LAB NOTE

Subject: Digital Design Principles

Topic: Counters

Student: Minh Quan Tran

July 12th, 2024

Table of Contents

TABLE OF FIGURES

1. Objectives

- Design a Counters using Quartus software.
- Write a VHDL and Verilog code from the design.
- Push the code and run to the SCEMA5F31C6N board.

2. Theory and Design

2.1 Theory

2.1.1 Ring Counter

A ring counter is a type of counter composed of a circular shift register, where the output of the last flip-flop is fed back to the input of the first flip-flop. In a ring counter, only one flip-flop is set to '1' (or 'high') at any time, and this '1' circulates around the ring. For an N-bit ring counter, the sequence repeats every N states. This type of counter is simple but has limited use due to the fixed number of states equal to the number of flip-flops.

2.1.2 Johnson Counter

A Johnson counter, also known as a twisted ring counter or Möbius counter, is a modified version of the ring counter. In a Johnson counter, the complement of the output of the last flip-flop is fed back to the input of the first flip-flop. This configuration creates a sequence of states that is twice the number of flip-flops, effectively doubling the counting states compared to a standard ring counter. For an N-bit Johnson counter, the sequence repeats every 2N states, making it more efficient in terms of state utilization.

2.2 Requirement

- Use VHDL and Verilog to implement a Counters and display to LED [0:7].
- Introduction to the VHDL Case statement.
- **Above and beyond:** Adding Complement mode

2.3 Solution

To design this system, first need to identify input and output:

- Input will be from BTN0 to BTN3
- Output will be LED[8:0].

Then identify what the output will be, from the 2 inputs,

Straight ring/*Overbeck counter*					Twisted ring/*Johnson counter*				
State	Q0	Q1	Q2	Q3	State	Q0	Q1	Q2	Q3
0	1	0	0	0	0	0	0	0	0
1	0	1	0	0	1	1	0	0	0
2	0	0	1	0	2	1	1	0	0
3	0	0	0	1	3	1	1	1	0
0	1	0	0	0	4	1	1	1	1
1	0	1	0	0	5	0	1	1	1
2	0	0	1	0	6	0	0	1	1
3	0	0	0	1	7	0	0	0	1
0	1	0	0	0	*0*	0	0	0	0

Figure 2-1: Counters's Truth Table

3. VHDL and Verilog

3.1 VHDL code

3.1.1 Top entity

```
library ieee;
use ieee.std_logic_1164.ALL;

entity MTran_Lab8_VHDL_Counters is
port
(
    -- Inputs
    BTN          : in std_logic;
    Reset        : in std_logic;
    Complement     : in std_logic;

    -- Outputs
    LED_Ring     : out std_logic_vector (3 downto 0);
    LED_Johnson : out std_logic_vector (3 downto 0);

    HEX0         : out std_logic_vector (6 downto 0);
    HEX4         : out std_logic_vector (6 downto 0)

);
end entity;

architecture Behavioral of MTran_Lab8_VHDL_Counters is

--Signal Declaration
signal state: integer:= 0;

signal RingCounterState   : integer := 0;
signal JohnsonCounterState: integer := 0;

-- Module Declaration
component MTran_Lab8_VHDL_RingCounter is
port
(
    -- Inputs
    CLK             : in     std_logic;
    Reset           : in  std_logic;
    Complement      : in  std_logic;

    -- Outputs
    Q                  : out std_logic_vector (3 downto 0);
    stateNumber : out integer
);
end component;

component MTran_Lab8_VHDL_JohnsonCounter is
port
(
    -- Inputs
    CLK             : in  std_logic;
    Reset           : in  std_logic;
    Complement      : in  std_logic;

    -- Outputs
    Q                  : out std_logic_vector (3 downto 0);
    stateNumber : out integer
```

```
);
end component;

-- Function declaration
function SevenSegmentDisplay (Number : integer) return std_logic_vector is
begin
    case (Number) is
    when 0       => return "1000000";
    when 1       => return "1111001";
    when 2       => return "0100100";
    when 3       => return "0110000";
    when 4       => return "0011001";
    when 5       => return "0010010";
    when 6       => return "0000010";
    when 7      => return "1111000";
    when 8      => return "0000000";
    when 9      => return "0010000";
    when others => return "1111111";
    end case;
end SevenSegmentDisplay;


begin
    -- Module Institiate
    module_RingCounter: MTran_Lab8_VHDL_RingCounter
    port map
    (
        -- Inputs
        CLK          => BTN ,
        Reset          => Reset,
        Complement   => Complement,

        -- Output
        Q                => LED_Ring,
        stateNumber => RingCounterState
    );

    module_JohnsonCounter: MTran_Lab8_VHDL_JohnsonCounter
    port map
    (
        -- Inputs
        CLK           => BTN ,
        Reset          => Reset,
        Complement     => Complement,

        -- Output
        Q            => LED_Johnson,
        stateNumber => JohnsonCounterState
    );

    -- Process
    process(RingCounterState, JohnsonCounterState) is
    begin
        HEX0 <= SevenSegmentDisplay(RingCounterState);
        HEX4 <= SevenSegmentDisplay(JohnsonCounterState);
    end process;

end Behavioral;
```

3.1.2 Component Ring Counter

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

entity MTran_Lab8_VHDL_RingCounter is
port
(
    -- Inputs
    CLK           : in    std_logic;
    Reset         : in    std_logic;
    Complement    : in    std_logic;

    -- Outputs
    Q             : out   std_logic_vector (3 downto 0);
    stateNumber   : out   integer

);
end entity;

architecture RTL of MTran_Lab8_VHDL_RingCounter is
-- Signal Declaration
signal state          : integer   := 0;
signal complementMode : std_logic := '0';

signal Output         : std_logic_vector (3 downto 0) := "0000";

-- Procedure
procedure Increment (signal Counter: inout integer) is
begin
    if Counter < 3 then
        Counter <= Counter + 1;
    else
        Counter <= 0;
    end if;
end procedure;


-- Main code
begin

    -- Process
    process(CLK,Reset)
    begin
        if Reset = '0' then
            state <= 0;
        else
            if rising_edge(CLK) then
                Increment(state);
            end if;
        end if;
    end process;

    process(state, complementMode)
    begin
        case(state) is
            when 0          => Output <= "1000";
            when 1          => Output <= "0100";
            when 2          => Output <= "0010";
            when 3          => Output <= "0001";
            when others => Output <= "0000";
        end case;
```

```
        stateNumber <= state;
    end process;

    -- Check BTN1 for complement mode
    process(Complement)
    begin
        if rising_edge(Complement) then
            complementMode <= not complementMode;
        end if;
    end process;

    -- Process complement mode
    process (complementMode) is
    begin
        if complementMode = '1' then
            Q <= not Output;
        else
            Q <= Output;
        end if;
    end process;

end RTL;
```

3.1.3 Component Ring Counter

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

entity MTran_Lab8_VHDL_JohnsonCounter is
port
(
    -- Inputs
    CLK           : in    std_logic;
    Reset         : in    std_logic;
    Complement    : in    std_logic;

    -- Outputs
    Q             : out  std_logic_vector (3 downto 0);
    stateNumber   : out  integer

);
end entity;
architecture RTL of MTran_Lab8_VHDL_JohnsonCounter is
-- Signal Declaration
signal state           : integer   := 0;
signal complementMode : std_logic := '0';

signal Output          : std_logic_vector (3 downto 0) := "0000";

-- Procedure
procedure Increment (signal Counter: inout integer) is
begin
    if Counter < 7 then
        Counter <= Counter + 1;
    else
        Counter <= 0;
    end if;
end procedure;


-- Main code
```

```
begin

    -- Process
    process(CLK,Reset)
    begin
        if Reset = '0' then
            state <= 0;
        else
            if rising_edge(CLK) then
                Increment(State);
            end if;
        end if;
    end process;

    process(state, complementMode)
    begin
        case(state) is
            when 0          => Output <= "0000";
            when 1          => Output <= "1000";
            when 2          => Output <= "1100";
            when 3          => Output <= "1110";
            when 4          => Output <= "1111";
            when 5         => Output <= "0111";
            when 6          => Output <= "0011";
            when 7          => Output <= "0001";
            when others => Output <= "0000";
        end case;

        stateNumber <= state;
    end process;

    -- Check BTN1 for complement mode
    process(Complement)
    begin
        if rising_edge(Complement) then
            complementMode <= not complementMode;
        end if;
    end process;

    -- Process complement mode
    process (complementMode) is
    begin
        if complementMode = '1' then
            Q <= not Output;
        else
            Q <= Output;
        end if;
    end process;

end RTL;
```

3.2 Verilog code

3.2.1 Top Module

```
module MTran_Lab7_Verilog_MagnitudeComparator
(
    input wire CLOCK_50,
    input wire [3:0] A,
    input wire [3:0] B,
    input mode,
     input speed,
```

```
    output reg [2:0] LED,
    output reg [6:0] HEX0,
    output reg [6:0] HEX1,
    output reg [6:0] HEX2,
    output reg [6:0] HEX3,
    output reg [6:0] HEX4,
    output reg [6:0] HEX5
);

// Signals declaration
reg [31:0] intA;
reg [31:0] intB;

reg Enable;

wire [31:0] Seconds1;
wire [31:0] Minutes1;
wire [31:0] Hours1;

wire [31:0] Seconds2;
wire [31:0] Minutes2;
wire [31:0] Hours2;

// Module Instantiation
MTran_Lab7_Verilog_Timer
#(
     .MAXSECONDS(60),
     .MAXMINUTES(60),
     .MAXHOURS(24),
     .CLKFREQUENCY(500000) // 500kHz => Clock 10 times faster
)

Tim1
(
    .Clk(CLOCK_50),
    .Enable(Enable),
    .Seconds(Seconds1),
    .Minutes(Minutes1),
    .Hours(Hours1)
);

MTran_Lab7_Verilog_Timer
#(
     .MAXSECONDS(60),
     .MAXMINUTES(60),
     .MAXHOURS(24),
     .CLKFREQUENCY(50000000) // 50 MHz => Normal Clk
)
Tim2
(
    .Clk(CLOCK_50),
    .Enable(Enable),
    .Seconds(Seconds2),
    .Minutes(Minutes2),
    .Hours(Hours2)
);

// Function for Seven Segment Display
function [6:0] SevenSegmentDisplay;
    input [3:0] Number;
    begin
        case (Number)
```

```
            4'h0: SevenSegmentDisplay = 7'b1000000;
            4'h1: SevenSegmentDisplay = 7'b1111001;
            4'h2: SevenSegmentDisplay = 7'b0100100;
            4'h3: SevenSegmentDisplay = 7'b0110000;
            4'h4: SevenSegmentDisplay = 7'b0011001;
            4'h5: SevenSegmentDisplay = 7'b0010010;
            4'h6: SevenSegmentDisplay = 7'b0000010;
            4'h7: SevenSegmentDisplay = 7'b1111000;
            4'h8: SevenSegmentDisplay = 7'b0000000;
            4'h9: SevenSegmentDisplay = 7'b0010000;
           default: SevenSegmentDisplay = 7'b1111111;
        endcase
    end
endfunction



always @(posedge CLOCK_50)
begin
    if (mode == 1'b0)
     begin
        // Disable Timer
        Enable = 1'b0;

        // Reset HEX2
        HEX2 = 7'b1111111;

        // Convert A and B from std_logic to integer
        intA = A;
        intB = B;

        // Display A and B in decimal
        // A
        HEX0 = SevenSegmentDisplay(intA % 10);
        HEX1 = SevenSegmentDisplay(intA / 10);

        // B
        HEX4 = SevenSegmentDisplay(intB % 10);
        HEX5 = SevenSegmentDisplay(intB / 10);

        // Compare A and B
        if (intA > intB)
              begin
                    LED  = 3'b100;
                    HEX3 = 7'b0100111;
              end
          else if (intA == intB)
              begin
                    LED  = 3'b010;
                    HEX3 = 7'b0110111;
              end
          else
              begin
                    LED  = 3'b001;
                    HEX3 = 7'b0110011;
              end

    end
     else
     begin
        Enable = 1'b1;
        LED    = 3'b000;
```

```
        if (speed == 1'b0) //normal mode
        begin
            // Display Second
            HEX0  = SevenSegmentDisplay(Seconds2 % 10);
            HEX1  = SevenSegmentDisplay(Seconds2 / 10);

            // Display Minute
            HEX2  = SevenSegmentDisplay(Minutes2 % 10);
            HEX3  = SevenSegmentDisplay(Minutes2 / 10);

            // Display Hour
            HEX4  = SevenSegmentDisplay(Hours2 % 10);
            HEX5  = SevenSegmentDisplay(Hours2 / 10);
          end
          else if (speed == 1'b1)
          begin
              // Display Second
            HEX0  = SevenSegmentDisplay(Seconds1 % 10);
            HEX1  = SevenSegmentDisplay(Seconds1 / 10);

            // Display Minute
            HEX2  = SevenSegmentDisplay(Minutes1 % 10);
            HEX3  = SevenSegmentDisplay(Minutes1 / 10);

            // Display Hour
            HEX4  = SevenSegmentDisplay(Hours1 % 10);
            HEX5  = SevenSegmentDisplay(Hours1 / 10);
          end
    end
end

endmodule
```

3.2.2 Module Timer

```
module MTran_Lab7_Verilog_Timer

#(
parameter MAXSECONDS      = 60,
parameter MAXMINUTES      = 60,
parameter MAXHOURS        = 24,
parameter CLKFREQUENCY    = 50000000
)

(
    input wire Clk,
    input wire Enable,
    inout reg [31:0] Seconds,
    inout reg [31:0] Minutes,
    inout reg [31:0] Hours
);

// Signal Declaration
reg [31:0] Ticks = 0;

// Main Logic
always @(posedge Clk)
begin
    if (Enable)
     begin
        // Increment Ticks
```

```
      Ticks = Ticks + 1;
        if (Ticks == CLKFREQUENCY - 1)
            begin
                Ticks   = 0;
                Seconds = Seconds + 1;
            end

        if (Seconds == MAXSECONDS)
            begin
                Seconds = 0;
                Minutes = Minutes + 1;
            end

        if (Minutes == MAXMINUTES)
            begin
                Minutes = 0;
                Hours   = Hours + 1;
            end

        if (Hours == MAXHOURS)
            begin
                Hours   = 0;
            end
   end
end

endmodule
```

4. Pin Planner

4.1 Input and Output

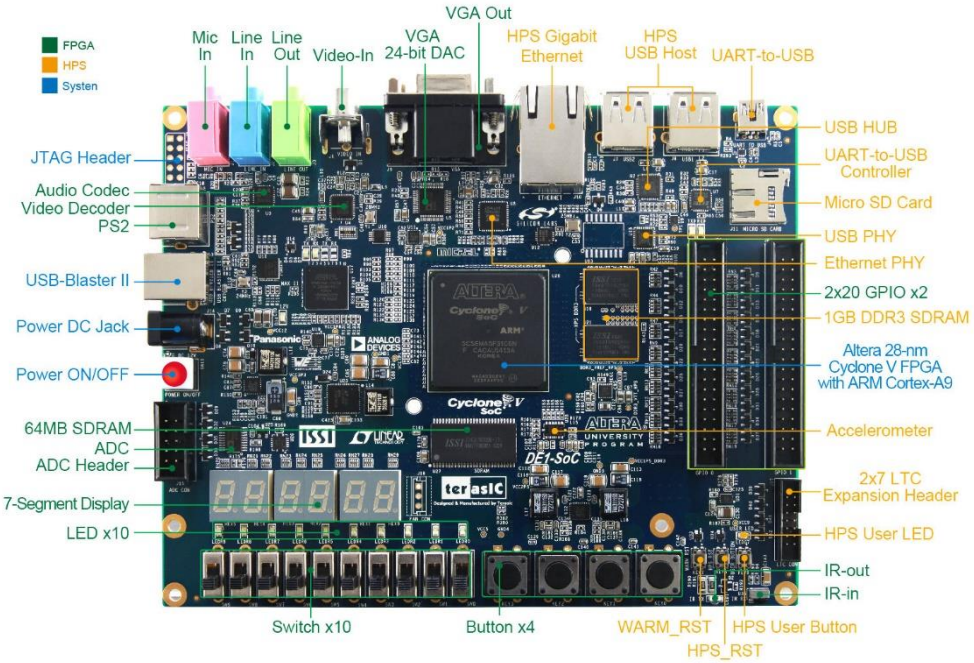

Figure 4-1: SCEMA5F31C6N board

Assigning:

SW[0]: SW0
SW[1]: SW1
SW[2]: SW2

Mode[0]: SW7
Mode[1]: SW8
EN: SW9

LED[0]: LEDR0
LED[1]: LEDR1
LED[2]: LEDR2
LED[3]: LEDR3
LED[4]: LEDR4
LED[5]: LEDR5
LED[6]: LEDR6
LED[7]: LEDR7

HEX0[0]: HEX0[0]
HEX0[1]: HEX0[1]
HEX0[2]: HEX0[2]
HEX0[3]: HEX0[3]
HEX0[4]: HEX0[4]
HEX0[5]: HEX0[5]
HEX0[6]: HEX0[6]

4. Pin Planner

From the DE1_SoC_User_Manual,

Signal Name	FPGA Pin No.	Description	I/O Standard
SW[0]	PIN_AB12	Slide Switch[0]	3.3V
SW[1]	PIN_AC12	Slide Switch[1]	3.3V
SW[2]	PIN_AF9	Slide Switch[2]	3.3V
SW[3]	PIN_AF10	Slide Switch[3]	3.3V
SW[4]	PIN_AD11	Slide Switch[4]	3.3V
SW[5]	PIN_AD12	Slide Switch[5]	3.3V

Figure 4-2: SW0 and SW1 Pin No

Signal Name	FPGA Pin No.	Description	I/O Standard
LEDR[0]	PIN_V16	LED [0]	3.3V
LEDR[1]	PIN_W16	LED [1]	3.3V

Figure 4-3: LEDR0's Pin No

Signal Name	FPGA Pin No.	Description	I/O Standard
HEX0[0]	PIN_AE26	Seven Segment Digit 0[0]	3.3V
HEX0[1]	PIN_AE27	Seven Segment Digit 0[1]	3.3V
HEX0[2]	PIN_AE28	Seven Segment Digit 0[2]	3.3V
HEX0[3]	PIN_AG27	Seven Segment Digit 0[3]	3.3V
HEX0[4]	PIN_AF28	Seven Segment Digit 0[4]	3.3V
HEX0[5]	PIN_AG28	Seven Segment Digit 0[5]	3.3V
HEX0[6]	PIN_AH28	Seven Segment Digit 0[6]	3.3V

Figure 4-4: HEX's Pin No

5. Result

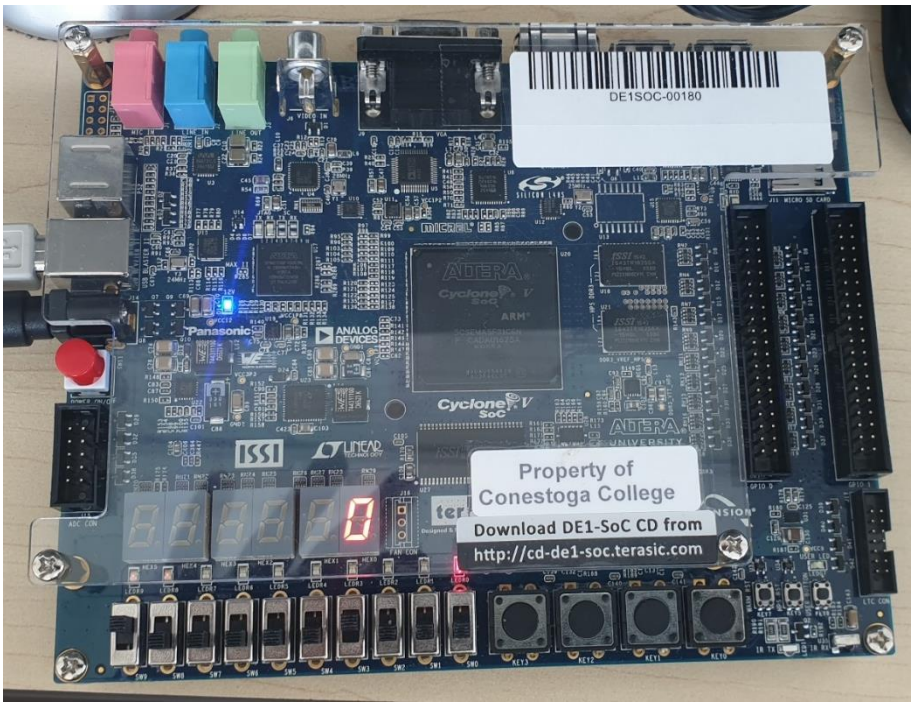

Figure 5-1: When input = ‘000’.

SW2 = 0, SW1 = 0, SW0 = 0 which should be 0, **LED0** asserted and the 7 segment LED displaying **0**.

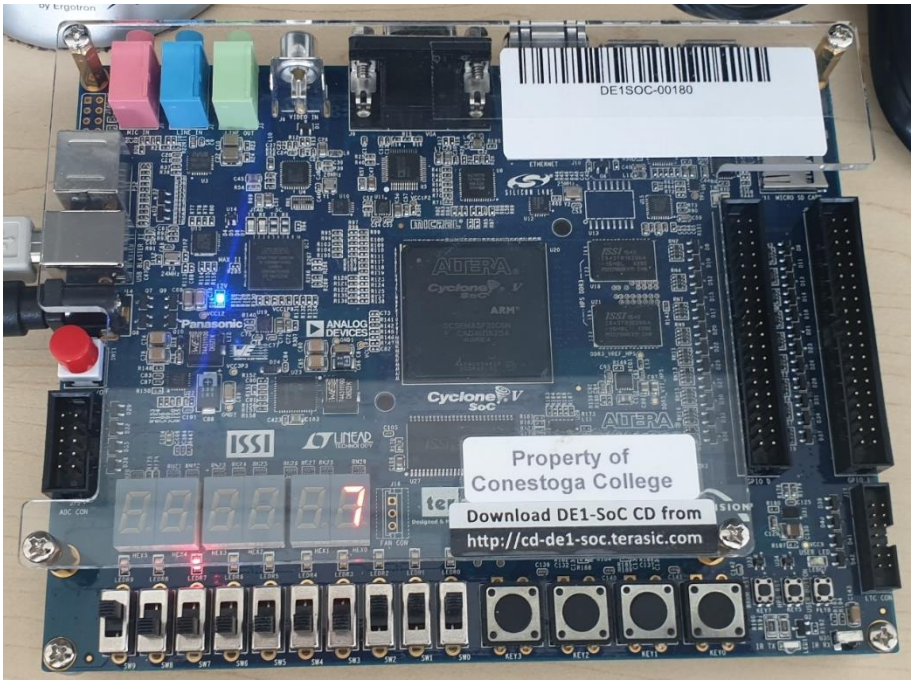

Figure 5-2: When input = ‘111’

SW2 = 1, SW1 = 1, SW0 = 1 which should be 7, **LED7** asserted and the 7 segment LED displaying **7**.

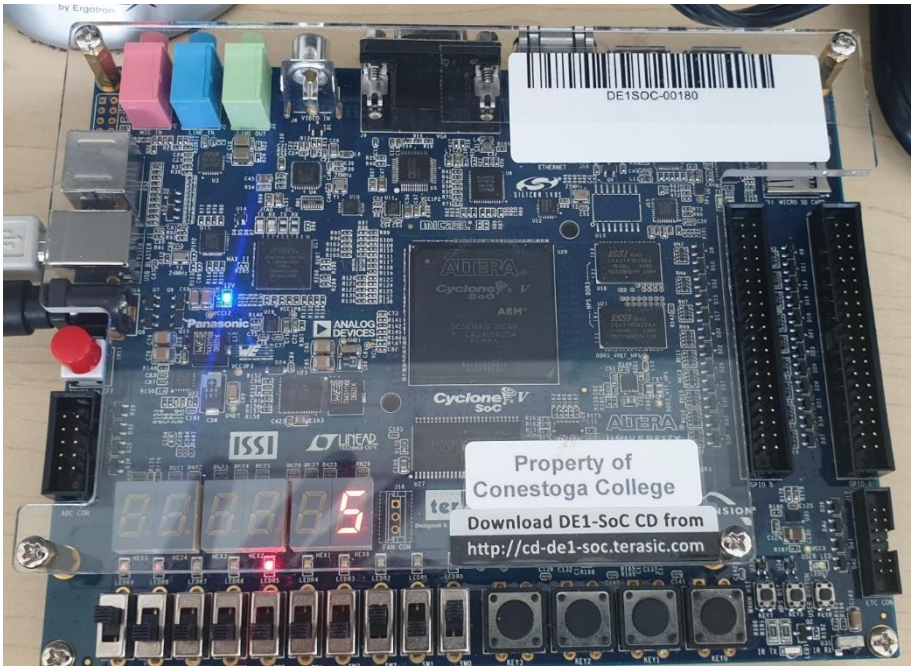

Figure 5-3: When input '101'

SW2 = 1, SW1 = 0, SW0 = 1 which should be 5, **LED5** asserted and the 7 segment LED displaying **5**.

Different mode:

- **Complement mode:**

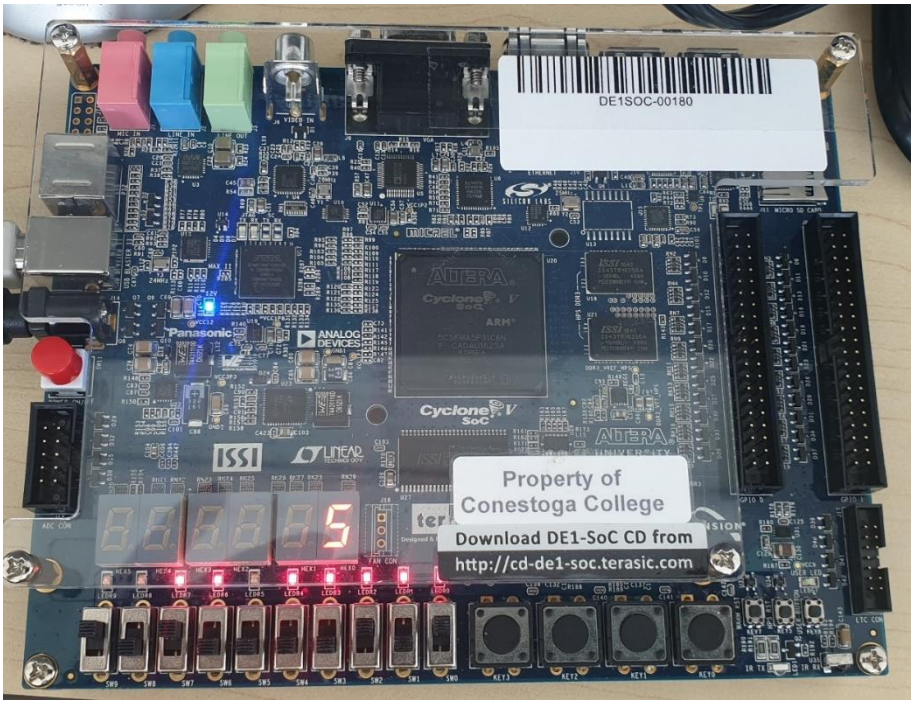

Figure 5-4: Input '101' with Complement mode

In this mode, rather than LED5 assert, all other LED except LED5 asserted. To activate this mode SW [8:7] = '01'.

5. Result

- **Multiline Mode:**

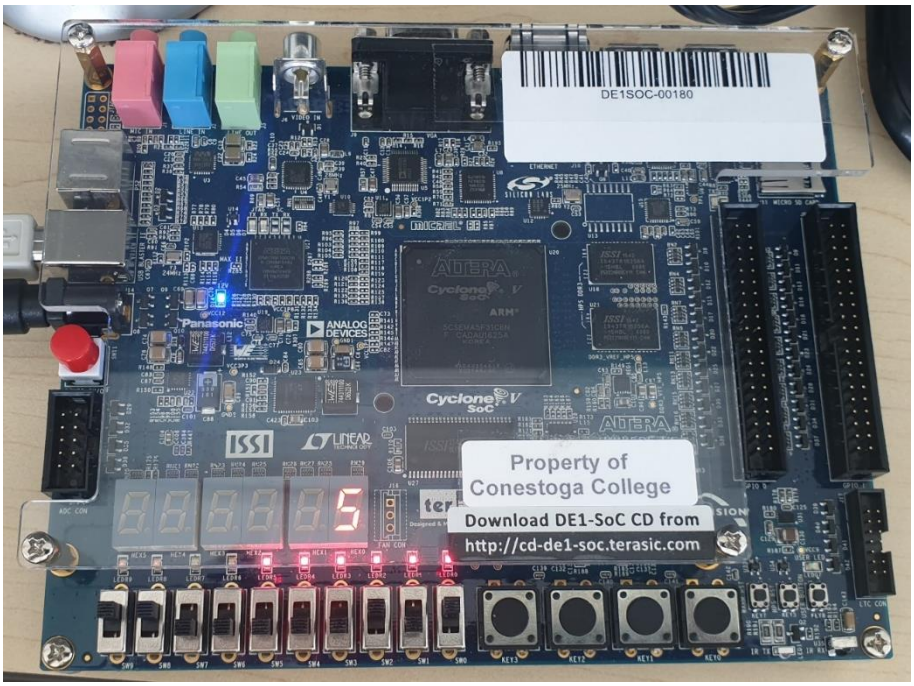

Figure 5-5: Input '101' with Multiline mode

In this mode, when input is '101' which is **5**, LED0 to LED5 lit up. To activate this mode SW [8:7] = '10'.

- **Complement Multiline Mode:**

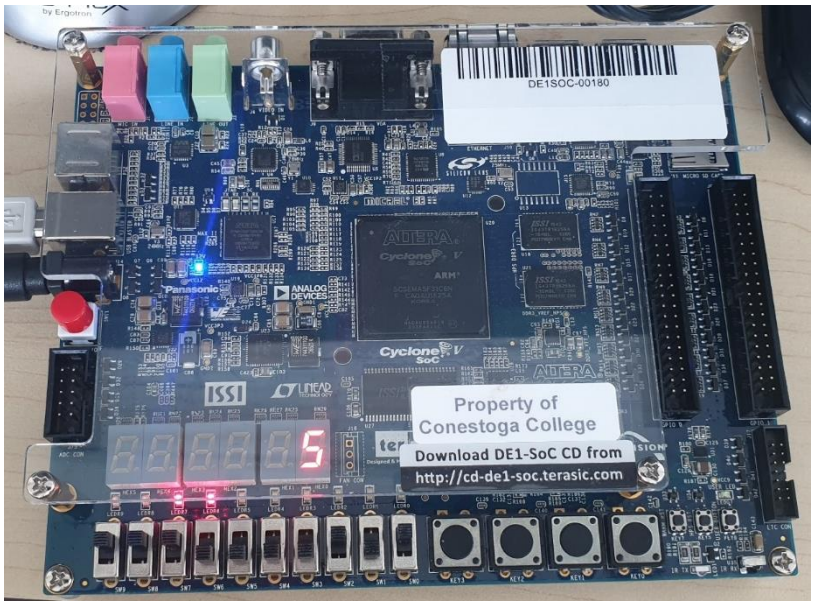

Figure 5-6: Result for adding them up

In this mode, it take the output of Multiline Mode and complemented it. To activate this mode SW [8:7] = '11'

REFERENCES