Bilkent University Electrical and Electronics Department

EE102-03 Lab 6 Report:

Greatest Common Divider

Cankut Bora Tuncer – 22001770

Purpose:

This lab aims to design a circuit that finds the greatest common divisor of two 8-bit numbers. There is 1 enable and 1 reset button and 16 switches for the 2 8-bit numbers. The output is projected to the LEDs on the Basys3.

Methodology:

There are many algorithms for finding the GCD of 2 numbers but the one it is chosen in this design is as follows(Figure 1):

- Step1: Compare the numbers.
- Step2: If the numbers are equal to each other then the gcd is one of the 2 numbers.
- Step3: If the numbers are not equal, then subtract the larger one with the other until the Step 2 is satisfied.

To initiate this operation, there used a Moore Machine with 3 states: "hold", "replace" and "substract". The purpose of the register states will be discussed in the following section. If the enable button is asserted and the reset button is '0', the algorithm initiates. When the necessary conditions are met, the indicator output, "ready", lights up and the output is projected over the LEDs.

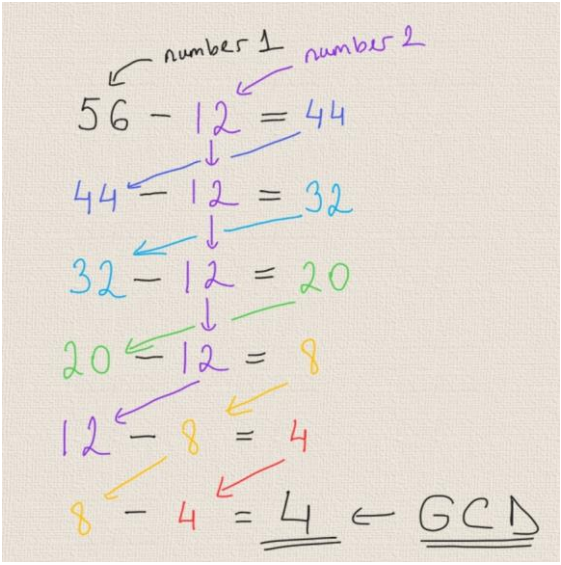

Figure 1 The GCD Algorithm

Design Specifications:

There are 5 inputs and 2 outputs. The first input is the clock value and it is fixed(100MHz). The second input is the reset selection, the output is cleared. 3rd and 4th inputs are the 8 bit number selections. For the last input, enable, is used to initiate the calculating sequence.

The design is constructed in a modular fashion. The top module is "gcd_main.vhd". The body of the Gcd algorithm is utilized in here. In the decided design it is used a Moore Machine for convinience. There are 3 states in the model:

- Hold: If both numbers are equal then it stays in this state, else it moves to the "Replace" state.
- Replace: If both numbers are equal then it moves to the "Hold" state, else if number2 is greater than number1, the register1 and register2 which stores the numbers are swapped with each other; then it moves to "Subtract" state.
- Subtract: The number1 is subtracted from number then it moves to "Replace" state.

In order to do the comparing and subtracting operation between 2 bitwise numbers, an ALU with 2 operations is constructed. The "comparator8bit.vhd" has 2 inputs and 1 output. It receives 2 8-bit numbers and outputs a 3-bit number. If; numbers are equal: "010", number1 is greater than number2: "001", number2 is greater than number1 "100". The "subtractor8bit.vhd" has 2 inputs and 1 output. It receives 2 8-bit numbers and outputs a 8-bit number. There used 8 full-subtractors to initiate the subtraction operation between 2 Unsigned numbers. Both the comparator and subtractor module is asynchronious, they are clock independent since the operation needs to be done at instant.

When the GCD is found, the FSM will stay in the "Hold" state. When it happens, the "ready" output lights up, indicating that the calculations are over, and the output is projected over the 8 LEDs, each representing each bit of the output.

Results:

For testing purposes, the initial step is to check that there are not multi-driven nets on the RTL Schematic. As seen in the Figure 2, 3 and 4 the connections are accurately matched with each other with no errors.

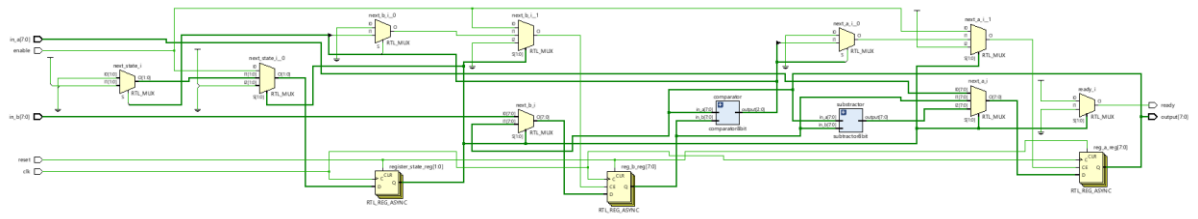

Figure 2 RTL Schematic of the "gcd_main.vhd"

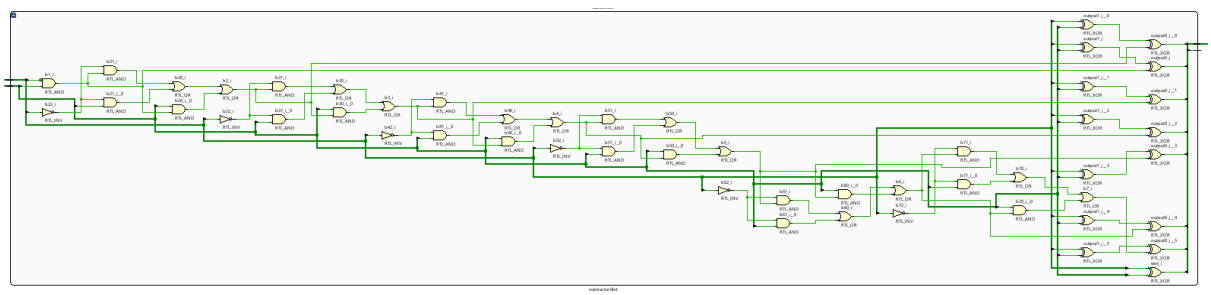

Figure 3 RTL Schematic of the "comparator8bit.vhd"

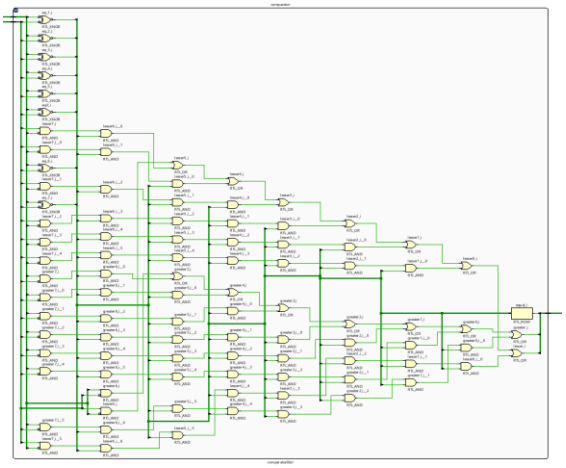

Figure 4 RTL Schematic of the "subtractor8bit.vhd"

The test bench is tested for the inputs in_a = "10001100" and in_b = "00001100" (Figure 5). The enable is asserted between 0ns and 100ns. The output is ready when the "ready" output is '1'. The test bench gives the result output = "00000100".

Figure 5 The waveform of the "mainTB.vhd"

Finally, the design is implemented on the Basys 3 board. The board configuration can be seen in Figure 6.

Figure 6 The Basys 3 configuration

When the reset button is asserted:

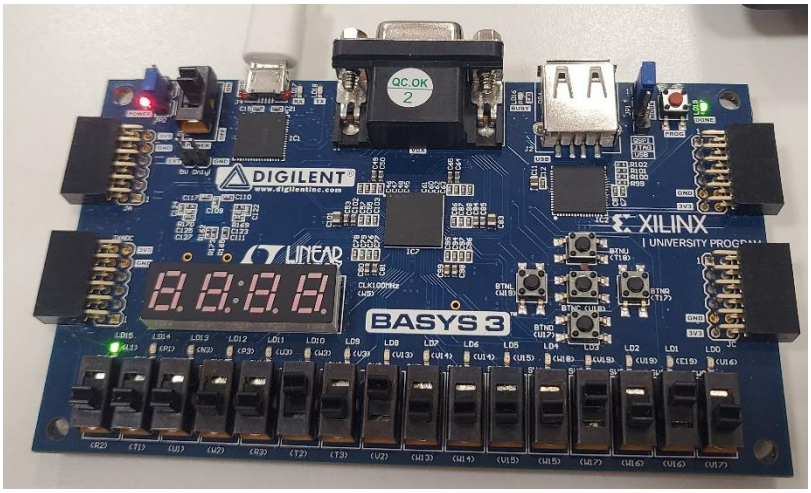

Figure 7 Configuration Number 1

When the enable is asserted and released and the inputs and the outputs are:

in_a: “00001010”

reset: ‘0’

in_b: “00000101”

ready: ‘1’

output: “00000101”

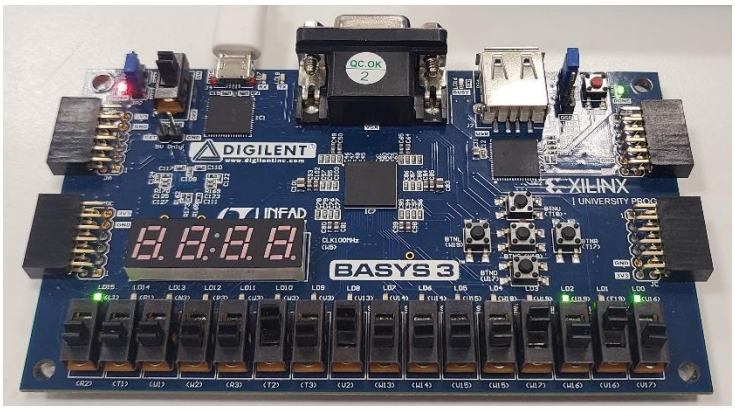

Figure 8 Configuration Number 2

When the enable is asserted and released and the inputs are:

in_a: “00000101”

reset: ‘0’

in_b: “00001010”

ready: ‘1’

output: “00000101”

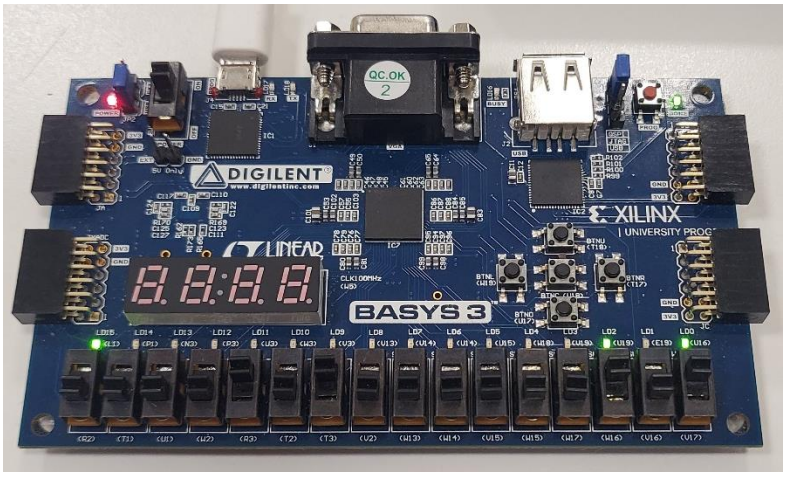

Figure 9 Configuration Number 3

When the enable is asserted and released and the inputs are:

in_a: “10001100”

reset: ‘0’

in_b: “00001100”

ready: ‘1’

output: “00000100”

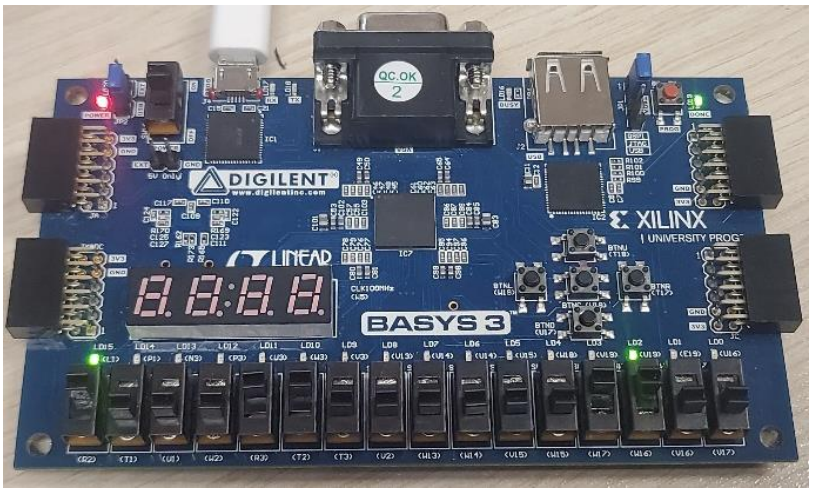

Figure 10 Configuration Number 4

Conclusion:

The purpose of this lab was to design a circuit that finds the greatest common divider of two 8-bit number as a 8-bit output. The simulation and implementation results were consistent with the expected results. The algorithm used in the design is the famous Simplified Euclidean Algorithm for finding GCD. The module used in the design is an FSM, Moore machine. A combinational circuit would be faster and better and in performance compared to an FSM. However, since a combinational circuit cannot store data, in order to calculate the gcd, the designing method would start from constructing the truth table. But it would be a hard task since there are total of 18-bit input an 8 bit output data (262144

combinations). Although in smaller tasks, combinational circuits are cheaper than FSMs, in cases like this, it is possible to cost more to design a combinational circuit than an FSM, but at the end combinational would be faster. In this design, both FSM and combinational circuits are used. For arithmetic operations such as substraction and comparison, combinational circuit is used because it is cheaper and faster. If there used an FSM model for arithmetic operations, since they are clock driven, the operation would take much more time and have more cost. It takes 296 clock ticks to find the gcd of 140 and 12. The results can be optimized by reducing the clock dependent operations. Furthermore, a Meally machine would complete the operation in lesser clock ticks. But the faster result will be given by a combinational circuit design.

References:

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

https://brilliant.org/wiki/greatest-common-divisor/

https://www.geeksforgeeks.org/full-subtractor-in-digital-logic/

Appendix:

gcd_main.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
------------------------------------------------------------
entity gcd_main is
    port(
        clk, reset: in std_logic;
```

```
        enable: in std_logic;
        in_a, in_b: in std_logic_vector(7 downto 0);
        ready: out std_logic;
        output: 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_state, next_state : state_type;

    signal reg_a, reg_b, next_a, next_b: unsigned(7 downto 0);
    signal comp_a,comp_b : unsigned(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);

begin
    ----------------------------------------------------------------
    comparator: entity work.comparator8bit(rtl_comparator)
        port map(in_a => comp_a, in_b=> comp_b, output=> comp_out);

    substractor: entity work.subtractor8bit(rtl_subtractor8bit)
        port map(in_a => subs_a, in_b => subs_b, output => subs_out);
    ----------------------------------------------------------------
    process(clk,reset)
    begin
        if reset='1' then
            register_state <= hold;
```

```
        reg_a <= (others=>'0');
        reg_b <= (others=>'0');
    elsif (rising_edge(clk)) then
        register_state <= next_state;
        reg_a <= next_a;
        reg_b <= next_b;
    end if;
end process;
-----------------------------------------------------------------
process(register_state,reg_a,reg_b,enable,in_a,in_b)
begin
    next_a <= reg_a;
    next_b <= reg_b;
    comp_a <= reg_a;
    comp_b <= reg_b;
    subs_a <= std_logic_vector(reg_a);
    subs_b <= std_logic_vector(reg_b);

    case register_state is
        ---------------------------------------------------------------
        when hold =>
            if enable = '1' then
                next_a <= unsigned(in_a);
                next_b <= unsigned(in_b);
                next_state <= replace;
            else
                next_state <= hold;
            end if;
        ---------------------------------------------------------------
        when replace =>
```

```
            if (comp_out(1) = '1') then
               next_state <= hold;
            else
               if(comp_out(2) = '1') then
                  next_a <= reg_b;
                  next_b <= reg_a;
               end if;
               next_state <= substract;
            end if;
         -------------------------------------------------------------
         when substract =>
            next_a <= unsigned(subs_out);
            next_state <= replace;
      end case;
   end process;
   -------------------------------------------------------------
   ready <= '1' when register_state = hold  else '0';
   output <= std_logic_vector(reg_a);
   -------------------------------------------------------------
end rtl_main;
```

comparator8bit.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
-------------------------------------------------------------
entity comparator8bit is
   port (
      in_a, in_b: in unsigned(7 downto 0);
```

```
        output: out std_logic_vector(2 downto 0)
    );
end entity;
-------------------------------------------------------------
architecture rtl_comparator of comparator8bit is
    signal eq: std_logic_vector(7 downto 0);
    signal grtr1, grtr2: std_logic;
    signal result: std_logic_vector(2 downto 0);
    signal equal,greater, lesser: std_logic;
begin
    ---------------------------------------------------------------
    eq(0) <= (not in_a(0)) xnor (not in_b(0));
    eq(1) <= (not in_a(1)) xnor (not in_b(1));
    eq(2) <= (not in_a(2)) xnor (not in_b(2));
    eq(3) <= (not in_a(3)) xnor (not in_b(3));
    eq(4) <= (not in_a(4)) xnor (not in_b(4));
    eq(5) <= (not in_a(5)) xnor (not in_b(5));
    eq(6) <= (not in_a(6)) xnor (not in_b(6));
    eq(7) <= (not in_a(7)) xnor (not in_b(7));

    equal<= '1' when eq = x"FF" else '0';
    ---------------------------------------------------------------
    greater <= (in_a(7) and not(in_b(7)))or
            (in_a(6) and not(in_b(6)) and eq(7))or
            (in_a(5) and not(in_b(5)) and eq(7) and eq(6))or
            (in_a(4) and not(in_b(4)) and eq(7) and eq(6) and eq(5))or
            (in_a(3) and not(in_b(3)) and eq(7) and eq(6) and eq(5) and eq(4))or
            (in_a(2) and not(in_b(2)) and eq(7) and eq(6) and eq(5) and eq(4) and eq(3))or
            (in_a(1) and not(in_b(1)) and eq(7) and eq(6) and eq(5) and eq(4) and eq(3) and
eq(2))or
```

```
        (in_a(0) and not(in_b(0)) and eq(7) and eq(6) and eq(5) and eq(4) and eq(3) and
eq(2) and eq(1));
    ------------------------------------------------------------
    lesser <= (in_b(7) and not(in_a(7)))or
            (in_b(6) and not(in_a(6)) and eq(7))or
            (in_b(5) and not(in_a(5)) and eq(7) and eq(6))or
            (in_b(4) and not(in_a(4)) and eq(7) and eq(6) and eq(5))or
            (in_b(3) and not(in_a(3)) and eq(7) and eq(6) and eq(5) and eq(4))or
            (in_b(2) and not(in_a(2)) and eq(7) and eq(6) and eq(5) and eq(4) and eq(3))or
            (in_b(1) and not(in_a(1)) and eq(7) and eq(6) and eq(5) and eq(4) and eq(3) and
eq(2))or
            (in_b(0) and not(in_a(0)) and eq(7) and eq(6) and eq(5) and eq(4) and eq(3) and
eq(2) and eq(1));
    ------------------------------------------------------------
    result(2) <= lesser;
    result(0) <= greater;
    result(1) <= equal;
    output <= result;
    ------------------------------------------------------------
end rtl_comparator;
```

subtractor8bit.vhd

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


entity subtractor8bit is
    port (in_a, in_b: IN STD_LOGIC_VECTOR ( 7 DOWNTO 0);
        output : OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0);
        test: out std_logic
       );
end subtractor8bit;
```

```
architecture rtl_subtractor8bit of subtractor8bit is
    signal b1,b2,b3,b4,b5,b6,b7,b8: std_logic;

BEGIN
    test <= in_a(0) XOR in_b(0) xor '0';
    output(0) <= in_a(0) XOR in_b(0) xor '0';
    b1 <= (not(in_a(0)) and '0') or (not(in_a(0)) and in_b(0)) or (in_b(0) and '0');

    output(1) <= in_a(1) XOR in_b(1) xor b1;
    b2 <= (not(in_a(1)) and b1) or (not(in_a(1)) and in_b(1)) or (in_b(1) and b1) ;

    output(2) <= in_a(2) XOR in_b(2) xor b2;
    b3 <= (not(in_a(2)) and b2) or (not(in_a(2)) and in_b(2)) or (in_b(2) and b2);

    output(3) <= in_a(3) XOR in_b(3) xor b3;
    b4 <= (not(in_a(3)) and b3) or (not(in_a(3)) and in_b(3)) or (in_b(3) and b3);

    output(4) <= in_a(4) XOR in_b(4) xor b4;
    b5 <= (not(in_a(4)) and b4) or (not(in_a(4)) and in_b(4)) or (in_b(4) and b4);

    output(5) <= in_a(5) XOR in_b(5) xor b5;
    b6 <= (not(in_a(5)) and b5) or (not(in_a(5)) and in_b(5)) or (in_b(5) and b5);

    output(6) <= in_a(6) XOR in_b(6) xor b6;
    b7 <= (not(in_a(6)) and b6) or (not(in_a(6)) and in_b(6)) or (in_b(6) and b6);

    output(7) <= in_a(7) XOR in_b(7) xor b7;

END rtl_subtractor8bit;
```

mainTB.vhd

```
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------------------------
entity mainTB is
end mainTB;
--------------------------------------------------------------------
architecture rtl_tb of mainTB is
    signal clk, reset,enable,ready: std_logic;
    signal in_a, in_b, output     : std_logic_vector (7 downto 0);

    type state_type is (hold, replace, subtract);
    signal register_state, next_state : state_type;

begin
--------------------------------------------------------------------
    dut: entity work.gcd_main(rtl_main)
        port map(clk,reset,enable,in_a,in_b,ready,output);
--------------------------------------------------------------------
    clock_process: process begin
        clk <= '0';
        wait for 10ns;
        clk <= '1';
        wait for 10ns;
    end process;
--------------------------------------------------------------------
    stim_process: process begin
        enable <= '1';
        in_a <= "10001100";
        in_b <= "00001100";
```

```
        reset <= '0';
        wait for 100ns;
        enable <= '0';
        wait;
    end process;
-------------------------------------------------------------------------
end rtl_tb;
-------------------------------------------------------------------------
configuration rtl_tb of mainTB is
    for rtl_tb
    end for;
end rtl_tb;
```