Computer Engineering Department	
Course Name: Digital design Lab 2	**Number: 10636391**
Lab Report Grading Sheet	

Instructor: Dr Ashraf Armoush	Experiment #: 3
Academic Year: 2020/2021	Experiment Name: Counters
Semester: Summer	

Students	
1-Ashraf Habromman	2-Ra'ed Khwayerh
Performed on: 22-24-29/6 - 1/7 / 2021	Submitted on: 8/7/2021

Report's Outcomes

ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %	ILO __ =( ) %

Evaluation Criterion	Grade	Points
Abstract answers of the questions: "What did you do? How did you do it? What did you find?"	0.5	
Introduction and Theory Sufficient, clear and complete statement of objectives. In addition to Presents sufficiently the theoretical basis.	1.5	
Apparatus/ Procedure Apparatus sufficiently described to enable another experimenter to identify the equipment needed to conduct the experiment.Procedure sufficiently described.	2	
Experimental Results and Discussion (In-Lab Worksheet) Crisp explanation of experimental results. Comparison of theoretical predictions to experimental results, including discussion of accuracy and error analysis in some cases.	4	
Conclusions and Recommendations Conclusions summarize the major findings from the experimental results with adequate specificity. Recommendations appropriate in light of conclusions. Correct grammar.	1	
Appearance Title page is complete, page numbers applied, content is well organized, correct spelling, fonts are consistent, good visual appeal.	1	
Total	10	

Counters

Introduction:

We learned in digital 2 about flips flops and we learned several types of flips flops as jk flip flop, T-flip flop and D-flip flop, and in **PART 1** of this experiment we want to use T-flip flop to build a 4-bit asynchronous ripple counter we used 4 T-flip flop after we build 4-bit asynchronous ripple counter and connected it, we added clk but we can’t connect clk direct because the system is operating on 100MHz frequency, so we used clock divider entity to divide the input clock (Y9) to generate an output clock with 1Hz frequency.

In **PART 2** we want to build a 4-bit up/down synchronous counter and it auto count but we want to build it behavioural not structural and replace enable with up/down to choose if we want increment or decrement count. As the first part, we can’t connect clk directly so we used a clock divider.

In **PART 3** we used the same counter in part 2 but we do not have to count automatically because we want to make it count with a push-button so we used a push button that existing on Zedboard, but we can’t connect it directly to the counter so we used debouncing module and it’s just a delay that ensures the correct state of the button. And we used a 10ns delay.

In **PART 4** In this part, we want to build a two-digit BCD counter that we learned theoretically and it counts from 00 to 99. Then we wanted to show it practically so we’ll install the bitstream on zedboard and display the result on a two-seven-segment display module connected to the kit.

Abstract:

We’ll learn:
1-Build a 4-bit asynchronous ripple counter by using a 4 components of T- flip flop then simulate it then connect clock divider and synthesis it then generate bitstream and install it on zedboard to show the results.
2-Build a 4-bit synchronous up/down auto counter then simulate it then connect the clock divider and synthesis it then generates bitstream and installs it on zedboard to show the results.
3-Use 4-bit synchronous up/down counter that we used in part 2 but no we won't it count auto so we’ll connect it to the push button and count when we press it.
4-Build BCD counter then displays the result on two 7-Segments.

Apparatus:

1- Vivado Design Suite HL WebPACK™ Edition.
2- ZedBoard.
3- 2 7-segment display.

Procedure:

Part 1: Asynchronous Ripple Counter:

1- T- flip flop:

T-flip flop component:

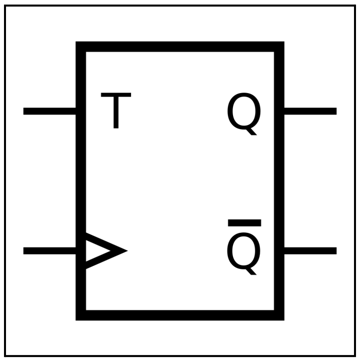

Fig1: T-flip flop

T- flip flop truth table:

T	Q (present state)	Q (next state)
0	Q	No change (Q)
1	Q	Toggle (Q')

T- flip flop code:

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

entity TFF is
   Port ( T : in STD_LOGIC;
          CLK : in STD_LOGIC;
          CLR : in STD_LOGIC;
          Q : out STD_LOGIC;
          QBAR : out STD_LOGIC);
end TFF;

architecture Behavioral of TFF is
          signal temp : std_logic := '0';
          begin
          Process (CLK, CLR)
```

```
            begin
                    if (CLR = '0') then                    -- Asynchronaus signal, which have high priority
                              temp <= '0' ;
                    elsif ( CLK' event and CLK = '1' ) then
                              if T='0' then
                                        temp <= temp;
                              elsif T='1' then
                                        temp<= not(temp);
                              end if ;
                    end if;
          end process;
          Q<=temp;
          QBAR<=not(temp) ;
end Behavioral;
```

2- 4-bit Asynchronous Counter:

4-bit Asynchronous Counter component:

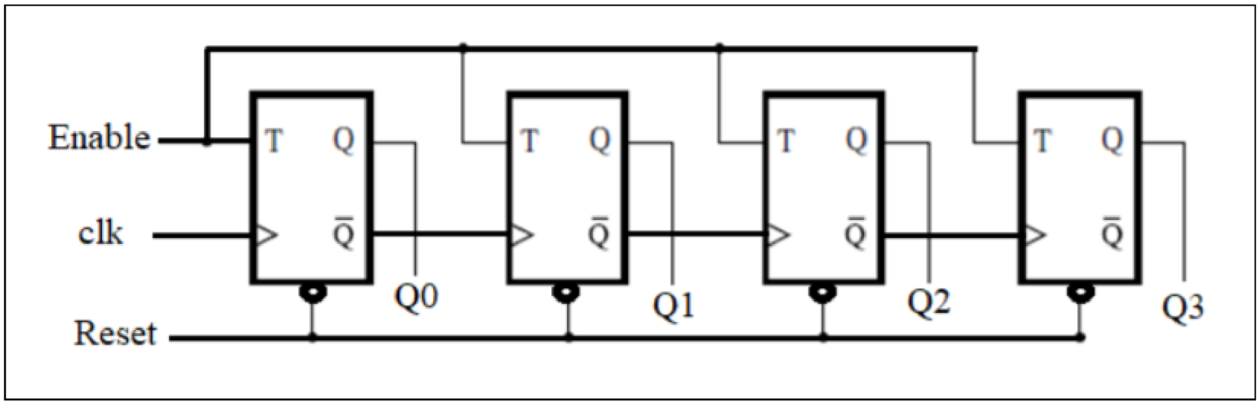

Fig2: 4-bit Asynchronous Counter component

4-bit Asynchronous Counter using T- flip flop code:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity counter is
   Port (
         CLK : in STD_LOGIC;
         ENABLE : in STD_LOGIC;
         RESET : in STD_LOGIC;
         Q : out STD_LOGIC_VECTOR (3 downto 0));
```

```
end counter;

architecture Behavioral of counter is
component TFF
    Port ( T : in STD_LOGIC;
           CLK : in STD_LOGIC;
           CLR : in STD_LOGIC;
           Q : out STD_LOGIC;
           QBAR : out STD_LOGIC);
end component;
SIGNAL SQ1,SQ2,SQ3:STD_LOGIC;
begin
    u1: TFF PORT MAP(T=>ENABLE,CLK=>CLK,CLR=>RESET,Q=>Q(0),QBAR=>SQ1);
    u2: TFF PORT MAP(T=>ENABLE,CLK=>SQ1,CLR=>RESET,Q=>Q(1),QBAR=>SQ2);
    u3: TFF PORT MAP(T=>ENABLE,CLK=>SQ2,CLR=>RESET,Q=>Q(2),QBAR=>SQ3);
    u4: TFF PORT MAP(T=>ENABLE,CLK=>SQ3,CLR=>RESET,Q=>Q(3));
end Behavioral;
```

4-bit Asynchronous Counter simulation part:

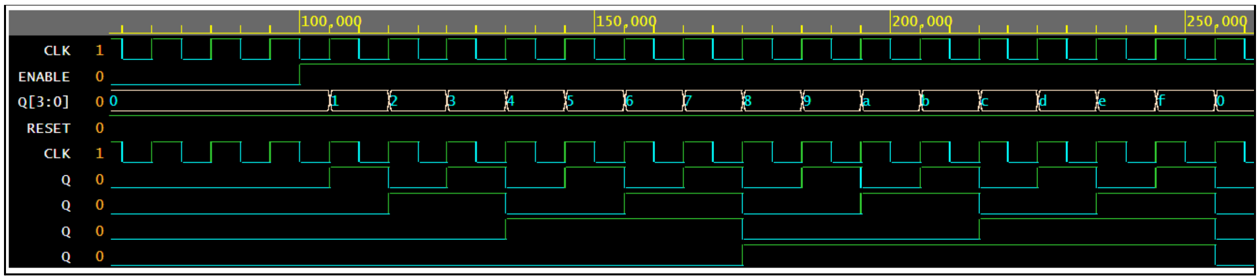

Fig3: 4-bit Asynchronous Counter signal

Testbench code part:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


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

architecture Behavioral of counter_tb is
component counter
 Port (
            CLK : in STD_LOGIC;
           ENABLE : in STD_LOGIC;
```

```
         RESET : in STD_LOGIC;
         Q : out STD_LOGIC_VECTOR (3 downto 0)
         );
end component;
signal CLK,ENABLE,RESET : std_logic ;
signal Q : STD_LOGIC_VECTOR (3 downto 0);
constant clk_period : time :=10 ns;
begin
         uut : counter port map(CLK=>CLK,RESET=>RESET,ENABLE=>ENABLE,Q=>Q);

         clk_generation : process
         begin
                  clk<='0';
                  wait for clk_period/2;
                  clk<='1';
                  wait for clk_period/2;
         end process;

         stimulus : process
         begin
                  RESET<='0';
                  ENABLE<='0';
                  wait for clk_period * 5;

                  RESET<='1';

                  ENABLE<='0';
                  wait for clk_period * 5;

                  ENABLE<='1';
                  wait for clk_period * 50;

                  ENABLE<='0';
                  wait for clk_period * 5;

                  RESET<='0';
                  wait for clk_period * 5;
                  wait;
         end process;
end Behavioral;
```

3- Clock Divider:

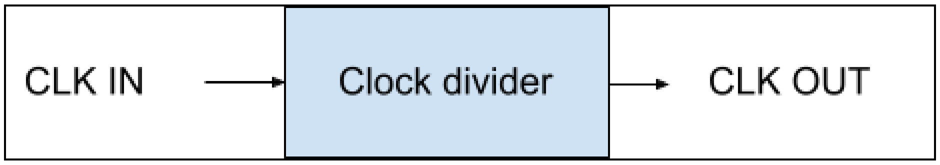

Fig4: Clock divider

The clock generator in the zedboard generate a clock cycle with a frequency 100MHz and the frequency wanted equals 1Hz, so the clock divider used to take the clock from the board and generate a clock with a frequency equals 1 Hz.

So, 100 MHz = X * 1Hz => X = 100 * 10^6 / 1 = 100,000,000 for 1 clock cycle, 50,000,000 as high level and 50,000,000 as low level.

Clock divider code:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity clock_divider is
   Port ( CLK_IN : in STD_LOGIC;
        CLK_OUT : out STD_LOGIC);
end clock_divider;

architecture Behavioral of clock_divider is
signal count: integer := 1;
signal tmp : std_logic :='0';
begin
        process(CLK_IN)
        begin
                if(CLK_IN'event and CLK_IN = '1') then
                        count <= count+1;
                        if (count = 50000000) then
                                tmp <= not(tmp);
                                count <= 1;
                        end if;
                end if;
                CLK_OUT <= tmp;
        end process;
end Behavioral;
```

4- Top-level entity of asynchronous ripple counter

The component of the top-level entity of asynchronous ripple counter:

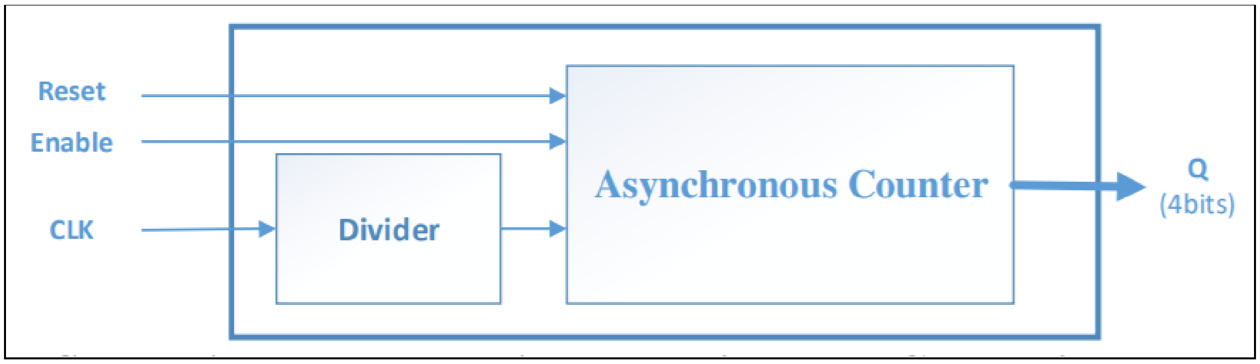

Fig5: Top-level entity of asynchronous ripple counter

The top-level entity of asynchronous ripple counter code:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity toplevelcounter is
    Port ( RESET : in STD_LOGIC;
           ENABLE : in STD_LOGIC;
           CLK : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end toplevelcounter;

architecture Behavioral of toplevelcounter is
component counter
   Port (
           CLK : in STD_LOGIC;
           ENABLE : in STD_LOGIC;
           RESET : in STD_LOGIC;
           Q : out STD_LOGIC_VECTOR (3 downto 0));
end component ;

component clock_divider
    Port ( CLK_IN : in STD_LOGIC;
           CLK_OUT : out STD_LOGIC);
```

```
end component;

signal clkod : std_logic;
Begin

u1: clock_divider port map (CLK_IN=>CLK,CLK_OUT=>clkod);
u2: counter port map(CLK=>clkod,RESET=>RESET,ENABLE=>ENABLE,Q=>Q);

end Behavioral;
```

5- Synthesis, Implementation, and Bitstream Generation for the top-level entity of the counter:

Connect each pin in the top-level entity with switches and LEDs on the board. Then do Synthesis and generate a bitstream.

Asynchronous ripple counter constraint file:

```
set_property IOSTANDARD LVCMOS33 [get_ports {Q[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Q[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Q[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Q[0]}]
set_property PACKAGE_PIN U21 [get_ports {Q[3]}]
set_property PACKAGE_PIN U22 [get_ports {Q[2]}]
set_property PACKAGE_PIN T21 [get_ports {Q[1]}]
set_property PACKAGE_PIN T22 [get_ports {Q[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property IOSTANDARD LVCMOS18 [get_ports ENABLE]
set_property IOSTANDARD LVCMOS18 [get_ports RESET]
set_property PACKAGE_PIN Y9 [get_ports CLK]
set_property PACKAGE_PIN G22 [get_ports ENABLE]
set_property PACKAGE_PIN F22 [get_ports RESET]
```

Part 2: AutoUpDown Synchronous Counter:

In this part of the experiment, the implementation was using behavioural architecture which is easier than structural in this case.

AutoUpDown synchronous counter component:

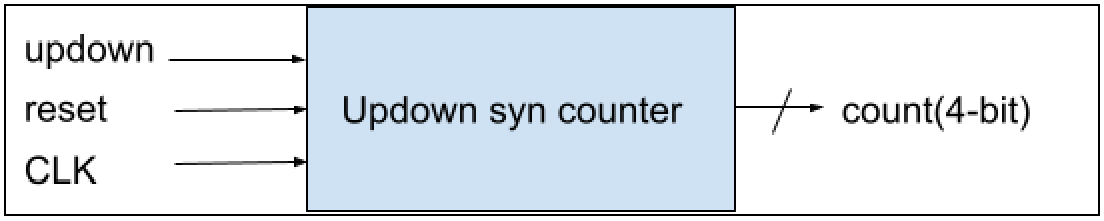

Fig6: AutoUpDown synchronous counter component:

AutoUpDown synchronous counter code:

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

entity updowncounter is
   Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          updown : in STD_LOGIC;
          count : out STD_LOGIC_VECTOR (3 downto 0));
end updowncounter;

architecture Behavioral of updowncounter is
signal tmp : std_logic_vector (3 downto 0):= "0000";
begin
process (clk,reset)
         begin
         if(reset='0') then
                  tmp<="0000";
         elsif (clk' event and clk='1') then
                  if(updown ='1') then
                           tmp <= tmp + '1';
                  else
                           tmp <= tmp - '1';
                  end if;
         end if;
         Count <= tmp;
end process;
end Behavioral;
```

AutoUpDown synchronous counter simulation:

Testbench code part:

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

entity updowncounter_tb is
end updowncounter_tb;

architecture Behavioral of updowncounter_tb is

component updowncounter is
    Port ( clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           updown : in STD_LOGIC;
           count : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal clk,Reset,UpDown:STD_LOGIC;
signal count : STD_LOGIC_VECTOR (3 downto 0);
constant clk_period : time :=10 ns;
begin

uut: updowncounter port map (clk=> clk, Reset=>Reset , UpDown=> UpDown, count=>count);


        clk_generation : process
        begin
                clk<='0';
                wait for clk_period/2;
                clk<='1';
                wait for clk_period/2;
        end process;

        stimulus : process
        begin
                reset<='0';
                updown<='0';
                wait for clk_period * 5;

                reset<='1';
                updown<='1'; -- increment
                wait for clk_period * 18;
                reset<='0';
                wait for clk_period * 5;

                reset<='1';
                updown<='0';     -- decrement
                wait for clk_period * 18;

                reset<='0';
                Wait;
        end process;

end Behavioral;
```

Signals:

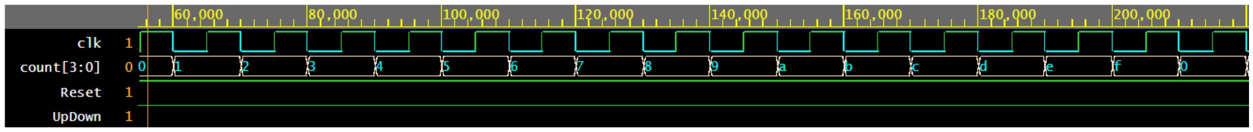

Fig7: AutoUpDown synchronous counter - UP part

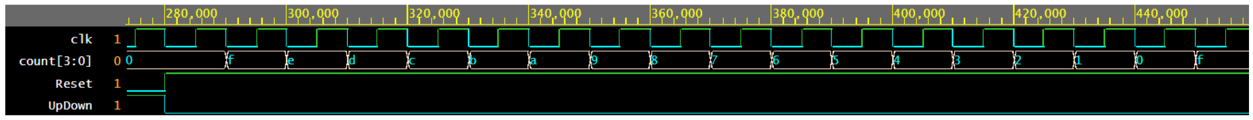

Fig7: AutoUpDown synchronous counter - DOWN part

Top-level entity of synchronous auto counter:

Because the frequency of the board is very high and we have to notice the counter we use a clock divider to generate a clock that has a frequency equals to 1Hz.

Note that the clock divider used here is the same clock divider that we used in the previous part.

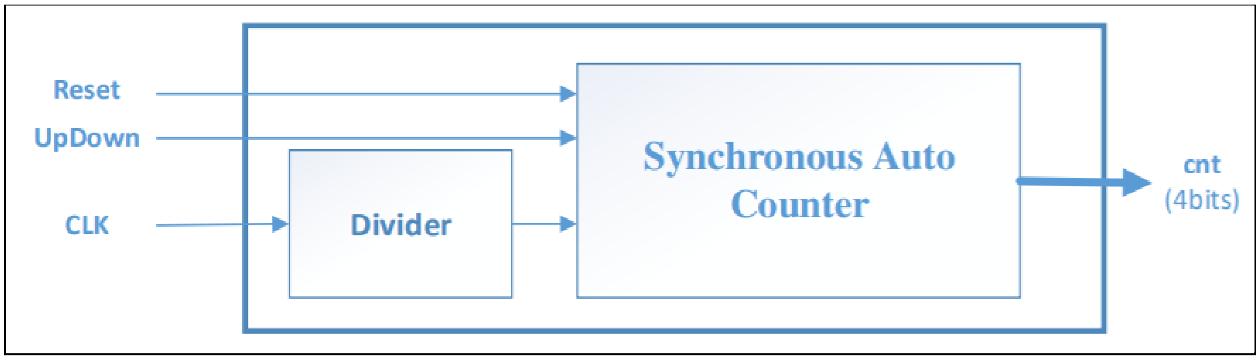

Fig8: Top-level entity of synchronous auto counter

Top-level of synchronous auto counter code:

```
--------------------------------------------------------------------------------
```

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

entity toplevel is
   Port ( RESET : in STD_LOGIC;
          UpDown : in STD_LOGIC;
          CLK : in STD_LOGIC;
          COUNT : out STD_LOGIC_VECTOR (3 downto 0));
end toplevel;

architecture Behavioral of toplevel is
component updowncounter
   Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          updown : in STD_LOGIC;
          count : out STD_LOGIC_VECTOR (3 downto 0));
end component;
component clock_divider
   Port ( CLK_IN : in STD_LOGIC;
          CLK_OUT : out STD_LOGIC);
end component;
signal clkod : std_logic;
begin
           u1: clock_divider port map (CLK_IN=>CLK,CLK_OUT=>clkod);
           u2: updowncounter port map
           (clk=>clkod,reset=>RESET,updown=>UpDown,count=>COUNT);
end Behavioral;
```

AutoUpDown Synchronous Counter constraint file:

```
set_property PACKAGE_PIN F22 (get_ports RESET)
set_property PACKAGE_PIN G22 (get_ports UpDown)
set_property PACKAGE_PIN Y9 (get_ports CLK)
set_property PACKAGE_PIN T22 (get_ports COUNT(0))
set_property PACKAGE_PIN T21 (get_ports COUNT(1))
set_property PACKAGE_PIN U22 (get_ports COUNT(2))
set_property PACKAGE_PIN U21 (get_ports COUNT(3))

set_property IOSTANDARD LVCOMS18 (get_ports RESET)
set_property IOSTANDARD LVCOMS18 (get_ports UpDown)
set_property IOSTANDARD LVCOMS33 (get_ports CLK)
set_property IOSTANDARD LVCOMS33 (get_ports COUNT(0))
set_property IOSTANDARD LVCOMS33 (get_ports COUNT(1))
set_property IOSTANDARD LVCOMS33 (get_ports COUNT(2))
set_property IOSTANDARD LVCOMS33 (get_ports COUNT(3))

set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[3]}]
set_property PACKAGE_PIN U21 [get_ports {COUNT[3]}]
set_property PACKAGE_PIN U22 [get_ports {COUNT[2]}]
set_property PACKAGE_PIN T21 [get_ports {COUNT[1]}]
set_property PACKAGE_PIN T22 [get_ports {COUNT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property IOSTANDARD LVCMOS18 [get_ports RESET]
set_property IOSTANDARD LVCMOS18 [get_ports UpDown]
set_property PACKAGE_PIN Y9 [get_ports CLK]
set_property PACKAGE_PIN G22 [get_ports RESET]
set_property PACKAGE_PIN F22 [get_ports UpDown]
```

Part 3: Push Button UpDown Counter:

In this part, we asked to design an up-down counter with a push-button when that push-button clicked the counter will increment/decrement with respect to the up-down signal. But the push-button has a debouncing problem so we have to add an additional component to solve that problem.

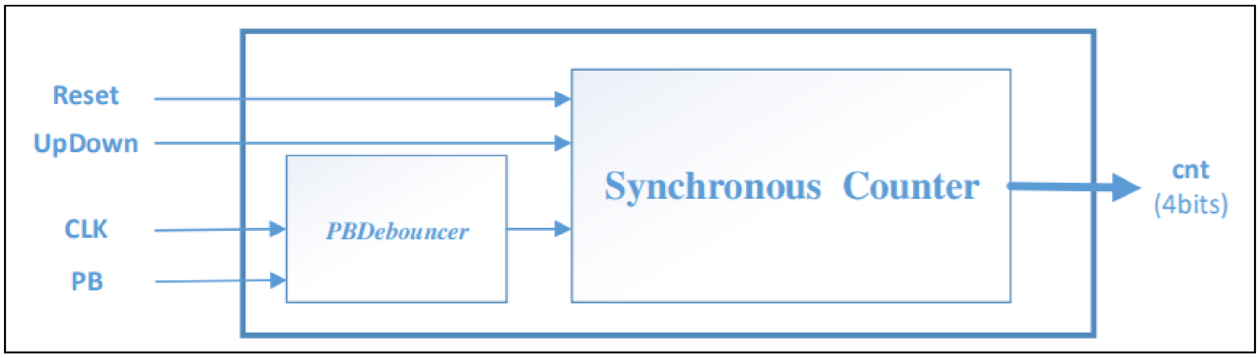

Fig9: Top-level of push-button UpDown counter:

The synchronous counter in this part of the experiment is the same one that used in the previous part.

PBDebouncer:

The debouncing module is just a delay that ensures the correct state of the button. In this part, the delay was 10 ms.

The frequency of pin Y9 on board equals to 100MHz, so to make delay equals 10 ms:

100MHz = 10 ns = 10 * 10^-9 s

10 ms = 10 * 10^-3 s

10 * 10^-9 * X = 10 * 10^-3

So X = 1000,000

X = 1000,000 means that we want to make a counter that counts million(10^6) raising edge of that clock which have a frequency of 100MHz.

PBDebouncer implementation code:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;

entity PBDebouncer is
   Port ( CLK : in STD_LOGIC;
          PB : in STD_LOGIC;
          PB_debounced : out STD_LOGIC);
end PBDebouncer;

architecture Behavioral of PBDebouncer is
begin
        process(CLK)
                variable Ton ,Pold : std_logic := '0';
                variable Timer: integer := 1;
                begin
                        if(CLK' event and CLK='1') then
                                if(Ton='0') then
                                        if(Pold/=PB)then
                                                Ton := '1';
                                                Timer := 1;
                                                Pold := not Pold;
                                        end if;
                                elsif (Ton='1') then
                                        if(Timer=1000000) then
                                                Ton:='0';
                                        else
                                                Timer:=Timer + 1;
                                        end if;
                                end if;
                        end if;
                PB_debounced<=Pold;
        end process;
end Behavioral;
```

The top-level entity of Push Button UpDown Counter:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity toplevel is
   Port ( reset : in STD_LOGIC;
          updown : in STD_LOGIC;
          clk : in STD_LOGIC;
          pb : in STD_LOGIC;
          cnt : out STD_LOGIC_VECTOR (3 downto 0));
end toplevel;

architecture Behavioral of toplevel is
component PBDebouncer
   Port ( CLK : in STD_LOGIC;
          PB : in STD_LOGIC;
          PB_debounced : out STD_LOGIC);
end component;

component updowncounter
   Port ( clk : in STD_LOGIC;
          reset : in STD_LOGIC;
          updown : in STD_LOGIC;
          count : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal pb_out:std_logic:='0';

begin
          u1: PBDebouncer port map(clk=>clk,pb=>pb,pb_debounced=>pb_out);
          u2: updowncounter port map (reset=>reset,updown=>updown,clk=>pb_out,count=>cnt);
end Behavioral;
```

Then do the synthesis and generate a bitstream file to program it on the board.

Push Button UpDown Counter constraint file :

```
set_property PACKAGE_PIN U21 [get_ports {COUNT[3]}]
set_property PACKAGE_PIN U22 [get_ports {COUNT[2]}]
set_property PACKAGE_PIN T21 [get_ports {COUNT[1]}]
set_property PACKAGE_PIN T22 [get_ports {COUNT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {COUNT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports CLK]
set_property PACKAGE_PIN Y9 [get_ports CLK]
set_property IOSTANDARD LVCMOS18 [get_ports PB]
set_property IOSTANDARD LVCMOS18 [get_ports RESET]
set_property IOSTANDARD LVCMOS18 [get_ports UpDown]
set_property PACKAGE_PIN P16 [get_ports PB]
set_property PACKAGE_PIN F22 [get_ports RESET]
set_property PACKAGE_PIN G22 [get_ports UpDown]
```

Part 4: Two-Digit BCD Counter:

In this part, a two-digit BCD counter has implemented and represented on 2 7-segment displays using many components as shown in the figure below:

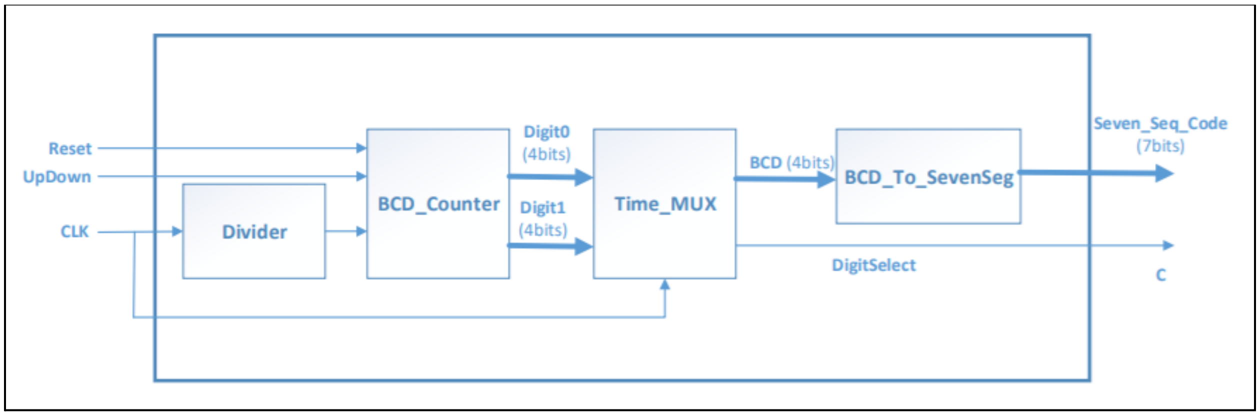

Fig10: Top-level of Two-Digit BCD Counter

The clock divider used in this part is the same one have used in the previous parts.

BCD Counter component:

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

entity BCD_Counter is
```

```
    Port ( clk : in STD_LOGIC;
           Reset : in STD_LOGIC;
           UpDown : in STD_LOGIC;
           Digit0 : out STD_LOGIC_VECTOR (3 downto 0);
           Digit1 : out STD_LOGIC_VECTOR (3 downto 0));
end BCD_Counter;

architecture Behavioral of BCD_Counter is
signal temp0,temp1:STD_LOGIC_VECTOR (3 downto 0):="0000";
begin
	process(clk,Reset)
	begin
		if(Reset='0') then
			temp0<="0000";
			temp1<="0000";
		elsif(clk' event and clk='1') then
			if(Updown='1') then
				if(temp0="1001") then
					temp0<="0000";
					if(temp1="1001")then
						temp1<="0000";
					else  temp1<=temp1+'1';
					end if;
				else temp0<=temp0+'1';
				end if;
			elsif (Updown='0') then
				if(temp0="0000") then
					temp0<="1001";
					if(temp1="0000")then
						temp1<="1001";
					else  temp1<=temp1-'1';
					end if;
				else temp0<=temp0-'1';
				end if;
			end if;
		end if;
	end process;
	Digit0<=temp0;
	Digit1<=temp1;
end Behavioral;
```

Simulation part of BCD counter:

Testbench code:

```
-- note that the testbench using is the same one used in the up-down counter, but the
-- component have changed

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity updowncounter_tb is

end updowncounter_tb;

architecture Behavioral of updowncounter_tb is

component BCD_Counter is
   Port ( clk : in STD_LOGIC;
          Reset : in STD_LOGIC;
          UpDown : in STD_LOGIC;
          Digit0 : out STD_LOGIC_VECTOR (3 downto 0);
          Digit1 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

signal clk,Reset,UpDown:STD_LOGIC;
signal Digit0,Digit1 : STD_LOGIC_VECTOR (3 downto 0);
constant clk_period : time :=10 ns;
begin

uut: BCD_Counter port map (clk=> clk, Reset=>Reset , UpDown=> UpDown, Digit0=>
Digit0,Digit1=>Digit1);


clk_generation : process
begin
          clk<='0';
          wait for clk_period/2;
          clk<='1';
          wait for clk_period/2;
end process;

stimulus : process
begin
          reset<='0';
          updown<='0';
          wait for clk_period * 5;
```

```
        reset<='1';
        updown<='1'; -- increment
        wait for clk_period * 100;

        reset<='0';
        wait for clk_period * 5;

        reset<='1';
        updown<='0';    -- decrement
        wait for clk_period * 100;

        reset<='0';
        Wait;
end process;

end Behavioral;
```

Time multiplexer:

This time mux has implemented to out one of the two inputs every 10 ms. In addition to out a select line which indicated which one of the inputs is outed.

As push-button debouncer part :

The frequency of pin Y9 on board equals to 100MHz, so to make delay equals 10 ms:

```
    100MHz = 10 ns = 10 * 10^-9 s
                 10 ms = 10 * 10^-3 s
                 10 * 10^-9 * X = 10 * 10^-3
                  So X = 1000,000
```

Time mux implementation:

```
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_unsigned.ALL;


entity time_mux is
   Port ( clk: in STD_LOGIC;
        Digit0 : in STD_LOGIC_VECTOR (3 downto 0);
        Digit1 : in STD_LOGIC_VECTOR (3 downto 0);
        BCD_Value : out STD_LOGIC_VECTOR (3 downto 0);
        DigSelect : out STD_LOGIC);
end time_mux;
```

```
architecture Behavioral of time_mux is

signal counter: integer:=1;
signal temp_digit: STD_LOGIC_VECTOR (3 downto 0);
signal flag : std_logic := '0';
begin
        process(clk)
        begin
                if(clk'event and clk='1')then
                        counter <= counter +1;
                        if(counter = 1000000)then
                                counter <= 0;
                                if(flag = '0')then
                                        temp_digit <= Digit1;
                                else temp_digit <= Digit0;
                                end if;
                                flag <= not flag;
                        end if;
                end if;
                BCD_Value <= temp_digit;
                DigSelect <= flag;
        end process;
end Behavioral;
```

BCD to seven-segment display code:

```
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity BCD_To_SevenSeg is
   Port ( BCD : in STD_LOGIC_VECTOR (3 downto 0);
        AA : out STD_LOGIC;
        AB : out STD_LOGIC;
        AC : out STD_LOGIC;
        AD : out STD_LOGIC;
        AE : out STD_LOGIC;
        AF : out STD_LOGIC;
        AG : out STD_LOGIC);
end BCD_To_SevenSeg;
```

```vhdl
architecture Behavioral of BCD_To_SevenSeg is
signal seven_temp: std_logic_VECTOR(6 downto 0);
begin
        process(BCD)
        begin
                case BCD is
                        when "0000"=>                           -- 0
                                seven_temp <="1111110";
                        when "0001"=>                           -- 1
                                seven_temp <="0110000";
                        when "0010"=>                           -- 2
                                seven_temp <="1101101";
                        when "0011"=>                           -- 3
                                seven_temp <="1111001";
                        when "0100"=>                           -- 4
                                seven_temp <="0110011";
                        when "0101"=>                           -- 5
                                seven_temp <="1011011";
                        when "0110"=>                           -- 6
                                seven_temp <="1011111";
                        when "0111"=>                           -- 7
                                seven_temp <="1110001";
                        when "1000"=>                           -- 8
                                seven_temp <="1111111";
                        when "1001"=>                           -- 9
                                seven_temp <="1110011";
                        when others =>
                                seven_temp <="0000000";
                end case;

        end process;

AA <= seven_temp(6);
AB <= seven_temp(5);
AC <= seven_temp(4);
AD <= seven_temp(3);
AE <= seven_temp(2);
AF <= seven_temp(1);
AG <= seven_temp(0);

end Behavioral;
```

The top-level entity of up-down BCD-counter:

```
--------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity top_level_entity is
   Port ( reset : in STD_LOGIC;
          updown : in STD_LOGIC;
          clk : in STD_LOGIC;
          seven_seg_code : out STD_LOGIC_VECTOR (6 downto 0);
          c : out STD_LOGIC);
end top_level_entity;

architecture Behavioral of top_level_entity is

component clock_divider
 Port ( CLK_IN : in STD_LOGIC;
          CLK_OUT : out STD_LOGIC);
end component;

component BCD_Counter
   Port ( clk : in STD_LOGIC;
          Reset : in STD_LOGIC;
          UpDown : in STD_LOGIC;
          Digit0 : out STD_LOGIC_VECTOR (3 downto 0);
          Digit1 : out STD_LOGIC_VECTOR (3 downto 0));
end component;

component time_mux
   Port ( clk : in STD_LOGIC;
          Digit0 : in STD_LOGIC_VECTOR (3 downto 0);
          Digit1 : in STD_LOGIC_VECTOR (3 downto 0);
          BCD_Value : out STD_LOGIC_VECTOR (3 downto 0);
          DigSelect : out STD_LOGIC);
end component;

component BCD_To_SevenSeg
   Port ( BCD : in STD_LOGIC_VECTOR (3 downto 0);
          AA : out STD_LOGIC;
          AB : out STD_LOGIC;
          AC : out STD_LOGIC;
```

```
        AD : out STD_LOGIC;
        AE : out STD_LOGIC;
        AF : out STD_LOGIC;
        AG : out STD_LOGIC);
end component;

signal sclk_out: std_logic;
signal sDigit0,sDigit1,sBCD : STD_LOGIC_VECTOR (3 downto 0):= "0000";

begin
        u1: clock_divider port map(CLK_IN=>clk,CLK_OUT=>sclk_out);

        u2: BCD_Counter port map(reset=>reset, updown=>updown, clk=>sclk_out,
        Digit0=>sDigit0,Digit1=>sDigit1);

        u3: time_mux port
        map(clk=>clk,Digit0=>sDigit0,Digit1=>sDigit1,BCD_Value=>sBCD,DigSelect=>c);

        u4: BCD_To_SevenSeg port map(BCD=>sBCD,AA=>seven_seg_code(6),
        AB=>seven_seg_code(5), AC=>seven_seg_code(4),
         AD=>seven_seg_code(3), AE=>seven_seg_code(2), AF=>seven_seg_code(1),
        AG=>seven_seg_code(0));

end Behavioral;
```

Two-Digit BCD Counter constraint file:

```
set_property IOSTANDARD LVCMOS33 [get_ports {seven_seg_code[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven_seg_code[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven_seg_code[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven_seg_code[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven_seg_code[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven_seg_code[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seven_seg_code[0]}]
set_property PACKAGE_PIN Y11 [get_ports {seven_seg_code[6]}]
set_property PACKAGE_PIN AA11 [get_ports {seven_seg_code[5]}]
set_property PACKAGE_PIN Y10 [get_ports {seven_seg_code[4]}]
set_property PACKAGE_PIN AA9 [get_ports {seven_seg_code[3]}]
set_property PACKAGE_PIN W12 [get_ports {seven_seg_code[2]}]
set_property PACKAGE_PIN W11 [get_ports {seven_seg_code[1]}]
set_property PACKAGE_PIN V10 [get_ports {seven_seg_code[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports c]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports reset]
set_property IOSTANDARD LVCMOS18 [get_ports updown]
set_property PACKAGE_PIN W8 [get_ports c]
set_property PACKAGE_PIN Y9 [get_ports clk]
set_property PACKAGE_PIN F22 [get_ports reset]
set_property PACKAGE_PIN G22 [get_ports updown]
```

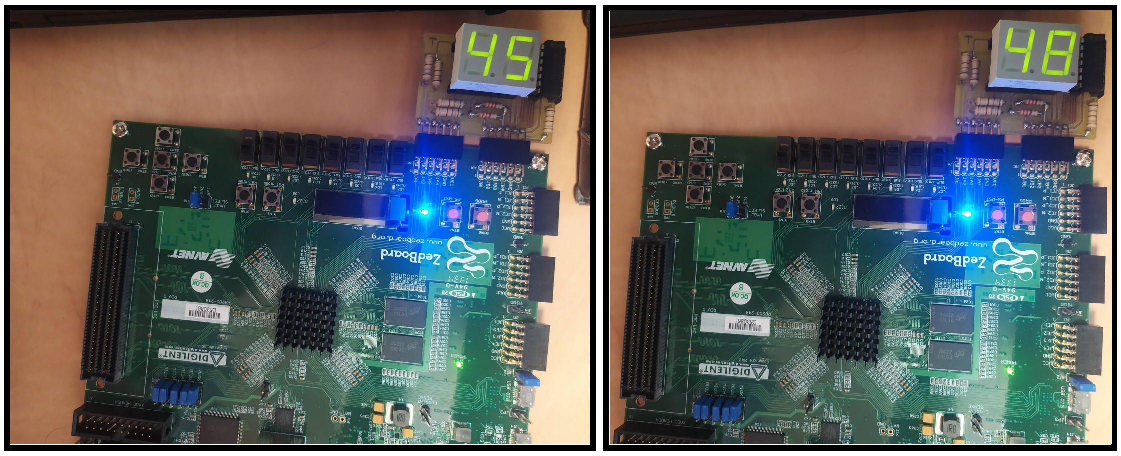

Fig11: Final result of BCD counter.

Conclusion:

In this experiment we learned how to build three kind of counters, first one is asynchronous counter that built with t-flip flop, second on is synchronous counter that we used in part 2,3 and we learned how to make it count auto or count by push button and this can increment or decrement count . The last one was the BCD counter that we displayed its result on two 7-segments. And we learned a new component (Clock divider) that we used in part 1,2,4 and it generates an output clock with 1Hz frequency.