EEE102-02 Lab 6 Report:

Greatest Common Divisor

Kaan Ermertcan - 22202823

13.11.2023

Purpose:

In this lab, we will design a circuit that calculates the greatest common divisor (GCD) of two 8-bit numbers. We will test it with a testbench and implement it on a Basys 3 board. We will learn about finite state machines and registers when designing our circuit.

Design Specifications:

My design will get two 8-bit unsigned binary numbers as inputs from Basys 3's all switches and start calculating their GCD after the center button is pressed and released. After the calculation is completed, the result will be displayed on the 8 LEDs starting from the rightmost LED. All numbers' LSB are on the right. Calculations happen on two registers that are updated with every clock cycle. A clock of 1KHz will used.

Methodology:

First, I decided on the algorithm for calculating the GCD. My algorithm is as follows: Let A and B be the two numbers we wish to find the gcd of. Then, subtract lesser number from greater one. After that, repeat the last step until both numbers are equal. The number they are both equal to is gcd(A, B). This algorithm holds because for each step gcd(a, b) = gcd(a-b, b) and gcd(a, b) = gcd(b, a) are true.

This algorithm can be applied to my design as a finite state machine with a modular structure created as described below:

main: This is the top module where inputs are taken and gcd is calculated. A and B are the inputs for the numbers, clockin is the internal clock, and button is the input for starting the calculation. First, signals are created for the register (named D_A, D_B, D_out for inputs of the registers, Q_A, Q_B, Q_out for the outputs of the registers), for the 1 KHz clock (named clk), and for the output of the subtractor (named diff). Output of the output register (Q_out) is directly connected to the output of the design (named gcd). After that, required components from submodules are initialized. Finally, there are two processes that describe the behavior of the design. The first process describes the registers. The registers are just a D-flip flop for each bit that is stored (3x8 bits). They are rising edge triggered. The second process

describes a combinational circuit that determines inputs of the registers depending on the current outputs of the registers. This process can be summarized with the following table:

Condition	Output
Button is pressed. (This has priority over others.)	Register A is loaded input A, Register B is loaded input B.
Q_A > Q_B	Register A is loaded A – B (diff signal). Register B keeps its value.
Q_A < Q_B	Numbers stored in A and B registers are swapped.
Q_A = Q_B	A and B keep their values. Value of A is copied to out register.

Table 1.1: Determining register inputs.

subtractor_8bit: This module is a modified version of my add-subtract module from ALU lab. add_subtract input is removed and it is set to do only subtraction.

clock_divider: This is the same clock divider module I wrote in the seven-segment display lab.

In conclusion, this design is an FSM, more specifically a Mealy machine with a single state because the output depends on only the input at that point in time. The machine neither changes its state nor the output depends on its current state (Şeker, 2011).

Results:

RTL schematic of my design is included below:

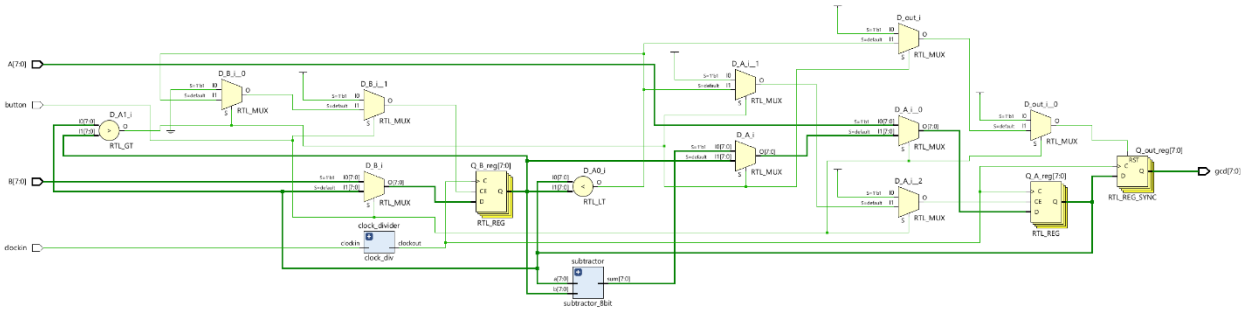

Figure 1.1: RTL Schematic.

As expected, a combinational circuit made up of muxes, comparators and components from other modules is created along with 3 registers: A, B, and out.

A testbench (gcd_test.vhd) is written to test and debug the design. Simulation graphs for three different inputs can be found below:

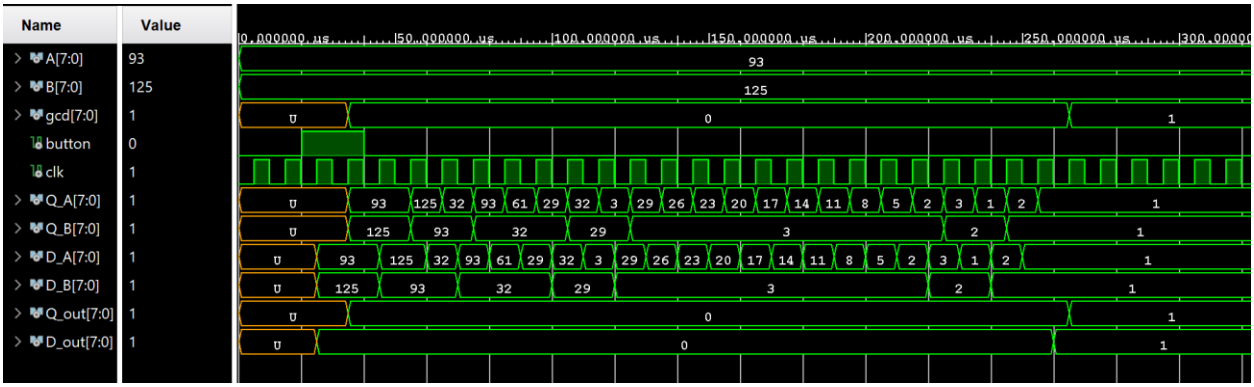

Figure 2.1: Simulation for gcd(93, 125) = 1.

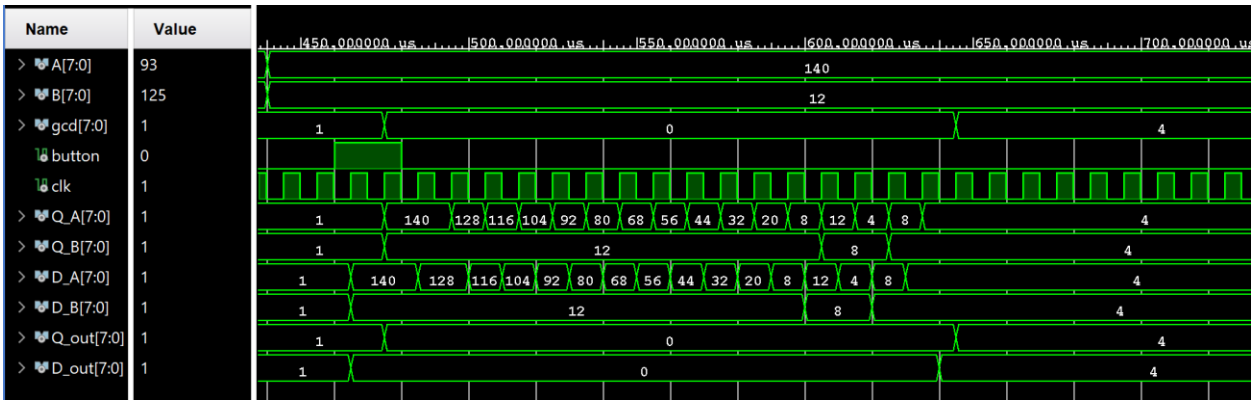

Figure 2.2: Simulation for gcd(140, 12) = 4.

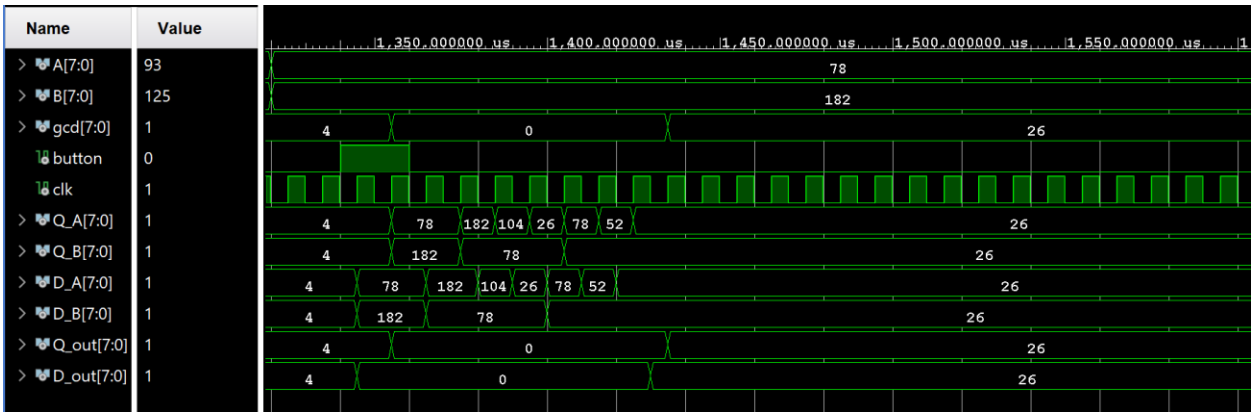

Figure 2.3: Simulation for gcd(78, 182) = 26.

Following table summarizes how many clock cycles is needed to calculate each test case:

Test Case	Clock Cycles Needed
gcd(93,125) = 1	22
gcd(140, 12) = 4	16
gcd(78, 182) = 26	7

Table 2.1: Clock cycles taken to calculate gcd for each simulation case.

My design does not require complex hardware, but it is not optimized for speed as it takes a lot of clock cycles to compute gcd. It can be improved by:

a) Doing the swapping of numbers at the same time with the last subtraction

b) instead of subtractor, a combinational modulus operator can be used to save a lot of clock cycles.

c) A fully combinational circuit can be designed to achieve the fastest result but in expense of hardware complexity / price.

I preferred to continue with this design for its simplicity. It is easy to design, understand and debug.

Finally, photos of the working Basys 3 are below:

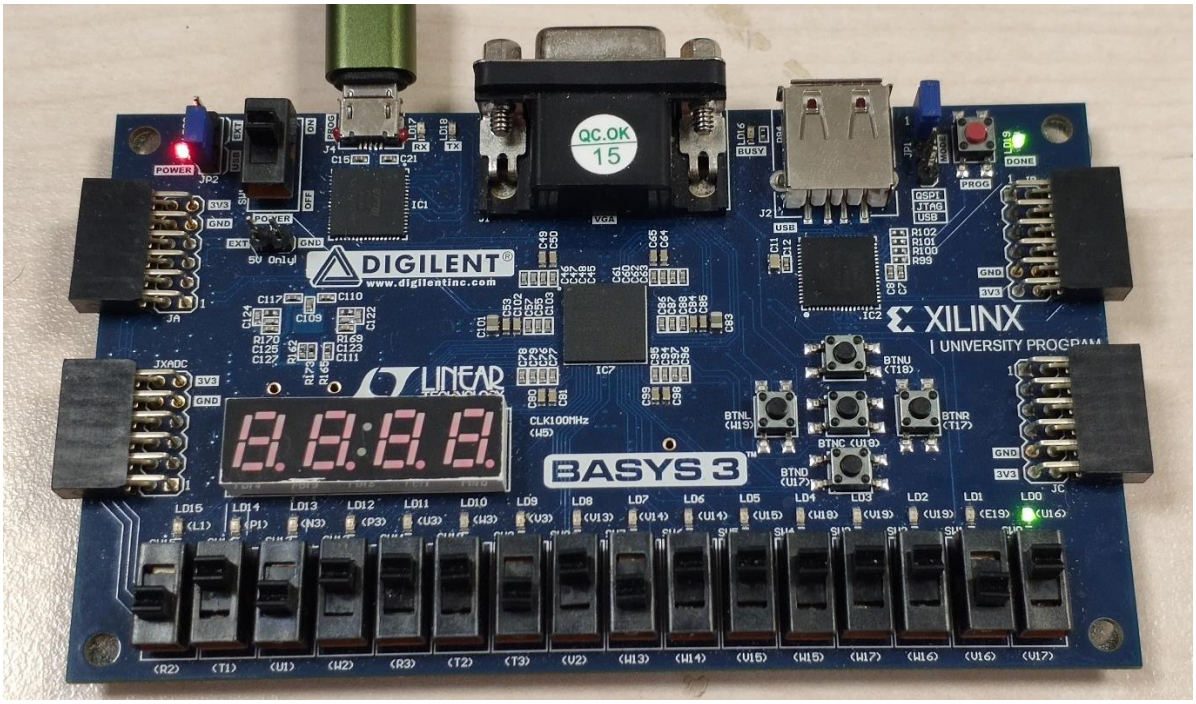

Figure 3.1: Test for gcd(93, 125) = 1.

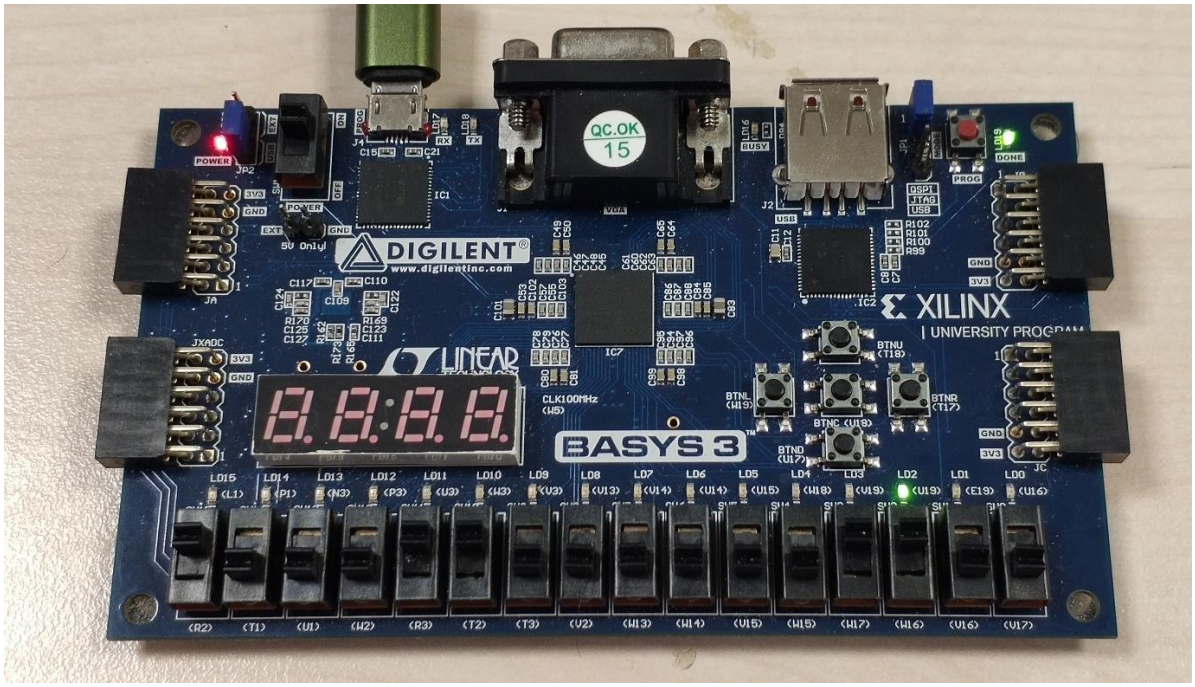

Figure 3.2: Test for gcd(140, 12) = 4.

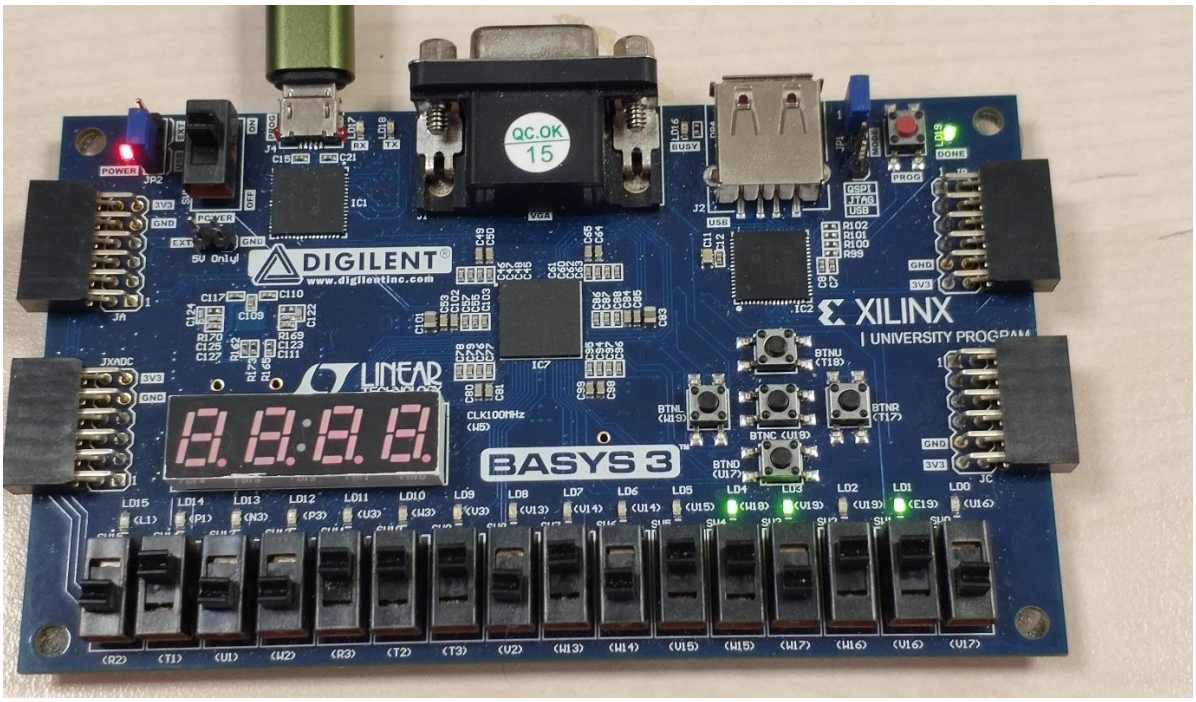

Figure 3.3: Test for gcd(78, 182) = 26.

In conclusion, my design works as expected, demonstrated by the simulation results and Basys 3 figures.

Conclusion:

In this lab, I have successfully implemented a design that calculates the gcd of two 8 bit numbers. I learned about registers and finite state machines.

References:

Şeker, Ş. E. (2011, January 26). *Mealy ve Moore makineleri (mealy and Moore machines)*. Bilgisayar Kavramları. https://bilgisayarkavramlari.com/2011/01/26/mealy-ve-moore-makineleri-mealy-and-moore-machines/

Appendices:

1) main.vhd

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

entity main is
   Port ( A : in STD_LOGIC_VECTOR (7 downto 0); -- Input A
          B : in STD_LOGIC_VECTOR (7 downto 0); --Input B
          gcd : out STD_LOGIC_VECTOR (7 downto 0); --Result
          clockin : in STD_LOGIC; -- Internal clock
          button : in STD_LOGIC); -- Button to load inputs to registers
end main;

architecture Behavioral of main is
component clock_div is
   Port(clockin : in STD_LOGIC;
        clockout : out STD_LOGIC);
end component;
component subtractor_8bit is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          sum : out STD_LOGIC_VECTOR (7 downto 0);
          borrow: out STD_LOGIC);
end component;

signal clk: std_logic; -- clock signal after division
```

```
signal Q_A, Q_B, Q_out, D_A, D_B, D_out: std_logic_vector(7 downto 0); -- Signals to be connected to
the registers
signal diff: std_logic_vector (7 downto 0); -- Q_A - Q_B
begin
gcd <= Q_out; -- Output Register connections
subtractor: subtractor_8bit Port map(a => Q_A, b => Q_B, sum => diff, borrow => open); -- 8 bit
subtractor
clock_divider: clock_div Port map(clockin => clockin, clockout => clk); -- 100 MHz to 1 KHz clock divider

Process(clk)
begin
   if rising_edge(clk) then -- Creating Registers
      Q_A <= D_A;
      Q_B <= D_B;
      Q_out <= D_out;
   end if;
end process;

process(clk)
begin
   D_out <= (others => '0'); -- Out register is defaulted to zeros
   if button = '1' then -- Load inputs to input registers
      D_A <= A;
      D_B <= B;
   else              -- gcd algorithm
      if Q_A > Q_B then
         D_A <= diff;
         D_B <= Q_B;
      elsif Q_A < Q_B then
         D_B <= Q_A;
         D_A <= Q_B;
      else
         D_A <= Q_A;
         D_B <= Q_B;
         D_out <= Q_A;
      end if;
   end if;
end process;
end Behavioral;
```

2) gcd_test.vhd

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


entity gcd_test is
end gcd_test;

architecture Behavioral of gcd_test is
component main is
   Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
          B : in STD_LOGIC_VECTOR (7 downto 0);
          gcd : out STD_LOGIC_VECTOR (7 downto 0);
          clockin : in STD_LOGIC;
          button : in STD_LOGIC);
end component;

signal A : std_logic_vector (7 downto 0);
signal B : std_logic_vector (7 downto 0);
signal gcd: STD_LOGIC_VECTOR (7 downto 0);
signal button : std_logic := '0';
signal clockin : std_logic := '0';
begin
uut: main port map(clockin => clockin, A => A, B => B, gcd => gcd, button => button);
process
constant clock_period : time := 10ns;
begin
clockin <= '0';
wait for clock_period/2;
clockin <= '1';
wait for clock_period/2;
end process;

process
begin
A <= "01011101";
B <= "01111101";
wait for 20020ns;
button <= '1';
wait for 20000ns;
button <= '0';
wait for 400us;
A <= "10001100";
```

```
B <= "00001100";
wait for 20020ns;
button <= '1';
wait for 20000ns;
button <= '0';
wait for 400us;
A <= "01001110";
B <= "10110110";
wait for 20020ns;
button <= '1';
wait for 20000ns;
button <= '0';
wait for 400us;
wait;
end process;
end Behavioral;
```