become equal.

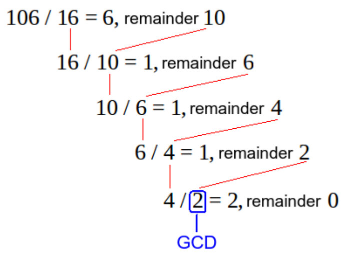

FIG. 1. Euclidean Algorithm

b. 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? An FSM Moore machine is used in this design. In Moore machine the output depends only the current state. Using FSM provides storage of data with registers therefore the GCD calculation steps can be done by using previous outputs as inputs. The same purpose can be completed by designing a combinational circuit as well, however since in the combinational circuit there isn't any data storage the output will be calculated only from the main inputs. This scenario requires a truth table with a very large amount of different combinations which occupies a larger space and gates. For smaller purposes combinational circuits can be more advantageous since they are more faster than FSM. In this design also, in smaller modules combinational circuits are used to calculate subtraction and comparison since they are more faster than clock dependent feature of FSM.

c. How many clock cycles did it take in your simulation to calculate the GCD? Do you think this can be optimized? How so? In order to find the GCD of 140 and 12 it takes 26 clock cycles. This number can be reduced by diminishing the clock dependent steps. The fastest result would be obtained by only using combinational circuits since there isn't any clock dependency, everything happens at the same time. If a Mealy machine is used, the operation would be completed with less clock cycles as well which reduces the calculation time.

III. METHODOLOGY

For this lab, the main algorithm as explained above is the Euclidean Algorithm. The top module consist of 5 inputs and 2 outputs. The clock input is fixed at 100 MHz that is the internal clock of Basys3, second input is the reset selection controlled by the button on FPGA when reset <= "1" the output is cleared. The two 8-bit numbers are the 3rd and 4th inputs and they are also user controlled with the switches of FPGA. The last input, enable selection when enable <= "1" the calculation initiates.

The design consist of 2 submodules (subtraction,comparison) and 1 main module. The main module has 3 states: Hold, Replace, Subtract

- Hold State

This state remains same if both numbers are equal, otherwise it moves to Replace state.

- Replace State

When the two numbers are equal, it moves back to Hold state. When there is inequality, in order to achieve a proper subtraction process, this state ensures that the greater number is stored in register1 and the smaller one is in register2. Therefore if the number in register2 is greater than the one in register 1, the values are swapped.Afterwards it moves to subtract state.

- Subtract State

The subtraction of the second (smaller) number from the first one takes place and it moves back to Replace state.

To achieve the subtraction and comparison processes a different code from the ALU of the previous lab is used. Since it wasn't forbidden to use any library, to make things easier numeric and unsigned library is used for both submodules.Both of the submodules are designed as combinational circuits therefore they work in asynchronous fashion,independent from the clock.

The comparator module is designed in vectoral form. When both numbers (7 downto 0) are equal the output becomes "010", if the first one is greater than the second it becomes "001" and lastly when the second one is greater the output becomes "100".

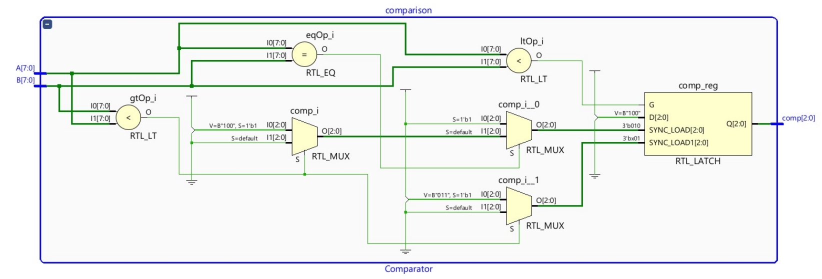

FIG. 2. comparator schematic

The subtractor module has also 2 inputs for 8 bit numbers and an output. Its schematic can be seen below.

The implementation on Basys3 is made using the switches and LEDs as can be seen from the figure below. For the reset and enable inputs the buttons are used.

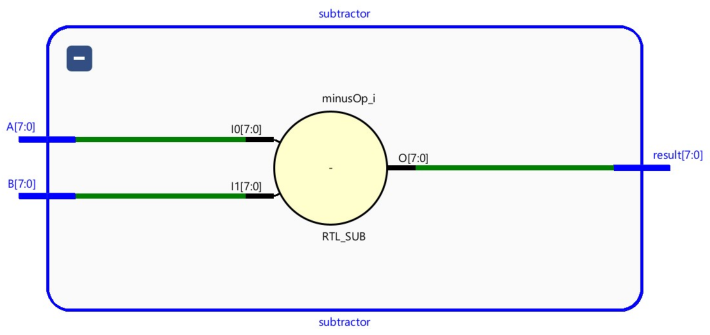

FIG. 3. subtractor schematic

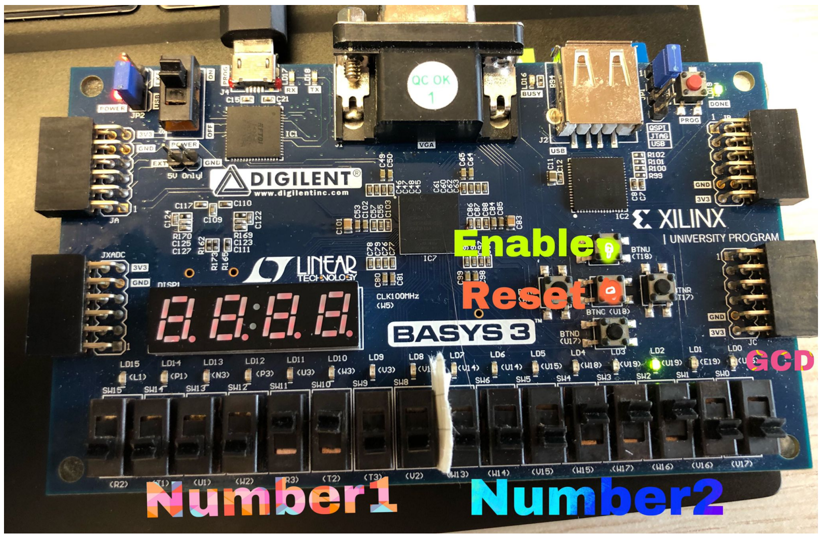

FIG. 4. Implementation on Bays3

IV. RESULTS

To verify the design a testbench is created for input values 140(0b10001100) and 12(0b00001100) respectively. As can be seen from the figure below the simulation results were matching with the expected results. Since the GCD of 140 and 12 is 4(0b00000100), the output is also 4.

Same values are tested on Basys3 and the result was also verifying that the design was correct. Additionaly when it is pressed to reset button it resets the LED combination on Bays3 to 0 for all, when pressed to enable button it initializes the process and the GCD value for given inputs is shown with LEDs.

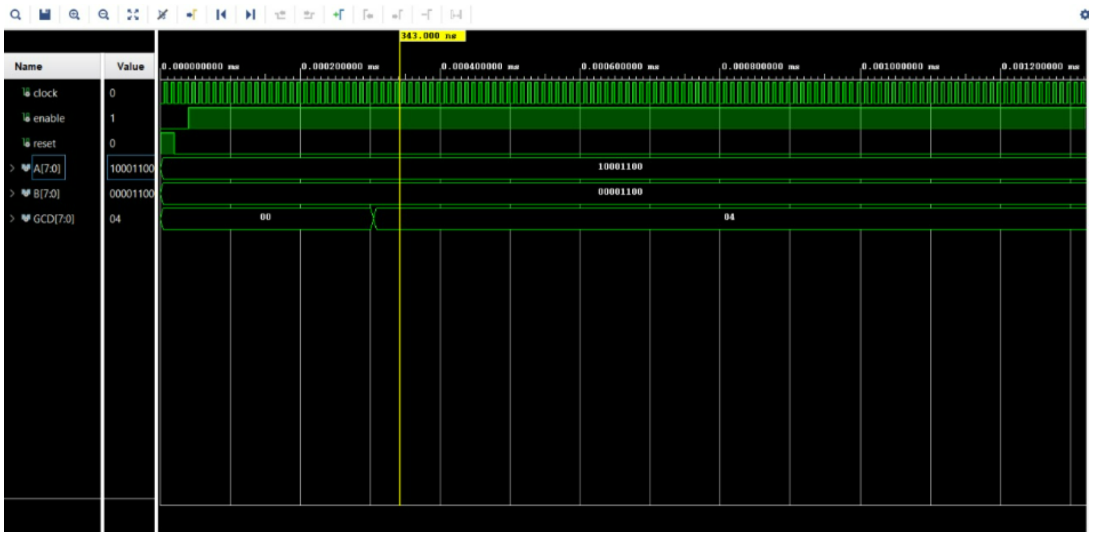

FIG. 5. Top module Test bench

- First number: 140 (0b10001100) & Second number: 12 (0b00001100) & Output: 4 (0b0b00000100)

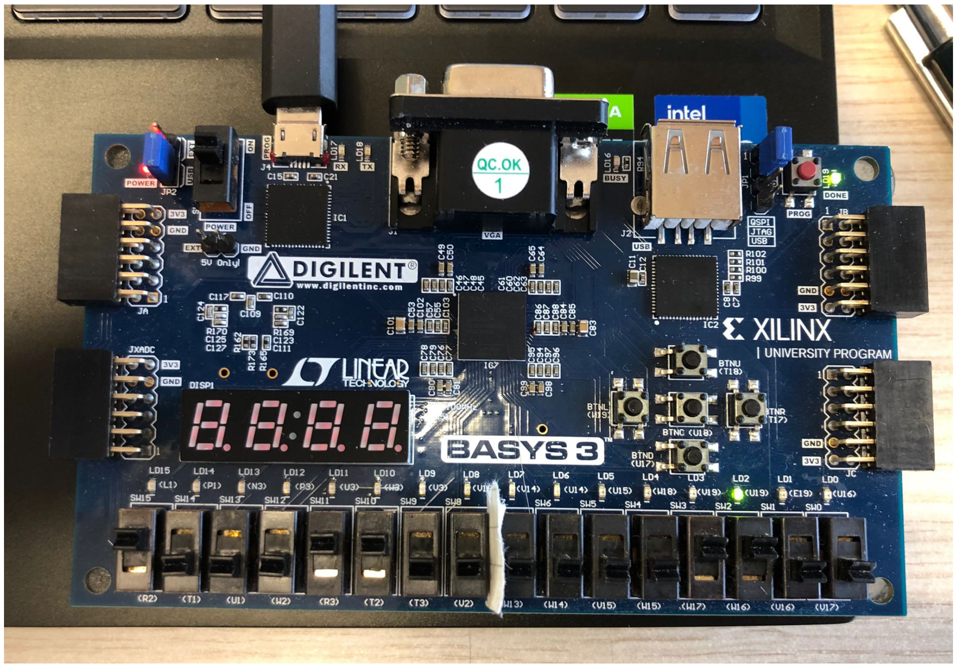

FIG. 6. Configuration-1-

- First number: 35 (0b00100011) & Second number: 20 (0b00010100) & Output: 5 (0b00000101)

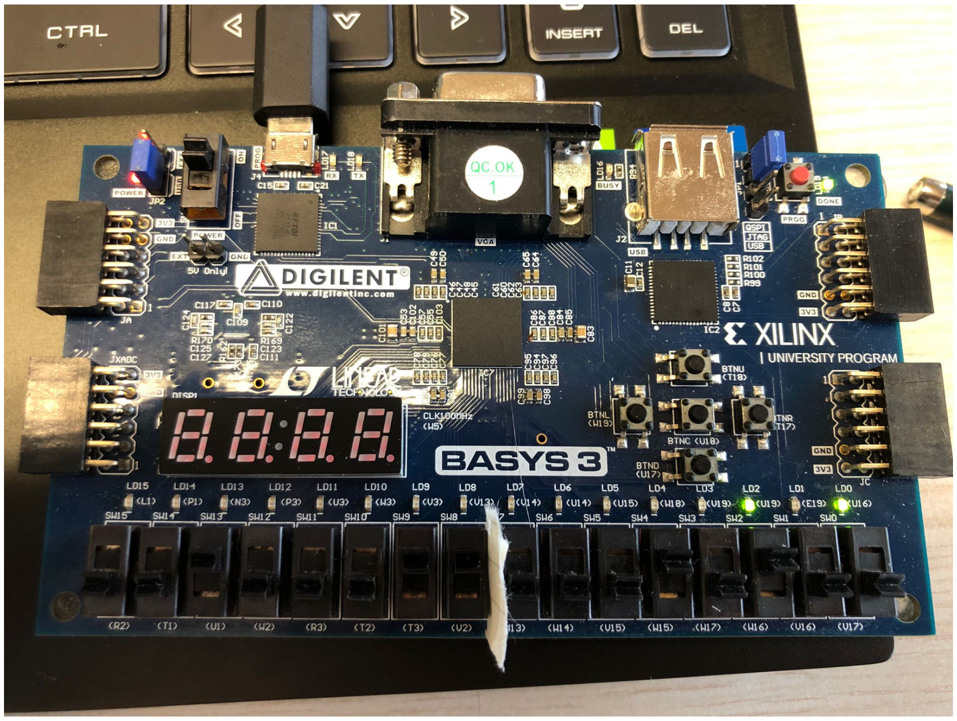
FIG. 7. Configuration-2-

- First number: 12 (0b00001100) & Second number: 28 (0b00011100) & Output: 4 (0b00000100)

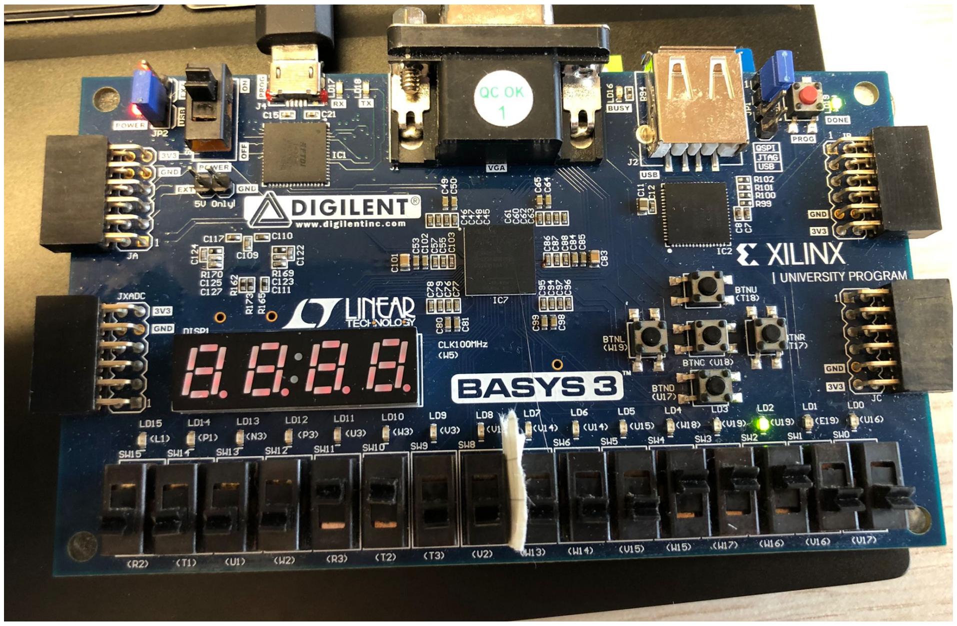
FIG. 8. Configuration-3-

V. CONCLUSION

Considering all the topics explained above, it can be said that a working GCD design is correctly simulated and implemented on FPGA.The crucial part was to understand the FSM and how to load and preserve data using it. This desing was a fusion of combinational circuits and FSM.

The codes for simulation and each module can be found in Appendix .

VI. REFERENCES

- https://www.engineersgarage.com/vhdl-tutorial-22-designing-a-1-bit-an-8-bit-comparator-by-using-vhdl/
- https://medium.com/@mlbors/what-is-a-finite-state-machine-6d8dec727e2c: :text=A20Finite

APPENDIX

A. Main GCD code

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


entity main_register is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           enable : in STD_LOGIC;
           A : in std_logic_vector ( 7 downto 0);
           B : in STD_LOGIC_vector ( 7 downto 0);
           GCD : out STD_LOGIC_vector ( 7 downto 0)
           );
end main_register;

architecture reg_bhv of main_register is
type state is (hold, replace, substract);
 signal reg_state, following_state : state;

 signal reg_a, reg_b, following_a, following_b:std_logic_vector(7 downto 0);
 signal comp_a,comp_b : std_logic_vector(7 downto 0);

 signal sub_b, sub_a, sub_out: std_logic_vector(7 downto 0);
 signal comp_out: std_logic_vector(2 downto 0);

begin


comparison: entity work.Comparator(comp_bhv)
 port map(A => comp_a, B=> comp_b, comp => comp_out);
 subtractor: entity work.subtractor(sub_bhv)
 port map(A => sub_a, B=> sub_b, result => sub_out);

 process(clk,reset)
 begin
 if reset='1' then
    reg_state <= hold;
    reg_a <= (others=>'0');
    reg_b <= (others=>'0');

 elsif (rising_edge(clk)) then
    reg_state <= following_state;
    reg_a <= following_a;
    reg_b <= following_b;
 end if;
 end process;



 process(reg_state,reg_a,reg_b,enable,A,B)
 begin
 following_a <= reg_a;
 following_b <= reg_b;
 comp_a <= reg_a;
 comp_b <= reg_b;
 sub_a <= std_logic_vector(reg_a);
 sub_b <= std_logic_vector(reg_b);

 case reg_state is
 when hold =>
 if enable = '1' then
 following_a <= std_logic_vector(A);
 following_b <= std_logic_vector(B);
 following_state <= replace;
 else
 following_state <= hold;
 end if;

 when replace =>
 if (comp_out = "010") then
 -- when A=B, state is hold
 following_state <= hold;
 else
 -- when A is less than B, A and B change their places to be used in the unsigned subtractor
 if(comp_out = "100") then
 following_a <= reg_b;
 following_b <= reg_a;
 end if;
 following_state <= substract;
 end if;

when substract =>
 following_a <= std_logic_vector(sub_out);
 -- in order to continue to subtraction process the result
 --of previous subtraction is replacd with the new input of the second subtraction process
 following_state <= replace;
 -- returning to replace state to do  comparison with the new inputs

 end case;
 end process;
 GCD <= std_logic_vector(reg_a);
```

```
end reg_bhv;
```

B. Comparator code

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


entity Comparator is
    Port ( A : in std_logic_vector(7 downto 0);
           B : in STD_LOGIC_vector (7 downto 0);
           comp : out std_logic_vector (2 downto 0));
end Comparator;

architecture comp_bhv of Comparator is

begin
 process
     begin
if A=B then
           comp <= "010";
       elsif A>B then
           comp <= "001";
      elsif A<B then
           comp <= "100";
      end if;
    end process;

end comp_bhv;
```

C. Subtractor code

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


use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity subtractor is
Port ( A,B : in STD_LOGIC_vector ( 7 downto 0);
       result : out std_logic_vector (7 DOWNTO 0));


end subtractor;

architecture sub_bhv of subtractor is

begin
    result <= A-B;
```

```
end sub_bhv;
```

D. Main TB

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


entity mainTB is
end mainTB;

architecture mainTB_bhv of mainTB is

signal clk,reset,enable : std_logic;
signal A, B, GCD : std_logic_vector (7 downto 0);

 type state_type is (hold, replace, subtract);
 signal reg_state, following_state : state;

begin

 dut: entity work.main_register(reg_bhv)
 port map(clk,reset,enable,A,B,GCD);


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

 tb_process: process begin
 enable <= '1';
 A <= "10001100";
 B <= "00001100";
 reset <= '0';
 wait for 100ns;
 enable <= '0';
 wait;
 end process;
 end mainTB_bhv;
```