# Report: Greatest Common Divisor

Bilkent University Electrical and Electronics Department
EE102-01 Lab 5

Semih Akkoç - 22103132

November 28, 2022

## Purpose:

The intent of this lab was to study the essentials of the finite state machine and design a system that has the ability to find the greatest common divisor of two eight-bit numbers. Additionally, getting familiar with this unit will grant a fundamental understanding of the Mealy and Moore machines and how it operates and the implementation of this circuit on FPGA using VHDL in a modular fashion.

## Questions:

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

An approach that incorporates the modulo operation has been chosen to calculate the greatest common divisor of two numbers. The way these algorithm works is simply easier way than the Euclidean algorithm. Further explanations and clarifications have been done in the "Methodology" part.

**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 design has used a Mealy machine, and therefore, this module is a finite state machine. Although combinational circuits can serve operations that need a smaller number of inputs with much cheaper costs and faster outcomes, finite-state machines can handle operations with a greater number

of inputs superior. Nonetheless, since finite-state machines are dependent on the clock cycles for most of their operations, this might yield more time for simple operations where combinational circuits can handle those quicker.

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

To obtain information about how many clock cycles it took to calculate GCD a simulation has been written and from the simulation, it has been observed that this process takes 15 clock cycles. The overall result is open to improvement and it can be optimized by either finding a better algorithm or creating a better, faster and cheaper design by reducing the operations that depend on the clock.

# Methodology:

In order to proceed with this lab work, one needed to comprehend how to compute the greatest common divisor, and there are several different ways to implement and conclude the result one can be described as a naive method where a division table is used to find their common prime numbers and multiply them. Another can be explained as doing subtraction until they are equal, and the number where they are equal is the greatest common divisor. Finally, an improved version of the previous idea can be described as taking modulo rather than doing subtraction. The last idea has been used in this lab work since it was easy to implement, and after implementation, the numbers have been obtained by switches and output, which is the result of these two eight-bit numbers greatest common divisor has been shown via utilization of LEDs. The way the greatest common divisor module works can be explained as using three states and implementing two registers as well as a comparator where these two numbers get compared, and the results are sent through a 2-bit standard logic vector. Furthermore, in order to register to work a clock and enable inputs has been defined. The algorithm used in this design can be boiled down to the following list.

- Control whether the numbers are the same or not.
- If they are not equal, find the bigger one and take apply a modulo operation with respect to the small one.
- Extend this process until the result of the modulo operation comes to zero, then the previous smaller number is the greatest common divisor.

# Design Specifications:

While implementing this greatest common divisor module, five inputs and two outputs were used, and this design can be summarized as a Mealy machine since the states include a loop therefore, this idea refers to the previously mentioned finite state machine. These five inputs can be briefly explained as one being the clock and the other being the reset input which resets the outputs also. In inputs, there are two eight-bit numbers, and an enable input where it loads the values gathered from switches. When it comes to outputs, one is 8-bit result outputs which is the outcome of these two 8-bit binary numbers' greatest common divisor. This design's inputs and outputs can be observed in the following table.

- GCD.vhd

```
 CLK: in std_logic;
RST: in std_logic;
ENABLE: in std_logic;
A: in std_logic_vector (7 downto 0);
B: in std_logic_vector (7 downto 0);
O: out std_logic_vector (7 downto 0); -- gcd
READY: out std_logic
```

Subsequently, other modules have been utilized in a modular fashion. "GCD.vhd" is the module that utilizes the Comparator and Modulo8Bit modules. Both of those modules can be inputs, and outputs can be found in the following list respectively:

- Comparator.vhd

```
A: in std_logic_vector (7 downto 0);
B: in std_logic_vector (7 downto 0);
R: out std_logic_vector(1 downto 0) -- result vector
```

R vector has three modes where "00" is A equals B, "01" is A is greater than B, "10" is A is lesser than B finally, "11" is don't care.

- Modulo8Bit.vhd

```
A: in std_logic_vector (7 downto 0);
B: in std_logic_vector (7 downto 0);
D: out std_logic_vector (7 downto 0)
```

# Results:

The suggested greatest common divisor modules design schematics and simulations of it, and photos of the implementation on BASYS3 can be found, and the codes regarding the modules can be found in the appendices part.

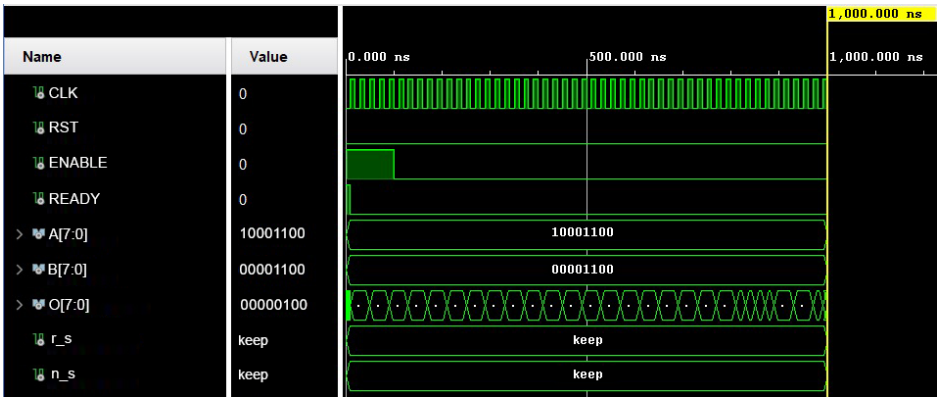



Figure 1: Simulation, main testbench of the design ($GCD_{t}estbench.vhd$).

The following part includes the schematic of greatest common divisors and the top modules RLT schematic Figure 2.

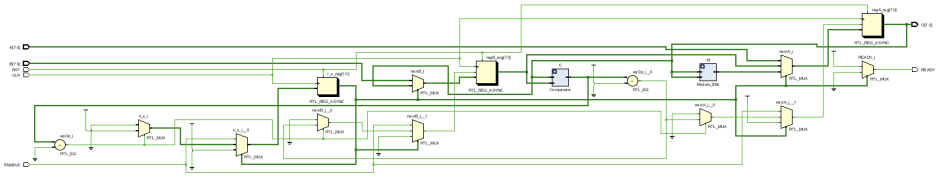


Figure 2: The schematic.

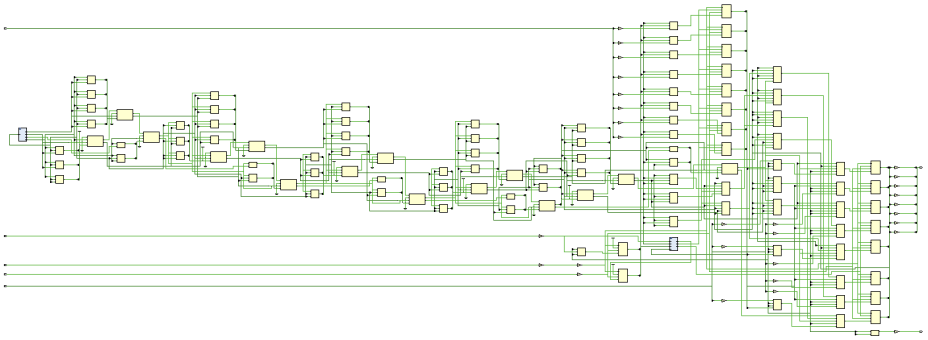
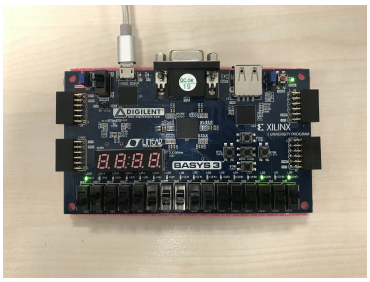
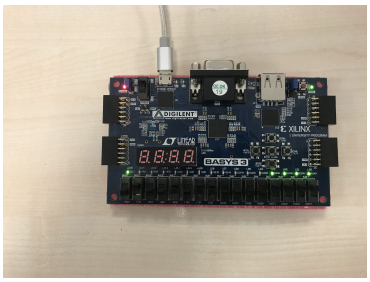


Figure 3: The RLT schematic.

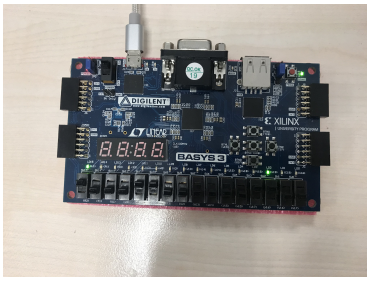
In favor of demonstrating the design on the BASYS3 board following images are chosen to show some of the operations done on the board.



(a) Greatest common divisor of 75 and 5 which is 5.



(b) Greatest common divisor of 75 and 5 which is 15.



(c) Greatest common divisor of 140 and 12 which is 4.

Figure 4: Working circuit on the BASYS3 board.

# Conclusion:

The greatest common divisor module has been implemented using VHDL with the modulo and the comparator modules. Throughout this lab work, several concepts have been reinforced, such as how a Mealy machine works, how to design and implement such a circuit, and how to fundamentally combine a combinational and a sequential circuit.

# Appendices:

**Note:**

Since all of the unnecessary comments in the codes have been excluded, the remaining pages may seem few. On the contrary, by discarding the redundant comments, the real work done by the student has been brought to light which shows more quality pages than useless ones.

## Code 1: (GCD.vhd)

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

entity GCD is
 Port (
    CLK: in std_logic;
    RST: in std_logic;
    ENABLE: in std_logic;
    A: in std_logic_vector (7 downto 0);
    B: in std_logic_vector (7 downto 0);
    O: out std_logic_vector (7 downto 0); -- greatest common
       divisior
    READY: out std_logic
    );
end GCD;


architecture Behavioral of GCD is

component Comparator
port (
    A: in std_logic_vector (7 downto 0);
    B: in std_logic_vector (7 downto 0);
    R: out std_logic_vector(1 downto 0)
);
end component;

component Modulo_8Bit
port (
    A: in std_logic_vector (7 downto 0);
    B: in std_logic_vector (7 downto 0);
    D: out std_logic_vector (7 downto 0)
);
end component;


signal compA : std_logic_vector (7 downto 0);
signal compB : std_logic_vector (7 downto 0);
```

```
signal compD : std_logic_vector (1 downto 0);

-- signal mod_A : std_logic_vector (0 downto 7);
-- signal mod_B : std_logic_vector (0 downto 7);
-- signal mod_res : std_logic_vector (0 downto 1);

signal modA : std_logic_vector (7 downto 0);
signal modB : std_logic_vector (7 downto 0);
signal modD : std_logic_vector (7 downto 0);

signal regA : std_logic_vector (7 downto 0);
signal regB : std_logic_vector (7 downto 0);

signal nextA : std_logic_vector (7 downto 0);
signal nextB : std_logic_vector (7 downto 0);

signal check : std_logic;

type state is (keep, change, t_m); -- keep, change, take modulo
signal r_s : state;
signal n_s : state;

begin

C : Comparator PORT MAP(compA ,compB, compD);
M : Modulo_8Bit PORT MAP(modA ,modB ,modD);

process (CLK, RST) begin
    if (RST='1') then
        r_s <= keep;
        regA <= (others=>'0');
        regB <= (others=>'0');
    elsif (rising_edge(CLK)) then
        r_s <= n_s;
        regA <= nextA;
        regB <= nextB;
    end if;
end process;

process (r_s, regA, regB, ENABLE, A, B) begin
    nextA <= regA;
    nextB <= regB;
    compA <= regA;
```

```
    compB <= regB;

    modA <= regA;
    modB <= regA;

    case r_s is
        when keep =>
            if (ENABLE = '1') then
                nextA <= A;
                nextB <= B;
                n_s <= change;
            else
                n_s <= keep;
            end if;

        when change =>
            if (compD = "00") then
                n_s <= keep;
            else
                if (compD = "10") then
                    nextA <= regB;
                    nextB <= regA;
                end if;
                n_s <= t_m;
            end if;
        when t_m =>
            nextA <= modD;
            n_s <= change;
    end case;
end process;

READY <= '1' when r_s = keep else '0';
O <= regA;

end Behavioral;
```

## Code 2: (Comparator.vhd)

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

```
use IEEE.NUMERIC_STD.ALL;

entity Comparator is
 Port (
   A : in std_logic_vector(7 downto 0);
   B : in std_logic_vector(7 downto 0);
   R : out std_logic_vector(1 downto 0)
 );
end Comparator;

architecture Behavioral of Comparator is

begin

process (A, B) begin
   if (A=B) then
      R <= "00";
   elsif (A>B) then
      R <= "01";
   elsif (A<B) then
      R <= "10";
   end if;
end process;

end Behavioral;
```

## Code 3: (Module8Bit.vhd)

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


entity Modulo_8Bit is
    Port (
        A: in std_logic_vector (7 downto 0);
        B: in std_logic_vector (7 downto 0);
        D: out std_logic_vector (7 downto 0)
    );
```

```
end Modulo_8Bit;

architecture Behavioral of Modulo_8Bit is

signal A_int : integer;
signal B_int : integer;
signal OUT_int : integer;

begin

A_int <= to_integer(unsigned(A));
B_int <= to_integer(unsigned(B));

OUT_int <= A_int mod B_int;

D <= std_logic_vector(to_unsigned(OUT_int, D'length));

end Behavioral;
```

## Code 4: ($\mathbf{GCD}_{testbench.vhd}$)

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


entity GCD_testbench is
--  Port ( );
end GCD_testbench;

architecture Behavioral of GCD_testbench is

component GCD
Port (
    CLK: in std_logic;
    RST: in std_logic;
    ENABLE: in std_logic;
    A: in std_logic_vector (0 downto 7);
    B: in std_logic_vector (0 downto 7);
    READY: out std_logic;
    O: out std_logic_vector (0 downto 7) -- greatest common divisior
    );
```

```
end component;

signal CLK, RST, ENABLE, READY: std_logic;
signal A, B, O : std_logic_vector (7 downto 0);

type state is (keep, change, t_m);
signal r_s, n_s : state;

begin

G : PORT MAP(CLK,RST,ENABLE,A,B,READY,O);

CLK_PROCESS: process begin
CLK <= '0';
wait for 10ns;
CLK <= '1';
wait for 10ns;
end process;

STIM_PROCESS: process begin
ENABLE <= '1';
A <= "10001100";
B <= "00001100";
RST <= '0';
wait for 100ns;
ENABLE <= '0';
wait;
end process;

end Behavioral;
```

# References:


- vhdl_registers_and_counters.pdf Accessed from Moodle.
- https://github.com/CankutBoraTuncer/Bilkent-EEE102-Labs