Verilog Meetup in LA - Interview Preparation

Yuri Panchul
2024-08-31

To run a checker for your answers on your own computer, please do the following:

git clone https://github.com/yuri-panchul/basics-graphics-music
cd basics-graphics-music
git checkout new_graphics
cd labs/9_meetup_in_la

Then, for each problem, cd to the problem directory and run the script 02_simulate_rtl.bash.
If the following problems are too complex for you at the moment, you can start with more simple problems in the repository https://github.com/yuri-panchul/systemverilog-homework

Problem 1. Implement serial adder with valid and last signal

```
module serial_adder_with_vld_and_last
(
  input  clk,
  input  rst,
  input  vld,
  input  a,
  input  b,
  input  last,
  output sum
);
```

Problem 2. Implement 1-to-2 gearbox with flow control

```
module gearbox_1_to_2
# (
  parameter width = 0
)
(
  input                     clk,
  input                     rst,

  input                     up_vld,    // upstream
  output                    up_rdy,
  input  [    width - 1:0] up_data,

  output                    down_vld,  // downstream
  input                     down_rdy,
  output [2 * width - 1:0] down_data
);
```

Problem 3. Implement 2-to-1 gearbox with flow control

```
module gearbox_2_to_1
# (
  parameter width = 0
)
(
  input                     clk,
  input                     rst,

  input                     up_vld,    // upstream
  output                    up_rdy,
  input  [2 * width - 1:0] up_data,

  output                    down_vld,  // downstream
  input                     down_rdy,
  output [    width - 1:0] down_data
);
```

Problem 4. Implement an adder of two numbers with separate flow control for arguments and the result.

```
module a_plus_b_with_flow_control
# (
    parameter width = 8, depth = 10
)
(
    input                clk,
    input                rst,

    input                a_valid,
    output               a_ready,
    input  [width - 1:0] a_data,

    input                b_valid,
    output               b_ready,
    input  [width - 1:0] b_data,

    output               sum_valid,
    input                sum_ready,
    output [width - 1:0] sum_data
);
```

Problem 5. Modify the following shift register design to reduce dynamic power

```
module shift_register_with_valid_and_debug
# (
    parameter width = 256, depth = 10
)
(
    input                clk,
    input                rst,

    input                in_valid,
    input  [width - 1:0] in_data,

    output               out_valid,
    output [width - 1:0] out_data
);

    logic [depth - 1:0] valid;
    logic [width - 1:0] data [0: depth - 1];

    always_ff @ (posedge clk or posedge rst)
        if (rst)
            valid <= '0;
        else
            valid <= { valid [$left (valid) - 1:0], in_valid };

    always_ff @ (posedge clk)
    begin
        data [0] <= in_data;

        for (int i = 1; i < depth; i ++)
            data [i] <= data [i - 1];
    end

    assign out_valid = valid [depth - 1];
    assign out_data  = data  [depth - 1];

endmodule
```