Q1

Problem description

Implement a ***T flip-flop*** with ***reset*** function using the ***provided JK flip-flop***.

Input: ***T*** (input signal of flip-flop, the output is ***reversed if T is high*** while the clock is rising, and ***held if T is low***), ***clk*** (clock signal, ***rising edge sensitivity***), ***rst_n*** (reset signal, ***synchronous reset, active low***).

Output: ***Q*** (output signal of flip-flop), ***Qn*** (Output reversal signal).

All of the above ports have a bit width of ***1***.

NOTE: It's asked do the circuit design in structural manner in Q1, which means there shouldn't be "always" or "assign" in your submit code to OJ

Example code

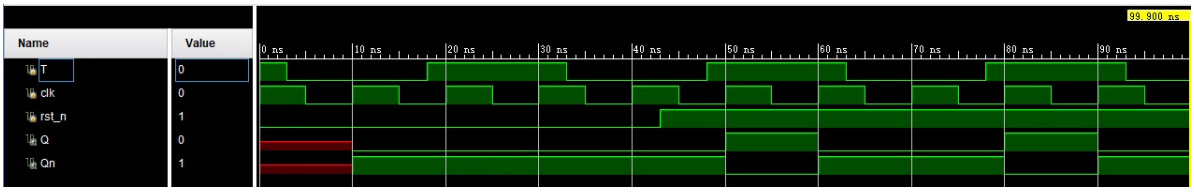
```
module T_FF_pos_rst_n_by_JKFF(
input T, clk, rst_n,
output  Q,Qn

    );
endmodule
```

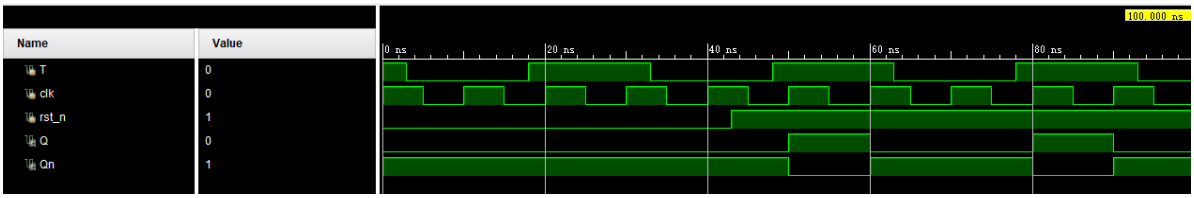
provided JK flip-flop

```
module JK_FF_Pos(
input J,K, clk,
output reg Q,
output Qn
    );
    assign Qn = ~Q;
    always@(posedge clk) begin
        case({J,K})
            2'b00: Q<=Q;
            2'b01: Q<=1'b0;
            2'b10: Q<=1'b1;
            2'b11: Q<=~Q;
        endcase
    end
endmodule
```

Tips: while run the simulation on the testbench to test the circuit, the expected waveform is like:

Due to the differences in simulation results caused by different design style, the following two simulation results are correct, oj will be compared from the position of the first rising edge of the clock (10ns)





Q2

Problem description

Build a ***testbench*** to simulate and test the reference circuits, and the modules being tested is named ***T_FF_pos_rst_n_by_JKFF***.

T_FF_pos_rst_n_by_JKFF have 3 input ports ***T, clk, rst_n*** and two output ports ***Q, Qn***. The bit widths of ports are ***all 1***.

T in the testbench connects to port T of T_FF_pos_rst_n_by_JKFF, ***initial value is 1*** and ***set to 0 after 3 ns*** then ***changes every 15 ns***;

clk in the testbench connects to port clk of T_FF_pos_rst_n_by_JKFF, ***initial value is 1*** and ***changes every 5 ns***;

rst_n in the testbench connects to port rst_n of T_FF_pos_rst_n_by_JKFF, ***initial value is 0*** and ***set to 1 after 43 ns***;

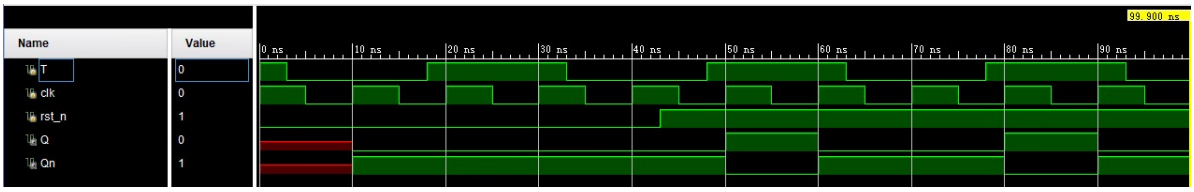
Q in the testbench connects to port Q of T_FF_pos_rst_n_by_JKFF;

Qn in the testbench connects to port Qn of T_FF_pos_rst_n_by_JKFF;

Note: Total simulation time is 100 ns so $finish is needed in your submission. Before submitting your code to Verilog-OJ, please make sure to use the monitor statement only once in initial block.

Example code

```
module tb_T_FF_pos_rst_n();
    reg T, clk, rst_n;
    wire Q, Qn;
    T_FF_pos_rst_n_by_JKFF tff(T, clk, rst_n, Q, Qn);

  initial begin
    $monitor ("%d %d %d %d %d", T, clk, rst_n, Q, Qn);

    end

endmodule
```



Q3

Problem description

Implement a ***2-state, asynchronous reset moor*** mode FSM.

Input: ***x*** (change state signal of FSM, ***when x is 1, the state remains unchanged, when x is 0, a state transition occurs***), ***clk*** (clock signal, ***rising edge sensitivity***), ***rst*** (reset signal, ***rising edge sensitivity*** and ***active high***).

Output: ***state*** (state of FSM with value ***2'b01*** or ***2'b10***), ***next_s*** (next state of FSM with value ***2'b01*** or ***2'b10***).

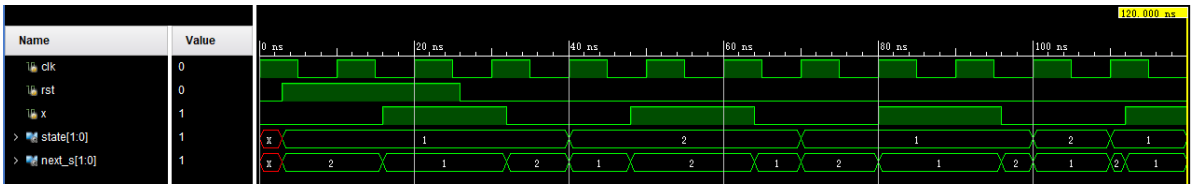
Ports ***x, clk***, and ***rst*** have ***1*** bit width, ***state*** and ***next_s*** have ***2*** bit widths.

NOTE: It's asked to do the design in 2-state manner, which means there should be an "always" statement block corresponding to sequential logic, an "always" statement block corresponding to combinatorial logic, with blocking assignments in the combinatorial logic statement block and non blocking assignments in the sequential logic statement block

Example code

```
module moor_s1s2_rst_asyn(
input clk, rst, x,
output reg [1:0]state, next_s
    );

endmodule
```

Tips: while run the simulation on the testbench to test the circuit, the expected waveform is like:



Q4

Problem description

Implement a ***2-state, synchronous reset moor*** mode FSM.

Input: ***x*** (change state signal of FSM, ***when x is 1, the state remains unchanged, when x is 0, a state transition occurs***), ***clk*** (clock signal, ***rising edge sensitivity***), ***rst*** (reset signal, ***active high***).

Output: ***state*** (state of FSM with value ***2'b01*** or ***2'b10***), ***next_s*** (next state of FSM with value ***2'b01*** or ***2'b10***).

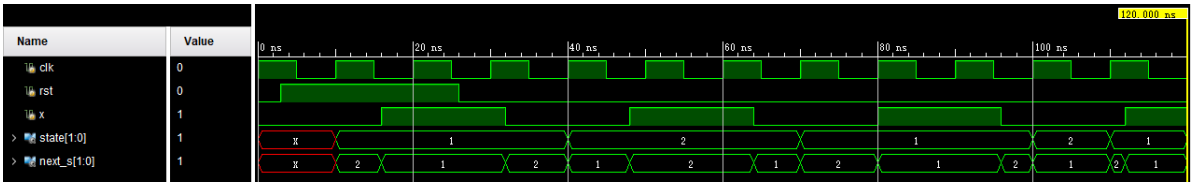
Ports ***x, clk***, and ***rst*** have ***1*** bit width, ***state*** and ***next_s*** have ***2*** bit widths.

Example code

```
module moor_s1s2_rst_syn(
input clk, rst, x,
output reg [1:0]state, next_s
    );

endmodule
```

NOTE: It's asked to using blocking assignments in the combinatorial logic "always" statement block and non blocking assignments in the sequential logic "always" statement block

Tips: while run the simulation on the testbench to test the circuit, the expected waveform is like:



Q5

Problem description

Build a ***testbench*** to simulate and test the reference circuits, and the modules being tested is named ***moor_s1s2_rst_asyn*** and ***moor_s1s2_rst_syn***. Your testbench should test both modules at the same time.

moor_s1s2_rst_asyn and moor_s1s2_rst_syn have 3 input ports ***x, clk, rst*** and two output ports ***state, next_s***. The bit widths of ports ***x, clk, rst*** are ***1***, ***state, next_s*** are ***2***.

x in the testbench connects to port x of moor_s1s2_rst_asyn and moor_s1s2_rst_syn, ***initial value is 0*** and ***changes every 16 ns***;

clk in the testbench connects to port clk of moor_s1s2_rst_asyn and moor_s1s2_rst_syn, ***initial value is 1*** and ***changes every 5 ns***;

rst in the testbench connects to port rst of moor_s1s2_rst_asyn and moor_s1s2_rst_syn, ***initial value is 0*** and ***set to 1 when time == 3ns*** and then ***set to 0 when time == 26ns***;

state in the testbench connects to port state of moor_s1s2_rst_syn;

next_s in the testbench connects to port next_s of moor_s1s2_rst_syn;

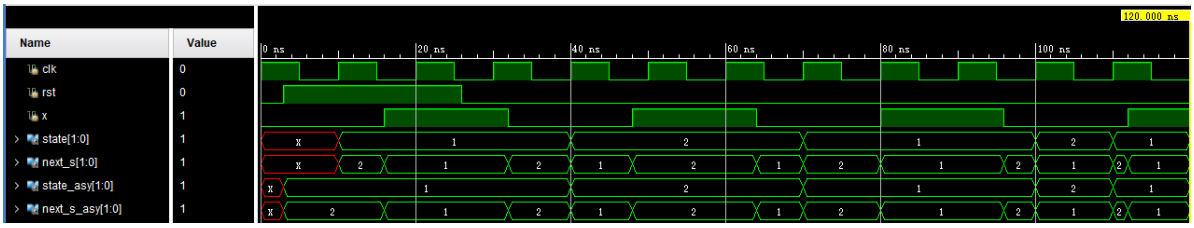
state_asy in the testbench connects to port state of moor_s1s2_rst_asyn;

next_s_asy in the testbench connects to port next_s of moor_s1s2_rst_asyn;

Example code

```
module tb_moor_s1s2_rst_syn_asyn(  );
reg clk, rst, x;
wire [1:0] state, next_s,  state_asy, next_s_asy;
```

```
moor_s1s2_rst_syn  dut_sy(clk, rst, x,state, next_s);
moor_s1s2_rst_asyn  dut_asyn(clk, rst, x,state_asy, next_s_asy);

initial begin
    $monitor ("%d %d %d %d %d %d %d", x, clk, rst, state, next_s,  state_asy,
next_s_asy);

end


endmodule
```



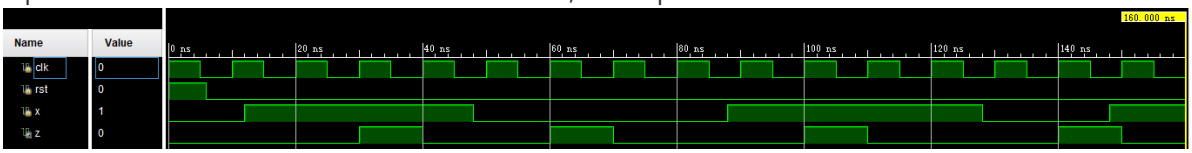
#**Q6:**

Problem description

Utilize Verilog to design a module named "*check_dif*" that comprises 3 inputs and 1 output. The inputs are **clk**, **rst**, and **x**, while the output is denoted as **z**. All inputs and outputs are 1 bit wide. The purpose of this module is to examine whether there is a difference in the input x between two consecutive positive edges of clk. If x differs between two consecutive positive edges of clk, then z ought to be set to 1 at the subsequent positive edge of clk. Conversely, if x remains the same, z should be set to 0 at the next positive edge of clk. The circuit should utilize synchronous reset in **rst**, and the reset signal should be active high. In your code, make sure to set the **state** to its initial value and set **z** to 1'b0 when a reset occurs.

NOTE: It's asked to using blocking assignments in the combinatorial logic "always" statement block and non blocking assignments in the sequential logic "always" statement block

Tips:While run the simulation on the testbench, the expected waveform is like.



#**Q7:**

##problem description

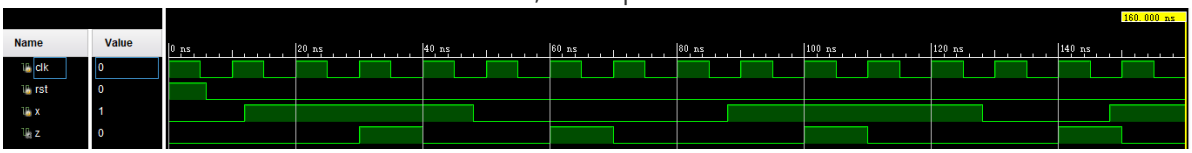
Construct the testbench for "check_dif" mentioned in Q6. The testbench should be named "**tb_check_dif**" and have 3 inputs and 1 output. The 3 inputs are **clk**, **rst**, and **x**, while the output is **z**. All inputs and outputs have a width of 1 bit. The testbench should be capable of generating a clock signal with a period of **10ns**. Initially, the value of clk should be set to 1'b1. The initial value of **rst** should be 1'b1 and transition to 1'b0 after 6ns, maintaining this value until the end of the simulation. The initial value of **x** should be 1'b0 and subsequently toggled at 12ns, 48ns, 88ns, 128ns, and 148ns. The testbench should end after 160ns.

Tips:

1.use **$monitor** as follow:

```
$monitor ("%d %d %d %d", clk, rst, x, z);
```

2.While run the simulation on the testbench, the expected waveform is like.



#**Q8.**

Problem description

Use verilog to design a module named `clock_div_7` that has **2** inputs and **1** output. The module works as a frequency divider that divides the input signal `clk` by **7**.

Inputs: `clk`, `rst_n`. Outputs: `clk_out`. All are **1** bit wide.

`clk`: input clock signal.
`rst_n`: reset signal, **asynchronous reset**, active low.
`clk_out`: output clock signal, divided by **7**.

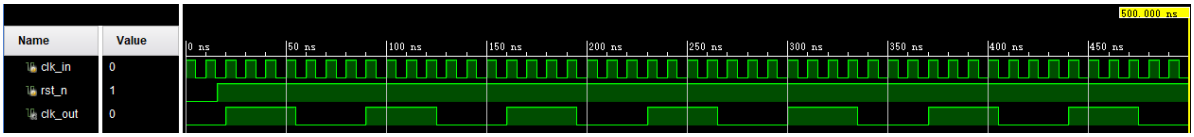
NOTE: It's asked to using blocking assignments in the combinatorial logic "always" statement block and non blocking assignments in the sequential logic "always" statement block

Note: The duty cycle of the input clock `clk` is 50%, and the output clock `clk_out` after division also has a duty cycle of 50%.

Example code:

```
module clock_div_7(
input clk, rst_n,
output  clk_out
);
```

Tips: The testbench should be like (please refers to **Q9**):



#**Q9.**

##Problem description
Build the testbench for `clock_div_7` in **q8**.

The testbench has 2 inputs and 1 output. The module being tested is named `clock_div_7`.

`clock_div_7`: The inputs are `clk` and `rst_n`. The output is `clk_out`.

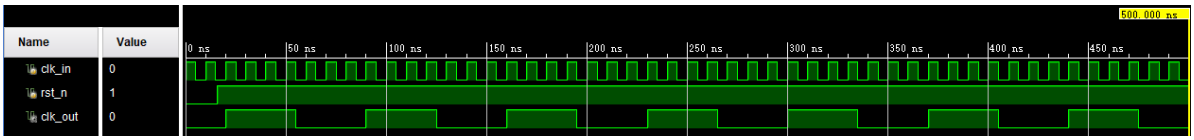
`clk` in the testbench connects to port `clk` of `clock_div_7`. The clock period should be **10ns**, and the value should be initialized with **1**.

`rst_n` in the testbench connects to port `rst_n` of `clock_div_7`. The value should be initialized with **0**. After **16ns**, set `rst_n` to **1**.

`clk_out` in the testbench connects to port `clk_out` of `clock_div_7`.

The testbench needs to end after **500ns**.

Tips: The expected waveform is like:



Example code:

```
module tb_clk_div7(    );
    /*
    module clock_div_7(
    input clk, rst_n,
    output  clk_out
        );
    */
    reg clk,rst_n;
    wire clk_out;
    clock_div_7 dut(clk,rst_n,clk_out);
    initial begin
    $monitor ("%d %d %d", clk, rst_n, clk_out);


    end
```

Q10

##Description
In question 10, you are required to implement a memory cell as well as a **read-and-write** function. Please make sure that your design is strictly in accordance with the following requirements.

write

- input rw == 0 represents a write operation
- write operation is triggered by **clk's positive edge**
- the reset signal (*rst_n*) should be **asynchronous** and **active low**, and you should reset not only the data in your memory but also the other signals including *data_out* and *data_valid*

read

- input rw == 1 represents a read operation
- read operation should be implemented as **combinational logic**
- only set *data_valid* to 1 when a read operation is performed, otherwise it should always be 0

###standard code
Please make sure the naming of your variables and module are strictly in line with the requirements

```
module MemUnit16_8(
input clk, rw, rst_n,  // A rw of 1 indicates a read operation, and a rw of 0
indicates a write operation.
input [3:0] addr, // address to be read or written
input [7:0] data_in, // when in the write state, assign the value to the
corresponding address
output reg [7:0]  data_out, // Data read in read state
output reg data_valid //represents the data_out is valid
    );
```

##Result
The result should looks like this \:)
please notice that at 240,000 ns, *data_out* had already became a valid output and had a value of *00*