CRN: 22840

CET3136 – Logic Devices Programming

Spring 2025

Experiment 8

Counters

Performed By:

Anthony Paul Sevarino

Submitted to:

Prof. Ashley Evans

Department of Electrical & Computer Engineering Technology (ECET)

School of Engineering, Technology, and Advanced Manufacturing (ETAM)

Valencia College

Date Submitted

03/13/2025

Introduction

The objective of this experiment is to create counters using various different methodologies on the MAX 10 DE-Lite Development board. The first iteration of a counter is a simple 5-bit counter which iterates on a clock with an asynchronous reset. The second iteration is an 8-bit counter utilizing 8 positive-edge triggered T-flipflops. Finally, the third clock iteration is based off the 50MHz clock located on the MAX 10 development board. Due to the variety of solutions present in this experiment, it is vital to understand both the VHDL aspect of these solutions, as well as the behavior of the physical components of the development board, such as the clock or the HEX displays. Both buttons and switches will be used as inputs throughout the study, and HEX displays will serve as the primary output means. Also, functional simulations of the lab will be analyzed and each system's Fmax observed as well.

LIST OF EQUIPMENT/PARTS/COMPONENTS/SOFTWARE

- DE10-Lite MAX-10 Dev Board
- USB 2.0 Type B Cable
- Windows Desktop
- Quartus FPGA Design Software

PROCEDURE / DISCUSSION

Part A – The 5-bit Counter

There is no schematic provided for this part of the experiment, however a sample VHDL statement is offered:

```
Q <= Q + 1;
```

Figure 1 - Sample VHDL Statement for Part A

The first part of the experiment demands a 5-bit counter which increments based off of a falling edge clock signal, with an active-low asynchronous reset. Thus, the code must give precedence to the reset, with clock controlling the rest of the logic of the program. Since this is a 5-bit counter, the upper boundary of the system should be 32, or 31 within the program (as index 0 is the lower boundary.)

Code Explanation Part A

```
1   ----------MAIN BLOCK----------
2   ------------------------------------
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.numeric_std.all;
6
7   entity counter5bitVHD is
8   port  (
9         clk    :  in std_logic;
10        rst_bar  :  in std_logic;
11        output:  out std_logic_vector(4 downto 0)
12        );
13  end counter5bitVHD;
14
15  architecture behavior of counter5bitVHD is
16
17  signal Q : integer range 0 to 31 := 0;
18
19  begin
20
21  process(clk, rst_bar)
22
23  begin
24
25     if falling_edge(clk) and rst_bar = '1' then
26        if Q = 31 then
27           Q <= 0;
28        else
29           Q <= Q + 1;
30        end if;
31     end if;
32
33     if rst_bar = '0' then
34        Q <= 0;
35     end if;
36
37  end process;
38
39  output <= std_logic_vector(to_unsigned(Q, 5));
40
41
42  end behavior;
43  ------------------------------------
44  ----------MAIN BLOCK----------
```

Figure 2 - Completed Code for D Latch

----------MAIN BLOCK----------

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

entity counter5bitVHD is
port	(
		clk	:	in std_logic;
		rst_bar	:	in std_logic;
		output:	out std_logic_vector(4 downto 0)
		);
end counter5bitVHD;

architecture behavior of counter5bitVHD is

signal Q : integer range 0 to 31 := 0;

begin

process(clk, rst_bar)

begin

	if falling_edge(clk) and rst_bar = '1' then
		if Q = 31 then
			Q <= 0;
		else
			Q <= Q + 1;
		end if;
	end if;

	if rst_bar = '0' then
		Q <= 0;
	end if;

end process;

output <= std_logic_vector(to_unsigned(Q, 5));


end behavior;
-------------------------------------
-------------MAIN BLOCK-------------
```

The entity portion of the program takes 2 inputs, a clock and a reset (clk and rst_bar respectively). The reset is active low, thus the chosen nomenclature. The system has only one output, acting as a 5 bit vector called *output*.

The architecture begins with a signal *Q* of range 0 to 31, defined initially as 0. A sensitivity list process then accepts *clock* and *rst_bar* as parameters. This means that any time either of the two inputs changes, the process will run (thus the asynchronous component is completed). Within this process, a conditional if statement is used to check if *clock* input is a falling edge AND if reset is set to 1 (recall that the reset is logic low.) If so, as long as *Q* is not full (at it's upper boundary), it iterates *Q*. Otherwise, if *rst_bar* is set to 0, then *Q* is reset to 0. *Q* is also reset to 0 if it has reached

it's upper boundary. Finally, the value *Q* is converted to an unsigned binary number of magnitude 5, then converted to a logic vector for transfer to it's output.

Below is the block diagram for this part of the experiment. There are no pin assignments as this part of the study is not ported to the board:

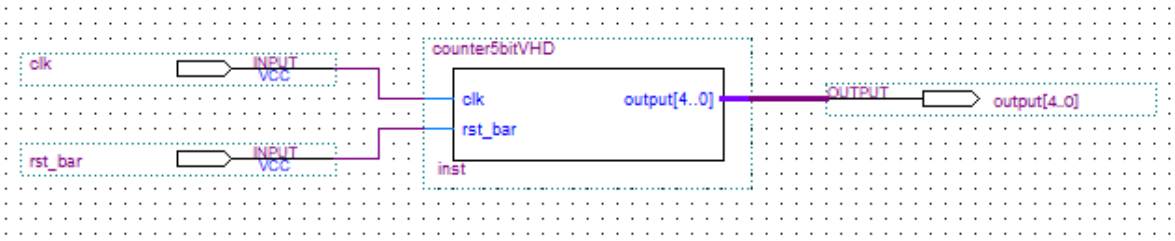

Figure 3 - Part A Block Diagram

Finally, the Fmax and total logic elements for the circuit are provided below:

Fitter Status	Successful - Wed Mar 12 10:15:53 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	counter5bitVHD
Top-level Entity Name	counter5bitVHD
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	6 / 49,760 (< 1 %)
Total registers	5
Total pins	7 / 360 (2 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

Figure 4 -Compilation Report Showing Logic Elements for Part A

	Fmax	Restricted Fmax	Clock Name	Note
1	486.38 MHz	250.0 MHz	clk	limit due to minimum period restriction (max I/O toggle rate)

Figure 5 - Fmax Result for Part A

Part B – The 8-bit T-flipflop Counter

For part B, the following schematic is provided:

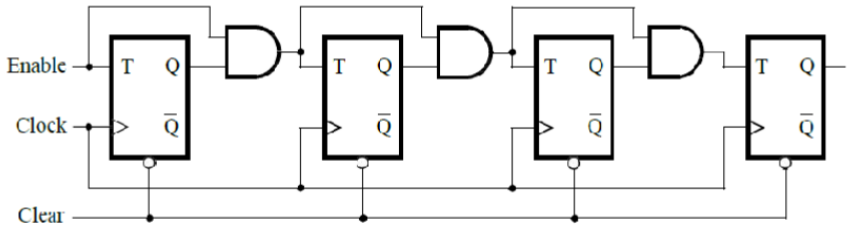

Figure 4 – 4-bit T-flopflop Counter

The structure of this schematic lays the foundation for the derived solution for part B of the experiment. Note the positive-edge clock incrementation, ANDing of the enable with previous Q output for new enable, and the asynchronous clear attached to each flip-flop.

Code Explanation Part B

```
1   ------------MAIN BLOCK------------
2   ----------------------------------
3   library ieee;
4   use ieee.std_logic_1164.all;
5   use ieee.numeric_std.all;
6
7   entity eightbitcounter is
8   port (
9       enable, clock, clear_bar : in STD_LOGIC;
10      q_out : out STD_LOGIC_VECTOR(7 downto 0);
11      hex1, hex0: out STD_LOGIC_VECTOR(6 downto 0)
12  );
13  end eightbitcounter;
14
15  architecture structure of eightbitcounter is
16
17  component tFlipFlop is
18  port (
19      t, clk, clr_bar : in std_logic;
20      q : out std_logic
21  );
22  end component;
23
24  signal q_link : std_logic_vector(7 downto 0);
25  signal t_link : std_logic_vector(7 downto 0);
26
27  begin
28
29      -- First Flip-Flop directly toggles on enable
30      t1 : tFlipFlop port map (t => enable, clk => clock, clr_bar => clear_bar, q => q_link(0));
31
32      -- enables are conditional for remaining flip flops
33      t_link(0) <= enable and q_link(0);
34      t2 : tFlipFlop port map (t => t_link(0), clk => clock, clr_bar => clear_bar, q => q_link(1));
35
36      t_link(1) <= t_link(0) and q_link(1);
37      t3 : tFlipFlop port map (t => t_link(1), clk => clock, clr_bar => clear_bar, q => q_link(2));
38
39      t_link(2) <= t_link(1) and q_link(2);
40      t4 : tFlipFlop port map (t => t_link(2), clk => clock, clr_bar => clear_bar, q => q_link(3));
41
42      t_link(3) <= t_link(2) and q_link(3);
43      t5 : tFlipFlop port map (t => t_link(3), clk => clock, clr_bar => clear_bar, q => q_link(4));
44
45      t_link(4) <= t_link(3) and q_link(4);
46      t6 : tFlipFlop port map (t => t_link(4), clk => clock, clr_bar => clear_bar, q => q_link(5));
47
48      t_link(5) <= t_link(4) and q_link(5);
49      t7 : tFlipFlop port map (t => t_link(5), clk => clock, clr_bar => clear_bar, q => q_link(6));
50
51      t_link(6) <= t_link(5) and q_link(6);
52      t8 : tFlipFlop port map (t => t_link(6), clk => clock, clr_bar => clear_bar, q => q_link(7));
53
54      -- Assign outputs
55      q_out <= q_link;
56
57      --Hexadecimal Hex Display Select Statements
58      with q_link(3 downto 0) select
59        hex0 <= "1000000" when "0000",
60                "1111001" when "0001",
61                "0100100" when "0010",
62                "0110000" when "0011",
63                "0011001" when "0100",
64                "0010010" when "0101",
65                "0000010" when "0110",
66                "1111000" when "0111",
67                "0000000" when "1000",
68                "0010000" when "1001",
69                "0001000" when "1010",
70                "0000011" when "1011",
71                "1000110" when "1100",
72                "0100001" when "1101",
73                "0000110" when "1110",
74                "0001110" when "1111",
75                "1111111" when others;
76
77      with q_link(7 downto 4) select
78         hex1 <= "1000000" when "0000",
79                 "1111001" when "0001",
80                 "0100100" when "0010",
81                 "0110000" when "0011",
82                 "0011001" when "0100",
83                 "0010010" when "0101",
84                 "0000010" when "0110",
85                 "1111000" when "0111",
86                 "0000000" when "1000",
87                 "0010000" when "1001",
88                 "0001000" when "1010",
89                 "0000011" when "1011",
90                 "1000110" when "1100",
91                 "0100001" when "1101",
92                 "0000110" when "1110",
93                 "0001110" when "1111",
94                 "1111111" when others;
```

```
end structure;
------------------------------------
-------------MAIN BLOCK-------------

-------------T-FlipFlop-------------
------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity tFlipFlop is
port (
    t, clk, clr_bar : in std_logic;
    q : out std_logic
);
end tFlipFlop;

architecture behavior of tFlipFlop is
    signal q_temp : std_logic := '0';  -- holder for q result
begin
    process(clk, clr_bar)
    begin
        if clr_bar = '0' then      -- Asynchronous reset when clear is active
            q_temp <= '0';
        elsif rising_edge(clk) then -- On the rising edge of the clock
            if t = '1' then
                q_temp <= not q_temp; -- Toggle Q when T is high
            end if;
        end if;
    end process;

    q <= q_temp; -- output q
end behavior;
------------------------------------
-------------T-FlipFlop-------------
```

Figure 6 - Completed Code for D Latch

```
-------------MAIN BLOCK-------------
------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity eightbitcounter is
port (
   enable, clock, clear_bar : in STD_LOGIC;
   q_out : out STD_LOGIC_VECTOR(7 downto 0);
   hex1, hex0: out STD_LOGIC_VECTOR(6 downto 0)
);
end eightbitcounter;

architecture structure of eightbitcounter is

component tFlipFlop is
port (
   t, clk, clr_bar : in std_logic;
   q : out std_logic
);
end component;

signal q_link : std_logic_vector(7 downto 0);
signal t_link : std_logic_vector(7 downto 0);

begin

   -- First Flip-Flop directly toggles on enable
   t1 : tFlipFlop port map (t => enable, clk => clock, clr_bar => clear_bar, q => q_link(0));

   -- enables are conditional for remaining flip flops
   t_link(0) <= enable and q_link(0);
   t2 : tFlipFlop port map (t => t_link(0), clk => clock, clr_bar => clear_bar, q => q_link(1));
```

```
t_link(1) <= t_link(0) and q_link(1);
t3 : tFlipFlop port map (t => t_link(1), clk => clock, clr_bar => clear_bar, q => q_link(2));

t_link(2) <= t_link(1) and q_link(2);
t4 : tFlipFlop port map (t => t_link(2), clk => clock, clr_bar => clear_bar, q => q_link(3));

t_link(3) <= t_link(2) and q_link(3);
t5 : tFlipFlop port map (t => t_link(3), clk => clock, clr_bar => clear_bar, q => q_link(4));

t_link(4) <= t_link(3) and q_link(4);
t6 : tFlipFlop port map (t => t_link(4), clk => clock, clr_bar => clear_bar, q => q_link(5));

t_link(5) <= t_link(4) and q_link(5);
t7 : tFlipFlop port map (t => t_link(5), clk => clock, clr_bar => clear_bar, q => q_link(6));

t_link(6) <= t_link(5) and q_link(6);
t8 : tFlipFlop port map (t => t_link(6), clk => clock, clr_bar => clear_bar, q => q_link(7));

-- Assign outputs
q_out <= q_link;

        --Hexadecimal Hex Display Select Statements
        with q_link(3 downto 0) select
                hex0 <= "1000000" when "0000",
        "1111001" when "0001",
        "0100100" when "0010",
        "0110000" when "0011",
        "0011001" when "0100",
        "0010010" when "0101",
        "0000010" when "0110",
        "1111000" when "0111",
        "0000000" when "1000",
        "0010000" when "1001",
        "0001000" when "1010",
        "0000011" when "1011",
        "1000110" when "1100",
        "0100001" when "1101",
        "0000110" when "1110",
        "0001110" when "1111",
        "1111111" when others;

        with q_link(7 downto 4) select
                hex1 <= "1000000" when "0000",
        "1111001" when "0001",
        "0100100" when "0010",
        "0110000" when "0011",
        "0011001" when "0100",
        "0010010" when "0101",
        "0000010" when "0110",
        "1111000" when "0111",
        "0000000" when "1000",
        "0010000" when "1001",
        "0001000" when "1010",
        "0000011" when "1011",
        "1000110" when "1100",
        "0100001" when "1101",
        "0000110" when "1110",
```

```
            "0001110" when "1111",
            "1111111" when others;

end structure;
-------------------------------------
-------------MAIN BLOCK-------------

-------------T-FlipFlop-------------
-------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity tFlipFlop is
port (
   t, clk, clr_bar : in std_logic;
   q : out std_logic
);
end tFlipFlop;

architecture behavior of tFlipFlop is
   signal q_temp : std_logic := '0';  -- holder for q result
begin
   process(clk, clr_bar)
   begin
      if clr_bar = '0' then      -- Asynchronous reset when clear is active
         q_temp <= '0';
      elsif rising_edge(clk) then -- On the rising edge of the clock
         if t = '1' then
            q_temp <= not q_temp; -- Toggle Q when T is high
         end if;
      end if;
   end process;

   q <= q_temp; -- output q
end behavior;
-------------------------------------
-------------T-FlipFlop-------------
```

There are two components to this solution, the main block, and the T-flipflop block. In the flip flop block, the entity takes three inputs, those being *t* (the enable), *clk*, and *clr_bar* (the logic low reset), as well as a single *q* output. The main block entity also has three inputs *enable*, *clock*, and *clear_bar*, representing the same connections as the flip flop. This entity also has *q_out* as an output, but differs from the previous entity as it also has two 7 digit vector outputs representing the HEX display outputs.

The architecture for the T-flipflop is simple, a sensitivity list process takes clear bar as the highest precedence element, and if 0, changes a standard logic signal *q_temp* to 0. This signal represents the temporary value of *q* which will be ported to the true output element. Next, if the clock arrives at a rising edge, if enable is set, then *q_temp* is inverted, or toggled. Finally, outside of the process, *q_temp* is assigned to *q*.

In the main block architecture, the T-flipflop is instantiated 8 times, while in between the required Boolean logic is applied to the enable and previous q outputs for each new enable input. Finally, the Hex displays are illuminated according to the Q results.

Below is the block diagram and pin assignments for this part of the experiment:

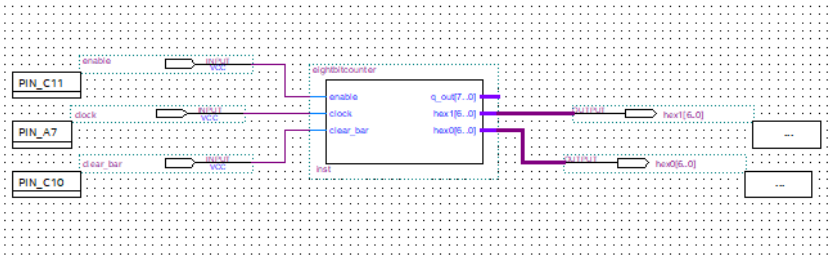

Figure 7 - Part B Block Diagram

clear_bar	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL
clock	Input	PIN_A7	7	B7_N0	PIN_A7	3.3 V Sc... Trigger
enable	Input	PIN_C11	7	B7_N0	PIN_C11	3.3-V LVTTL
hex0[6]	Output	PIN_B22	6	B6_N0	PIN_B22	3.3-V LVTTL
hex0[5]	Output	PIN_C22	6	B6_N0	PIN_C22	3.3-V LVTTL
hex0[4]	Output	PIN_B21	6	B6_N0	PIN_B21	3.3-V LVTTL
hex0[3]	Output	PIN_A21	6	B6_N0	PIN_A21	3.3-V LVTTL
hex0[2]	Output	PIN_B19	7	B7_N0	PIN_B19	3.3-V LVTTL
hex0[1]	Output	PIN_A20	7	B7_N0	PIN_A20	3.3-V LVTTL
hex0[0]	Output	PIN_B20	6	B6_N0	PIN_B20	3.3-V LVTTL
hex1[6]	Output	PIN_E17	6	B6_N0	PIN_E17	3.3-V LVTTL
hex1[5]	Output	PIN_D19	6	B6_N0	PIN_D19	3.3-V LVTTL
hex1[4]	Output	PIN_C20	6	B6_N0	PIN_C20	3.3-V LVTTL
hex1[3]	Output	PIN_C19	7	B7_N0	PIN_C19	3.3-V LVTTL
hex1[2]	Output	PIN_E21	6	B6_N0	PIN_E21	3.3-V LVTTL
hex1[1]	Output	PIN_E22	6	B6_N0	PIN_E22	3.3-V LVTTL
hex1[0]	Output	PIN_F21	6	B6_N0	PIN_F21	3.3-V LVTTL
q_out[7]	Output	PIN_D14	7	B7_N0	PIN_D14	3.3-V LVTTL
q_out[6]	Output	PIN_E14	7	B7_N0	PIN_E14	3.3-V LVTTL
q_out[5]	Output	PIN_C13	7	B7_N0	PIN_C13	3.3-V LVTTL
q_out[4]	Output	PIN_D13	7	B7_N0	PIN_D13	3.3-V LVTTL
q_out[3]	Output	PIN_B10	7	B7_N0	PIN_B10	3.3-V LVTTL
q_out[2]	Output	PIN_A10	7	B7_N0	PIN_A10	3.3-V LVTTL
q_out[1]	Output	PIN_A9	7	B7_N0	PIN_A9	3.3-V LVTTL
q_out[0]	Output	PIN_A8	7	B7_N0	PIN_A8	3.3-V LVTTL

Figure 8 - Part B Pin Assignments

Finally, the Fmax and total logic elements for the circuit are provided below:

Total logic elements	26 / 49,760 (< 1 %)
Total registers	8

Figure 9 -Compilation Report Showing Logic Elements for Part B

	Fmax	Restricted Fmax	Clock Name	Note
1	534.19 MHz	250.0 MHz	clock	limit due to minimum period restriction

Figure 10 - Fmax for Part B

Part C – The 50MHz 0-to-9 Counter

This section of the experiment deals with a new concept, using the 50MHz clock on the MAX 10

board rather than a standard logic input acting as the clock. Researching the correct pin assignment and behavior of this clock is crucial to completing this portion of the study.

Note the clock operates on 50MHz, or 50 million times per second. Thus, a rising edge should at that point. An interim counter variable can be used to keep track of where in the cycle the program is, which is demonstrated in the code explanation below.

Code Explanation Part C

```
1   ----------MAIN BLOCK----------
2   ------------------------------------
3    library ieee;
4    use ieee.std_logic_1164.all;
5
6   entity onesecondcounter is
7   port   (
8          input_clock, reset : in std_logic;
9          hex0 : out std_logic_vector(6 downto 0)
10         );
11   end onesecondcounter;
12
13  architecture behavior of onesecondcounter is
14
15   signal clk_spread : integer range 0 to 49999999 := 0; --50MHz means 50 million times per second
16   signal currentNum : integer range 0 to 9 := 0;
17
18  begin
19
20     process(input_clock, reset)
21     begin
22
23        if reset = '1' then
24           clk_spread <= 0;
25           currentNum <= 0;
26        elsif rising_edge(input_clock) then
27           if clk_spread = 49999999 then     --once clock reaches 1 second, reset clock, then perform logic
28              clk_spread <= 0;
29              if currentNum = 9 then
30                 currentNum <= 0;
31              else
32                 currentNum <= currentNum + 1;
33              end if;
34           else
35              clk_spread <= clk_spread + 1;
36           end if;
37        end if;
38     end process;
39
40     with currentNum select
41        hex0 <= "1000000" when 0,
42                "1111001" when 1,
43                "0100100" when 2,
44                "0110000" when 3,
45                "0011001" when 4,
46                "0010010" when 5,
47                "0000010" when 6,
48                "1111000" when 7,
49                "0000000" when 8,
50                "0010000" when 9,
51                "1111111" when others;
52
53  end behavior;
54  ------------------------------------
55  ----------MAIN BLOCK----------
56
```

Figure 11 - Completed Code for Section C of the Experiment

```
----------MAIN BLOCK----------
------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity onesecondcounter is
port      (
                  input_clock, reset : in std_logic;
                  hex0 : out std_logic_vector(6 downto 0)
                  );
end onesecondcounter;

architecture behavior of onesecondcounter is

signal clk_spread : integer range 0 to 49999999 := 0;          --50MHz means 50 million times per second
```

```
signal currentNum : integer range 0 to 9 := 0;

begin

	process(input_clock, reset)
	begin

		if reset = '1' then
			clk_spread <= 0;
			currentNum <= 0;
		elsif rising_edge(input_clock) then
			if clk_spread = 49999999 then		--once clock reaches 1 second, reset clock,
then perform logic
				clk_spread <= 0;
				if currentNum = 9 then
					currentNum <= 0;
				else
					currentNum <= currentNum + 1;
				end if;
			else
				clk_spread <= clk_spread + 1;
			end if;
		end if;
	end process;

	with currentNum select
		hex0 <= "1000000" when 0,
	"1111001" when 1,
	"0100100" when 2,
	"0110000" when 3,
	"0011001" when 4,
	"0010010" when 5,
	"0000010" when 6,
	"1111000" when 7,
	"0000000" when 8,
	"0010000" when 9,
	"1111111" when others;

end behavior;
-------------------------------------
-------------MAIN BLOCK-------------
```

The entity portion of this code takes two inputs as *input_clock* and *reset*, with a single output acting as the HEX display output vector.

The architecture begins by defining two signals, one to define the upper boundary of the number counter, and one to define the upper boundary of the cycle tracker for the clock. As reset takes precedence, in a sensitivity list process accepting *input_clock* and *reset* as parameters, the clock counter and number counter are both set to 0 if *reset* is set to 1 (active high reset.) Then, if there is a rising edge on the *input_clock* (which according to the pin assigner is given to the 50MHz clock on the development board,) if the clock has reach it's upper boundary, then *clk_spread* is reset and *currentNum* state is checked. If full, it resets, otherwise it increments. The clock then increments after this all completes. Finally, the HEX display is illuminated accordingly using a select statement.

Below is the block diagram for the experiment.

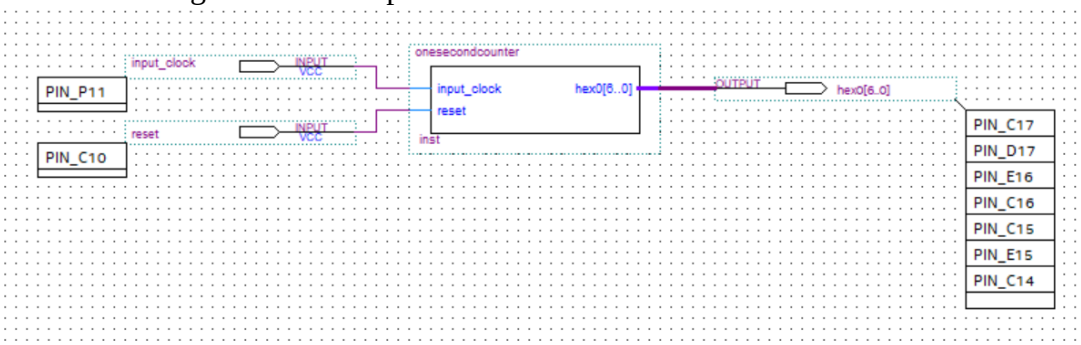

Figure 12 - Part C Block Diagram

out hex0[6]	Output	PIN_C17	7	B7_N0	PIN_C17	3.3-V LVTTL
out hex0[5]	Output	PIN_D17	7	B7_N0	PIN_D17	3.3-V LVTTL
out hex0[4]	Output	PIN_E16	7	B7_N0	PIN_E16	3.3-V LVTTL
out hex0[3]	Output	PIN_C16	7	B7_N0	PIN_C16	3.3-V LVTTL
out hex0[2]	Output	PIN_C15	7	B7_N0	PIN_C15	3.3-V LVTTL
out hex0[1]	Output	PIN_E15	7	B7_N0	PIN_E15	3.3-V LVTTL
out hex0[0]	Output	PIN_C14	7	B7_N0	PIN_C14	3.3-V LVTTL
in input_clock	Input	PIN_P11	3	B3_N0	PIN_P11	3.3-V LVTTL
in reset	Input	PIN_C10	7	B7_N0	PIN_C10	3.3-V LVTTL
<<new node>>						

Figure 13 - Pin Assignments for Part C

Finally, the Fmax and total logic elements for the circuit are provided below:

Total logic elements	60 / 49,760 (< 1 %)

Figure 13 -Compilation Report Showing Logic Elements for Part C

1	267.59 MHz	250.0 MHz	input_clock	limit due to minimum period restriction (

Figure 14 - Fmax for Part C

VALIDATION OF DATA

Functional Simulation Analysis

Below are the functional simulation analysis results for each part of the Lab, labelled accordingly. Note that these simulations were conducted in Modelsim-Altera.

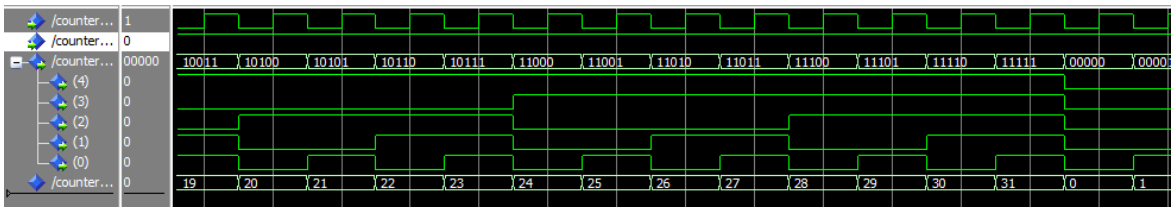

Figure 15 – Part A Functional Simulation (filled reset)

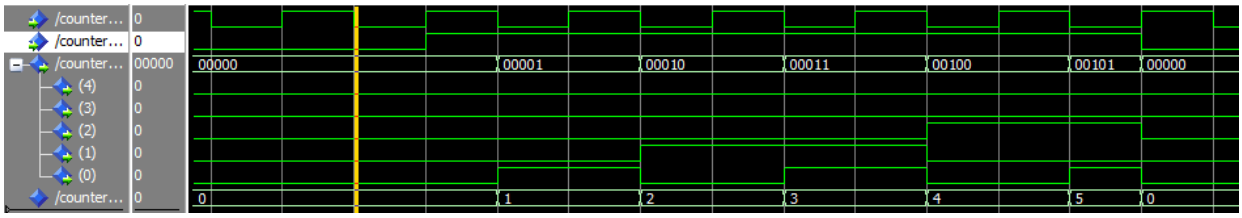

Figure 16 – Part A Functional Simulation (rst reset)

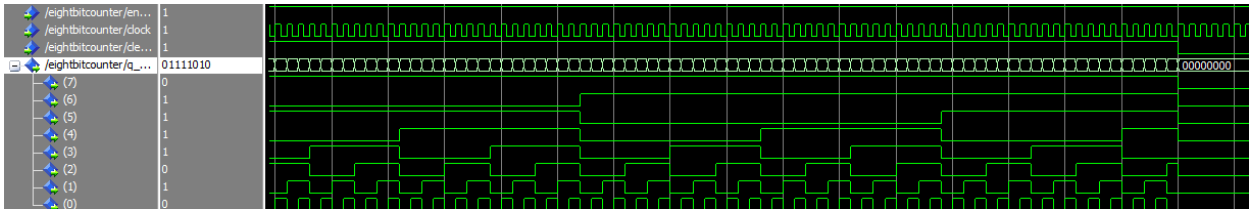

Figure 17 – Part B Functional Simulation (filled reset)

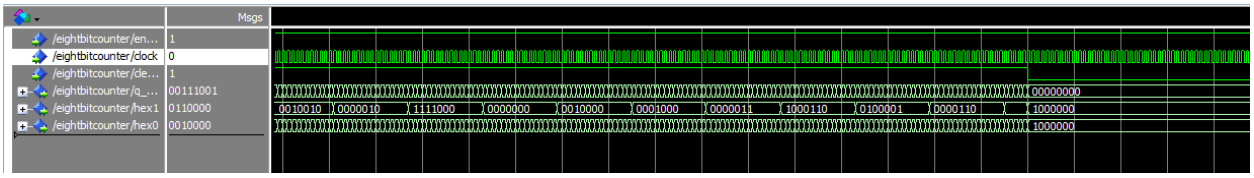

Figure 18 – Part B Functional Simulation (rst reset)

CONCLUSION

The experiment was conducted successfully and efficiently demonstrated three different methods of creating a counter using VHDL and the MAX10 DE-Lite development board. Understanding the behavior of the local 50MHz clock aided greatly in successfully completing part C of the experiment, while strong foundations in behavioral and structural coding assisted in the general completion of all parts of the experiment. Also, utilizing different types of flip-flops, such as the toggle flip-flop in part B, demonstrates their flexibility within embedded systems and how foundational logic and circuit components can work together to create a broader, wider use-case system. While Part A did not have a component which was ported onto the board, utilizing functional simulation waveforms allowed both for visualization of the data produced by each circuit, as well as served as a valuable troubleshooting tool for use in detecting false logic and repairing it. If this experiment were to be conducted again, it would prove beneficial to observe the behavior of other types of counters, such as combining the large 8-bit hexadecimal counter in part B with the time clock counter found in part C.

REFERENCES

[1] Professor Ashley Evans, *Logic Devices Programming Lab Manual*, 1st ed. Orlando, FL: Valencia College, 2025.