

Leveraging Bing GPT-4 for Digital Design and Verification with SystemVerilog and Assertions

1. Introduction

ChatGPT, as a language model, has been trained on a wide range of text data from the internet, including technical content. It has demonstrated potential in generating SystemVerilog (SV) modules based on English requirements and creating SystemVerilog Assertions (SVA) from module descriptions and English requirements. To assess its capabilities, accuracy, and usefulness, this paper aims to evaluate two models:

1. A simple counter with unconventional requirements.
2. A traffic light controller with two additional requirements to increase its complexity:
 - The North-South sensors serve as the main road and should be given higher priority and more traffic time.
 - The East-West sensors act as the side road and should be active for a brief period after the North-South light completes a full cycle.
 - An emergency sensor triggers an immediate transition to a safe state, meaning that when the sensor is activated, a currently green light quickly changes to yellow and then red.

The paper will delve into the experimentation, analysis, and evaluation of these models to gauge the potential benefits and challenges of employing ChatGPT in the context of digital design and verification with SystemVerilog and assertions.

2. Summary and Conclusions

- **Value in the process:** The process of expressing requirements in English and having Bing create a design and testbench module with its comments, along with the translation of English into SVA assertions, enhances human awareness of actual requirements and potential misunderstandings. This becomes particularly significant when the machine misinterprets or fails to articulate what was initially envisioned in your mind.
- **Accuracy:** GPT exhibits a reasonable understanding of digital design concepts and the SystemVerilog with SVA language, although it is not flawless. GPT is capable of making mistakes, but it graciously accepts corrections and guidance. I observed three forms of guidance:
 1. Correcting SystemVerilog syntax errors.
 2. Suggesting architectural changes, such as utilizing nonblocking assignments and registering specific signals.
 3. Breaking down the architecture into smaller, simpler machines, potentially as separate modules, and then requesting GPT to connect all the modules into a single top-level model.
- **Potential Pitfalls:** 1) If GPT generates a suboptimal architecture, it may be challenging to guide it towards the desired outcome. One can easily get lost in a multitude of tuning instructions, leading to confusion and potential difficulties in achieving the intended design. 2) Due to security concerns, business organizations may be hesitant to communicate design requirements and designs to GPT. However, it is possible to explore special arrangements to ensure privacy between GPT and businesses.

- **Recommendations:** Despite being in the early stages of development for processing requirements and converting them into SystemVerilog with SVA, GPT still holds value in its ability to shed light on ambiguous aspects of design and requirements. Additionally, GPT can generate additional assertions based on its comprehension of the requirements. While it may not consistently produce complete designs, it can offer templates for testbenches.

3. Counter with special requirements:

Related files:

https://systemverilog.us/vf/tlight_bing_last.sv // last iteration, model fails

https://systemverilog.us/vf/tlight_propsl.sv // From book, needs binding

<https://systemverilog.us/vf/trafficlightok.sv> // Works OK, one assertion needs the range to be extended

https://systemverilog.us/vf/Bing_gpt_for_design.pdf // this file

Here is a summary of my interaction with Bing GPT-4; GPT understood my typos.

[Me] Write a systemverilog code for a 4-bit counter that has the following requirements:

- 1) it is loadable counter with a value from data_in.
- 2) It is synchronously reset when rst_n
- 3) Its minimum count is 2 and its maximum count is 5.
- 4) It stays at 5 once it reaches 5
- 5) write a systemverilog assertion that verifies that the count must change at least every 8 cycles
- 6) write systemverilog assertions to define those requirements

[GPT] It produced a model with type reg insteard of logic, and incorrectly used the always in the assertions

[Ben] For that model, write an assertion to insure that if the couter value is stable, then with 1 to 8 clock cycles it should be unstable. Also rewrite all the assertions but this time do not use the "always" because that is incorrect

[GPT] Apologies for the confusion. Since the assertions should not use the "always" keyword, let's rewrite them accordingly:It created a model with the couter==1 at reset.

[Ben] Rewrite what you just created, but this time when you reset go for a value of 2. Also do NOT use the type "reg" as it is outdated; insteard use the type "logic"

[GPT] Certainly! Here's the revised code that includes the requested changes, using the "logic" type instead of "reg" and setting the reset value to 2:

[Comment] Code is at <https://www.edaplayground.com/x/nhu2>

[GPT]

```

module Counter (
    input wire clk,
    input wire rst_n,
    input wire load,
    input wire [3:0] data_in,
    output wire [3:0] count
);

    logic [3:0] count_reg;

    always_ff @(posedge clk, negedge rst_n) begin
        if (!rst_n)
            count_reg <= 4'b0010; // Synchronously reset to 2
        else if (load)
            count_reg <= data_in;
        else if (count_reg == 4'b0101)
            count_reg <= 4'b0101; // Stay at 5
        else
            count_reg <= count_reg + 1;
    end

    assign count = count_reg;

```

```
endmodule
```

```
module Testbench;
```

```
    logic clk;  
    logic rst_n;  
    logic load;  
    logic [3:0] data_in;  
    logic [3:0] count;
```

```
    // Instantiate the counter module
```

```
    Counter counter (  
        .clk(clk),  
        .rst_n(rst_n),  
        .load(load),  
        .data_in(data_in),  
        .count(count)  
    );
```

```
    // Clock generation  
    always #5 clk = ~clk;
```

```
    // Initial values
```

```
    initial begin  
        clk = 0;  
        rst_n = 1;  
        load = 0;  
        data_in = 0;  
        #10 rst_n = 0; // Assert reset  
        #20 rst_n = 1; // Deassert reset
```

```
    // Add test cases here  
    // ...
```

```
    #100 $finish; // End simulation  
end
```

```
[Ben] Below are the assertions it created. I corrected the ones with poor form
```

```
    // Assertion: The count should never exceed the maximum value of 5  
    // assert property (@(posedge clk) !$past(count) |-> count <= 4'b0101); // GPT  
    assert property (@(posedge clk) count <= 4'b0101); // Ben
```

```
    // Assertion: The count should be greater than or equal to the minimum value of 2  
    // assert property (@(posedge clk) !$past(count) |-> count >= 4'b0010); // GPT  
    assert property (@(posedge clk) count >= 4'b0010); // Ben
```

```
    // Assertion: The count should only change when the load signal is inactive  
    // assert property (@(posedge clk) disable iff (load) !$past(load) |-> !$changed(count)); // GPT  
    assert property (@(posedge clk) disable iff (rst_n) load |-> ##1 count==$past(data_in));
```

```
    // Assertion: The count should not change while the reset signal is active  
    // assert property (@(posedge clk) disable iff (!rst_n) !$past(rst_n) |-> !$changed(count)); // GPT  
    // [Ben] if reset then count==2  
    assert property (@(posedge clk) (!rst_n) |-> ##1 count==2);
```

```
// Assertion: If the counter value is stable, then within 1 to 8 clock cycles, it should become unstable
// assert property (@(posedge clk) disable iff (!$stable(count)) !$past($stable(count)) |-> ##[1:8] !$stable(count));
// [Ben] if counter is stable, it should change in value within 1 to 8 cycles
  assert property (@(posedge clk) disable iff(!rst_n) $stable(count)|-> ##[1:8] !$stable(count));
endmodule
```

```
// Testbench instantiation
module tb;
  Testbench tb();
endmodule
```

[GPT In this code, we have two modules: Counter and Testbench. The Counter module defines the 4-bit counter with the specified requirements. The Testbench module instantiates the Counter module, provides test stimuli, and includes the revised assertions.

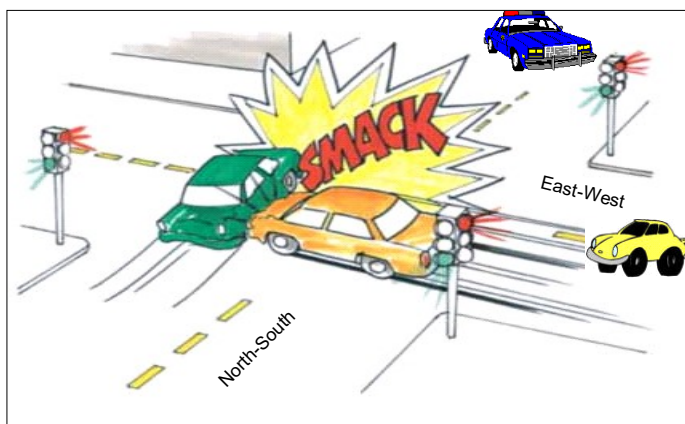
To run the simulation, you can use a SystemVerilog simulator and compile the code. The test cases can be added within the initial block in the Testbench module, and the simulation will run for a specific duration, as indicated by the \$finish statement.

Please note that you may need to adjust the test stimuli and add appropriate test cases to validate the counter's behavior according to your specific requirements. In this revised code, the "reg" type has been replaced with the "logic" type for improved compatibility with modern SystemVerilog. The reset value has been changed to 2, as requested, by updating the assignment in the reset condition.

The rest of the code remains the same, including the clock generation, initial values, and assertions.

4. Traffic Light Controller

This design represents a simple traffic light controller for a North-South and East-West intersection. The North-South is the main road, and is given the GREEN light unless a sensor on the East-West Street is activated. When that occurs, and the North-South light was GREEN for sufficient time, then the light will change to give way to the East-West traffic. The design also takes into account emergency vehicles that can activate an emergency sensor. When the emergency sensor is activated, then the North-South and East-West lights will turn RED, and will stay RED for a minimum period of 3 cycles.



The following are my interactions with GPT. The final code is shown at the end. Simulation demonstrated that the model created by GPT is incorrect and assertions failed: The created model failed these 2 assertions

```
// Safety property
property Never_NS_EW_ALL_GREEN;
    disable iff (!reset_n) not (ns_light==GREEN && ew_light==GREEN);
endproperty : Never_NS_EW_ALL_GREEN
Never_NS_EW_ALL_GREEN_1 : assert property(@ (posedge clk) Never_NS_EW_ALL_GREEN)

// GREEN-YELLOW at the same time
property NeverGreenYellow;
    not ((ew_light==GREEN && ns_light==YELLOW) || (ns_light==GREEN &&
ew_light==YELLOW));
endproperty : NeverGreenYellow
NeverGreenYellow_1: assert property (@ (posedge clk) NeverGreenYellow);
```

[ME] write a systemverilog module with the following requirements:

This design represents a simple traffic light controller for a North-South and East-West intersection. The North-South is the main road, and is given the GREEN light unless a sensor on the East-West Street is activated. When that occurs, and the North-South light was GREEN for sufficient time, then the light will change to give way to the East-West traffic. The design also takes into account emergency vehicles that can activate an emergency sensor. When the emergency sensor is activated, then the North-South and East-West lights will turn RED, and will stay RED for a minimum period of 3 cycles.

1. North-South remains GREEN unless one of the East-West sensors is activated.
2. If the North-South light is RED and the North-South GREEN timer is 3, then the light will switch to GREEN.
3. If the North-South light is YELLOW, it will switch to RED.
4. If the North-South light is GREEN, and the emergency sensor is activated, the light will switch to YELLOW. Also, if the North-South timer is 3, and the East-West sensor is activated, the North-South light will switch to YELLOW.
5. The North-South GREEN timer is reset to zero at reset, or when the North-South light is YELLOW. Otherwise, it increments at every clock until it reaches the maximum count of 3.
6. The East-West light switches from RED to a PRE_GREEN state (to allow the North-South light to go to YELLOW) if the North-South timer is 3 and the East-West sensor is activated.
7. If the East-West is PRE_GREEN, it will switch to GREEN at the next clock.
8. If the East-West light is YELLOW, it will switch to RED.
9. If the East-West is GREEN and either the emergency sensor is activated or the East-West timer reaches a count of 3 then it switches to YELLOW.
10. The East-West GREEN timer is reset to zero at reset or when the East-West light is YELLOW. Otherwise, it increments at every clock until it reaches the maximum count of 3.

Use this interface:

```
typedef enum { OFF, // power off //ch7/7.3/trafficlightok.sv
    RED, // red state YELLOW, // yellow state
    GREEN, // green state
    PRE_GREEN} // state before green
    _lights_t;
module trafficlight ( output _lights_t ns_light, // North/South light status, Main road
    output _lights_t ew_light, // East/West light status
    input ew_sensor, // East/West sensor for new car
    input emgcy_sensor, // emergency sensor
    input reset_n, // synchronous reset
```

```
input clk // master clock );
```

Use type logic instead of reg

use nonblocking assignments (i.e., the <=) in the always @(posedge clk)

[GPT] Here is a SystemVerilog module that meets the requirements you provided:

```
``systemverilog
typedef enum { OFF, RED, YELLOW, GREEN, PRE_GREEN } lights_t;

module trafficlight (
    output lights_t ns_light,
    output lights_t ew_light,
    input ew_sensor,
    input emgcy_sensor,
    input reset_n,
    input clk
);

// Internal state variables
localparam MAX_COUNT = 3;
logic [1:0] ns_timer;
logic [1:0] ew_timer;

always_ff @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        ns_light <= RED;
        ew_light <= RED;
        ns_timer <= 0;
        ew_timer <= 0;
    end else begin
        // North-South light state transitions
        case (ns_light)
            GREEN: begin
                if (emgcy_sensor) begin
                    ns_light <= YELLOW;
                end else if (ns_timer == MAX_COUNT && ew_sensor) begin
                    ns_light <= YELLOW;
                end else begin
                    ns_timer <= ns_timer + 1;
                end
            end
            YELLOW: begin
                ns_light <= RED;
                ns_timer <= 0;
            end
            RED: begin
                if (ns_timer == MAX_COUNT) begin
                    ns_light <= GREEN;
                end else begin
                    ns_timer <= ns_timer + 1;
                end
            end
        endcase
    end
endcase
```

```

// East-West light state transitions
case (ew_light)
  GREEN: begin
    if (emgcy_sensor || ew_timer == MAX_COUNT) begin
      ew_light <= YELLOW;
    end else begin
      ew_timer <= ew_timer + 1;
    end
  end
  YELLOW: begin
    ew_light <= RED;
    ew_timer <= 0;
  end
  RED: begin
    if (ns_timer == MAX_COUNT && ew_sensor) begin
      ew_light <= PRE_GREEN;
    end else begin
      ew_timer <= ew_timer + 1;
    end
  end
  PRE_GREEN: begin
    ew_light <= GREEN;
  end
endcase

// Emergency sensor overrides all other states and sets both lights to RED
for 3 cycles.
  if (emgcy_sensor) begin
    ns_light <= RED;
    ew_light <= RED;
    ns_timer <= 0;
    ew_timer <= 0;
  end

  end // if (!reset_n)
end // always_ff @(posedge clk or negedge reset_n)

endmodule // trafficlight

...

```

This module implements the state transitions for the North-South and East-West lights as described in the requirements. The `ns_timer` and `ew_timer` variables are used to keep track of the time that each light has been in the GREEN state. The emergency sensor overrides all other states and sets both lights to RED for a minimum of 3 cycles.

Is there anything else you would like me to add or change? 😊

[Ben] write SystemVerilog assertions for this module

[GPT] Here are some SystemVerilog assertions that can be used to verify the behavior of the `trafficlight` module:

```
// Both lights should never be GREEN at the same time
assert property (@(posedge clk) disable iff (!reset_n)
    !(ns_light == GREEN && ew_light == GREEN));
```

```
// Both lights should never be YELLOW at the same time
assert property (@(posedge clk) disable iff (!reset_n)
    !(ns_light == YELLOW && ew_light == YELLOW));
```

```
// If the emergency sensor is activated, both lights should turn RED within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n) emgcy_sensor |->
    (ns_light == RED && ew_light == RED) within (2));
```

ERROR

```
// If the East-West sensor is activated and the North-South light is GREEN for 3 cycles,
// then the North-South light should turn YELLOW within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n) ew_sensor && ns_timer == 3
    |-> ns_light == YELLOW within (2));
```

ERROR

```
// If the East-West sensor is activated and the North-South light is GREEN for 3 cycles,
// then the East-West light should turn PRE_GREEN within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n) ew_sensor && ns_timer == 3 |->
    ew_light == PRE_GREEN within (2));
```

ERROR

These assertions check for some of the key behaviors of the `trafficlight` module, such as ensuring that both lights are never GREEN or YELLOW at the same time, and that the emergency sensor and East-West sensor cause the appropriate state transitions.

Is there anything else you would like me to add or change?

[Ben] give me many more assertions

[GPT] Here are some additional SystemVerilog assertions that can be used to verify the behavior of the `trafficlight` module:

```
// If the North-South light is GREEN for 3 cycles and the East-West sensor is activated,
// then the North-South light should turn YELLOW within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n)
    ns_timer == 3 && ew_sensor |-> ns_light == YELLOW within (2));
```

```
// If the North-South light is GREEN for 3 cycles and the East-West sensor is activated,
// then the East-West light should turn PRE_GREEN within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n) ns_timer == 3 && ew_sensor |->
    ew_light == PRE_GREEN within (2));
```

```
// If the East-West light is GREEN for 3 cycles, then it should turn YELLOW within 2 clock
cycles
```



```

assert property (@(posedge clk) disable iff (!reset_n) ew_timer == 3
  |-> ew_light == YELLOW
  Here are some more SystemVerilog assertions that can be used to verify the behavior of
  the trafficlight module:
  // If the North-South light is RED for 3 cycles, then it should turn GREEN within 2 clock
  cycles
assert property (@(posedge clk) disable iff (!reset_n)
ns_light == RED && ns_timer == 3 |-> ns_light == GREEN within (2));

// If the East-West light is RED for 3 cycles and the East-West sensor is activated,
// then it should turn PRE_GREEN within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n)
ew_light == RED && ew_timer == 3 && ew_sensor |-> ew_light == PRE_GREEN within (2));

// If the East-West light is PRE_GREEN, then it should turn GREEN within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n) ew_light == PRE_GREEN |->
  ew_light == GREEN within (2));

// If the North-South light is YELLOW, then it should turn RED within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n) ns_light == YELLOW |->
ns_light == RED within (2));

// If the East-West light is YELLOW, then it should turn RED within 2 clock cycles
assert property (@(posedge clk) disable iff (!reset_n) ew_light == YELLOW |->
ew_light == RED within (2));

```

These assertions check for additional behaviors of the `trafficlight` module, such as ensuring that the North-South and East-West lights turn GREEN or PRE_GREEN when their respective timers reach a count of 3 and their sensors are activated, and that the lights turn RED when they are in the YELLOW state.

Is there anything else you would like me to add or change?

----- LATER ON -----

[ME] **Recalculate the model by also considering the following assertions**

parameter FAIL = 1'b0;

```

// sequence definition
sequence qEmgcy;
  @(posedge clk) emgcy_sensor;
endsequence : qEmgcy

// *****
// Safety property
property Never_NS_EW_ALL_GREEN;
  disable iff (!reset_n)
  not (ns_light==GREEN && ew_light==GREEN);
endproperty : Never_NS_EW_ALL_GREEN
Never_NS_EW_ALL_GREEN_1 : assert property(@(posedge clk) Never_NS_EW_ALL_GREEN);
// *****

```

```

// State of lights at reset
property nsLightAtReset;
  // disable iff (!reset_n)
  reset_n==1'b0 | => ns_light==OFF;
endproperty : nsLightAtReset
nsLightAtReset_1 : assert property(@ (posedge clk) nsLightAtReset);
//
property ewLightAtReset;
  // disable iff (!reset_n)
  reset_n==1'b0 | => ew_light==OFF;
endproperty : ewLightAtReset
ewLightAtReset_1 : assert property(@ (posedge clk) ewLightAtReset);
// *****

// State of lights during emergency
// Lights switch from GREEN to YELLOW to RED
property NsLightsWhenEmergency;
  disable iff (!reset_n)
  emgcy_sensor | => `true[*2] ##1 ns_light==RED;
endproperty : NsLightsWhenEmergency
NsLightsWhenEmergency_1 : assert property(@ (posedge clk) NsLightsWhenEmergency);
//
property EwLightsWhenEmergency;
  disable iff (!reset_n)
  emgcy_sensor | => `true[*2] ##1 ew_light==RED;
endproperty : EwLightsWhenEmergency
EwLightsWhenEmergency_1 : assert property(@ (posedge clk) EwLightsWhenEmergency);
// *****

// Safety, GREEN to RED is illegal. Need YELLOW
property NsNeverFromGreenToRed;
  disable iff (!reset_n)
  not(ns_light==GREEN ##1 ns_light==RED);
endproperty : NsNeverFromGreenToRed
NsNeverFromGreenToRed_1 : assert property(@ (posedge clk) NsNeverFromGreenToRed);
//
property EwNeverFromGreenToRed;
  disable iff (!reset_n)
  not(ew_light==GREEN ##1 ew_light==RED);
endproperty : EwNeverFromGreenToRed
EwNeverFromGreenToRed_1 : assert property(@ (posedge clk) EwNeverFromGreenToRed);
// *****

// The NorthSouth light is the main street light.
// If ns is green and no emergency or ew sensor, then next cycle is also GREEN
property NsGreenNext; /// Problem with this assertion because it ignores a latched ew_request
  (ns_light==GREEN) && ($past(emgcy_sensor)==1'b0 && reset_n==1'b1
//      && ew_green_req == 1'b0) | => ns_light==GREEN;
  ) | => ns_light==GREEN;
endproperty : NsGreenNext
// NsGreenNext_1: assert property (@ (posedge clk) NsGreenNext);

// *****

// East-West light to RED errata 3/23/05
/// property EwLightToRed;
///   disable iff (!reset_n || emgcy_sensor)
///   (ew_green_timer==2'b11) | =>
///   `true ##1 ew_light==RED; // abort emgcy_sensor==1'b1);
/// endproperty : EwLightToRed

```

```

/// EwLightToRed_1 : assert property (EwLightToRed);

// GREEN-YELLOW at the same time
property NeverGreenYellow;
  not ((ew_light==GREEN && ns_light==YELLOW) ||
        (ns_light==GREEN && ew_light==YELLOW));
endproperty : NeverGreenYellow
NeverGreenYellow_1: assert property (@ (posedge clk) NeverGreenYellow);

// NEW PROPERTIES 09/10/09
// *****
// The NorthSouth light is the main street light.
// It must remain GREEN for ns_green_timer == 3 before it can switch.
// Timer ns_green_timer will count to 3, and remain at 3 until light changes.
property NsGreenForMin3Cycles;
  @ (posedge clk) disable iff (!reset_n || emgcy_sensor)
  $rose(ns_light==GREEN) && !$past(emgcy_sensor) |=>
  ns_light==GREEN[*2]; // abort emgcy_sensor);
// ns_light==GREEN ##1 ns_light==GREEN; // abort emgcy_sen

```

[ME] The `EwLightsWhenEmergency` property is not being handled correctly in the current implementation

[GPT] //Here is the updated SystemVerilog module that includes the assertion you specified:
// typedef enum { OFF, RED, YELLOW, GREEN, PRE_GREEN } lights_t;
// You are correct. The `EwLightsWhenEmergency` property is not being handled correctly in the current implementation of the traffic light controller.
Here is an updated version of the module that correctly handles the `EwLightsWhenEmergency` property:
....

[ME] register the *emgcy_sensor* into register *emgcy_sensor_r* when *emgcy_sensor*==1. Clear *emgcy_sensor_r* when both *ns_light* and *es_light* are RED.

Also, fix from

```

property LightsTurnRedWhenEmergency;
  disable iff (!reset_n)
  emgcy_sensor |=> ((ns_light == RED) && (ew_light == RED)) ;

```

TO

```

property LightsTurnRedWhenEmergency;
  disable iff (!reset_n)
  emgcy_sensor |=> s_eventually ((ns_light == RED) && (ew_light == RED));

```

----- LATER ON

[ME] You often misuse the character { for the begin
Also, please add within the module as many assertions as possible

[GPT] I apologize for any confusion caused by my use of the { character instead of the begin keyword.
Here is an updated version of the SystemVerilog module that uses the begin keyword and includes additional assertions:
Here is the final code produced by GPT; it includes assertion that I told it about plus assertions that it generated.

```

`define true 1
typedef enum { OFF, RED, YELLOW, GREEN, PRE_GREEN } lights_t;

module trafficlight (

```

```

output lights_t ns_light,
output lights_t ew_light,
input ew_sensor,
input emgcy_sensor,
input reset_n,
input clk
);

parameter MAX_COUNT = 3;
parameter FAIL = 1'b0;

logic [1:0] ns_timer;
logic [1:0] ew_timer;
logic emgcy_sensor_r;

always_ff @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        ns_light <= RED;
        ew_light <= RED;
        ns_timer <= 0;
        ew_timer <= 0;
        emgcy_sensor_r <= 0;
    end else begin
        if (emgcy_sensor) begin
            emgcy_sensor_r <= 1;
        end else if (ns_light == RED && ew_light == RED) begin
            emgcy_sensor_r <= 0;
        end

        case (ns_light)
            GREEN: begin
                if (emgcy_sensor_r) begin
                    ns_light <= YELLOW;
                    ns_timer <= 0;
                end else if (ns_timer == MAX_COUNT && ew_sensor) begin
                    ns_light <= YELLOW;
                    ns_timer <= 0;
                end else begin
                    ns_timer <= ns_timer + 1;
                end
            end
            YELLOW: begin
                if (emgcy_sensor_r) begin
                    ns_light <= RED;
                end else begin // {
                    ns_light <= RED;
                    ns_timer <= 0;
                end
            end
            RED: begin
                if (ns_timer == MAX_COUNT) begin
                    ns_light <= GREEN;
                end else begin // {
                    ns_timer <= ns_timer + 1;
                end
            end
        end
    end
end

```

```

endcase

case (ew_light)
  GREEN: begin
    if (emgcy_sensor_r || ew_timer == MAX_COUNT) begin
      ew_light <= YELLOW;
      ew_timer <= 0;
    end else begin // {
      ew_timer <= ew_timer + 1;
    end
  end
  YELLOW: begin
    if (emgcy_sensor_r) begin
      ew_light <= RED;
    end else begin // {
      ew_light <= RED;
      ew_timer <= 0;
    end
  end
  RED: begin
    if (ns_timer == MAX_COUNT && ew_sensor) begin
      ew_light <= PRE_GREEN;
    end else begin // {
      ew_timer <= ew_timer + 1;
    end
  end
  PRE_GREEN: begin
    ew_light <= GREEN;
  end
endcase

end

end

// sequence definition
sequence qEmgcy;
  @ (posedge clk) emgcy_sensor;
endsequence : qEmgcy

// Safety property
property Never_NS_EW_ALL_GREEN;
  disable iff (!reset_n) not (ns_light==GREEN && ew_light==GREEN);
endproperty : Never_NS_EW_ALL_GREEN
Never_NS_EW_ALL_GREEN_1 : assert property(@ (posedge clk) Never_NS_EW_ALL_GREEN);

// State of lights at reset
property nsLightAtReset;
  reset_n==1'b0 | => ns_light==OFF;
endproperty : nsLightAtReset
nsLightAtReset_1 : assert property(@ (posedge clk) nsLightAtReset);

property ewLightAtReset;
  reset_n==1'b0 | => ew_light==OFF;
endproperty : ewLightAtReset
ewLightAtReset_1 : assert property(@ (posedge clk) ewLightAtReset);

```

```

// State of lights during emergency
// Lights switch from GREEN to YELLOW to RED
property NsLightsWhenEmergency;
  disable iff (!reset_n) emgcy_sensor |=> `true[*2] ##1 ns_light==RED;
endproperty : NsLightsWhenEmergency
NsLightsWhenEmergency_1 : assert property(@ (posedge clk) NsLightsWhenEmergency);

property EwLightsWhenEmergency;
  disable iff (!reset_n) emgcy_sensor |=> `true[*2] ##1 ew_light==RED;
endproperty : EwLightsWhenEmergency
EwLightsWhenEmergency_1 : assert property(@ (posedge clk) EwLightsWhenEmergency);

// Safety, GREEN to RED is illegal. Need YELLOW
property NsNeverFromGreenToRed;
  disable iff (!reset_n) not(ns_light==GREEN ##1 ns_light==RED);
endproperty : NsNeverFromGreenToRed
NsNeverFromGreenToRed_1 : assert property(@ (posedge clk) NsNeverFromGreenToRed);

property EwNeverFromGreenToRed;
  disable iff (!reset_n) not(ew_light==GREEN ##1 ew_light==RED);
endproperty : EwNeverFromGreenToRed
EwNeverFromGreenToRed_1 : assert property(@ (posedge clk) EwNeverFromGreenToRed);

// The NorthSouth light is the main street light.
// If ns is green and no emergency or ew sensor, then next cycle is also GREEN
property NsGreenNext;
  (ns_light==GREEN) && ($past(emgcy_sensor)==1'b0 && reset_n==1'b1) |=> ns_light==GREEN;
endproperty : NsGreenNext

// GREEN-YELLOW at the same time
property NeverGreenYellow;
  not ((ew_light==GREEN && ns_light==YELLOW) || (ns_light==GREEN && ew_light==YELLOW));
endproperty : NeverGreenYellow
NeverGreenYellow_1 : assert property (@ (posedge clk) NeverGreenYellow);

// The NorthSouth light is the main street light.
// It must remain GREEN for ns_green_timer == 3 before it can switch.
// Timer ns_green_timer will count to 3, and remain at 3 until light changes.
property NsGreenForMin3Cycles;
  @ (posedge clk) disable iff (!reset_n || emgcy_sensor) $rose(ns_light==GREEN) && !$past(emgcy_sensor) |=>
ns_light==GREEN[*2];
endproperty : NsGreenForMin3Cycles

// If the emergency sensor is activated, the North-South and East-West lights should eventually turn RED
property LightsTurnRedWhenEmergency;
  disable iff (!reset_n)
  emgcy_sensor |=> s_eventually ((ns_light == RED) && (ew_light == RED));
endproperty : LightsTurnRedWhenEmergency
LightsTurnRedWhenEmergency_1 : assert property (@ (posedge clk) LightsTurnRedWhenEmergency);

// The North-South light should never go directly from RED to YELLOW
property NsNeverFromRedToYellow;
  disable iff (!reset_n) not(ns_light==RED ##1 ns_light==YELLOW);
endproperty : NsNeverFromRedToYellow
NsNeverFromRedToYellow_1 : assert property(@ (posedge clk) NsNeverFromRedToYellow);

```

```

// The East-West light should never go directly from RED to YELLOW
property EwNeverFromRedToYellow;
  disable iff (!reset_n) not(ew_light==RED ##1 ew_light==YELLOW);
endproperty : EwNeverFromRedToYellow
EwNeverFromRedToYellow_1 : assert property(@ (posedge clk) EwNeverFromRedToYellow);

// The North-South light should never go directly from YELLOW to GREEN
property NsNeverFromYellowToGreen;
  disable iff (!reset_n) not(ns_light==YELLOW ##1 ns_light==GREEN);
endproperty : NsNeverFromYellowToGreen
NsNeverFromYellowToGreen_1 : assert property(@ (posedge clk) NsNeverFromYellowToGreen);

// The East-West light should never go directly from YELLOW to GREEN
property EwNeverFromYellowToGreen;
  disable iff (!reset_n) not(ew_light==YELLOW ##1 ew_light==GREEN);
endproperty : EwNeverFromYellowToGreen
EwNeverFromYellowToGreen_1 : assert property(@ (posedge clk) EwNeverFromYellowToGreen);

// The North-South light should never go directly from PRE_GREEN to RED
property NsNeverFromPreGreenToRed;
  disable iff (!reset_n) not(ns_light==PRE_GREEN ##1 ns_light==RED);
endproperty : NsNeverFromPreGreenToRed
NsNeverFromPreGreenToRed_1 : assert property(@ (posedge clk) NsNeverFromPreGreenToRed);

// The East-West light should never go directly from PRE_GREEN to RED
property EwNeverFromPreGreenToRed;
  disable iff (!reset_n) not(ew_light==PRE_GREEN ##1 ew_light==RED);
endproperty : EwNeverFromPreGreenToRed
EwNeverFromPreGreenToRed_1 : assert property(@ (posedge clk) EwNeverFromPreGreenToRed);

// The North-South light should never go directly from PRE_GREEN to YELLOW
property NsNeverFromPreGreenToYellow;
  disable iff (!reset_n) not(ns_light==PRE_GREEN ##1 ns_light==YELLOW);
endproperty : NsNeverFromPreGreenToYellow
// NsNeverFromPreGreen STOPPED HERE !
// Manual completion
NsNeverFromPreGreenToYellow_1: assert property(@ (posedge clk) NsNeverFromPreGreenToYellow );
endmodule

// Ben's testbench
module trafficlight_tb;
  `include "uvm_macros.svh" import uvm_pkg::*;
  // Declare the signals
  lights_t ns_light;
  lights_t ew_light;
  logic ew_sensor;
  logic emgcy_sensor;
  logic reset_n;
  logic clk=0;

  // Instantiate the trafficlight module
  trafficlight uut (
    .ns_light(ns_light),
    .ew_light(ew_light),
    .ew_sensor(ew_sensor),

```

```

        .emgcy_sensor(emgcy_sensor),
        .reset_n(reset_n),
        .clk(clk)
    );

    // Clock generation
    always #5 clk = ~clk;

    // Reset generation
    initial begin
        reset_n = 0;
        #10 reset_n = 1;
    end

    // Stimulus
    initial begin
        ew_sensor = 0;
        emgcy_sensor = 0;
        repeat (200) begin
            @(posedge clk); // { OFF, RED, YELLOW, GREEN, PRE_GREEN } lights_t;
            if (randomize( ew_sensor, emgcy_sensor) with {
                ew_sensor dist {1'b1 := 3, 1'b0 := 1};
                emgcy_sensor dist {1'b1 := 1, 1'b0 := 50};
            }) `uvm_error("MYERR", "This is a randomize error");
        end
        #20;
        $finish;
    end
endmodule

```

Here is a list of the assertions that were included in the updated SystemVerilog module for the traffic light controller:

1. `Never_NS_EW_ALL_GREEN`: This assertion checks that the North-South and East-West lights are never both `GREEN` at the same time.
2. `nsLightAtReset`: This assertion checks that the North-South light is `OFF` at reset.
3. `ewLightAtReset`: This assertion checks that the East-West light is `OFF` at reset.
4. `NsLightsWhenEmergency`: This assertion checks that if the emergency sensor is activated, the North-South light will eventually turn `RED`.
5. `EwLightsWhenEmergency`: This assertion checks that if the emergency sensor is activated, the East-West light will eventually turn `RED`.
6. `NsNeverFromGreenToRed`: This assertion checks that the North-South light never goes directly from `GREEN` to `RED`.
7. `EwNeverFromGreenToRed`: This assertion checks that the East-West light never goes directly from `GREEN` to `RED`.
8. `NsGreenNext`: This assertion checks that if the North-South light is currently `GREEN` and there is no emergency or East-West sensor activation, then it will remain `GREEN` in the next cycle.
9. `NeverGreenYellow`: This assertion checks that the North-South and East-West lights are never both `GREEN` and `YELLOW` at the same time.
10. `NsGreenForMin3Cycles`: This assertion checks that if the North-South light turns `GREEN`, it must remain `GREEN` for at least 3 cycles before it can switch to another state.
11. `LightsTurnRedWhenEmergency`: This assertion checks that if the emergency sensor is activated, both the North-South and East-West lights will eventually turn `RED`.
12. `NsNeverFromRedToYellow`: This assertion checks that the North-South light never goes directly from `RED` to `YELLOW`.
13. `EwNeverFromRedToYellow`: This assertion checks that the East-West light never goes directly from `RED` to `YELLOW`.

14. `NsNeverFromYellowToGreen`: This assertion checks that the North-South light never goes directly from `YELLOW` to `GREEN`.
15. `EwNeverFromYellowToGreen`: This assertion checks that the East-West light never goes directly from `YELLOW` to `GREEN`.
16. `NsNeverFromPreGreenToRed`: This assertion checks that the North-South light never goes directly from `???`

```
[Ben Simulation results
# ** Error: Assertion error.
#   Time: 15 ns Started: 5 ns nsLightAtReset_1 Expr: ns_light==OFF
# ** Error: Assertion error.
#   Time: 65 ns Started: 65ns NeverGreenYellow_1 Expr: ns_light==YELLOW
# ** Error: Assertion error.
#   Time: 75 ns Started: 45 ns EwLightsWhenEmergency_1 Expr: ew_light==RED
...
# ** Error: Assertion error.
#   Time: 885 ns Started: 885 ns Never_NS_EW_ALL_GREEN_1 Expr: ew_light==GREEN
```

5. Conclusions

While GPT may not consistently generate accurate results when converting English to SystemVerilog modules and assertions, its utilization does offer valuable insights into the requirements.