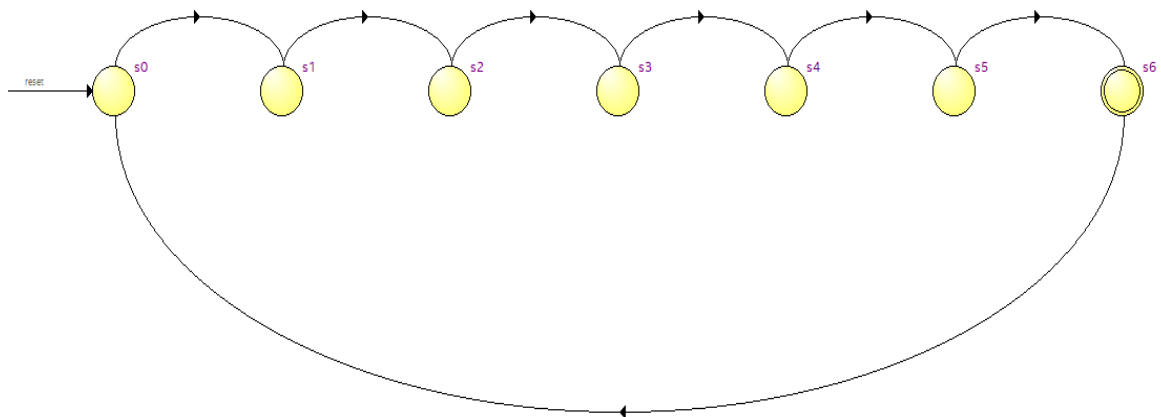


PROBLEM STATEMENT:

Design a clock frequency divider module that decelerates the given clock the given clock to obtain the given frequency.

DESIGN IMPLEMENTATION:

I have used finite state machine design to implement this module. I have taken the clock division ratio to be 7:1, but it can be set to any desired value by changing the value assigned to the parameter. This module defines a clock frequency divider with a specified division ratio. It utilizes a seven-state finite state machine to sequence through states representing each division stage. The module takes an input clock and a reset signal and outputs a signal that toggles at the desired divided frequency. The division ratio is set by the parameter `DIVISION_RATIO`. The module transitions between states on the rising edge of the clock or the rising edge of the reset signal, with the output signal toggling when the current state matches the specified division ratio minus one. Overall, the module efficiently divides the input clock frequency and generates an output signal with the desired frequency division.



State Encoding Table

State	Binary Encoding	Decimal Equivalent
S0	000	0
S1	001	1
S2	010	2
S3	011	3
S4	100	4
S5	101	5
S6	110	6

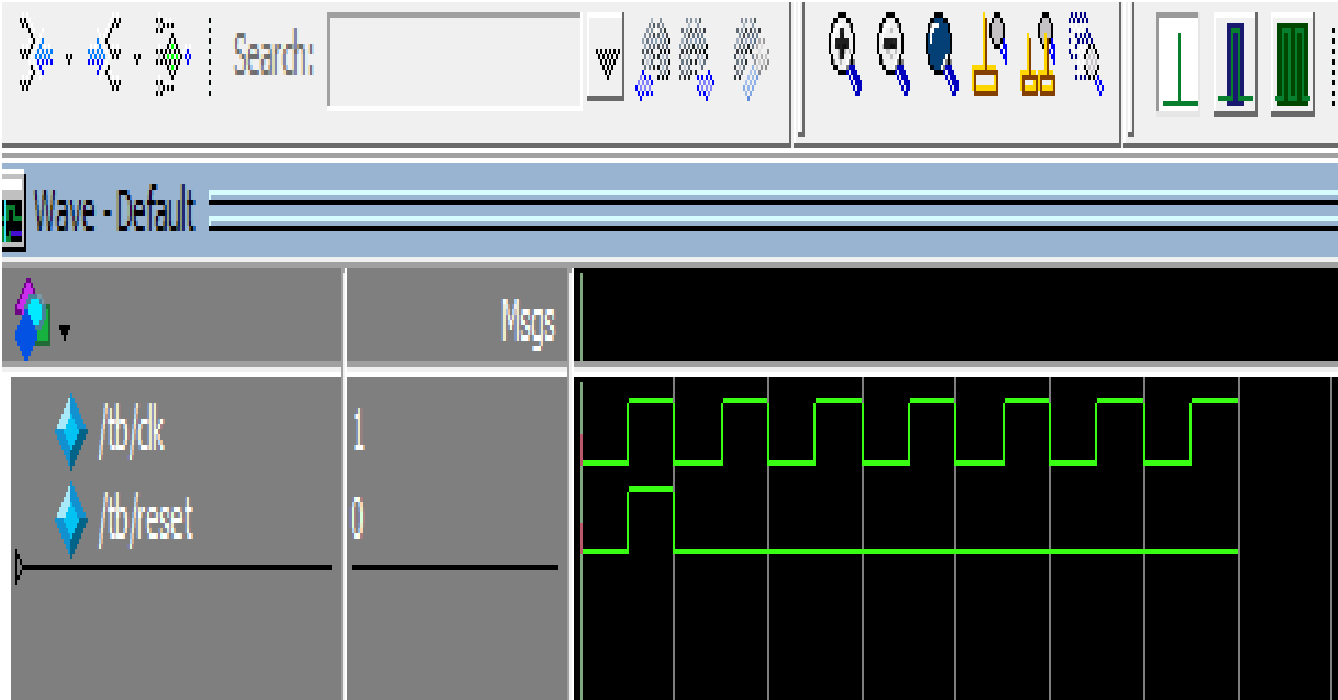
Output Table

State	Output
S0	0
S1	0
S2	0
S3	0
S4	0
S5	0
S6	1

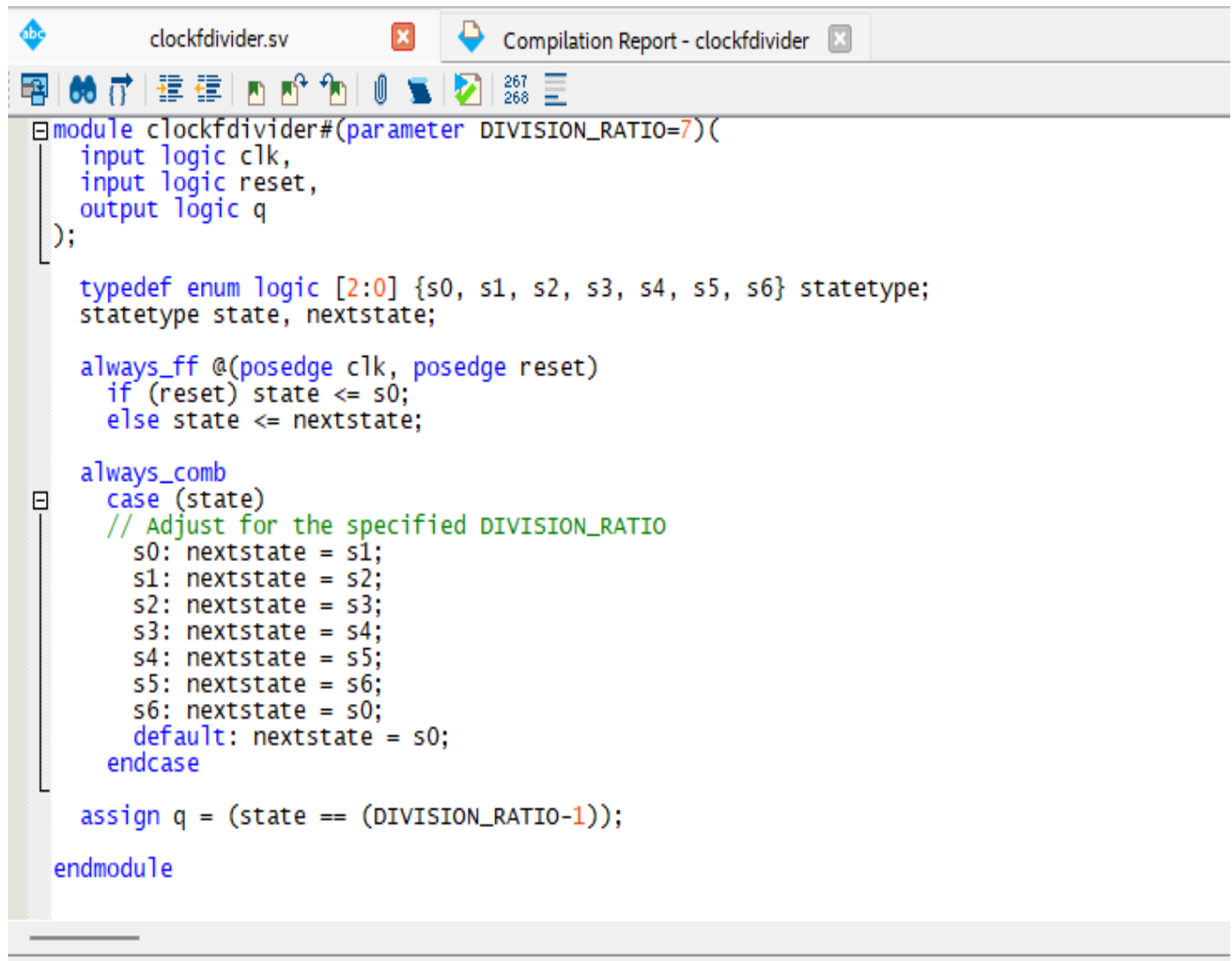
	Source State	Destination State
2	s1	s2
3	s2	s3
4	s3	s4
5	s4	s5
6	s5	s6
7	s6	s0

Transitions
Encoding

SIMULATION RESULTS:



DESIGN CODE:



The screenshot shows a Verilog code editor with a file named 'clockfddivider.sv'. The code implements a clock divider module. The module has three inputs: 'clk' (logic), 'reset' (logic), and 'q' (logic). It has a parameter 'DIVISION_RATIO' set to 7. The code uses a state machine to divide the clock. It defines a state type 'statetype' with states 's0' through 's6'. The state machine is implemented using 'always_ff' for the next state calculation and 'always_comb' for the output 'q'. The output 'q' is set to 1 when the state reaches 'DIVISION_RATIO-1' (6) and 0 otherwise.

```
module clockfddivider#(parameter DIVISION_RATIO=7)(
    input logic clk,
    input logic reset,
    output logic q
);

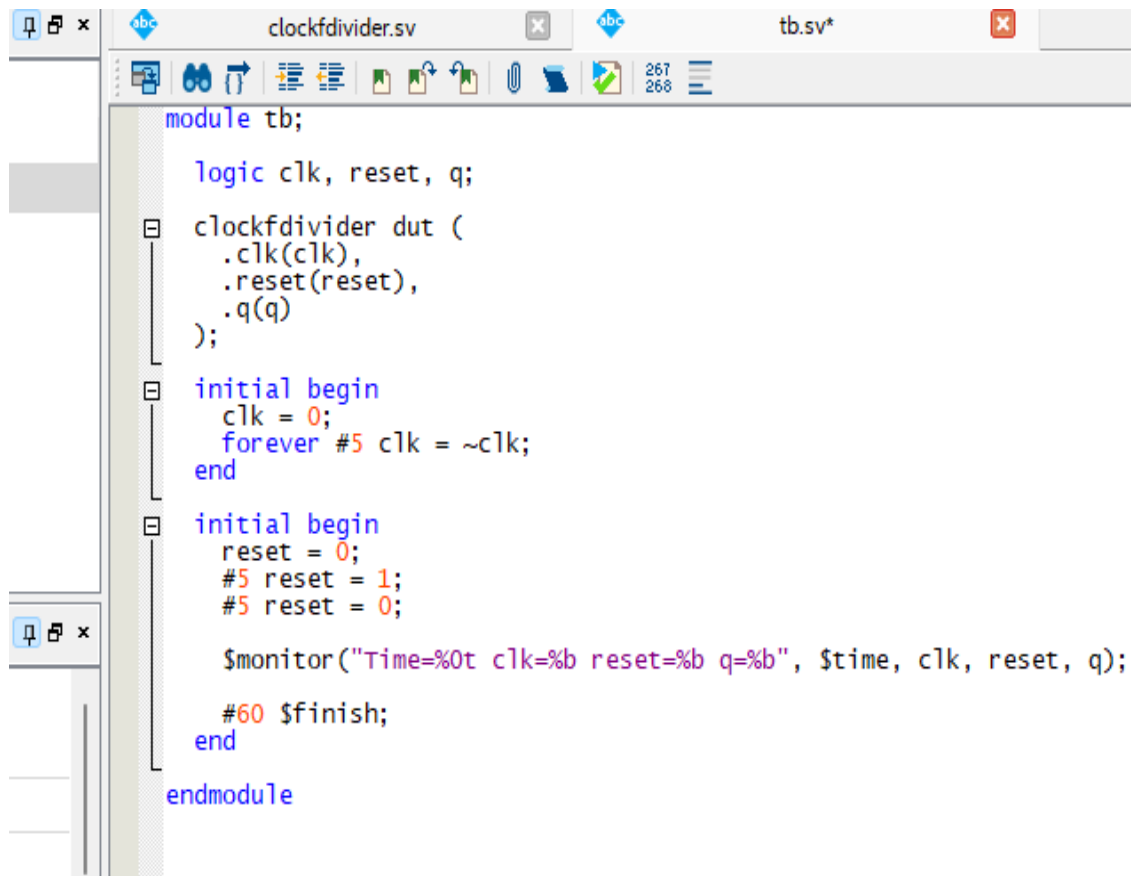
    typedef enum logic [2:0] {s0, s1, s2, s3, s4, s5, s6} statetype;
    statetype state, nextstate;

    always_ff @(posedge clk, posedge reset)
        if (reset) state <= s0;
        else state <= nextstate;

    always_comb
    case (state)
        // Adjust for the specified DIVISION_RATIO
        s0: nextstate = s1;
        s1: nextstate = s2;
        s2: nextstate = s3;
        s3: nextstate = s4;
        s4: nextstate = s5;
        s5: nextstate = s6;
        s6: nextstate = s0;
        default: nextstate = s0;
    endcase

    assign q = (state == (DIVISION_RATIO-1));
endmodule
```

TEST BENCH CODE:



The screenshot shows a software editor window with two tabs: 'clockfddivider.sv' and 'tb.sv*'. The 'tb.sv*' tab is active, displaying the following Verilog testbench code:

```
module tb;
    logic clk, reset, q;

    clockfddivider dut (
        .clk(clk),
        .reset(reset),
        .q(q)
    );

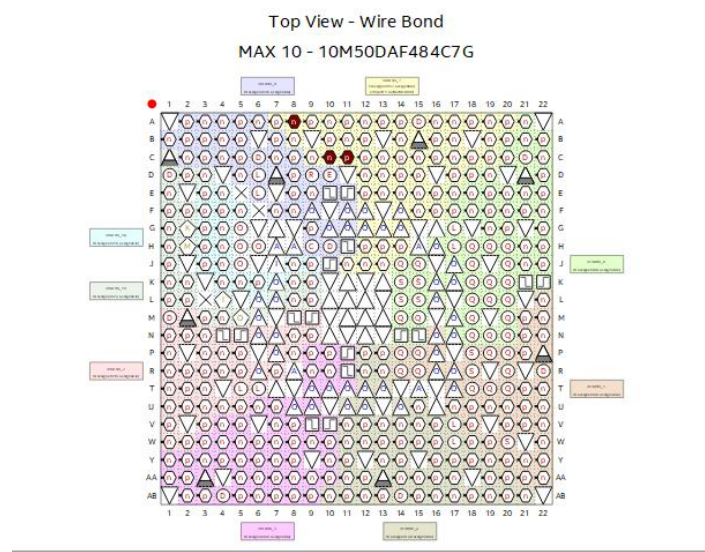
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        reset = 0;
        #5 reset = 1;
        #5 reset = 0;

        $monitor("Time=%0t clk=%b reset=%b q=%b", $time, clk, reset, q);
        #60 $finish;
    end
endmodule
```

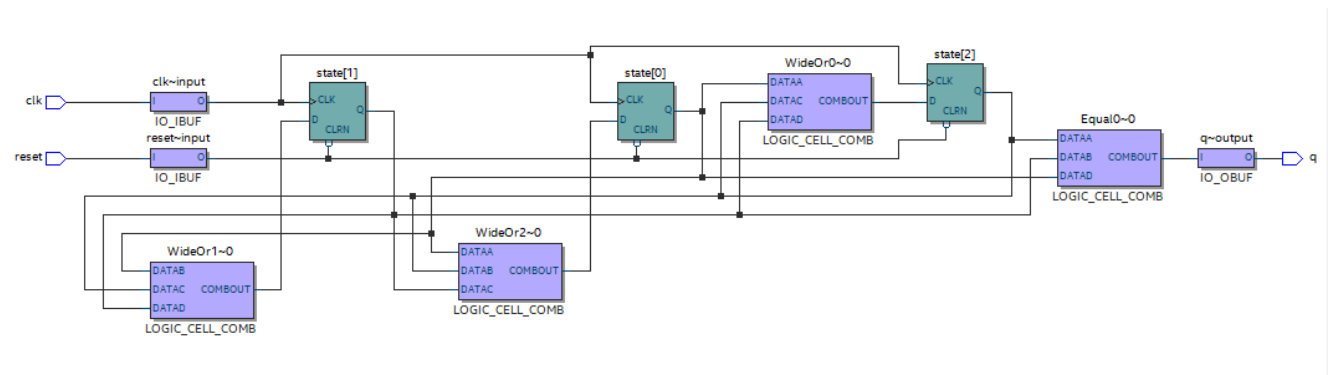
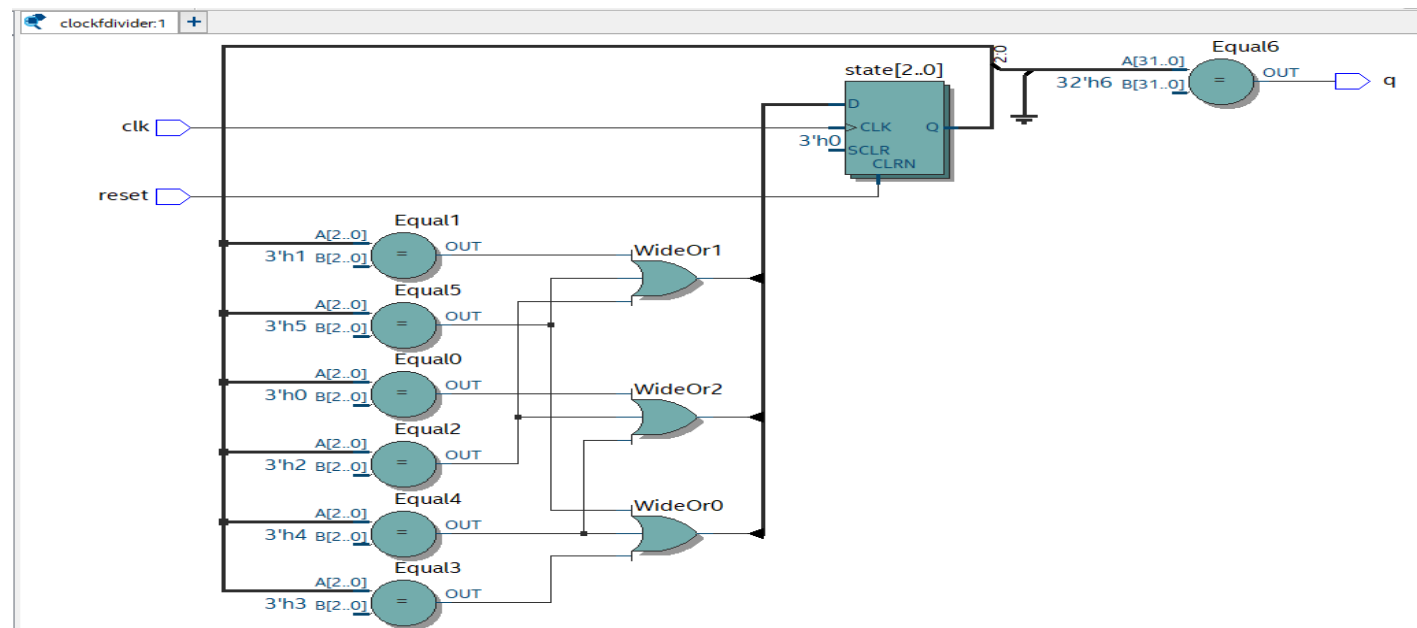
The code defines a testbench module 'tb' that instantiates the 'clockfddivider' module. It sets up a clock signal 'clk' that toggles every 5 time units, a reset signal 'reset' that pulses high for 5 time units, and a data signal 'q'. A monitor is used to display the values of 'clk', 'reset', and 'q' over time, and the simulation is terminated after 60 time units.

PIN PLANNER, RTL VIEWER, TECHNOLOGY MAP VIEWER:



Named: * Edit: X

Node Name	Direction	Location
in clk	Input	PIN_C10
out q	Output	PIN_A8
in reset	Input	PIN_C11
<<new node>>		



RESULTS:

The simulation output of the clock frequency divider module exhibits the expected behavior. As the input clock signal is applied, the output signal (**q**) transitions according to the specified division ratio, resulting in a frequency-divided waveform. The state transitions and output toggling align with the defined finite state machine, accurately reflecting the desired frequency division. Additionally, the reset functionality works as intended, initializing the state machine and ensuring proper operation after a reset event. The simulation confirms the functionality of the module in achieving the specified clock frequency division.