Mustafa Cankan BALCI

22101761

Section 1

28 November 2022

Lab 6: Greatest Common Divisor

Purpose

The objective of lab is to design a circuit which calculates the greatest common divisor (GCD) of two 8- bit numbers. The lab also aimed to creating circuits via using Finite State Machine.

Design Specifications

The algorithm to calculate the greatest common divisor of two 8-bit numbers is Euclidian algorithm. The algorithm is based on subtracting highest value number form the lowest value number. If the result and subtrahend become equal, the final result is the values. Otherwise, there is a two condition. If the result is bigger than subtrahend, the result changes with becomes minuend and the subtrahend does not change. However, the result changes with becomes subtrahend and the minuend becomes the result if the result is lower than subtrahend. This cycle is done until the subtrahend and minuend becomes the same since the subtraction between them gives 0. The minuend becomes the greatest common divisor of two numbers.

36 - 16 = 20

20 - 16 = 4

16 - 4 = 12

12 - 4 = 8

8 - 4 = 0

4 - 4 = 0

Equation 1 The example GCD operation between 36 and 16

The Euclidian algorithm is used in the lab for designing a Finite State Machine. The type of finite state machine is Moore machine since the outputs of states determine the states. There are 3 states in design: IDLE, CHANGE, and SUBTRACT. To begin with, The first state is called idle. Moreover, the two 8-bit numbers is compared to obtain the bigger number , and it is loaded into registers. If the numbers are equal, the next state is idle. This state is called “change” in the FSM module. Afterward, the bigger number subtracts from the lower number by using the subtractor module, and the state moves back to “change” .This state is called subtract state.

Registers are designed by using the positive edge D flip flop. If the reset is zero, the two 8-bit numbers are loaded in the 8-bit register. Otherwise, the state stays at idle, and the register loads with 0000000 if the reset is one. The clock of the registers is 100MHz, the internal clock frequency of BASYS 3.

The lab module contains 19 inputs and 9 outputs. The inputs are clock signal, reset, enable and two 8-bit binary . The reset is assigned to W19 button, and the enable button is assigned to T17. The clock signal is defined to W5 pin. The first input A has 8 inputs ,and their switches are V17, V16,W16,W17,W15,V15, W14,W13 respectively. The second input B has 8 inputs, which are R2, T1, U1, W2, R3, T2, T3 and V2. The outputs consists of 8 outputs led for showing the result of gcd, which are U16, E19, U19, V19, W18, U15, U14 and V1 LEDs on BASYS 3. The “go” output pin is L1 led on BASYS 3.

Methodology

First of all, the 8-bit subtractor and 8-bit comparator module are created. The full adder and half adder codes is same as Lab 4(Arithmetic Logic Unit). The full adders are used the create 8-bit subtractor. The first full adder carry in is 1. The second input is connected to not gate before going inside the full adder. In the design of comparator is different from the Lab 4. It has 3 outputs that are a equal, less or greater. The eight_bit_equal and eight_bit_gt files are used as a low module for making 8-bit comparator. The eight_bit_equal is used XNOR and AND gates. The eight_bit_gt

module is designed by using 4-bit equal and 4-bit greater than module. Additionally, the 4 bit greater than module is coded by 2-bit equal and 2-bit greater than. 2-bit equal and 2-bit greater than is coded in Lab 4(Arithmetic Logic Unit).

Furthermore, the GCD module file is designed. The 8-bit subtractor and 8-bit comparator used as a low level module inside the file. The FSM states in design sources is declared in this module. Unsigned data type is used to get more accurate results.

Results

The RTL schematic of greatest common divisor module of two 8-bit numbers is same as expected. There are registers, multiplexers, and module files in schematics. The more detailed photos of 8-bit comparator and 8-bit subtractor are in appendix.

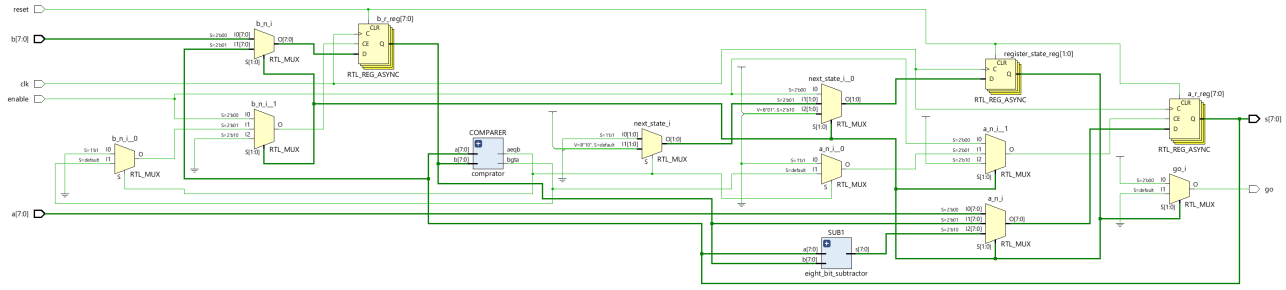

Figure 1 RTL Schematics of Greatest Common Divisor

The testbench graph gave the expected results in the graph. The inputs of GCD module is 140 and 12. The clock is triggered at each 10 nanosecond. The final result is 4, which is desired result. The total number of clock cycle is 27.

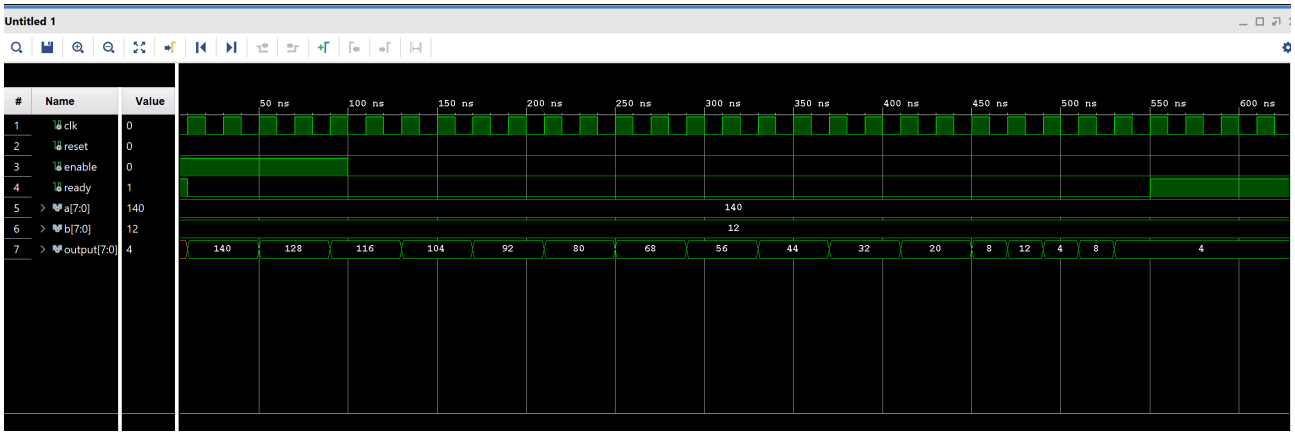

Figure 2 Testbench Graph when a = 140 and b = 12 in hexadecimal base

The application of the lab module on BASYS 3 does not give different result. For instance, the result was 4 when the input a was 12 and input b was 140 when the enable button was pressed . Other example photos of different input for a experiment setup BASYS 3 implementation is in appendix.

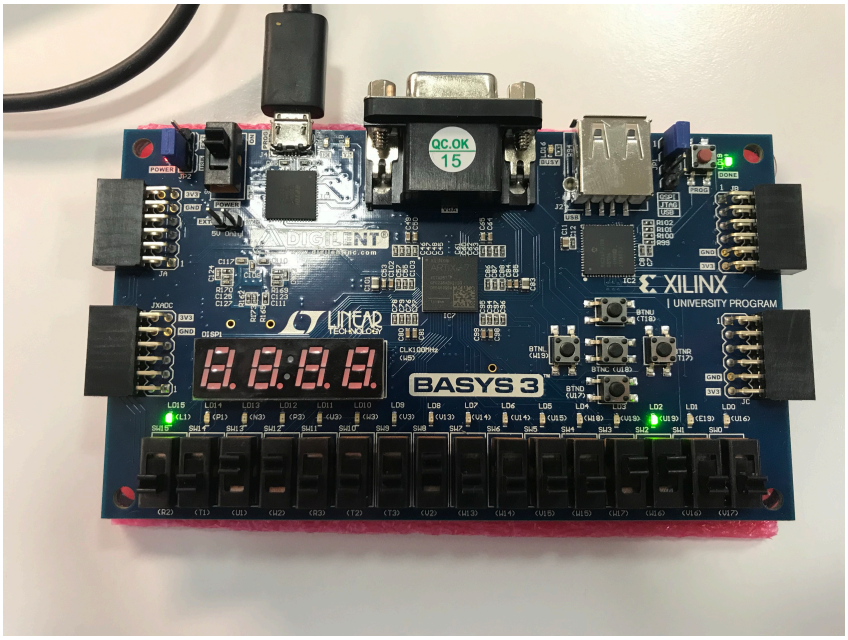

Figure 3 L1 and U19 is on when U1,T1 and W16 and W17 is pulled when enable is pressed

Conclusion

The experiment will aim to design a module that finds the greatest common divisor of two 8-bit binary numbers. Also, this lab focuses on creating FSM designs by using VHDL. The algorithm to calculate the gcd of two 8-bit numbers is applied by using FSM. The type of FSM is a Moore Machine because the output determines the state. The combinational circuit will be faster because FSM has a clock due to registers. The code is hard to understand via a combinational circuit and only applies to 8-bit numbers. However, FSM code can be modified by any bit number easily. The combinational circuit might be cheaper since FSM model contains data hold which is requires more logic gates. The clock cycle is 27, which could be a better result. If the process connected the clock is decreased, the clock cycle will be decrease.

Appendix

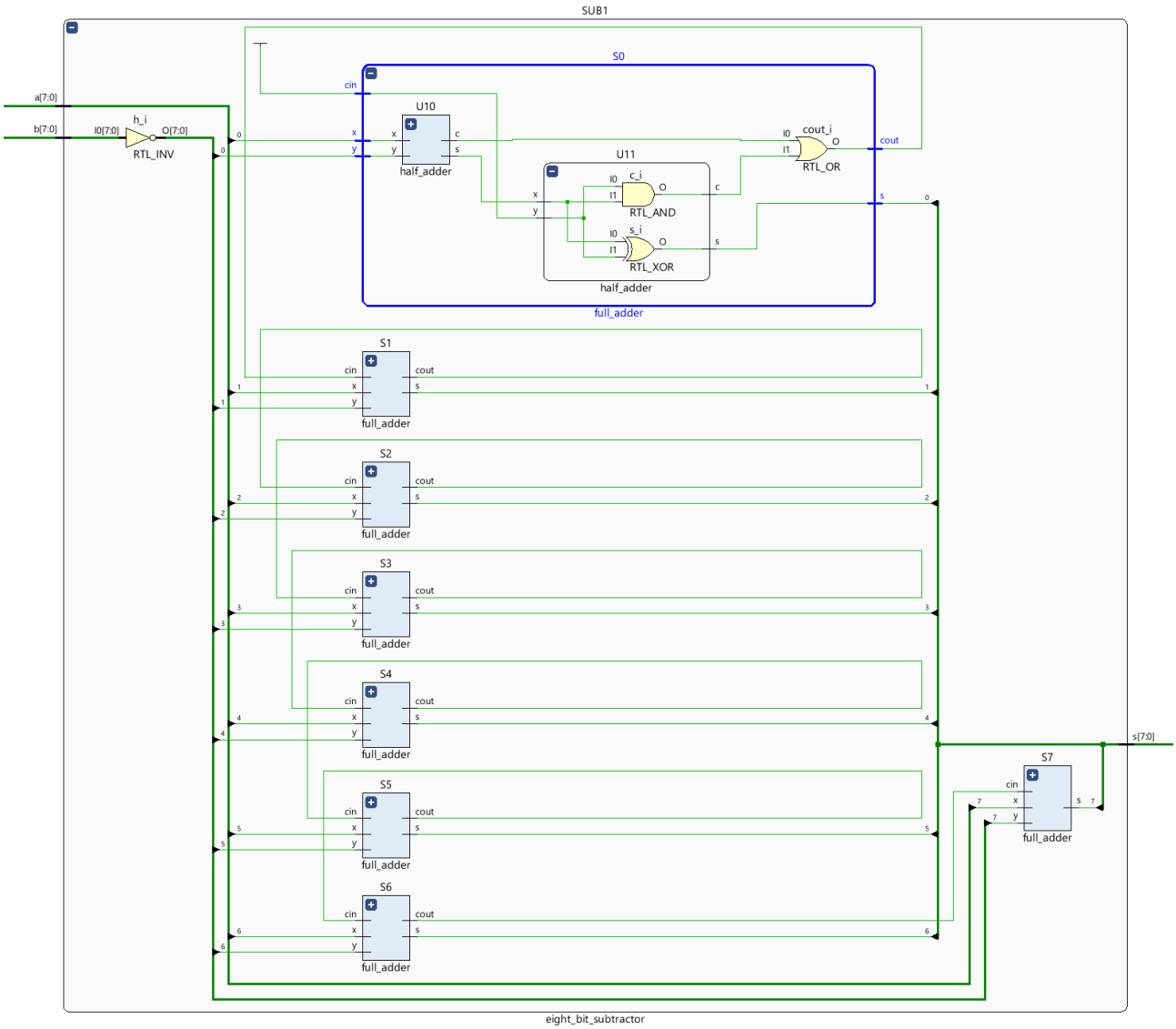

Figure 4 RTL Schematics of 8-bit Subtractor

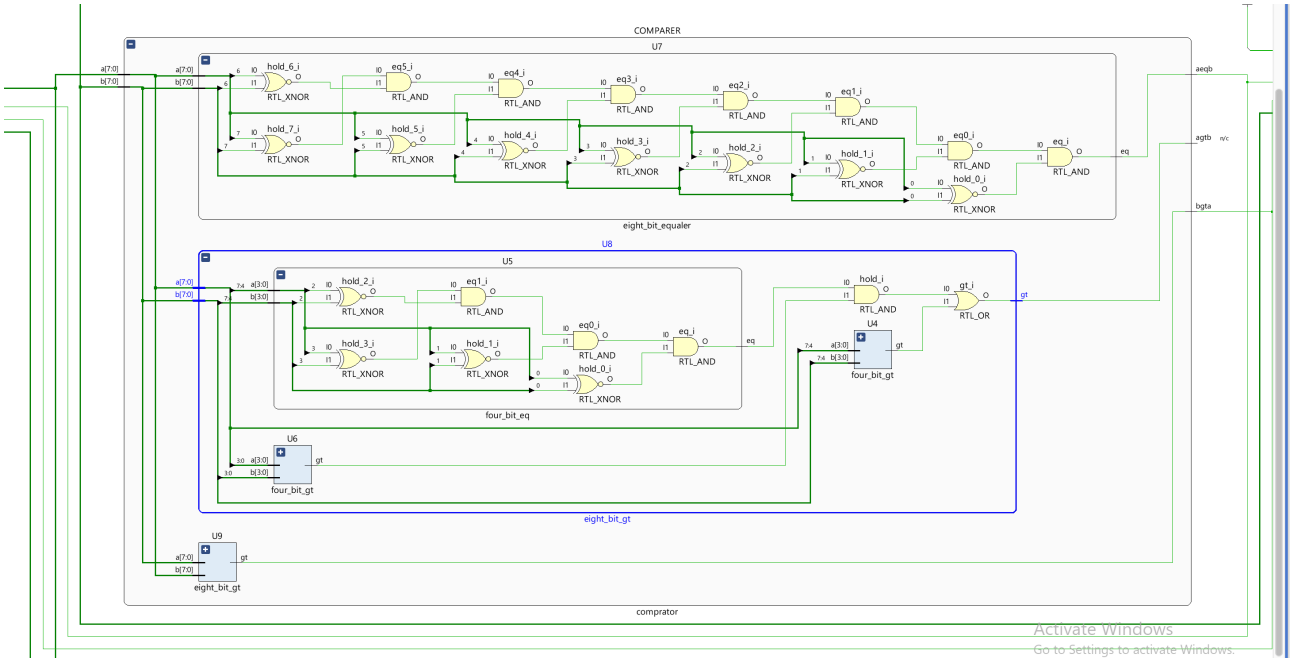

Figure 5 RTL Schematics of 8-bit Comparator

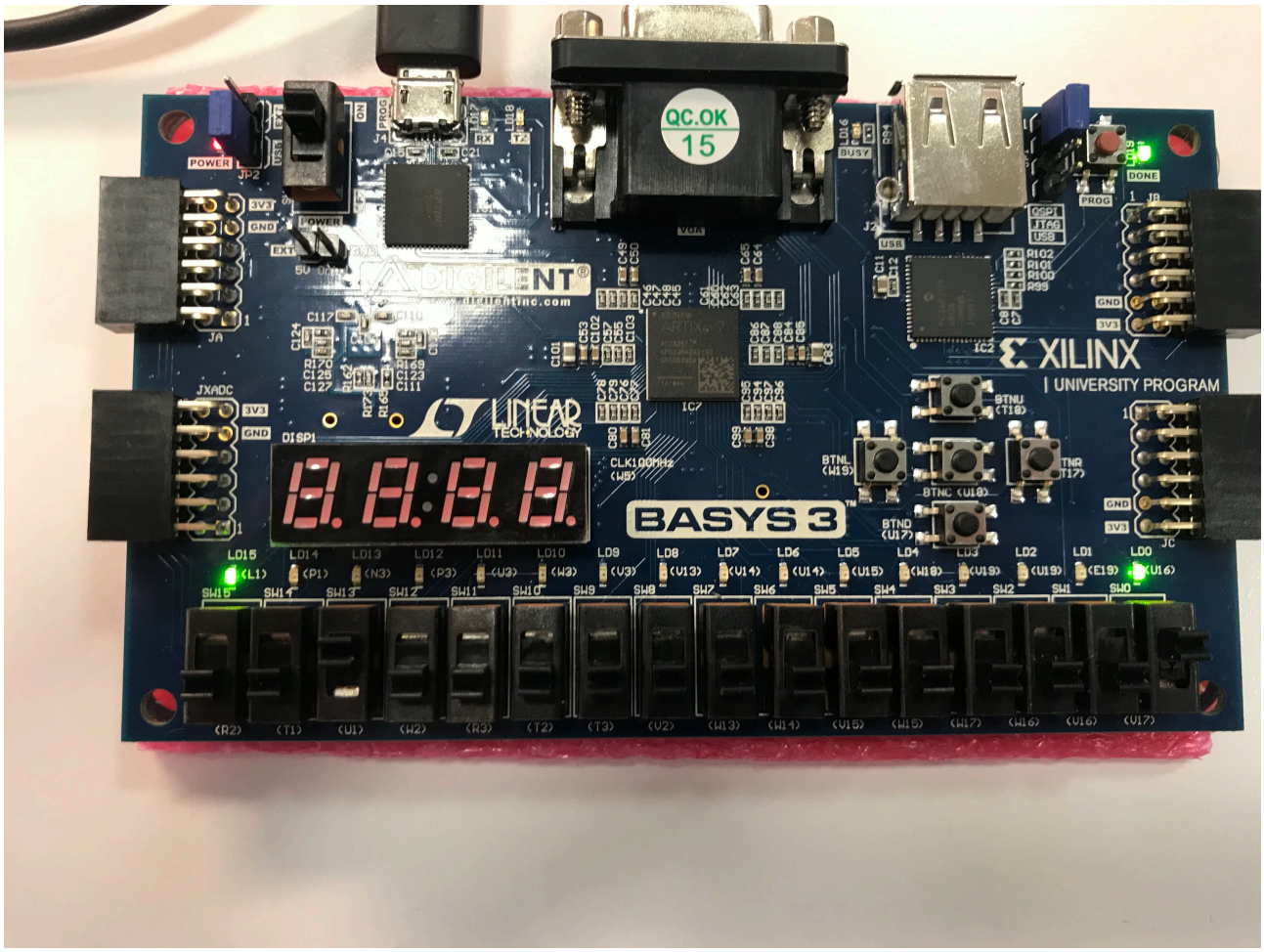

Figure 6 L1 and U16 is on when U1 and V17 is pulled when enable is pressed

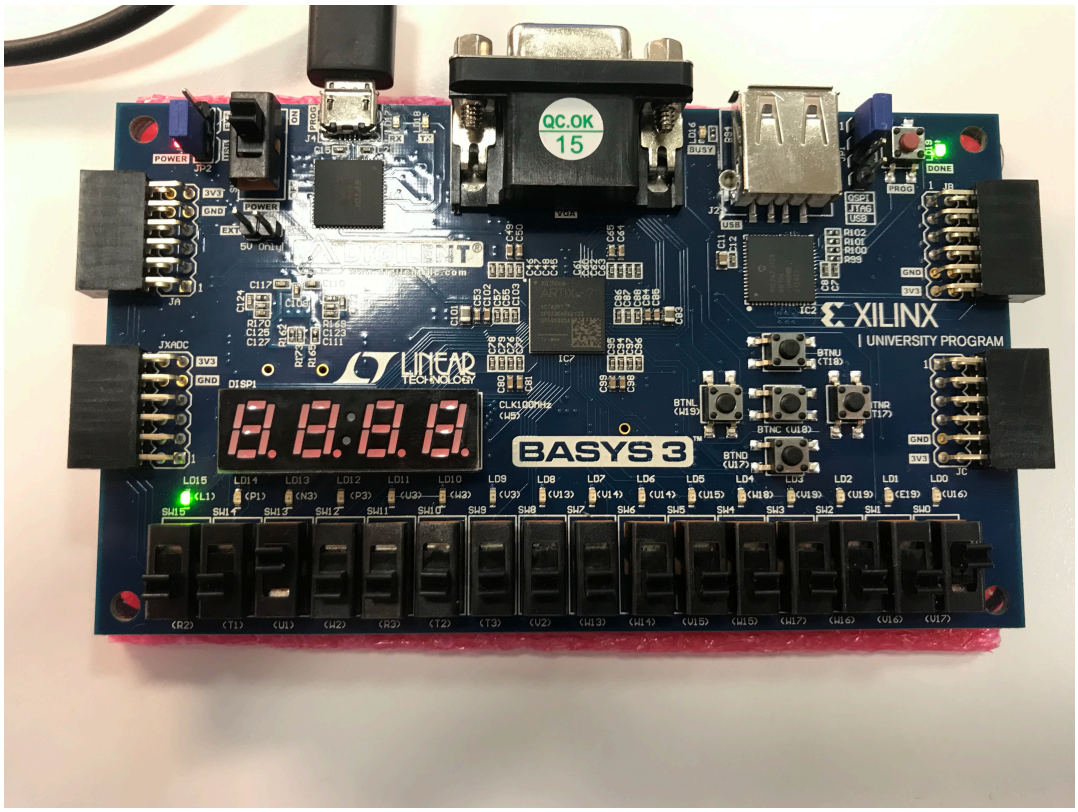

Figure 7 L1 is on when U1 and V17 is pulled when reset is pressed

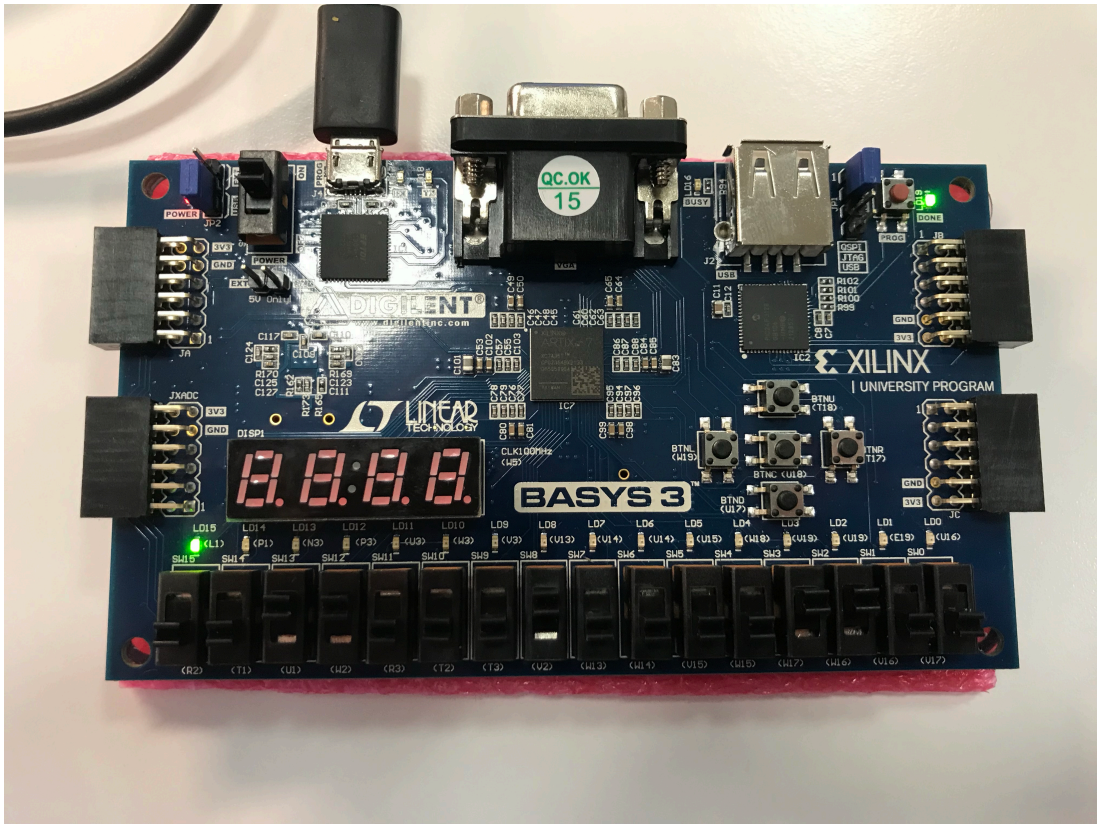

Figure 8 L1 is on when U1, W2,V2,W16 and V17 is pulled when reset is pressed

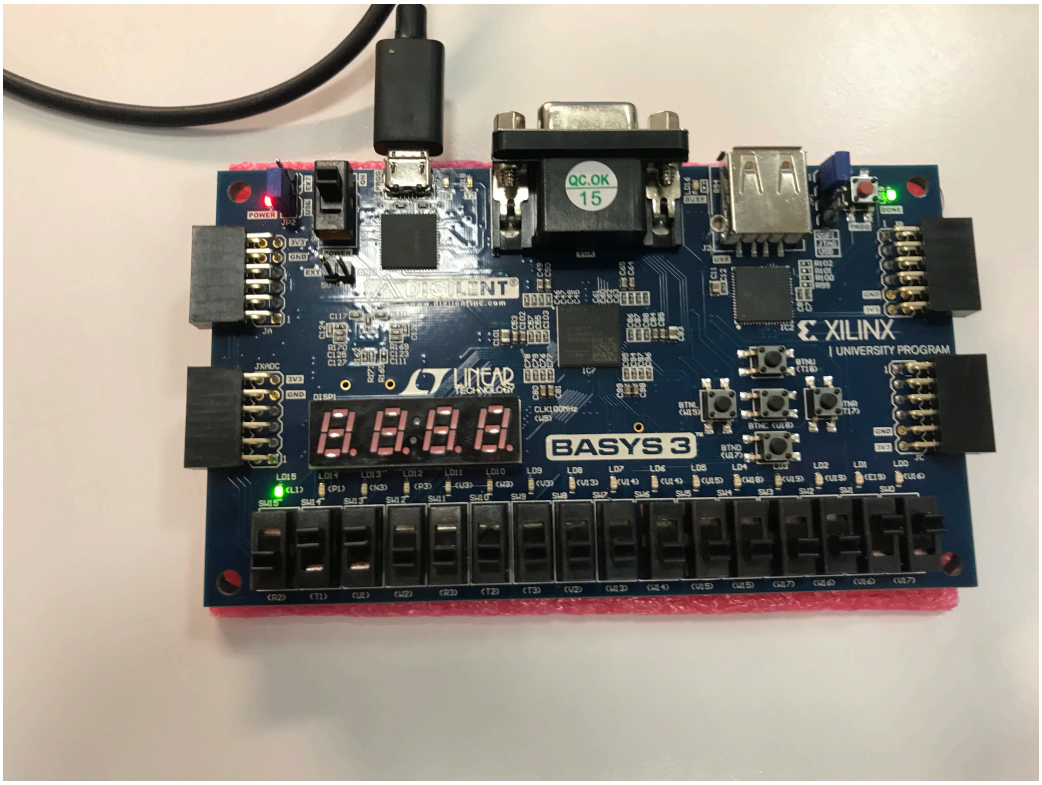

Figure 9 L1 is on when T1,U1, V16 and V17 is pulled when reset is pressed

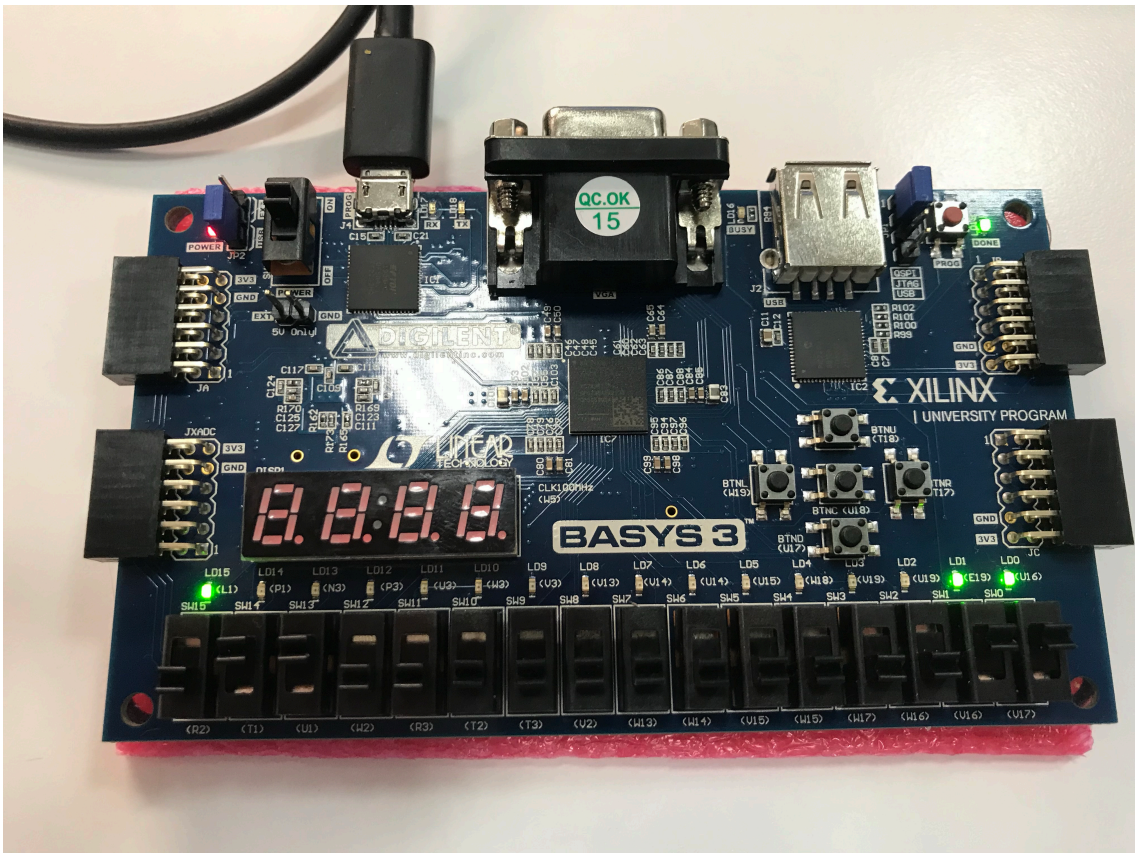

Figure 10 L1,E19 and U16 is on when T1,U1, V16 and V17 is pulled when enable is pressed

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity half_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           s : out STD_LOGIC;
           c : out STD_LOGIC);
end half_adder;
architecture Behavioral of half_adder is
begin
s <= x XOR y;
c <= y and x;
end Behavioral;
```

Code 1: Design Code of Half Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity full_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           cin : in STD_LOGIC;
           s : out STD_LOGIC;
           cout : out STD_LOGIC);
end full_adder;

architecture Behavioral of full_adder is
```

```
Signal s0, c0, c1 : std_logic;


Component half_adder is
    PORT(
    x, y: in std_logic;
    s, c: out std_logic
    );
 end component;
begin
U10: half_adder PORT MAP (x,y,s0,c0);
U11: half_adder PORT MAP (s0,cin, s, c1);
cout <= c0 or c1;
end Behavioral;
```

Code 2: Design Code of Full Adder

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
entity eight_bit_subtractor is
    Port ( a : in std_logic_vector (7 downto 0);
           b : in std_logic_vector (7 downto 0);
           s : out STD_LOGIC_VECTOR (7 downto 0));
end eight_bit_subtractor;


architecture Behavioral of eight_bit_subtractor is


signal c: std_logic_vector(8 downto 0);
```

```
signal h: std_logic_vector(7 downto 0);



Component full_adder is
    Port ( x : in STD_LOGIC;
           y : in STD_LOGIC;
           cin : in STD_LOGIC;
           s : out STD_LOGIC;
           cout : out STD_LOGIC);
end component;

begin
c(0) <= '1';
h <= not b;
S0: full_adder Port Map(x=> a(0), y=> h(0), cin => c(0),s=> s(0),cout => c(1));
S1: full_adder Port Map(x=> a(1), y=>h(1), cin => c(1),s=> s(1),cout => c(2));
S2: full_adder Port Map(x=> a(2), y=> h(2), cin => c(2),s=> s(2),cout => c(3));
S3: full_adder Port Map(x=> a(3), y=>h(3), cin => c(3),s=> s(3),cout => c(4));
S4: full_adder Port Map(x=> a(4), y=> h(4), cin => c(4),s=> s(4),cout => c(5));
S5: full_adder Port Map(x=> a(5), y=>h(5), cin => c(5),s=> s(5),cout => c(6));
S6: full_adder Port Map(x=> a(6), y=> h(6), cin => c(6),s=> s(6),cout => c(7));
S7: full_adder Port Map(x=> a(7), y=>h(7), cin => c(7),s=> s(7),cout => c(8));
end Behavioral;
```

Code 3: Design Code of 8-bit Subtractor

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

```
entity eight_bit_equaler is
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
           b : in STD_LOGIC_VECTOR (7 downto 0);
           eq : out STD_LOGIC);
end eight_bit_equaler;

architecture Behavioral of eight_bit_equaler is
signal hold: std_logic_vector(7 downto 0);
begin
hold(7) <= a(7) xnor b(7);
hold(6) <= a(6) xnor b(6);
hold(5) <= a(5) xnor b(5);
hold(4) <= a(4) xnor b(4);
hold(3) <= a(3) xnor b(3);
hold(2) <= a(2) xnor b(2);
hold(1) <= a(1) xnor b(1);
hold(0) <= a(0) xnor b(0);

eq <= hold(7) and hold(6) and hold(5) and hold(4) and hold(3) and hold(2) and hold(1) and hold(0);
end Behavioral;
```

Code 4: Design Code of 8-bit Equaler

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

```
entity two_bit_gt is
    Port ( a : in STD_LOGIC_VECTOR (1 downto 0);
           b : in STD_LOGIC_VECTOR (1 downto 0);
           gt : out STD_LOGIC);
end two_bit_gt;

architecture Behavioral of two_bit_gt is
signal holder: std_logic_vector(1 downto 0);
begin
holder(1) <= a(1) and (not b(1));
holder(0) <= (a(1) xnor b(1)) and (a(0) and (not b(0)));
gt <= holder(1) or holder(0);

end Behavioral;
```

Code 5: Design Code of 2-bit Greater Than

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity two_bit_eq is
    Port ( a : in STD_LOGIC_VECTOR (1 downto 0);
           b : in STD_LOGIC_VECTOR (1 downto 0);
           eq : out STD_LOGIC);
end two_bit_eq;
architecture Behavioral of two_bit_eq is

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

```
begin
hold(1) <= a(1) xnor b(1);
hold(0) <= a(0) xnor b(0);
eq <= hold(1) and hold(0);
end Behavioral;
```

Code 6: Design Code of 2-bit Equaler

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity four_bit_gt is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (3 downto 0);
           gt : out STD_LOGIC);
end four_bit_gt;
architecture Behavioral of four_bit_gt is
signal hold : std_logic_vector(2 downto 0);
signal keeper: std_logic;
Component two_bit_eq is
Port ( a : in STD_LOGIC_VECTOR (1 downto 0);
           b : in STD_LOGIC_VECTOR (1 downto 0);
           eq : out STD_LOGIC);
end component;
Component two_bit_gt is
    Port ( a : in STD_LOGIC_VECTOR (1 downto 0);
           b : in STD_LOGIC_VECTOR (1 downto 0);
           gt : out STD_LOGIC);
end component;
```

```
begin
U1: two_bit_gt Port Map (a(3 downto 2),b(3 downto 2),hold(0));
U2: two_bit_eq Port Map (a(3 downto 2),b(3 downto 2),hold(1));
U3: two_bit_gt Port Map (a(1 downto 0),b(1 downto 0),hold(2));
keeper <= hold(1) and hold(2);
gt <= hold(0) or keeper;
end Behavioral;
```

Code 7: Design Code of 4-bit Greater Than

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


entity four_bit_eq is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (3 downto 0);
           eq : out STD_LOGIC);
end four_bit_eq;



architecture Behavioral of four_bit_eq is
signal hold: std_logic_vector(3 downto 0);
begin
hold(3) <= a(3) xnor b(3);
hold(2) <= a(2) xnor b(2);
hold(1) <= a(1) xnor b(1);
hold(0) <= a(0) xnor b(0);
eq <= hold(3) and hold(2) and hold(1) and hold(0);
```

```
end Behavioral;
```

Code 8: Design Code of 4-bit Equaler

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


entity eight_bit_gt is
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
           b : in STD_LOGIC_VECTOR (7 downto 0);
           gt: out STD_LOGIC);
end eight_bit_gt;

architecture Behavioral of eight_bit_gt is

signal keep: std_logic_vector(2 downto 0);

signal hold: std_logic;



Component four_bit_eq is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
           b : in STD_LOGIC_VECTOR (3 downto 0);
           eq : out STD_LOGIC);
 end component;

Component four_bit_gt is
    Port ( a : in STD_LOGIC_VECTOR (3 downto 0);
```

```
        b : in STD_LOGIC_VECTOR (3 downto 0);
        gt : out STD_LOGIC);
 end component;

begin
U4: four_bit_gt Port Map(a(7 downto 4), b(7 downto 4), keep(0));
U5: four_bit_eq Port Map(a(7 downto 4), b(7 downto 4), keep(1));
U6: four_bit_gt Port Map(a(3 downto 0), b(3 downto 0), keep(2));

hold <= keep(1) and keep(2);
gt <= hold or keep(0);

end Behavioral;
```

Code 9: Design Code of 8-bit Greater Than

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



entity comprator is
    Port ( a : in std_logic_vector (7 downto 0);
        b : in std_logic_vector  (7 downto 0);
        aeqb : out STD_LOGIC;
        agtb : out STD_LOGIC;
        bgta : out STD_LOGIC);
```

```
end comprator;

architecture Behavioral of comprator is

Component eight_bit_equaler is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          eq : out STD_LOGIC);
end component;

Component eight_bit_gt is
   Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
          b : in STD_LOGIC_VECTOR (7 downto 0);
          gt : out STD_LOGIC);
end component;

begin

U7: eight_bit_equaler Port Map(a => std_logic_vector(a), b=> std_logic_vector(b),eq=> aeqb);
U8: eight_bit_gt Port Map(a=>std_logic_vector(a), b=>std_logic_vector(b),gt=>agtb);
U9: eight_bit_gt Port Map(a=>std_logic_vector(b), b=>std_logic_vector(a),gt=>bgta);

end Behavioral;
```

Code 10: Design Code of 8-bit Comparator

```
library IEEE;
```

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

entity eight_bit_gcd is
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
           b : in STD_LOGIC_VECTOR (7 downto 0);
           reset : in STD_LOGIC;
           enable : in STD_LOGIC;
           clk : in STD_LOGIC;
           go : out std_logic;
           s : out STD_LOGIC_VECTOR (7 downto 0));
end eight_bit_gcd;

architecture Behavioral of eight_bit_gcd is

type state_type is (idle, change, subtract);
signal register_state, next_state: state_type;

signal a_r, a_n, b_r, b_n: unsigned(7 downto 0);

signal subs_out: std_logic_vector(7 downto 0);
signal carry_aeqb: std_logic;
signal carry_agtb: std_logic;
signal carry_bgta: std_logic;

Component eight_bit_subtractor is
```

```
        Port ( a : in std_logic_vector (7 downto 0);
        b : in std_logic_vector (7 downto 0);
        s : out STD_LOGIC_VECTOR (7 downto 0));
end component;

Component comprator is
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
        b : in STD_LOGIC_VECTOR (7 downto 0);
        aeqb : out STD_LOGIC;
        agtb : out STD_LOGIC;
        bgta : out STD_LOGIC);
end component;

begin

SUB1: eight_bit_subtractor Port
Map(a=>std_logic_vector(a_r),b=>std_logic_vector(b_r),s=>subs_out);
COMPARER: comprator Port Map(a=> std_logic_vector(a_r), b=> std_logic_vector(b_r),
aeqb=>carry_aeqb,agtb=>carry_agtb,bgta=>carry_bgta);


process(clk,reset)
  begin
    if reset='1' then
      register_state <= idle;
      a_r <= "00000000";
```

```
        b_r <= "00000000";
      elsif rising_edge(clk) then
        register_state <= next_state;
        a_r <= a_n;
        b_r <= b_n;
      end if;
 end process;

process(register_state,a_r,b_r,enable,a,b, carry_aeqb,carry_bgta,carry_agtb,subs_out)
    begin
    a_n <= a_r;
    b_n <= b_r;

    case register_state is
       when idle =>
          if enable = '1' then
             a_n <= unsigned(a);
             b_n <= unsigned(b);
             next_state <= change;
          else
             next_state <= idle;
          end if;

       when change =>
          if carry_aeqb = '1' then
             next_state <= idle;
```

```
            else
                if carry_bgta = '1' then
                  a_n <= b_r;
                  b_n <= a_r;
                end if;
                next_state <= subtract;
            end if;
            when subtract =>
            a_n <= unsigned(subs_out);
            next_state <= change;
        end case;
    end process;
    go <= '1' when register_state=idle else '0';
    s <= std_logic_vector(a_r);
end Behavioral;
```

Code 11: Design Code of GCD Module

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

entity tb3 is
--  Port ( );
end tb3;

architecture Behavioral of tb3 is

Component eight_bit_gcd is
```

```
    Port ( a : in STD_LOGIC_VECTOR (7 downto 0);
           b : in STD_LOGIC_VECTOR (7 downto 0);
           reset : in STD_LOGIC;
           enable : in STD_LOGIC;
           clk : in STD_LOGIC;
           go : out std_logic;
           s : out STD_LOGIC_VECTOR (7 downto 0));
end component;



signal clk, reset, enable, ready: std_logic;
signal a,b, output: std_logic_vector(7 downto 0);

begin
uut: eight_bit_gcd Port Map(a=> a, b=> b, reset=> reset, enable=>enable, clk=> clk, go=>ready,
s=> output );

clk_process: process
begin
clk <= '0';
wait for 10ns;
clk <= '1';
wait for 10ns;
end process;
```

```
stim_proc: process
begin
enable<= '1';
a<="10001100";
b<="00001100";
reset<= '0';
wait for 100ns;
enable <= '0';
wait;
end process;
end Behavioral;
```

Code 12: Testbench Code of GCD Module