**LAB 6 REPORT**

**EEE 102**

Robin Umut Kızıl

22003260

**Purpose of Lab:**

The purpose of this lab is to find the GCD Greatest common divisor) of 2 8-bit numbers after commanding with the start button, when using basys3. The GCD of the numbers will be displayed on the leds and when the process is complete, the ready led will also light up.

**Design Specification:**

2 8-bit numbers are first entered on basys 3 with switches. Then, when the start button is pressed, these 2 numbers are entered into the registers in the system as input_1 and input_2 and the GCD starts to be calculated. After the calculation is finished, the ready button and the result are shown as output by leds.

The inputs are as follows:

- clk : The clock input
- reset : Reset input
- start : Start button to start calculating GCD
- input_1 : First 8-bit number
- input_2 : Second 8-bit number

The outputs are as follows:

- output : output that demonstrates with led to show GCD
- ready : Ready led to able to see calculation process

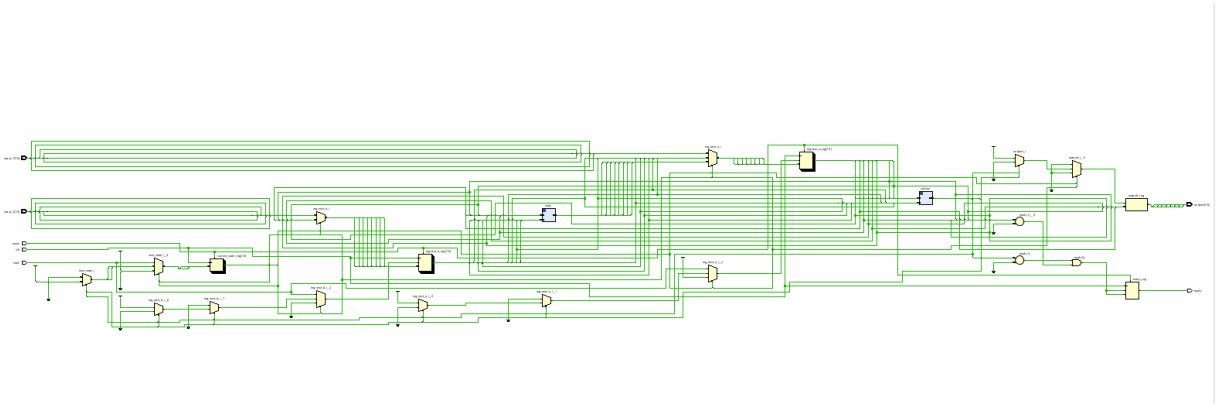

*Figure 1 : RTL Schematic of GCD Calculator*

**Methodology:**

First of all, 3 states were determined and current_state and next_state were added. then one register and 1 next_register were added for every 2 8-bit numbers. A clock process was then written, at each rising edge clock, overwriting the next information with the current information. It was later written as the Euclidean Algorithm moore machine. every time the state changes, it moves to the next state according to the current information. When the calculation is finished, it returns to the defaultt state and transfers the desired information to the output and turns the ready led on.

(note: subtractor and comparator component taken from the Lab 4 and redesign to operating on 8-bits numbers)

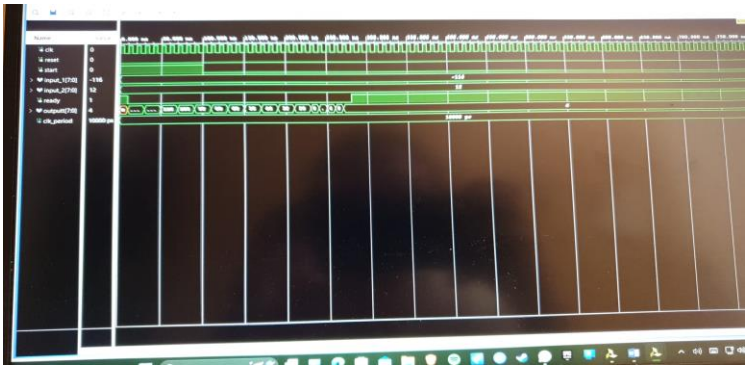

*Figure 2 : Testbench of GCD with the input of 140 and 12*

**Conclusion:**

The Euclidean Algorithm was written using the moore machine. When the input is entered using switches and pressed at the start, the information is loaded into the registers and operations are performed. When the process is finished, the ready led lights up and the output is shown by the leds, so the lab is successful. For example, when we enter the number 140 and 12, we can get the number 4 (see in figure 2, figure3 and figure 4).


*Figure 3 : before to pushing 'Start' button*

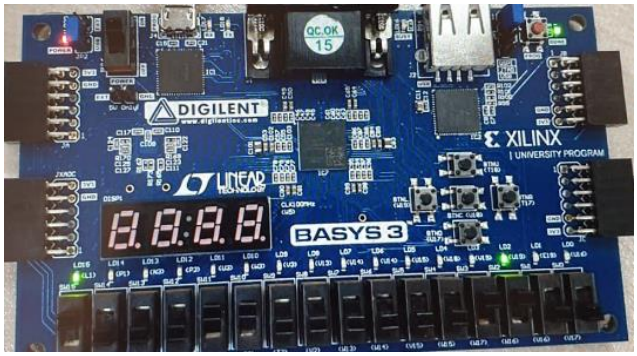


*Figure 4 : after to pushing 'Start' button*

**Answers for Questions:**

1. **What was your algorithm to calculate the GCD?**

   I use Euclidean Algorithm in order to calculate GCD. Euclidean Algorithm is that finds GCD by continuously subtracting the smaller number from the larger number until the remaining numbers are the same.

2. **What was your algorithm to calculate the GCD? 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 module I created is a FSM and it's a moore machine since the output depends on the current state. The FSM version is possibly cheaper than combinational because minimalizing the 2^16 input circuit will need a lot of gates. Since the combinational version delay just the gate, it’s probably faster than FSM which is depends on clock rates. A combinational circuit has the drawback of being expensive whereas an FSM has the drawback of being slow.

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

   When the start key changes from '1' to '0', the ready led lights up in 18 rising edges, which means 18 clock cycles, so GCD is calculated. If we remove the subtract state and write it under the charged as a combinational, the time will be shortened. but as I said in the previous question, the cost of the circuit increases.

**Appendices A:**


*Figure 5: number with 6 and 180, conclusion is 6*

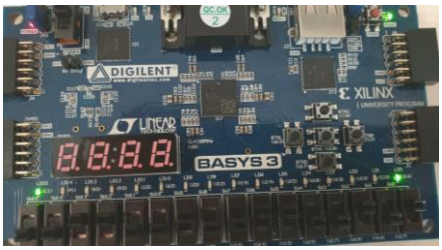


*Figure 6 : number with 50 and 101, conclusion is 1*

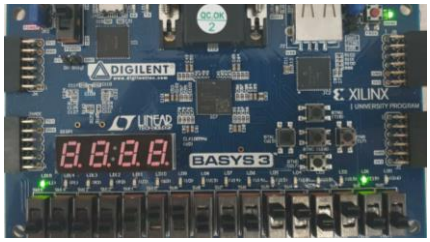


*Figure 7 : number with 14 and 52, conclusion is 2*

**Appendices B:**

**GCD_8bit_Top**

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.numeric_std;



entity GCD_8bit_Top is

    Port ( clk     : in std_logic ;

           reset   : in std_logic;
```

```
        start   : in std_logic ;
        input_1 : in STD_LOGIC_VECTOR (7 downto 0);
        input_2 : in STD_LOGIC_VECTOR (7 downto 0);
        outputt : out STD_LOGIC_VECTOR (7 downto 0);
        ready   : out std_logic
        );
end GCD_8bit_Top;

architecture Behavioral of GCD_8bit_Top is

component Comparator is
Port ( in_a : in STD_LOGIC_vector(7 downto 0);
        in_b : in STD_LOGIC_vector(7 downto 0);
        result : out STD_LOGIC_VECTOR (1 downto 0)); -- "00" equal , "01" a>b , "10" a<b
end component;

component subtraction is
    Port ( x : in STD_LOGIC_VECTOR (7 downto 0);
```

```
        y : in STD_LOGIC_VECTOR (7 downto 0);
        outz : out STD_LOGIC_VECTOR (7 downto 0));
end component;



type state_type is (defaultt, charged, subtracting);
signal current_state, next_state : state_type;

Signal register_a, register_b,reg_next_a,reg_next_b : std_logic_vector(7 downto 0);
signal subtractout: std_logic_vector(7 downto 0);
signal comout: std_logic_vector (1 downto 0);

begin

comps : Comparator  port map(in_a => register_a, in_b =>register_b, result =>comout);

subs  : subtraction port map(x => register_a , y => register_b , outz => subtractout);
```

```
process(clk,reset)

begin

if reset = '1' then

current_state <= defaultt;

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

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

ready <='0';

elsif rising_edge(clk) then

current_state <= next_state;

register_a <= reg_next_a;

register_b <= reg_next_b;

if comout = "00" and register_a /="00000000" then
```

```
ready <= '1';

end if;




end if;

end process;









process(current_state, start)

begin



reg_next_a <= register_a;

reg_next_b <= register_b;



case current_state is
```

```
when defaultt =>

    if start ='1' then

        reg_next_a <= input_1;

        reg_next_b <= input_2;

        next_state <= charged;

    else

        next_state <= defaultt;

    end if;

when charged =>
```

```
    if comout = "00" then
        next_state <= defaultt;
        outputt <= register_a;


    else
        if comout = "10" then
        reg_next_a <= register_b;
        reg_next_b <= register_a;
        end if;
    next_state <= subtracting;
    end if;


when subtracting =>
reg_next_a <= subtractout;
next_state <= charged;
```

```
end case;

end process;

end Behavioral;
```

**Comparator**

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

entity Comparator is

    Port ( in_a : in STD_LOGIC_vector(7 downto 0);
           in_b : in STD_LOGIC_vector(7 downto 0);
           result : out STD_LOGIC_VECTOR (1 downto 0)); -- "00" equal , "01" a>b , "10" a<b
```

```
end Comparator;

architecture Behavioral of Comparator is

signal e: STD_LOGIC_vector(7 downto 0);
signal AeqB,AgtB,AltB: STD_LOGIC;

begin
e(0) <= not(in_a(0) xor in_b(0));
e(1) <= not(in_a(1) xor in_b(1));
e(2) <= not(in_a(2) xor in_b(2));
e(3) <= not(in_a(3) xor in_b(3));
e(4) <= not(in_a(4) xor in_b(4));
e(5) <= not(in_a(5) xor in_b(5));
e(6) <= not(in_a(6) xor in_b(6));
e(7) <= not(in_a(7) xor in_b(7));

AeqB <= e(0) and e(1) and e(2) and e(3) and e(4) and e(5) and e(6);
```

AgtB <= (in_a(7) and NOT in_b(7)) or (e(7) and in_a(6) and NOT in_b(6)) or (e(7) and e(6) and in_a(5) and NOT in_b(5)) or (e(7) and e(6) and e(5) and in_a(4) and NOT in_b(4)) or (e(6) and e(5) and e(4) and in_a(3) and NOT in_b(3)) or (e(7) and e(6) and e(5) and e(4) and e(3) and in_a(2) and NOT in_b(2)) or (e(7) and e(6) and e(5) and e(4) and e(3) and e(2) and in_a(1) and NOT in_b(1)) or (e(7) and e(6) and e(5) and e(4) and e(3) and e(2) and e(1) and in_a(0) and NOT in_b(0));

AltB <= (in_b(7) and NOT in_a(7)) or (e(7) and in_b(6) and NOT in_a(6)) or (e(7) and e(6) and in_b(5) and NOT in_a(5)) or (e(7) and e(6) and e(5) and in_b(4) and NOT in_a(4)) or (e(6) and e(5) and e(4) and in_b(3) and NOT in_a(3)) or (e(7) and e(6) and e(5) and e(4) and e(3) and in_b(2) and NOT in_a(2)) or (e(7) and e(6) and e(5) and e(4) and e(3) and e(2) and in_b(1) and NOT in_a(1)) or (e(7) and e(6) and e(5) and e(4) and e(3) and e(2) and e(1) and in_b(0) and NOT in_a(0));

process(AeqB,AgtB,AltB)

begin

if AeqB = '1' then

result <= "00";

```
    elsif  AgtB = '1' then
        result <= "01";



    elsif AltB = '1' then
        result <= "10";


     else
        result <= "00";


    end if;
    end process;
end Behavioral;
```

**subtraction**

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

```
entity subtraction is
    Port ( x : in STD_LOGIC_VECTOR (7 downto 0);
           y : in STD_LOGIC_VECTOR (7 downto 0);
           outz : out STD_LOGIC_VECTOR (7 downto 0));
end subtraction;



architecture Behavioral of subtraction is



component one_bit_adder
    port(x,y,c_in : in std_logic;
         sum,c_out    : out std_logic
    );
end component;



signal c0,c1,c2,c3,c4,c5,c6,c7,trash : std_logic;
```

```
signal noty0,noty1,noty2,noty3,noty4,noty5,noty6,noty7 : std_logic;



begin

c0 <= '1';


noty0 <= not y(0);

noty1 <= not y(1);

noty2 <= not y(2);

noty3 <= not y(3);

noty4 <= not y(4);

noty5 <= not y(5);

noty6 <= not y(6);

noty7 <= not y(7);



st1 : one_bit_adder port map(

    x => x(0),y => noty0,c_in => c0,sum => outz(0),c_out => c1
```

```
);

st2 : one_bit_adder port map(
    x => x(1),y => noty1,c_in => c1,sum => outz(1),c_out => c2
);

st3 : one_bit_adder port map(
    x => x(2),y => noty2,c_in => c2,sum => outz(2),c_out => c3
);

st4 : one_bit_adder port map(
    x => x(3),y => noty3,c_in => c3,sum => outz(3),c_out => c4
);

st5 : one_bit_adder port map(
    x => x(4),y => noty4,c_in => c4,sum => outz(4),c_out => c5
);
```

```
st6 : one_bit_adder port map(
    x => x(5),y => noty5,c_in => c5,sum => outz(5),c_out => c6
);

st7 : one_bit_adder port map(
    x => x(6),y => noty6,c_in => c6,sum => outz(6),c_out => c7
);

st8 : one_bit_adder port map(
    x => x(7),y => noty7,c_in => c7,sum => outz(7),c_out => trash
);

end Behavioral;
```

**one_bit_adder**

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

```
entity one_bit_adder is
    Port ( x     : in STD_LOGIC;
           y     : in STD_LOGIC;
           c_in  : in STD_LOGIC;
           sum   : out STD_LOGIC;
           c_out : out STD_LOGIC);
end one_bit_adder;

architecture Behavioral of one_bit_adder is

begin

sum   <= x xor y xor c_in;
c_out <= (x and y) or (y and c_in) or (x and c_in);

end Behavioral;
```