BILKENT UNIVERSITY ELECTRICAL AND ELECTRONICS ENGINEERING DEPARTMENT

EEE102 LAB 6 REPORT

GREATEST COMMON DIVISER

Atalay ÜSTÜNDAĞ-22203554

A.Purpose:

The objective of this laboratory exercise is to construct a circuit that calculates the greatest common divisor of two 8-bit numbers. The setup involves utilizing 16 switches for inputting the two 8-bit numbers, along with an enable button and a reset button. The output is displayed using the LEDs on the Basys3.

B.Questions and Answers:

1. What was your algorithm to calculate the GCD?

In order to determine the greatest common divisor of two numbers, a modulo operation method was used. This approach is more straightforward to use than the Euclidean one. More details and justifications for this methodology may be found in the "Methodology" section, specifically under addition.

2. 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?

This part uses a Mealy machine as part of its architecture to function as a finite state machine. Unlike combinational circuits, which are excellent at handling operations that require fewer inputs with cost-effectiveness and speedier outputs, finite-state machines are skilled in managing activities. It is important to remember, nevertheless, that the functioning of finite-state machines is largely dependent on clock cycles. As a result, certain basic operations that combinational circuits can do faster might take longer in a finite-state machine.

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

A built-in simulation was used to determine how many clock cycles are required for GCD computation, and the results showed that 15 clock cycles are required. Optimizing the final result can be accomplished by investigating better algorithms or by creating a faster, more affordable, and more effective design by reducing the number of clock-dependent processes.

C. Methodology of the Experiment:

The multiplexer, register, datapath, and state component have all been integrated into the design of the GCD unit. To provide a controlled procedure, the multiplexer in this design chooses register inputs according to their predefined inputs. In the design, two multiplexers have been included to handle two numbers. Three registers total—two for the input numbers and one for the greatest common divisor (GCD)—play an additional function in this architecture. The output values of these registers are determined by their present state. The algorithm has seven states that make up the state component. The term "start state" refers to the FSM's initial state, which is stateless until it receives the command to start. The input state adds a delay to allow registers and multiplexers to properly prepare the signal for processing, even though it is not changing in terms of input or output. By evaluating the two numbers' inequality, the test state produces two outputs depending on the signal relationship. Before going back to the test state, the Euclidean Algorithm is used to refresh the outputs in the update state. This cycle continues until all the numbers are equal. Since the clear input of this FSM is an asynchronous signal that sets the next state to the start state, it is classified as a Mealy machine. The FSM design is more expensive than a possible combinational circuit model, but because of its clock dependency, it performs less well. Implementing this procedure involves a complex combinational circuit with multiple logic gates like AND, OR, NAND, NOR, etc. Consequently, the design cost is higher than that of the FSM design.

D. Results:

After finalizing the design phase, a simulation has been executed to validate the functionality of the code. The simulation outcome corresponds with the expected behavior of the design. Presented below is the waveform graph depicting the simulation results.

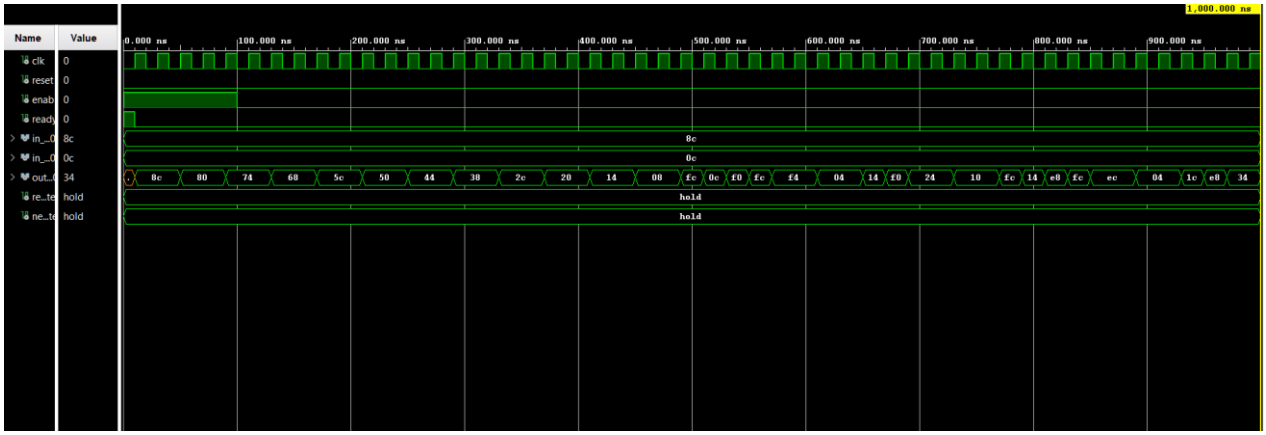

Figure 1: Waveform graph

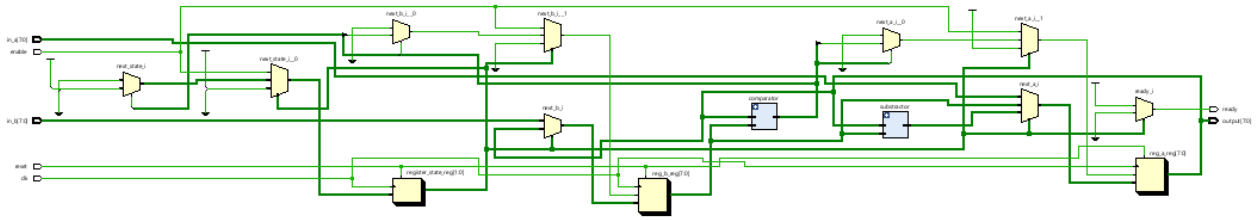

Figure 2: The General Schematic

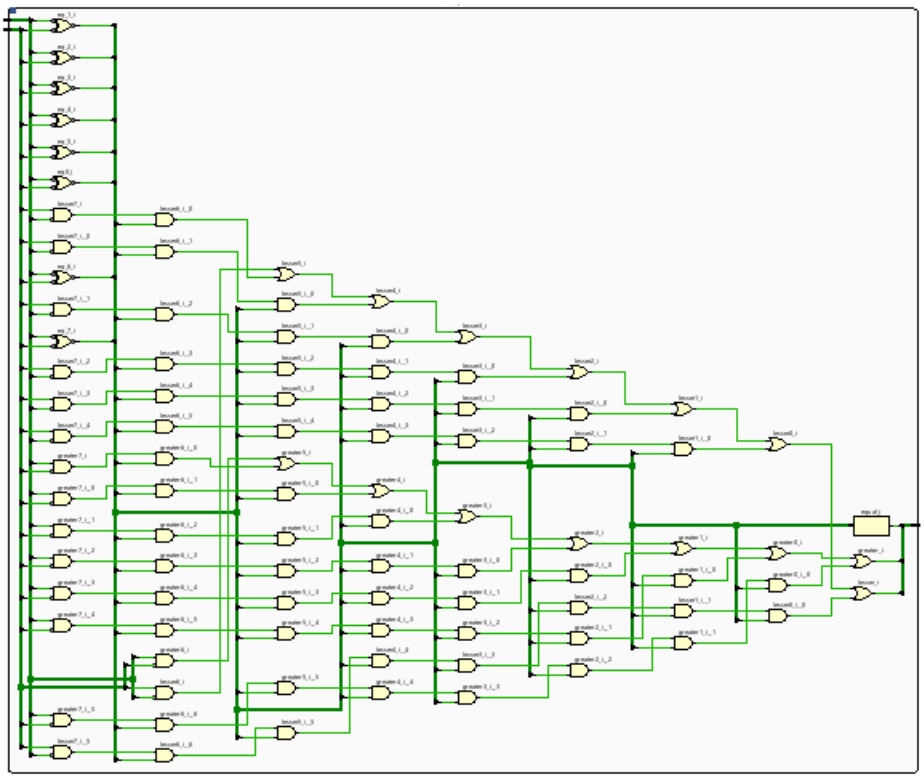

Figure 3: Schematics for the comparitor

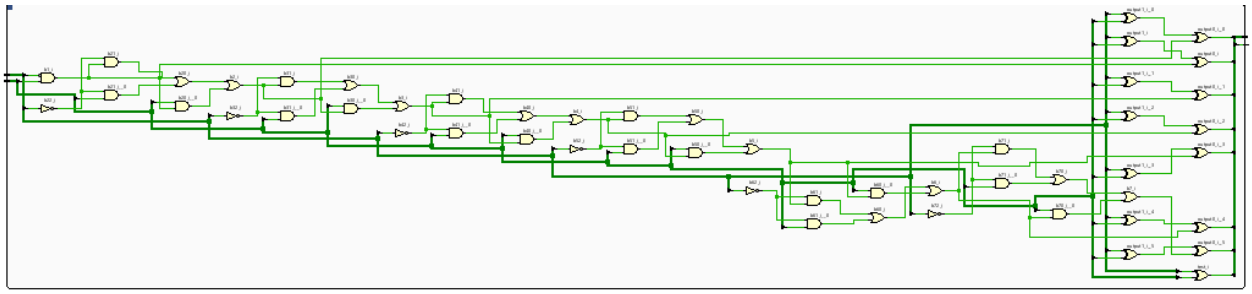

Figure 4: Schematics for the substractor

Following figures are some examples of the greatest common divisor of the two input numbers.

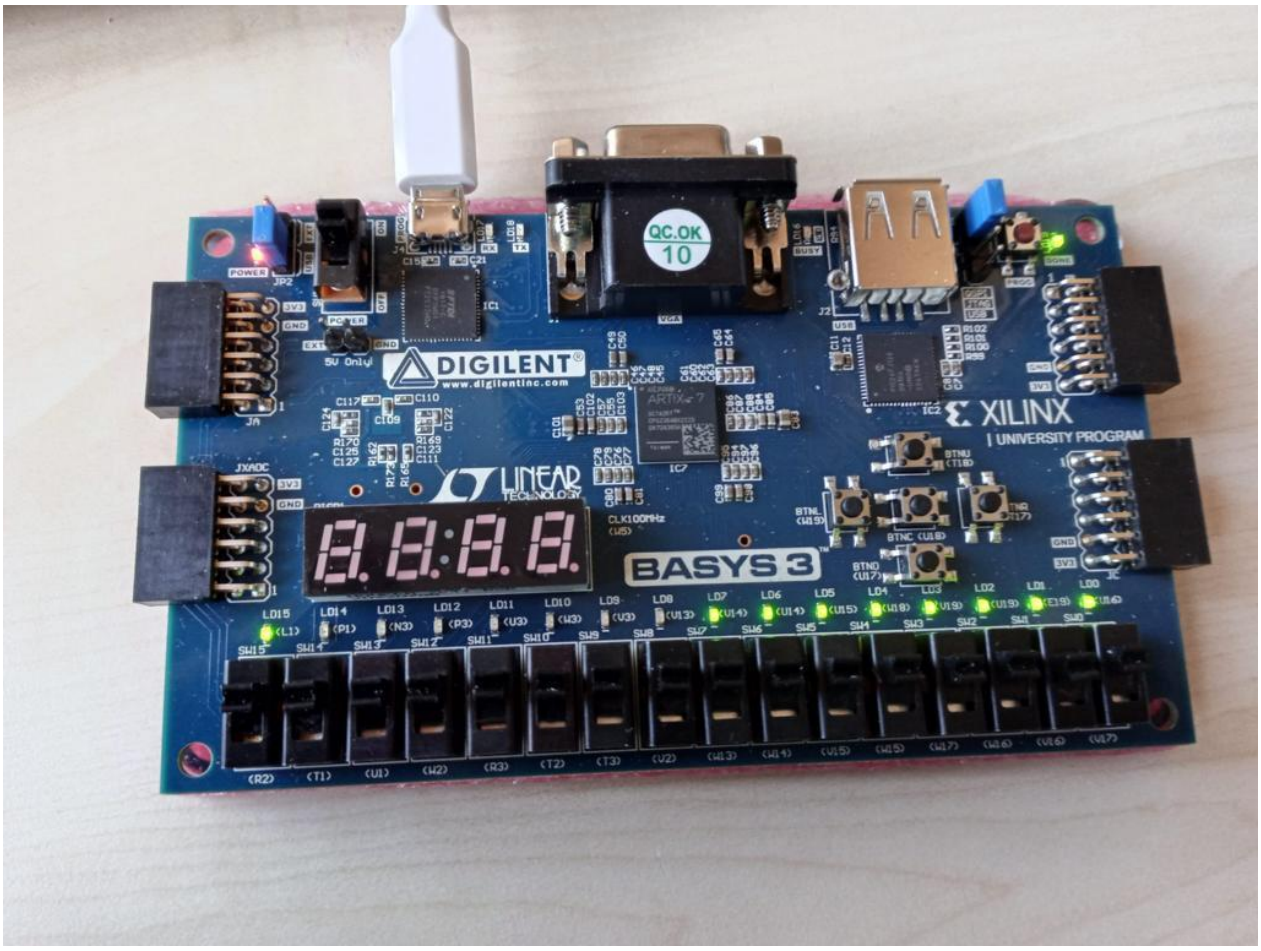

Figure 5: Numbers are 11111111 and 11111111. GCD is 11111111.

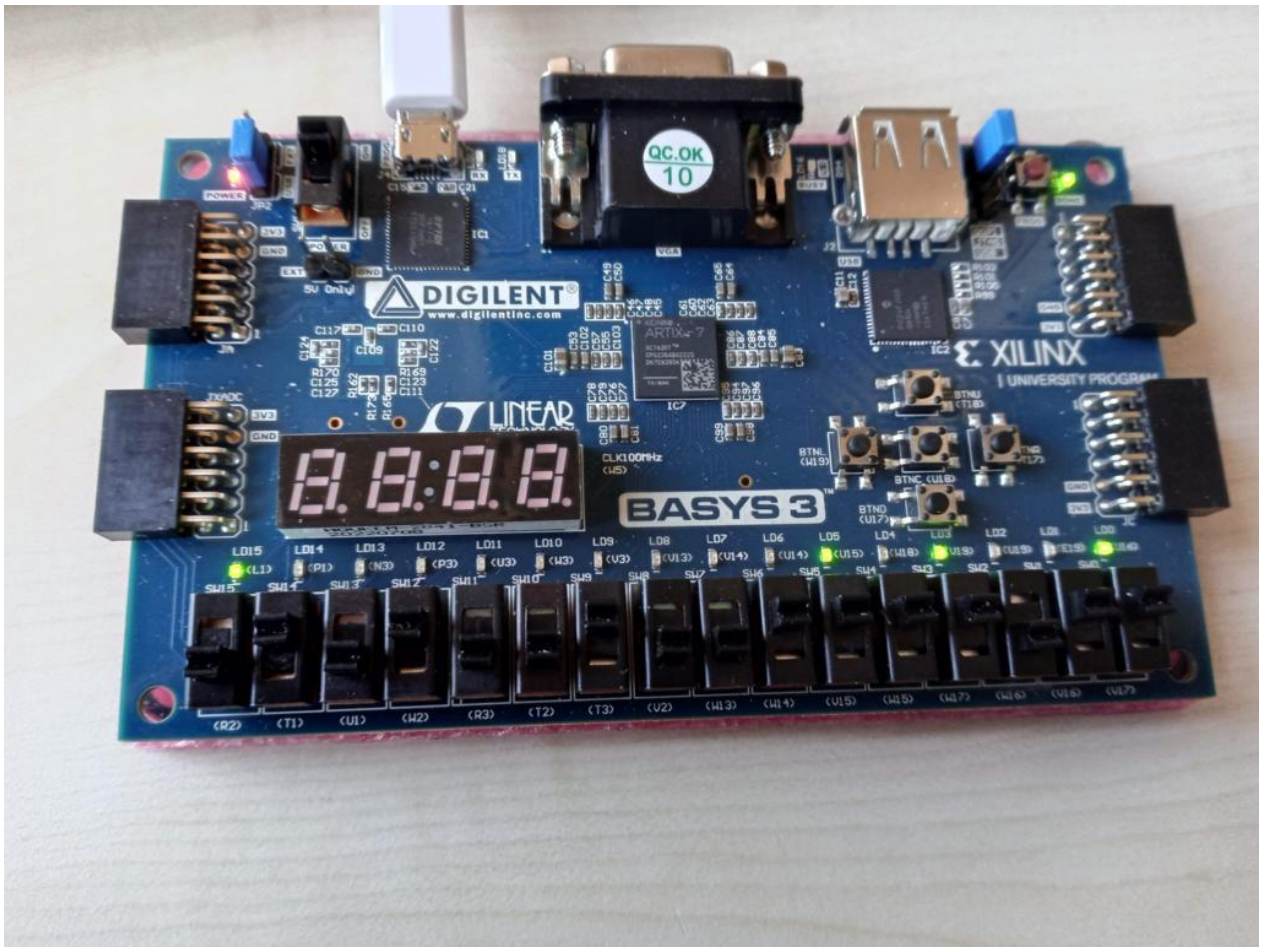

Figure 6: Numbers are 01010010 and 01111011. GCD is 00101001.

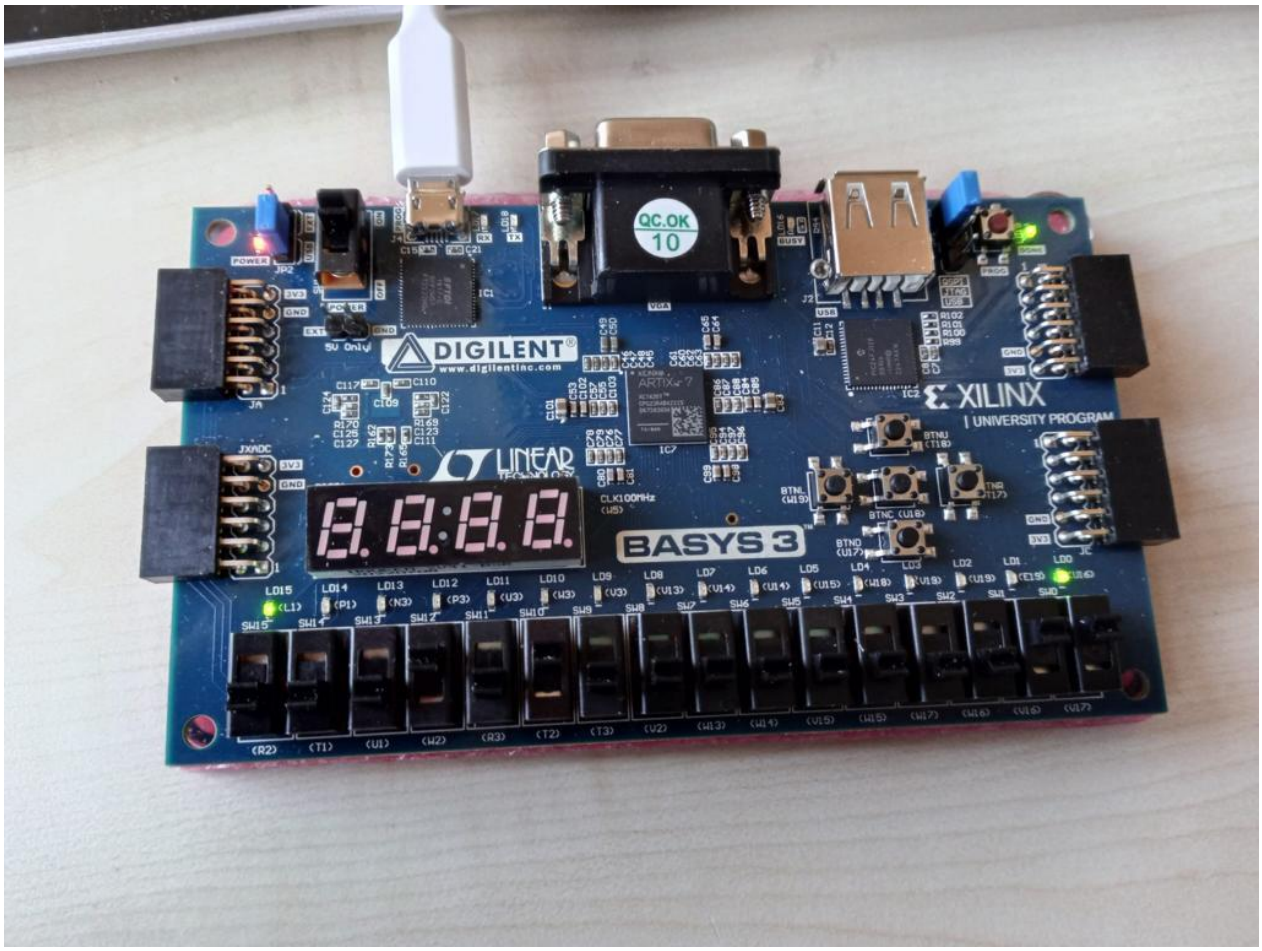

Figure 7: Numbers are 00010100 and 00000011. GCD is 00000001.

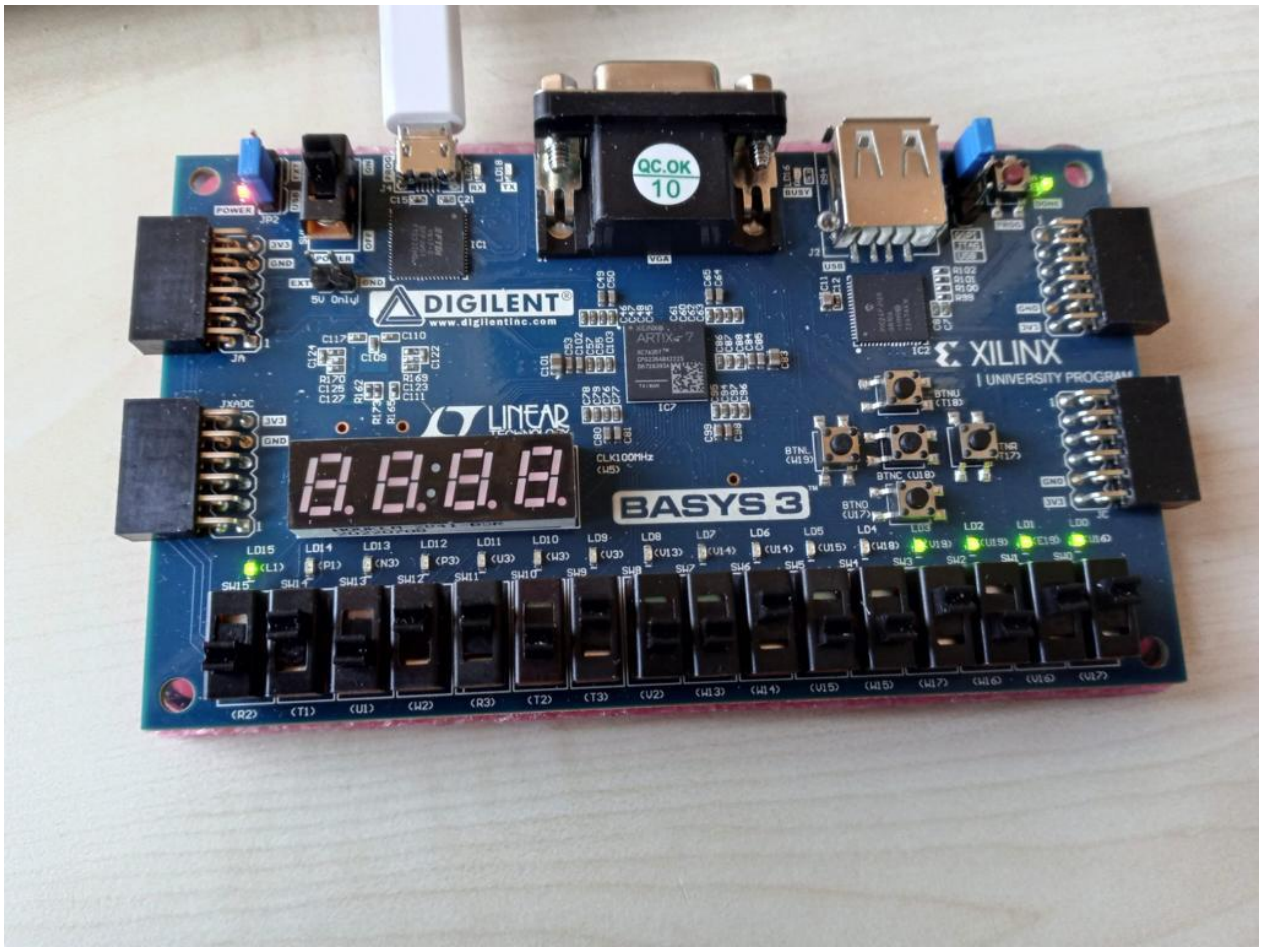

Figure 8: Numbers 01011010 are 01001011. GCD is 00001111

E. Conclusion:

The objective of the lab is to build a circuit that enables the computation of the greatest common divisor (GCD) between two eight-bit numbers. First, an algorithm was developed. Additionally, the Moore Machine (FSM) circuit was designed to be appropriate for data storage. While combinational circuit design might be faster, it would also be more difficult and expensive. GCD of 140 and 12 were found with 296 clock ticks. The Mealy machine might be utilized to operate in a shorter amount of clock ticks. This lab helped us learn about and experience with registers and FSM, and it could be valuable for our project as well.

F.References:

http://quitoart.blogspot.com/2017/11/vhdl-greatest-common-divisor-gcd.html
https://www.geeksforgeeks.org/full-subtractor-in-digital-logic/
https://github.com/CankutBoraTuncer/Bilkent-EEE102-Labs/blob/main/Lab_6/EE102Lab_6%20(4).pdf

G. Appendix:

Code of gsd_main:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity gcd_main is
 port(
 clock, reset: in std_logic;
 enable: in std_logic;
 in_first_number, in_second_number: in std_logic_vector(7 downto 0);
 rdy: out std_logic;
 out_num: out std_logic_vector(7 downto 0)
 );
end gcd_main ;

architecture rtl_main of gcd_main is
 type state_type is (hold, replace, substract);
 signal register_signal, signal_of_next_state : state_type;

 signal register_of_A, register_of_B, signal_next_A, signal_next_B: unsigned(7 downto 0);
 signal comp_a,comp_b : unsigned(7 downto 0);

 signal B_substution, A_substution, out_substution: std_logic_vector(7 downto 0);
```

```
 signal out_com: std_logic_vector(2 downto 0);


begin


 comparator: entity work.comparator8bit(rtl_comparator)

 port map(in_first_number => comp_a, in_second_number=> comp_b, out_num=> out_com);

 substractor: entity work.subtractor8bit(rtl_subtractor8bit)

 port map(in_first_number => A_substution, in_second_number => B_substution, out_num =>
out_substution);

process(clock,reset)

 begin

 if reset='1' then

 register_signal <= hold;

 register_of_A <= (others=>'0');

 register_of_B <= (others=>'0');

 elsif (rising_edge(clock)) then

 register_signal <= signal_of_next_state;

 register_of_A <= signal_next_A;

 register_of_B <= signal_next_B;

 end if;

 end process;

process(register_signal,register_of_A,register_of_B,enable,in_first_number,in_second_number)

 begin

 signal_next_A <= register_of_A;

 signal_next_B <= register_of_B;
```

```
comp_a <= register_of_A;
comp_b <= register_of_B;
A_substution <= std_logic_vector(register_of_A);
B_substution <= std_logic_vector(register_of_B);

case register_signal is

when hold =>
if enable = '1' then
signal_next_A <= unsigned(in_first_number);
signal_next_B <= unsigned(in_second_number);
signal_of_next_state <= replace;
else
signal_of_next_state <= hold;
end if;
when replace =>
if (out_com(1) = '1') then
signal_of_next_state <= hold;
else
if(out_com(2) = '1') then
signal_next_A <= register_of_B;
signal_next_B <= register_of_A;
end if;
signal_of_next_state <= substract;
end if;
```

```
 when substract =>
 signal_next_A <= unsigned(out_substution);
 signal_of_next_state <= replace;
 end case;
 end process;

 rdy <= '1' when register_signal = hold else '0';
 out_num <= std_logic_vector(register_of_A);

end rtl_main;
```

Code of 8-bit-comparator:

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity comparator8bit is
 port (
 in_first_number, in_second_number: in unsigned(7 downto 0);
 out_num: out std_logic_vector(2 downto 0)
 );
end entity;
architecture rtl_comparator of comparator8bit is
 signal equal_num: std_logic_vector(7 downto 0);
 signal greaterfirst, greatersecond: std_logic;
 signal result_number: std_logic_vector(2 downto 0);
```

```
 signal equal,greater, lesser: std_logic;
begin

 equal_num(0) <= (not in_first_number(0)) xnor (not in_second_number(0));
 equal_num(1) <= (not in_first_number(1)) xnor (not in_second_number(1));
 equal_num(2) <= (not in_first_number(2)) xnor (not in_second_number(2));
 equal_num(3) <= (not in_first_number(3)) xnor (not in_second_number(3));
 equal_num(4) <= (not in_first_number(4)) xnor (not in_second_number(4));
 equal_num(5) <= (not in_first_number(5)) xnor (not in_second_number(5));
 equal_num(6) <= (not in_first_number(6)) xnor (not in_second_number(6));
 equal_num(7) <= (not in_first_number(7)) xnor (not in_second_number(7));

 equal<= '1' when equal_num = x"FF" else '0';

 greater <= (in_first_number(7) and not(in_second_number(7)))or
 (in_first_number(6) and not(in_second_number(6)) and equal_num(7))or
 (in_first_number(5) and not(in_second_number(5)) and equal_num(7) and equal_num(6))or
 (in_first_number(4) and not(in_second_number(4)) and equal_num(7) and equal_num(6) and
equal_num(5))or
 (in_first_number(3) and not(in_second_number(3)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4))or
 (in_first_number(2) and not(in_second_number(2)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4) and equal_num(3))or
 (in_first_number(1) and not(in_second_number(1)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4) and equal_num(3) and
equal_num(2))or
```

```
 (in_first_number(0) and not(in_second_number(0)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4) and equal_num(3) and

equal_num(2) and equal_num(1));


 lesser <= (in_second_number(7) and not(in_first_number(7)))or

 (in_second_number(6) and not(in_first_number(6)) and equal_num(7))or

 (in_second_number(5) and not(in_first_number(5)) and equal_num(7) and equal_num(6))or

 (in_second_number(4) and not(in_first_number(4)) and equal_num(7) and equal_num(6) and
equal_num(5))or

 (in_second_number(3) and not(in_first_number(3)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4))or

 (in_second_number(2) and not(in_first_number(2)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4) and equal_num(3))or

 (in_second_number(1) and not(in_first_number(1)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4) and equal_num(3) and

equal_num(2))or

 (in_second_number(0) and not(in_first_number(0)) and equal_num(7) and equal_num(6) and
equal_num(5) and equal_num(4) and equal_num(3) and

equal_num(2) and equal_num(1));


 result_number(2) <= lesser;

 result_number(0) <= greater;

 result_number(1) <= equal_num;

 out_num <= result_number;


end rtl_comparator;
```

Code of 8-bit subtractor:

```
library ieee;

use ieee.std_logic_1164.all;

entity subtractor8bit is

 port (in_first_number, in_second_number: IN STD_LOGIC_VECTOR ( 7 DOWNTO 0);

 out_num : OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0);

 test: out std_logic

 );

end subtractor8bit;

architecture rtl_subtractor8bit of subtractor8bit is

 signal first_bit,second_bit,third_bit,fourth_bit,fifth_bit,sixth_bit,seventh_bit,b8: std_logic;

BEGIN

 test <= in_first_number(0) XOR in_second_number(0) xor '0';

 out_num(0) <= in_first_number(0) XOR in_second_number(0) xor '0';

 first_bit <= (not(in_first_number(0)) and '0') or (not(in_first_number(0)) and
in_second_number(0)) or (in_second_number(0) and '0');

 out_num(1) <= in_first_number(1) XOR in_second_number(1) xor first_bit;

 second_bit <= (not(in_first_number(1)) and first_bit) or (not(in_first_number(1)) and
in_second_number(1)) or (in_second_number(1) and first_bit) ;

 out_num(2) <= in_first_number(2) XOR in_second_number(2) xor second_bit;

 third_bit <= (not(in_first_number(2)) and second_bit) or (not(in_first_number(2)) and
in_second_number(2)) or (in_second_number(2) and second_bit);

 out_num(3) <= in_first_number(3) XOR in_second_number(3) xor third_bit;

 fourth_bit <= (not(in_first_number(3)) and third_bit) or (not(in_first_number(3)) and
in_second_number(3)) or (in_second_number(3) and third_bit);

 out_num(4) <= in_first_number(4) XOR in_second_number(4) xor fourth_bit;
```

```
 fifth_bit <= (not(in_first_number(4)) and fourth_bit) or (not(in_first_number(4)) and
in_second_number(4)) or (in_second_number(4) and fourth_bit);

 out_num(5) <= in_first_number(5) XOR in_second_number(5) xor fifth_bit;

 sixth_bit <= (not(in_first_number(5)) and fifth_bit) or (not(in_first_number(5)) and
in_second_number(5)) or (in_second_number(5) and fifth_bit);

 out_num(6) <= in_first_number(6) XOR in_second_number(6) xor sixth_bit;

 seventh_bit <= (not(in_first_number(6)) and sixth_bit) or (not(in_first_number(6)) and
in_second_number(6)) or (in_second_number(6) and sixth_bit);

 out_num(7) <= in_first_number(7) XOR in_second_number(7) xor seventh_bit;

END rtl_subtractor8bit;
```

Code of Test Bench:

```
library ieee;

use ieee.std_logic_1164.all;

entity mainTB is

end mainTB;

architecture rtl_tb of mainTB is

 signal clock, reset_signal,enable,rdy: std_logic;

 signal in_first_number, in_second_number, out_num : std_logic_vector (7 downto 0);

 type state_type is (hold, replace, subtract);

 signal reset_signal, signal_of_next_state : state_type;

begin

dut: entity work.gcd_main(rtl_main)

 port map(clock,reset_signal,enable,in_first_number,in_second_number,rdy,out_num);

stim_process: process begin

 enable <= '1';
```

```
 in_first_number <= "10001100";
 in_second_number <= "00001100";
 reset_signal <= '0';
 wait for 100ns;
 enable <= '0';
 wait;
 end process;
clock_process: process begin
 clock <= '0';
 wait for 10ns;
 clock <= '1';
 wait for 10ns;
 end process;


configuration rtl_tb of mainTB is
 for rtl_tb
 end for;
end rtl_tb;
```