Lab 6: Greatest Common Divisor

1) Purpose

Aim of this lab is to understand how to implement registers on the Basys 3 FPGA board and design a digital circuit that calculates greatest common divisor of 2, 8 bit numbers as it requires an iterative process it is reasonable to use registers instead of combinational circuit design. Different algorithms can be used to calculate GCD of two numbers in the project Euclidean algorithm is used.

2) Questions

a) What was your algorithm to calculate the GCD?

Euclidean algorithm that says 1) take two numbers, 2) if they are equal the GCD is the value of them, 3) if they are not equal then find absolute difference of them and take the difference and the smaller number and go to step 1.

b) 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 circuit is a FSM as it is not a combinational circuit, it is a sequential circuit including finite number of states and it is a Moore machine as its output is determined by only its current state. Its inputs can be changed using button signal to set and reset. It might be harder to

implement circuits doing iterative operations using combinational circuits but they are faster as they don't relay on a clock signal to operate otherwise only once in a clock cycle the circuit operates.

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

To calculate the GCD of 140 and 12 it take 14 clock cycles. It might be optimized using more effective algorithms than Euclidean algorithm.

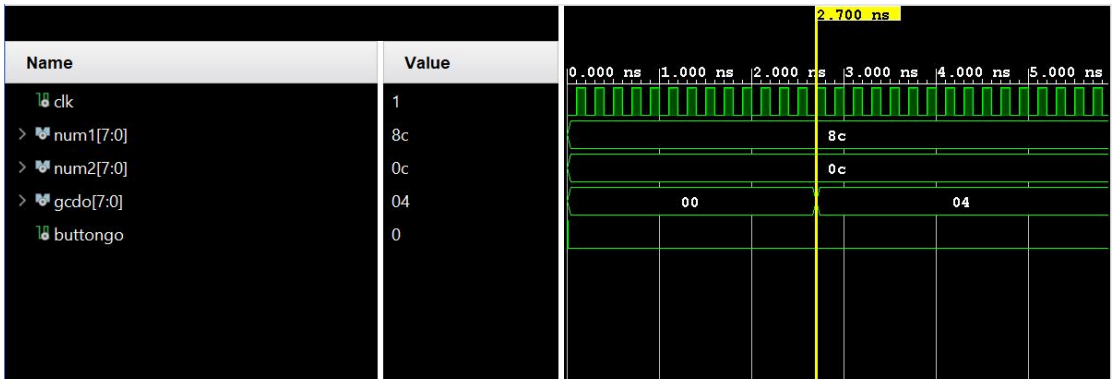

Figure 2.1 Simulation result for 140 and 12, it take 13-14 clock cycles to calculate GCD of them as 4.

3) Methodology

The designed circuit takes 2, 8 bit numbers from the 16 switches on the board and when the button is pushed it is designed to initiate set functioning meaning the algorithm starts to work and the second push of button makes it get to reset mode which connects 0's for the number inputs to registers and eventually the algorithm stops working. The set and reset functioning achieved by using an additional register for button, when it is clicked it sets the ser signal to 1, reset to 0 and a flag signal to 1, when it is pushed after that as the flag signal is 1 it sets the set signal to 0, reset signal

to 1 and flag to 0. as the reset and set signals are not declared in the sensitivity list of the registers they are synchronous signals. While the algorithm working combinational circuits that make comparison and subtraction operations are needed and for that ieee.std_logic_unsigned library is used as these operators is defined as built in functions. Also multiplexers to set D values for the d flip flops and for the showing the results are designed. Schematic of the design can be seen in the figure below. And the VHDL design, constraint and test bench codes written in the appendix .

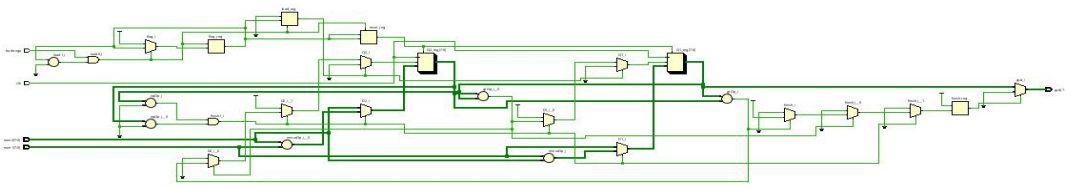

Figure 3.1 schematic of GCD circuit. 1 clock, 2 8bit numbers and a button signal inputs and 1, 8 bit GCD output

4) Results

The results of the BASYS 3 implementation can be seen figures below. Dfferent 8 bit numbers given and set reset functions of the design also exhibited.

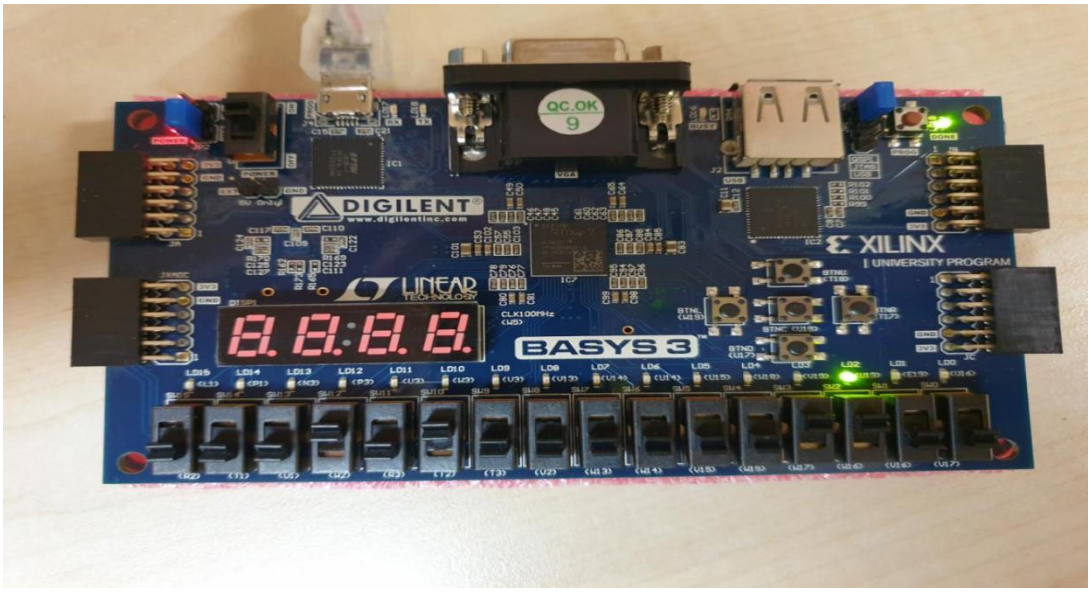

Figure 4.1: num1 = 12, num2 = 20, GCD = 4 after button pushed to set.

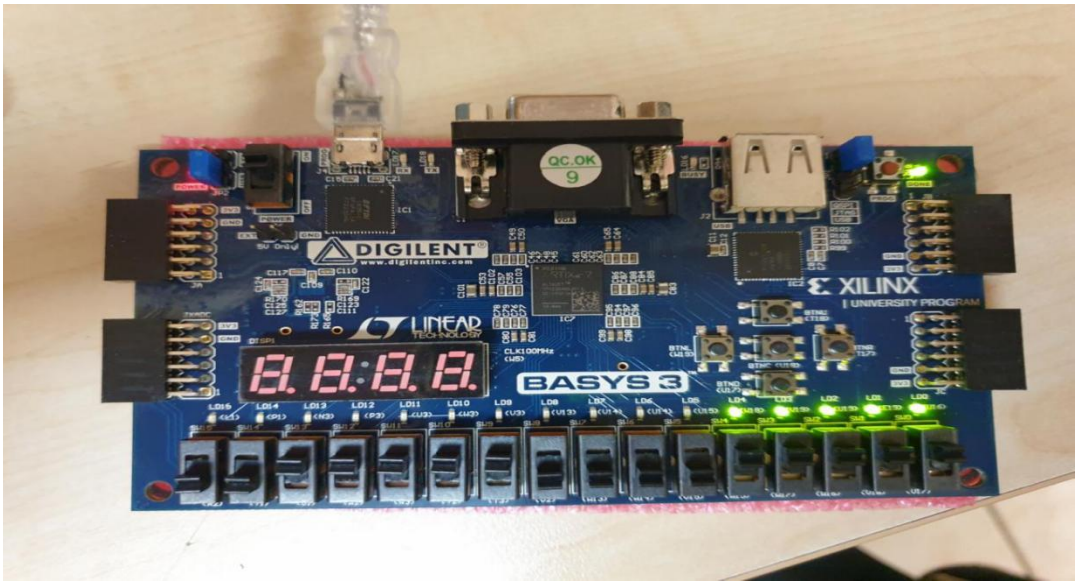

Figure 4.2: num1 = 31, num2 = 62, GCD = 31 after button pushed to set.

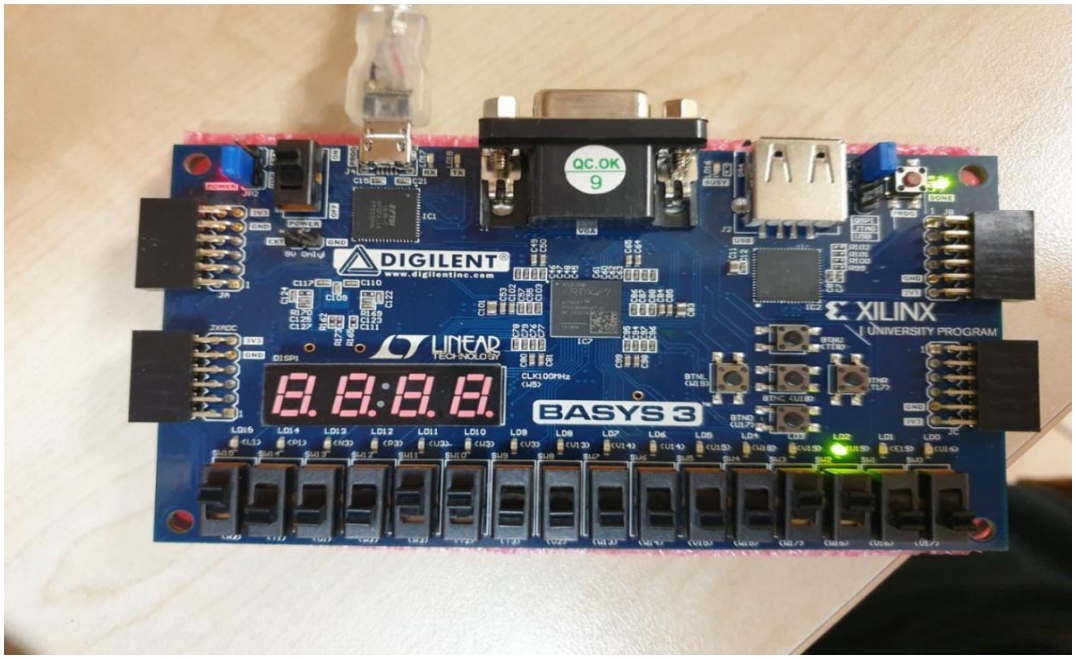

Figure 4.3 num1 = 12, num2 = 140, GCD = 4 after button pushed once.

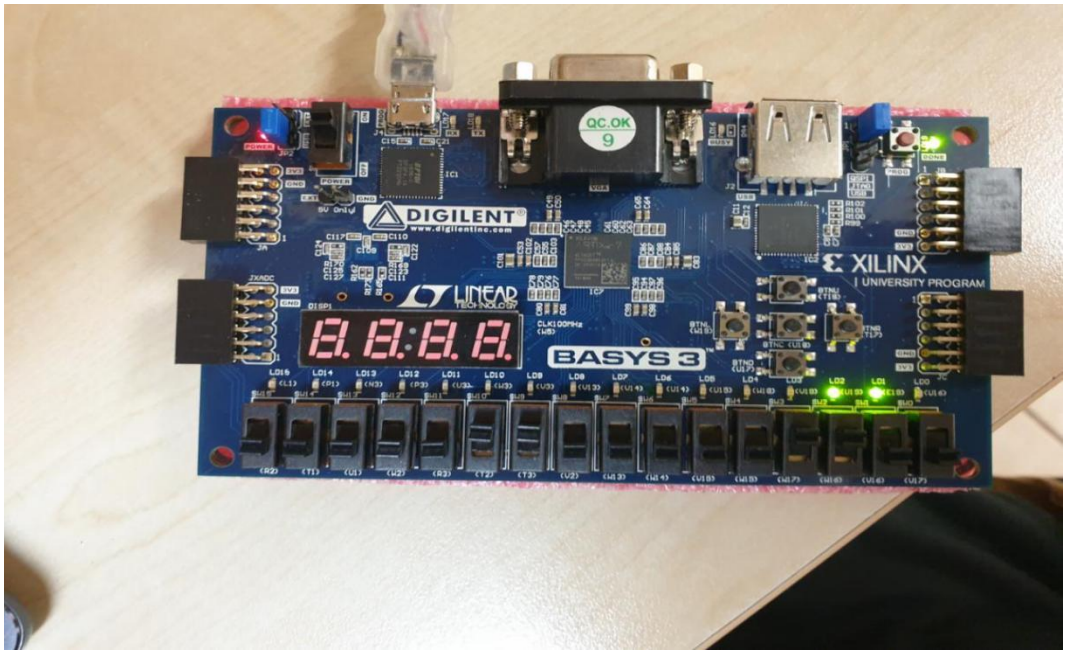

Figure 4.4 num1 = 12, num2 = 6, GCD = 6 after button pushed once.

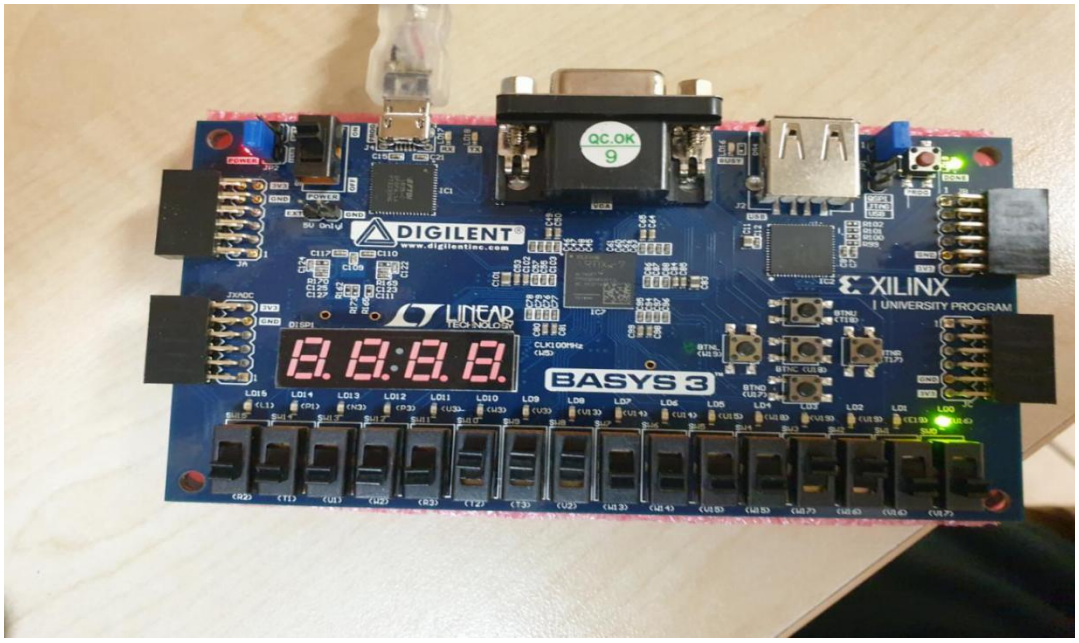

Figure 4.5 num1 = 12, num2 = 7, GCD = 1 after button pushed once.

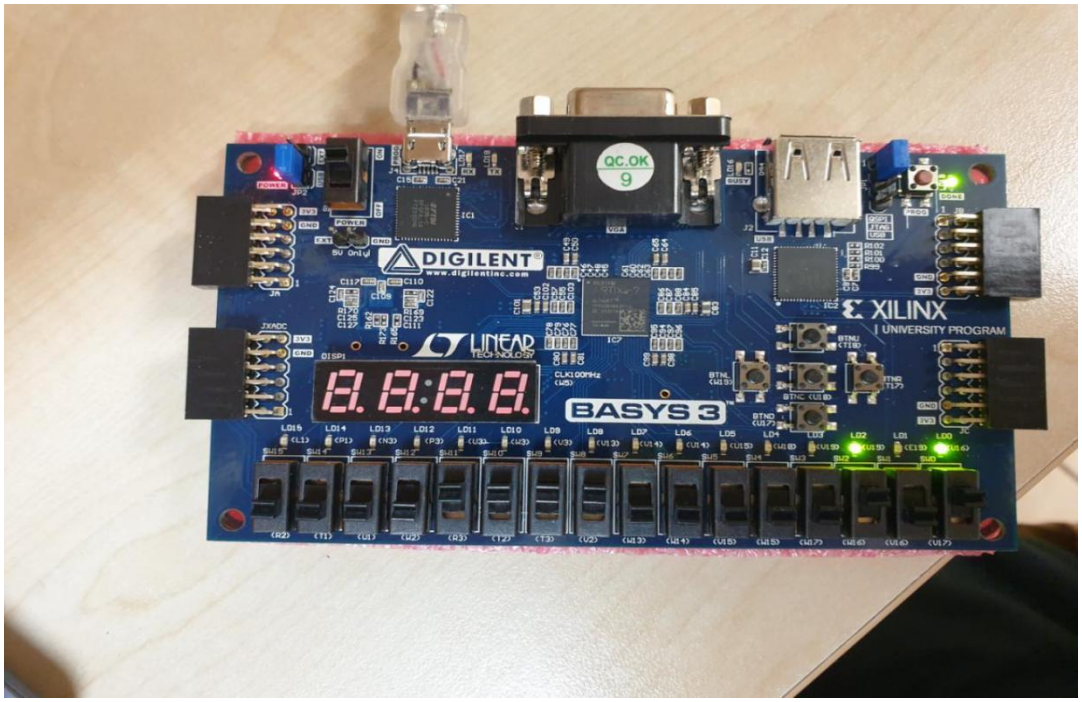

Figure 4.6 num1 = 5, num2 = 15, GCD = 5 after button pushed once.

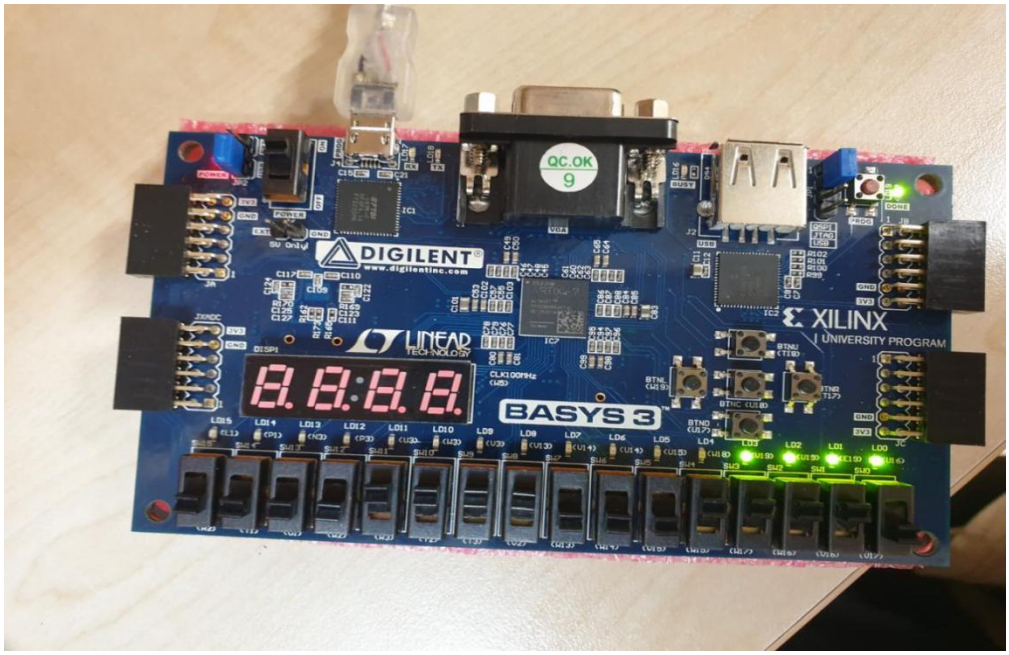

Figure 4.7 num1 = 30, num2 = 15, GCD = 15 after button pushed once.

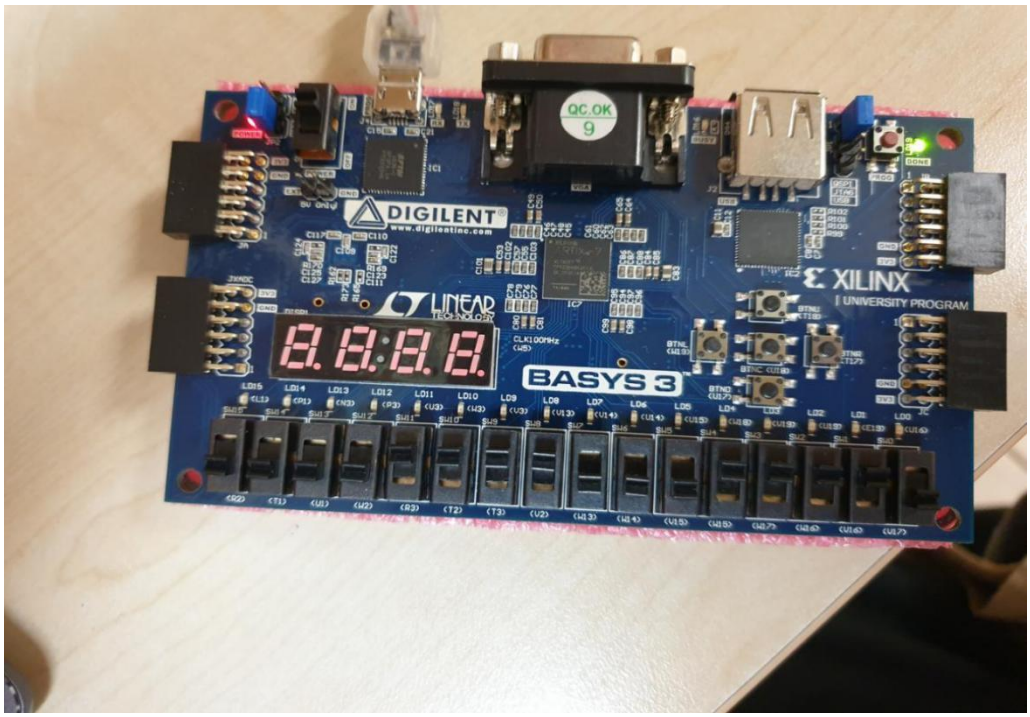

Figure 4.8 num1 = 30, num2 = 15, GCD = 0 as the reset button is pushed GDC is 0.

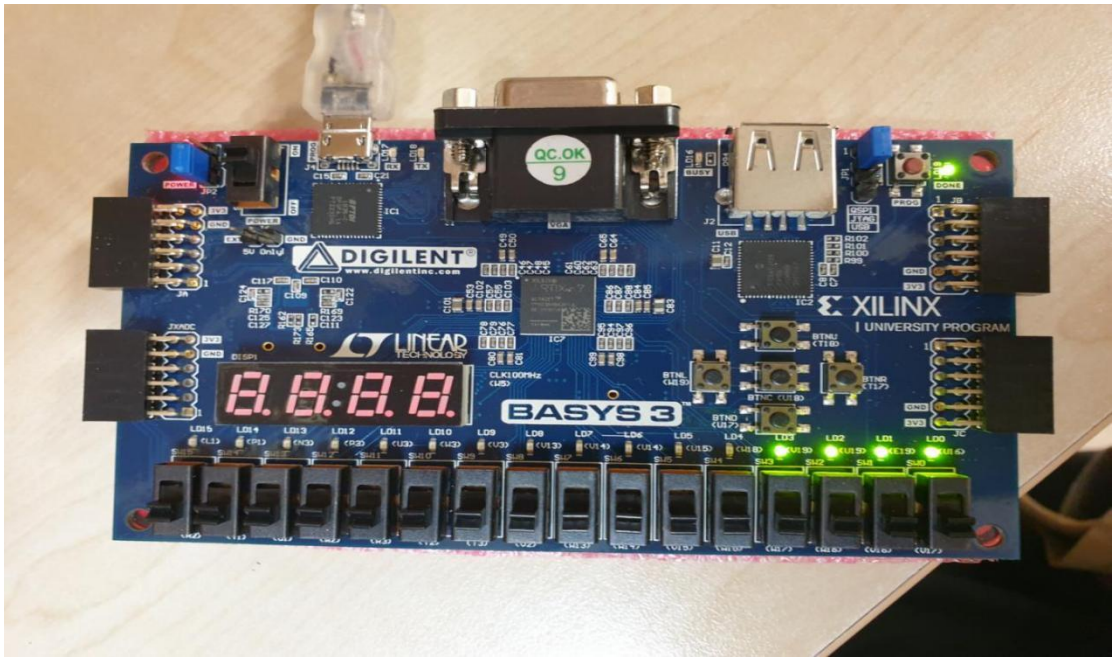

Figure 4.9 num1 = 0, num2 = 0, GCD = 15 as the reset button is not pushed it shows the GCD of previous inputs which are 15 and 30.

5) Conclusion

In conclusion we learned how to implement registers on FPGA board using VHDL and we also learned how sequential circuit designs can be implemented and used for practical problems such as finding GCD of two numbers.

6) Appendix

- Design code

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


entity gcd is
```

```
    Port ( clk : in std_logic;
           buttongo: in std_logic;
           num1 : in STD_LOGIC_VECTOR (7 downto 0);
           num2 : in STD_LOGIC_VECTOR (7 downto 0);
           gcd : out STD_LOGIC_VECTOR (7 downto 0));
end gcd;

architecture Behavioral of gcd is

signal Q1: std_logic_vector (7 downto 0):="00000000";
signal Q2: std_logic_vector (7 downto 0):="00000000";
signal D1: std_logic_vector (7 downto 0):="00000000";
signal D2: std_logic_vector (7 downto 0):="00000000";
signal finish : std_logic:='0';
signal reset: std_logic:='0';
signal load: std_logic:='0';
signal flag: std_logic:='0';
signal control: std_logic_vector (1 downto 0):="00";
begin

D1 <= num1 when Q1 = "00000000" and Q2="00000000" else
   Q1-Q2 when Q1>Q2 else
   Q1;
D2 <= num2 when Q1 = "00000000" and Q2="00000000" else
   Q2-Q1 when Q2>Q1 else
```

```
   Q2;
finish <= '1' when Q1=Q2 and q1 /= "00000000" else '0';


process(buttongo)
begin
if (buttongo ='1' and flag='0') then
load<='1';
reset<='0';
flag <= '1';
elsif (buttongo ='1' and flag='1') then
load<='0';
reset<='1';
flag <= '0';

end if;
end process;
process(clk)
begin
 if (rising_edge(clk)) then
 if (reset = '1') then
 Q1 <= "00000000";
 elsif (load = '1') then
 Q1 <= D1;
 end if;
 end if;
```

```
end process;
process(clk)
begin
 if (rising_edge(clk)) then
 if (reset = '1') then
 Q2 <= "00000000";
 elsif (load = '1') then
 Q2 <= D2;
 end if;
 end if;
end process;
gcd<=Q1 when finish = '1'else "00000000";

end Behavioral;
```

- Test bench code

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



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

architecture Behavioral of sim is
component gcd is
```

```
    Port ( clk : in std_logic;
           buttongo: in std_logic;
           num1 : in STD_LOGIC_VECTOR (7 downto 0);
           num2 : in STD_LOGIC_VECTOR (7 downto 0);
           gcd : out STD_LOGIc_VECTOR (7 downto 0));
end component;
signal clk: std_logic;
signal num1: std_logic_vector (7 downto 0):="00000000";
signal num2: std_logic_vector (7 downto 0):="00000000";
signal gcdo: std_logic_vector (7 downto 0):="00000000";
signal buttongo: std_logic:='0';
begin
uut : gcd port
map(clk=>clk,num1=>num1,num2=>num2,gcd=>gcdo,buttongo=>buttongo);
clock: process
begin
clk<='0';
wait for 0.1 ns ;
clk <='1';
wait for 0.1 ns ;
end process;
test : process
begin
num1 <= "10001100";
num2 <= "00001100";
```

```
buttongo <='1';
wait for 0.01ns;
buttongo <='0';
wait for 50 ns;
buttongo <='1';
wait for 0.01ns;
buttongo <='0';
wait for 1ns;
buttongo <='1';
wait for 0.01ns;
buttongo <='0';

num1 <= "00001001";
num2 <= "00000011";
wait for 100 ns;
buttongo <='1';
wait for 0.01ns;
buttongo <='0';
wait for 1 ns;

end process;
end Behavioral;
```

- Constraint code

```
set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports
{num1[0]}]
```

```
set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports
{num1[1]}]
set_property -dict { PACKAGE_PIN W16   IOSTANDARD LVCMOS33 }
[get_ports {num1[2]}]
set_property -dict { PACKAGE_PIN W17   IOSTANDARD LVCMOS33 }
[get_ports {num1[3]}]
set_property -dict { PACKAGE_PIN W15   IOSTANDARD LVCMOS33 }
[get_ports {num1[4]}]
set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports
{num1[5]}]
set_property -dict { PACKAGE_PIN W14   IOSTANDARD LVCMOS33 }
[get_ports {num1[6]}]
set_property -dict { PACKAGE_PIN W13   IOSTANDARD LVCMOS33 }
[get_ports {num1[7]}]
set_property -dict { PACKAGE_PIN V2    IOSTANDARD LVCMOS33 } [get_ports
{num2[0]}]
set_property -dict { PACKAGE_PIN T3    IOSTANDARD LVCMOS33 } [get_ports
{num2[1]}]
set_property -dict { PACKAGE_PIN T2    IOSTANDARD LVCMOS33 } [get_ports
{num2[2]}]
set_property -dict { PACKAGE_PIN R3    IOSTANDARD LVCMOS33 } [get_ports
{num2[3]}]
set_property -dict { PACKAGE_PIN W2    IOSTANDARD LVCMOS33 } [get_ports
{num2[4]}]
```

```
set_property -dict { PACKAGE_PIN U1    IOSTANDARD LVCMOS33 } [get_ports
{num2[5]}]
set_property -dict { PACKAGE_PIN T1    IOSTANDARD LVCMOS33 } [get_ports
{num2[6]}]
set_property -dict { PACKAGE_PIN R2    IOSTANDARD LVCMOS33 } [get_ports
{num2[7]}]



set_property -dict { PACKAGE_PIN W5   IOSTANDARD LVCMOS33 } [get_ports
clk]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports
{gcd[0]}]
set_property -dict { PACKAGE_PIN E19   IOSTANDARD LVCMOS33 } [get_ports
{gcd[1]}]
set_property -dict { PACKAGE_PIN U19   IOSTANDARD LVCMOS33 } [get_ports
{gcd[2]}]
set_property -dict { PACKAGE_PIN V19   IOSTANDARD LVCMOS33 } [get_ports
{gcd[3]}]
set_property -dict { PACKAGE_PIN W18   IOSTANDARD LVCMOS33 }
[get_ports {gcd[4]}]
set_property -dict { PACKAGE_PIN U15   IOSTANDARD LVCMOS33 } [get_ports
{gcd[5]}]
```

```
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports {gcd[6]}]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports {gcd[7]}]

set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports buttongo]
```