perplexity

Verilog Code for Traffic Light Controller (Based on Blackboard Content)

Below is a complete Verilog code for a traffic light controller, modeled as a finite state machine (FSM) as depicted on the blackboard. The design uses four states (S0, S1, S2, S3), a single input sensor $ x $, and outputs for the traffic lights. The code structure and state transitions are based on the diagram and notes visible in the image[1] [2].

```
module traffic_light_controller(
    input clk,
    input reset,
    input x,           // Sensor input: 1 if vehicle present, 0 otherwise
    output reg [1:0] highway, // [^1]: Red, [^0]: Green
    output reg [1:0] farm     // [^1]: Red, [^0]: Green
);

// State encoding
parameter S0 = 2'b00; // Highway Green, Farm Red
parameter S1 = 2'b01; // Highway Yellow, Farm Red
parameter S2 = 2'b10; // Highway Red, Farm Green
parameter S3 = 2'b11; // Highway Red, Farm Yellow

reg [1:0] state, next_state;

// State register (sequential block)
always @(posedge clk or posedge reset) begin
    if (reset)
        state &lt;= S0;
    else
        state &lt;= next_state;
end

// Next state logic (combinational block)
always @(*) begin
    case (state)
        S0: begin
            if (x == 1)
                next_state = S1;
            else
                next_state = S0;
        end
        S1: begin
            next_state = S2; // After yellow, go to farm green
        end
        S2: begin
            if (x == 0)
                next_state = S3;
            else
                next_state = S2;
        end
```

```
        S3: begin
            next_state = S0; // After farm yellow, back to highway green
        end
        default: next_state = S0;
    endcase
end

// Output logic (output block)
always @(*) begin
    case (state)
        S0: begin // Highway Green, Farm Red
            highway = 2'b01; // Green
            farm    = 2'b10; // Red
        end
        S1: begin // Highway Yellow, Farm Red
            highway = 2'b11; // Yellow (assuming 2'b11 as yellow, adjust as needed)
            farm    = 2'b10; // Red
        end
        S2: begin // Highway Red, Farm Green
            highway = 2'b10; // Red
            farm    = 2'b01; // Green
        end
        S3: begin // Highway Red, Farm Yellow
            highway = 2'b10; // Red
            farm    = 2'b11; // Yellow (assuming 2'b11 as yellow, adjust as needed)
        end
        default: begin
            highway = 2'b10;
            farm    = 2'b10;
        end
    endcase
end

endmodule
```

Key Points:

- The states and transitions match the state diagram on the board, where `x` is the sensor input.
- Highway and farm outputs are 2-bit signals representing Red, Yellow, and Green (adjust encoding as per your actual hardware requirements).
- The code is modular, with clear sequential, combinational, and output blocks as referenced in the board notes.

Customization:

- You may need to add timers for yellow light durations or adjust output encoding for your specific hardware[2].
- The output encoding for yellow is assumed as `2'b11` (as per some conventions); adjust if your system uses a different encoding.

This code provides a direct implementation of the FSM and logic structure shown in your classroom blackboard image[1] [2].

⁂

1. image.jpg
2. https://github.com/N-O-O-B-Coder/State-Machine-based-Controller-For-Traffic-Signal