KARTHICK RAJA B
126004121

Given:

Rule 90 is a one-dimensional cellular automaton with interesting properties.
The rules are simple. There is a one-dimensional array of cells (on or off). At each time step, the next state of each cell is the XOR of the cell's two current neighbours. A more verbose way of expressing this rule is the following table, where a cell's next state is a function of itself and its two neighbours:

Left	Center	Right	Center's next state
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

(The name "Rule 90" comes from reading the "next state" column: 01011010 is decimal 90.)
In this circuit, create a 512-cell system (q[511:0]), and advance by one time step each clock cycle. The load input indicates the state of the system should be loaded with data[511:0]. Assume the boundaries (q[-1] and q[512]) are both zero (off).

Verilog code:

```
module rule90 (
    input clk,
    input reset,
    input load,
    input [511:0] data,
    output reg [511:0] q,
    reg [511:0] next_q
);

    integer i;
```

```
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            q <= 512'b0;
        end else if (load) begin
            q <= data;
        end else begin

            next_q[0] = q[1];
            next_q[511] = q[510];

            for (i = 1; i < 511; i = i + 1) begin
                next_q[i] = q[i-1] ^ q[i+1];
            end

            q <= next_q;
        end
    end
endmodule
```

Testbench code:

```
module tb_rule90();
    reg clk, reset, load;
    reg [511:0] data;
    wire [511:0] q;

    rule90 uut (
        .clk(clk),
        .reset(reset),
        .load(load),
        .data(data),
        .q(q)
    );

    always #5 clk = ~clk; /

    initial begin
        clk = 0; reset = 1; load = 0; data = 0;
        #10 reset = 0;
        #10 load = 1; data = 512'b1 << 256; /        #10 load = 0;

        #200 $stop;
```

```
    end
endmodule
```

Functional Simulation: