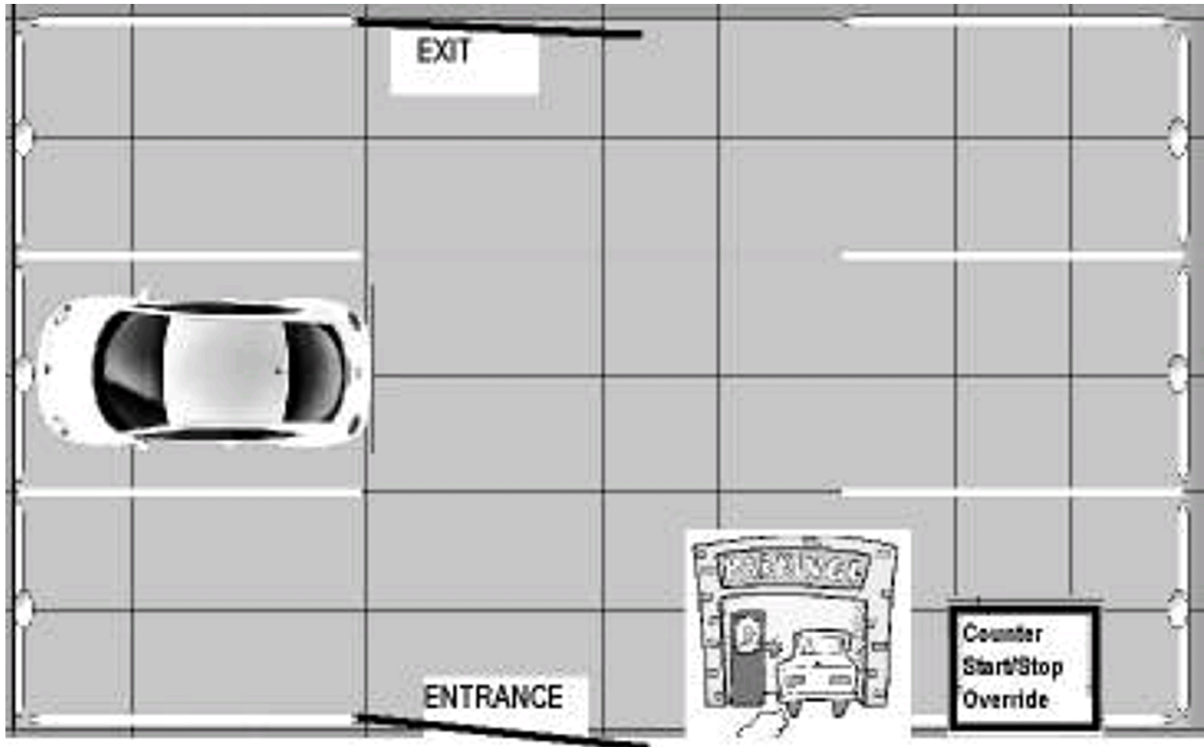


**Parking Lot SV and SVA experience with Bing GPT vs Human**  
*Requirements extracted from [http://bear.ces.cwru.edu/eecs\\_316/](http://bear.ces.cwru.edu/eecs_316/)*  
*Instructors: Prof. Chris Papachristou, Francis Wolff*



**Figure 6.ex1 Parking Lot overview**

## Requirements to GPT:

using systemverilog and assertions Write properties that define the following requirements. A package and an interface are provided below. /\* Problem Statement21 The example FSM controls a FIFO Queue but to be practical, consider that we want to design a simple digital controller that monitors cars entering and leaving a parking lot. The operation specs are as follows. \* The controller maintains the current COUNT of the total number of cars parked in the lot. \* The lot has a limited number of parking spots, for simplicity sixteen (16). \* There are two input sensors at the entrance and exit of the lot. The sensors provide input signals to the controller as cars pass by and the controller increments/decrements the current COUNT. What if two cars enter and exit simultaneously? \* If COUNT=16, then the controller issues a FULL output signal, otherwise the OPEN output signal is on. \* There is a manual input for the lot attendant to override the COUNT for special vehicles. \* A START/STOP input is also available to the attendant, with the STOP input causing a CLOSED output signal. Use this systemverilog structure for the interfaces

```
package parking_pkg;
    timeunit 1ns;
    timeprecision 100ps;
// Type definitions
    typedef enum bit[1:0] {NEUTRAL, STOP, START} gate_override_t;
    typedef enum bit {CLOSED, OPEN} gate_t;
    typedef enum bit {ENABLE_COUNTER, BYPASS_COUNTER} counter_cntrl_t;
endpackage : parking_pkg
```

```
interface parking_structure_if (input wire clk, reset_n);
```

```
    timeunit 1ns;
    timeprecision 100ps;
    gate_t      gate_status;
    bit         parking_full;
    bit         entrance_sensor;
    bit         exit_sensor;
    gate_override_t gate_override;
    counter_cntrl_t counter_cntrl;
```

```
    modport parking_cntrl_if (
        output gate_t      gate_status,
        output bit         parking_full,
        input bit          entrance_sensor,
        input bit          exit_sensor,
        input gate_override_t gate_override,
        input counter_cntrl_t counter_cntrl);
endinterface : parking_structure_if
```

[bing GPT-4] Here are some properties that define the requirements for the parking lot controller using SystemVerilog and assertions:

1. The `COUNT` property: `

```
assert property (@(posedge clk) disable iff (!reset_n)  
    (counter_cntrl == ENABLE_COUNTER) |-> (COUNT <= 16));`
```

This property ensures that the `COUNT` of cars parked in the lot never exceeds the maximum capacity of 16.

2. The `FULL` property: `

```
assert property (@(posedge clk) disable iff (!reset_n)  
    (COUNT == 16) |-> parking_full);`
```

This property ensures that when the `COUNT` reaches 16, the `parking\_full` signal is asserted.

3. The `OPEN` property: `

```
assert property (@(posedge clk) disable iff (!reset_n)  
    (COUNT < 16) |-> !parking_full);`
```

This property ensures that when the `COUNT` is less than 16, the `parking\_full` signal is de-asserted.

4. The `entrance\_sensor` and `exit\_sensor` properties:

```
`assert property (@(posedge clk) disable iff (!reset_n)  
    (entrance_sensor && !exit_sensor) |-> (COUNT == $past(COUNT)+1));`  
`assert property (@(posedge clk) disable iff (!reset_n)  
    (!entrance_sensor && exit_sensor) |-> (COUNT == $past(COUNT)-1));`
```

These properties ensure that when a car enters or exits the lot, the `COUNT` is incremented or decremented accordingly.

5. The `gate\_override` property:

```
`assert property (@(posedge clk) disable iff (!reset_n)  
    (gate_override == STOP) |-> gate_status == CLOSED);`
```

This property ensures that when the attendant issues a STOP command, the gate is closed.

Is there anything else you would like to know?

[Ben] create the RTL design for this parking lot  
Here is an example of an RTL design for the parking lot controller using SystemVerilog:

```
``systemverilog
module parking_cntlr (
    input wire clk,
    input wire reset_n,
    input wire entrance_sensor,
    input wire exit_sensor,
    input gate_override_t gate_override,
    input counter_cntrl_t counter_cntrl,
    output reg [3:0] COUNT,
    output reg gate_t gate_status,
    output reg parking_full
);

import parking_pkg::*;

always_ff @(posedge clk or negedge reset_n) begin
    if (!reset_n) begin
        COUNT <= 4'd0;
        gate_status <= OPEN;
        parking_full <= 1'b0;
    end else begin
        if (counter_cntrl == ENABLE_COUNTER) begin
            if (entrance_sensor && !exit_sensor) begin
                COUNT <= COUNT + 1;
            end else if (!entrance_sensor && exit_sensor) begin
                COUNT <= COUNT - 1;
            end
        end
        if (gate_override == STOP) begin
            gate_status <= CLOSED;
        end else if (gate_override == START) begin
            gate_status <= OPEN;
        end
        if (COUNT == 16) begin
            parking_full <= 1'b1;
        end else begin
            parking_full <= 1'b0;
        end
    end
end

endmodule : parking_cntlr
``
```

This design implements the requirements specified in the problem statement. The `COUNT` is incremented or decremented based on the `entrance\_sensor` and `exit\_sensor` inputs. The `gate\_status` is controlled by the `gate\_override` input. The `parking\_full` signal is asserted when the `COUNT` reaches 16.

Is there anything else you would like to know?

## [Ben] Assertion set I created

```
// The steps involved in translating these requirements to SVA include the following:
// 1.   Define objects and types for these objects
// 2.   Define interfaces for module.
// 3.   Map English requirements to SVA. Method used for this application is first copying each English requirement
into the module as comments, and then mapping the requirements to SVA.
// // Define objects and types for these objects
package parking_pkg;
    timeunit 1ns;
    timeprecision 100ps;
// Type definitions
    typedef enum bit[1:0] {NEUTRAL, STOP, START} gate_override_t;
    typedef enum bit {CLOSED, OPEN} gate_t;
    typedef enum bit {ENABLE_COUNTER, BYPASS_COUNTER} counter_cntrl_t;
endpackage : parking_pkg

// Define interfaces for module.
// compilation-unit scope
import parking_pkg::*;
interface parking_structure_if (input wire clk, reset_n);
    timeunit 1ns;
    timeprecision 100ps;
    gate_t      gate_status;
    bit         parking_full;
    bit         entrance_sensor;
    bit         exit_sensor;//CVC change ""-> ";
    
    gate_override_t gate_override;
    counter_cntrl_t counter_cntrl;

    modport parking_cntrl_if (
        output gate_t      gate_status,
        output bit         parking_full,
        input bit         entrance_sensor,
        input bit         exit_sensor,
        input gate_override_t gate_override,
        input counter_cntrl_t counter_cntrl);
endinterface : parking_structure_if

// compilation-unit scope
import parking_pkg::*;
//Module definition
module parking_structure (input clk, input reset_n, parking_structure_if.parking_cntrl_if p_if);

// Object definition that relate to requirements
// * The controller maintains the current COUNT of the total number of cars
//   parked in the lot.
// * The lot has a limited number of parking spots, for simplicity sixteen (16).
    logic [4:0] car_counter; // counter of parked cars
    parameter MAX_SPOTS = 16;
```

```

// A START/STOP input is also available to the attendant,
// with the STOP input causing a CLOSED output signal.
// use the gate_override

// * There are two input sensors at the entrance and exit of the lot.
// The sensors provide input signals to the controller as cars pass by and
// the controller increments/decrements the current COUNT. What if two cars
// enter and exit simultaneously?
// * If COUNT=16, then the controller issues a FULL output signal,
// otherwise the OPEN output signal is on.
//What if two cars enter and exit simultaneously?
property parking_full_to_count_chk;
  @ (posedge clk) car_counter == MAX_SPOTS |-> p_if.parking_full;
endproperty : parking_full_to_count_chk
assert property parking_full_to_count_chk;

// From Spec: Otherwise the OPEN output signal is on.
// * There is a manual input for the lot attendant to override the COUNT
// for special vehicles.
// * A START/STOP input is also available to the attendant, with the STOP
// input causing a CLOSED output signal.
property gate_open_status_chk;
  @ (posedge clk) p_if.gate_status == OPEN  |->
    (car_counter < MAX_SPOTS || $past(p_if.gate_override) == START);
endproperty : gate_open_status_chk
gate_open_status_chk_1 assert property gate_open_status_chk;

property gate_closed_status_chk;
  @ (posedge clk) p_if.gate_status == CLOSED  |->
    (car_counter == MAX_SPOTS || $past(p_if.gate_override) == STOP);
endproperty : gate_closed_status_chk
gate_closed_status_chk_1 : assert property gate_closed_status_chk;

// Value change in the car counter
property counter_val_change_chk;
  @ (posedge clk) (car_counter != $past(car_counter))  |->
    p_if.counter_cntrl==ENABLE_COUNTER && // control is enabled
    !($past(p_if.entrance_sensor) && $past(p_if.exit_sensor)) //No simultaneous entrance and exit
    ($past(p_if.entrance_sensor) || $past(p_if.exit_sensor)); // either an entrance or an exit
endproperty : counter_val_change_chk
counter_val_change_chk_1 : assert property counter_val_change_chk;

// Counter has no change
property counter_val_no_change_chk;
  @ (posedge clk) (car_counter == $past(car_counter))  |->
    !($past(p_if.entrance_sensor) or $past(p_if.exit_sensor)) or // no entrance or exit
    ($past(p_if.entrance_sensor) && $past(p_if.exit_sensor)) or // simultaneous entrance and exit
    p_if.counter_cntrl==BYPASS_COUNTER; // in BYPASS mode
endproperty counter_val_no_change_chk;
counter_val_no_change_chk_1 : assert property counter_val_no_change_chk;

```

```
// Overflow detection
property counter_overflow;
    @ (posedge clk) not(car_counter == MAX_SPOTS + 1);
endproperty : overflow
counter_overflow_1 : assert property counter_overflow else
    $display("\n %m counter overflow Prev. value %d Cur Value %d ",
        $past(car_counter), car_counter);
endmodule // parking_structure
```