Overview Task Description

Welcome to this year's *Digital Integrated Circuits* course! Within the course, you will have to solve some assignments that will teach you fundamental concepts of integrated circuits design.

We will demonstrate digital integrated circuit design on a step-by-step example of a digital counter. It is important that you get a feeling for the result of your design decisions, so we will draw attention on the results of the synthesis steps, which you will verify with several tools along the digital design flow.

So, let's get started! Your task is to design a synchronous counter that runs in two modes, based on a *mode* switch. The two modes are called *up* and *down*. In *up* mode, the counter increments the counter value by **5** at the rising edge of the clock signal, and in *down* mode, it decrements the counter value by **9** at the rising edge of the clock signal. The counter should implement reset behavior, such that it is set to an initial value of **-16** upon reset. The counter should never exceed a value of **244**, and should never go below a value of **-349** (the counter value sticks near the minimum/maximum value, until the counting direction changes). The counter value should never be **-5** (the counter should jump over this value).

Task 1: Properties first!

Your first task is to create a formal specification for your counter. Take the informal, human-language description of the counter, and formulate *SystemVerilog Assertion (SVA)* properties that constrain the behavior of your counter. Name the signals as follows:

Table 1: Signal names for the counter

Signal name	description	width
cnt	Counter value	10
mode	Counter mode	1
clk	Clock	1
rst	Reset	1

If the *mode* signal is '0', the counter is in *down* mode, and in *up* mode otherwise.

Specify properties for your counter as given in in Listing 1 and provide them in a file called *properties.sv*. Your properties should check that:

1. The counter value always is in the specified range, and that it never takes the *invalid* value
2. The counter increment is always as specifed, except when the *invalid* value is jumped over
3. The counter decrement is always as specifed, except when the *invalid* value is jumped over
4. The increment/decrement can be different after the first clock cycle, when *rst* goes from '1' to '0'
5. The increment/decrement can be different when the counter value remains near the minimum/maximum value

Listing 1 shows a skeleton file into which you can add more properties. Two properties are already given: The first one checks whether the counter register is set to its correct initial value upon reset. The second property checks whether the counter value is always lower than or equal to the maximum value.

Your task is to extend this file with more properties. You will need a minimum of SystemVerilog for this task. Get yourself familiar with the SystemVerilog *if/else* statement. Also, the *$past* statement will be helpful to solve your task.

Listing 1: Skeleton file *properties.sv* to verify your counter

```
module ctb #(
        ) (
        input clk,
        input rst,
        input mode,
        output signed[10-1:0] cnt
        );

        counter i_counter (.*);

        bind
        counter : i_counter counter_sva i_sva (.*);

endmodule

```

```

module counter_sva (
	input clk,
	input rst,
	input mode,
	input signed[10-1:0] cnt
	);

	reg init = 1;

	always @(posedge clk) begin
		if (init)	assume(rst);
		else assume(!rst);
		init <= 0;
	end

	always @(posedge clk) begin

		if (rst) begin
			assert (cnt == -16);
		end

		if (!rst) begin

			//
			// Check if counter value is never lower than MIN, larger than MAX, or
			// equal to INV
			//
			assert (cnt <= 244);
			//
			// TODO: Add more properties here!

			//
			// Check if the counter value is correctly incremented and decremented
			//

			// Counting up
			//
			// TODO: Add your properties here!

			// Counting down
			//
			// TODO: Add your properties here!

		end

	end // Process

endmodule
```

We provide you a counter implementation in the file *counter.v* such that you can check your properties.

You can check the properties with *SymbiYosys*. Run it with the command given in Listing 2.

Listing 2: Command to invoke *SymbiYosys*, using the configuration file *counter.sby*

```
sby counter.sby
```

The command given in Listing 2 reads a configuration file *counter.sby*, which is given in Listing 3.

Listing 3: Command to invoke *SymbiYosys*, using the configuration file *counter.sby*

```
[tasks]
prove

[options]
prove: mode prove

[engines]
prove: abc pdr

[script]
read -formal properties.sv counter.v
verific -import ctb
prep -top ctb

[files]
properties.sv
counter.v
```

If the counter does not satisfy your properties (or vice versa), SymbiYosys generates a trace describing the error. You can inspect the waveform (*trace.vcd*) using GTKWave.