An Najah National University
Faculty of Engineering
Computer Engineering Department
Digital Design 2 LAB -10636391-
Experiment 3
Dr. Sufyan Samara and Dr. Ashraf Armoush

Part 1: Asynchronous Ripple Counter

In this part, you will build a 4-bit Asynchronous ripple counter using T flip-flops.

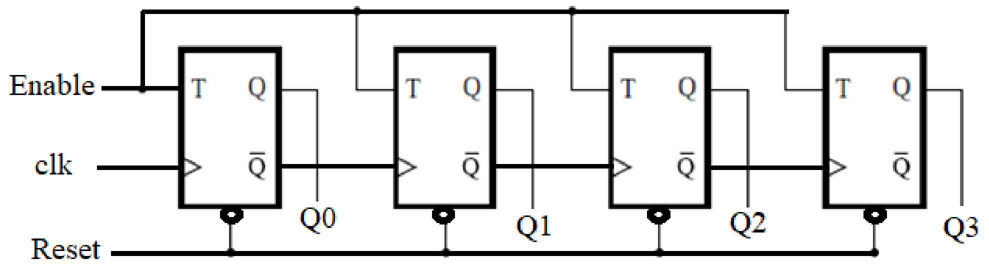
1. T Flip-Flop

Write a Verilog code to implement a T flip-flop with following pins:

- **T** : Synchronous Input (0: No change , 1: Toggle)
- **CLK**: positive edge trigger clock.
- **Clear**: Asynchronous active low clear.
- **Q** : output
- **QBAR** : output.

2. 4-bit Asynchronous Counter

- Design a 4-bit asynchronous ripple counter using the previous T flip-flop as a submodule.
- Your counter should include the flowing ports:
  - **CLK**
  - **Enable**: Active High Enable
  - **Reset**: Active Low Reset
  - **Q**: (4-bit output)



An Najah National University
Faculty of Engineering
Computer Engineering Department
Digital Design 2 LAB -10636391-
Experiment 3
Dr. Sufyan Samara and Dr. Ashraf Armoush

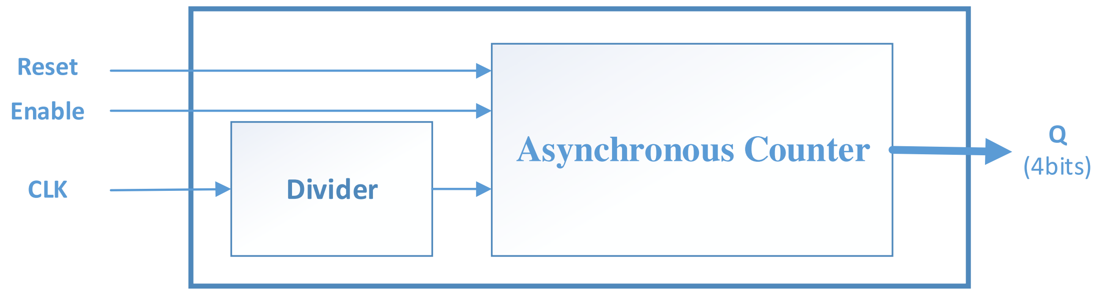
3. Clock Divider

Because the system is operating on 100MHz frequency, complete the following module to divide the input clock (Y9) to generate an output clock with 1Hz frequency.

```
module ClkDivider(
        input CLK_IN,
        output  CLK_OUT
);
```

4. Top-Level Entity:

Write the main Verilog file to implement the first part. Your file should include two submodules from the clock divider and the asynchronous counter as shown in the figure



5. Synthesis, Implementation, and Bitstream Generation:

Add a constraint file to assign the input and output ports as follows:

- **Reset**: SW0
- **Enable**: SW1
- **CLK**: Y9
- **Q** (4bits): LD3, LD2, LD1, LD0;

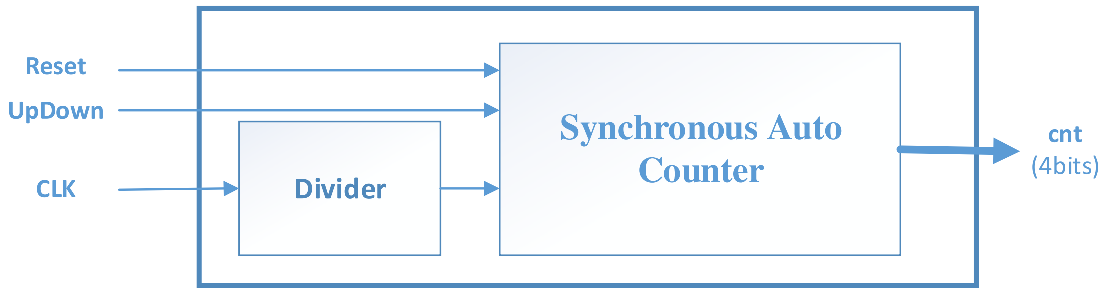
Inputs	**Package Pin**	**I/O Std**
SW0	F22	LVCMOS18
SW1	G22	LVCMOS18
CLK	Y9	LVCMOS33
Outputs	**Package Pin**	**I/O Std**
LD0	T22	LVCMOS33
LD1	T21	LVCMOS33
LD2	U22	LVCMOS33
LD3	U21	LVCMOS33

An Najah National University
Faculty of Engineering
Computer Engineering Department
Digital Design 2 LAB -10636391-
Experiment 3
Dr. Sufyan Samara and Dr. Ashraf Armoush

Part 2 : AutoUpDown Synchronous Counter

- In the AutoUpDownCounter.v, implement an automatic up-down counter.
- This is a 4-bit counter that increment/decrement automatically using the system clock.
- Because the system is operating on 100MHz frequency, we need to divide the clock so we can notice the change in LED's counting.

```
module AutoUpDownCounter (
        input CLK,
        input UpDown,
        input Reset,
       output [3:0] Q
  );
```



- Add a constraint file to assign the input and output ports as follows:

    - **Reset**: SW0
    - **UpDown**: SW1
    - **CLK**: Y9
    - **Q** (4bits): LD3, LD2, LD1, LD0;

Inputs	**Package Pin**	**I/O Std**
SW0	F22	LVCMOS18
SW1	G22	LVCMOS18
CLK	Y9	LVCMOS33
Outputs	**Package Pin**	**I/O Std**
LD0	T22	LVCMOS33
LD1	T21	LVCMOS33
LD2	U22	LVCMOS33
LD3	U21	LVCMOS33

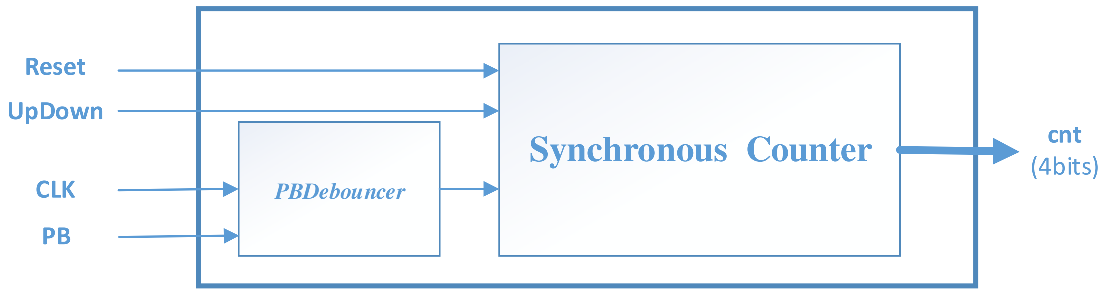
An Najah National University
Faculty of Engineering
Computer Engineering Department
Digital Design 2 LAB -10636391-
Experiment 3
Dr. Sufyan Samara and Dr. Ashraf Armoush

Part 3 : Push Button UpDown Counter

- The push buttons are to be debounced before used. In this section you are to design a 4-bit up-down counter that would increment/decrement manually using a push button.
- In the PBDebouncer.v, implement a module that would debounce a push button. The interface to this module is given below:

```
module PBDebouncer(
       input CLK,
       input Reset,
       input PB,    //the button state to be debounced
       output PB_debounced // the button state after debouncing
    );
```

- The debouncing module is just a delay that ensures the correct state of the button. Try a delay for 10 ms.
- In the ***PBUpDownCounter.v***, use the **`PBDebouncer`** module, which you implemented, to debounce the input from a push and use it to produce a manual event that increment/decrement a 4-bit synchronous counter, which is shown on the LEDs.



- The interface for the counter would look as following:

```
module PBUpDownCounter(
             input CLK,
             input Reset,
             input UpDown,
             input PB,
             output [3:0] cnt
       );
```

The ZedBoard provide 5 user push buttons on the PL-side of the FPGA. The pull-down connections provide a known default state, pushing each button connects to Vcco.

	Signal Name	Package Pin	I/O Std
Up Button	BTNU	T18	LVCMOS18
Right Button	BTNR	R18	LVCMOS18
Down Button	BTND	R16	LVCMOS18
Center Button	BTNC	P16	LVCMOS18
Left Button	BTNL	N15	LVCMOS18

An Najah National University
Faculty of Engineering
Computer Engineering Department
Digital Design 2 LAB -10636391-
Experiment 3
Dr. Sufyan Samara and Dr. Ashraf Armoush

Part 4 : Two-Digit BCD Counter

In this part, you will build a two-digit BCD counter that counts from 00 to 99. The result is to be displayed on a two-seven-segment display module connected to the kit. The circuit diagram for this module is shown in Fig1.

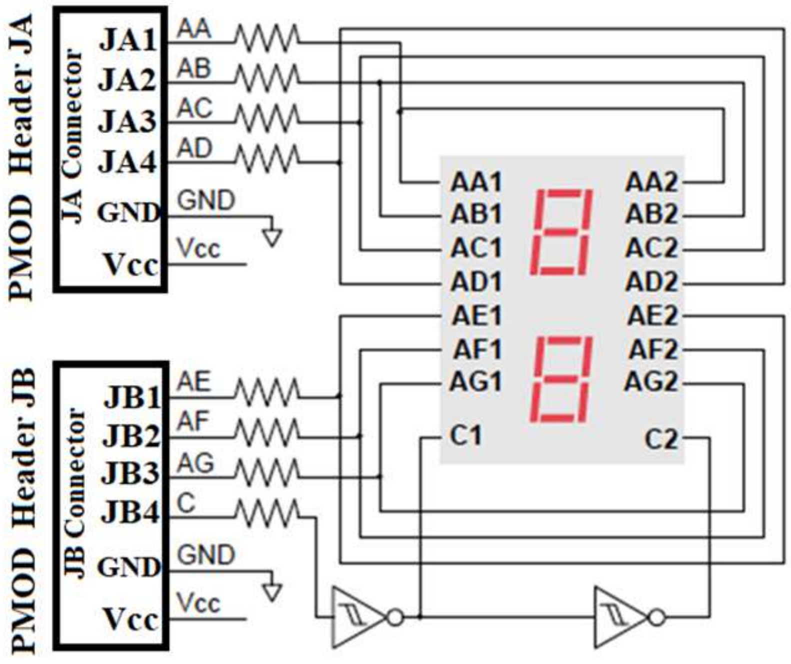
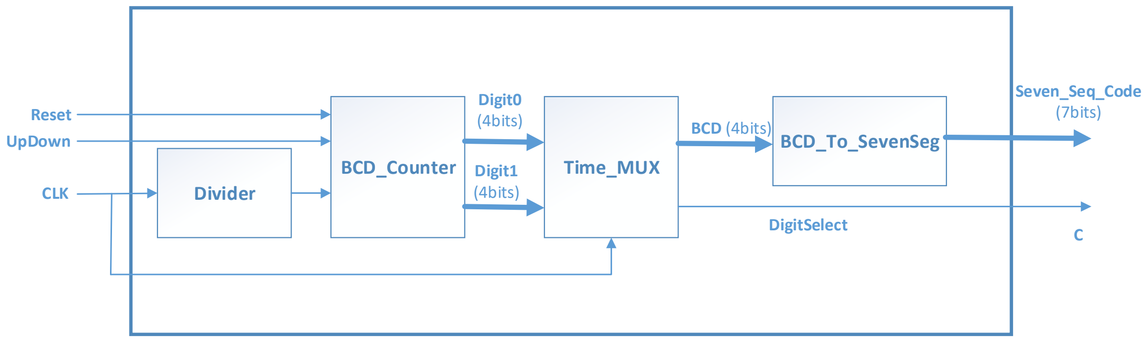


Fig1.

The seven segments in this module are common cathode. This means, to light a LED segment (e.g. AA) it should be pulled high.

Your design should include four submodules as shown in Fig2.



An Najah National University
Faculty of Engineering
Computer Engineering Department
Digital Design 2 LAB -10636391-
Experiment 3
Dr. Sufyan Samara and Dr. Ashraf Armoush

Fig2

1. Divider:

Because the system is operating on 100MHz frequency, this module is used to divide the input clock to generate an output clock with 1Hz frequency.

```
module ClkDivider(
     input CLK_IN,
     output  CLK_OUT
     );
```

2. BCD_Counter:

It is a two-digit (two-decade) BCD counter counts in decimal from 00 decimal to 99 decimal.

```
module BCD_Counter(
    input CLK,
    input Reset,
    input UpDown,
    output reg [3:0] Digit0,
    output reg [3:0] Digit1
    );
```

3. Time_MUX:

We cannot simultaneously display a digit on both 7-segment LED displays. Instead, by repeatedly and continuously display a digit on each display faster than the human eye can respond, both displays will appear to be illuminated at the same time. In this part, you will implement a multiplexer to select between the two digits (Tens and Ones) of the BCD_Counter. Each 7-segment LED display should be illuminated for **10 ms**. The output DigSelect signal is used to indicate the current active digit (0 and 1 for digit0 and digit1 respectively).

```
module Time_MUX(
    input CLK,
    input [3:0] Digit0,
    input [3:0] Digit1,
    output reg [3:0] BCD_Value,
    output reg DigSelect
    );
```

4. BCD_To_SevenSeg:

The BCD to 7 Segment Decoder converts 4 bit BCD number to a 7-bit code which can be displayed on a 7-segment display. The seven segments in this module are common cathode. Therefore, your output signals should be active high.
(Hint: you can use a ***case*** statement to implement this decoder)

```
module BCD_To_SevenSeg(
     input [3:0] BCD,
     output AA,
     output AB,
     output AC,
     output AD,
     output AE,
     output AF,
     output AG
    );
```

Note:
The seven-segment display connection diagram shows that the module pins can be connected to JA and JB connector.

Signal Name	Package Pin	I/O Std
JA1	Y11	LVCMOS33
JA2	AA11	LVCMOS33
JA3	Y10	LVCMOS33
JA4	AA9	LVCMOS33
JB1	W12	LVCMOS33
JB2	W11	LVCMOS33
JB3	V10	LVCMOS33
JB4	W8	LVCMOS33