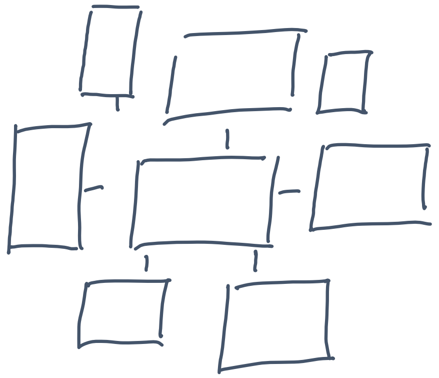


FPGA-based SoC Design

International Master of Science in Electrical Engineering

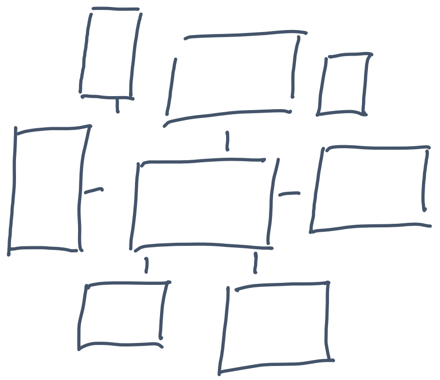
Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Design Verification

Today’s Agenda

Intended topics for today’s session

- Introduction to SystemVerilog – Basic Testbench Design
- Testbench based Functional Design Simulation using Intel Quartus Prime Lite, SystemVerilog HDL and Mentor ModelSim
- Live Demonstration



FPGA-based SoC Design

International Master of Science in Electrical Engineering

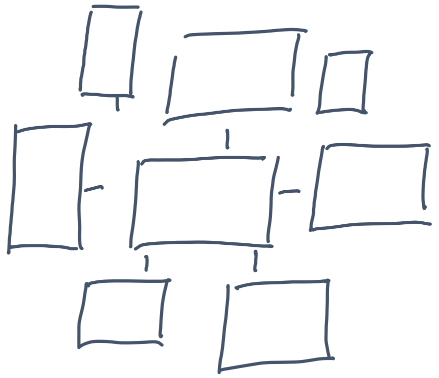
Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Design Verification

Recommended Readings

Textbooks, Application Notes, White Papers …

- Sutherland, S., "RTL Modeling with SystemVerilog for Simulation and Synthesis: Using SystemVerilog for ASIC and FPGA Design", CreateSpace Independent Publishing Platform, 2017
- Spear, C., "SystemVerilog for Verification: A Guide to Learning the Testbench Language Features", Springer, 3rd edition, 2012.



Introduction to SystemVerilog – Part#5

International Master of Science in Electrical Engineering

Prof. Dr. C. Jakob
University of Applied Sciences Darmstadt
h_da
Faculty of Electrical Engineering and Information Technology
fbeit

Basic Testbench Design using SystemVerilog HDL

Introduction to SystemVerilog

- Timing control in SystemVerilog HDL testbenches

Include unit delay times using the # operator ...

```
#20                      // Delay the execution of the respective block by 20ns...
#(10*CLK_PERIOD)         // ...
```

SystemVerilog Testbench Design - Delay operator

Event control – Rising/Falling Signal Edges

```
@(posedge clk)           // event control, wait for the rising edge of the clk
```

SystemVerilog Testbench Design – Event Control

Event control – Signal Levels ...

```
wait((state == IDLE) && (start_sys == 1’b1))     // Level sensitive event
                                                 // control ...
```

SystemVerilog Testbench Design – Event Control

Basic Testbench Design using SystemVerilog HDL

Introduction to SystemVerilog

- Tasks

Test Vector Generation

- Very often in testbenches, the same piece of code is repeated several times.
- This is often the case when different sets of inputs are applied.

```
task SEND_ACKNOWLEDGE;          // semicolon needed
   begin                        // begin-end necessary
      // Send an acknowledge
      tb_ack = 1;
      @(posedge tb_local_clock);
      # 1;
      tb_ack = 0;
      @(posedge tb_local_clock);
      # 1;
   end                          // begin-end necessary
endtask                         // no semicolon
```

SystemVerilog Task

Basic Testbench Design using SystemVerilog HDL

Introduction to SystemVerilog

- Basic Testbench configuration using Intel Quartus Prime Lite and Mentor ModelSim

Basic Testbench Configuration

using Intel Quartus Prime Lite and Mentor ModelSim

Launching **ModelSim** from Quartus Prime

1. Open the settings dialog box by clicking **Assignment | Settings** menu or by using the keyboard shortcut Ctrl+Shift+E
2. Select the **EDA Tool Settings | Simulation** option in the sidebar
3. Enable testbench by selecting the option Compile testbench
4. Click **Testbenches**… button to create testbenches
5. Click New button to create new testbench
6. In the **New Testbench Settings** dialog, provide a name for the testbench
7. Specify the top-level module in your testbench file
8. Add the testbench file by clicking the File name ellipsis button
9. Click **Ok** to create a new testbench
10. Afterwards, click **Apply** in Settings dialog box
11. To launch ModelSim simulator, you have to first synthesize your design by clicking the Start Analysis & Synthesis icon or by using the Ctrl+K shortcut (Make sure that your design is set as the current top-level entity - **Project | Set as Top-Level entity**)
12. After the Analysis & Synthesis process is successfully completed, launch the simulation by **Tools | Run Simulation | RTL Simulation**

Basic Testbench Design using SystemVerilog HDL

Introduction to SystemVerilog

- Time Base Generation Core – Using an extended testbench concept for LED simulation purposes

SystemVerilog Testbench ...

1/2

```
`timescale 1ns/1ps
`define HALF_CLOCK_PERIOD   10
`define RESET_PERIOD        200
`define SIM_DURATION        50000

module time_base_generation_tb();
  logic tb_q;
  // ### clock generation process ...
  logic tb_local_clock = 0;
  initial
     begin: clock_generation_process
        tb_local_clock = 0;
        forever
          #`HALF_CLOCK_PERIOD tb_local_clock = ~tb_local_clock;
     end

 logic tb_local_reset_n = 0;
```

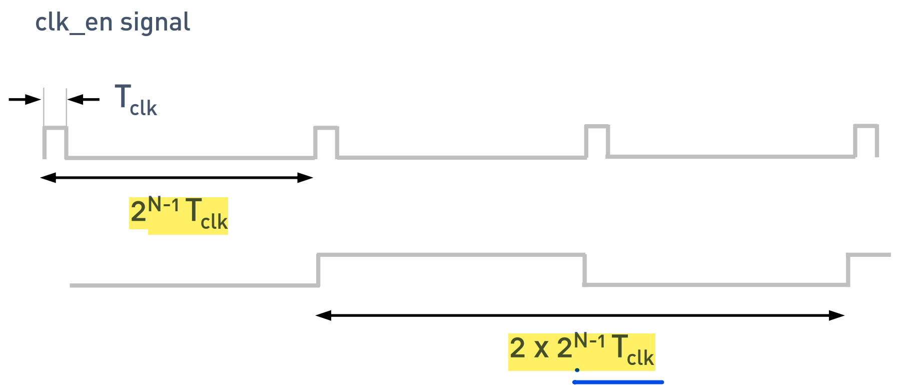
[SystemVerilog] Source Code: time_base_generattion_v2_tb.sv

SystemVerilog Testbench ...

2/2

```
initial
     begin: reset_generation_process
         $display ("Simulation starts ...");
         #`RESET_PERIOD tb_local_reset_n = 1'b1;
         #`SIM_DURATION
         $stop();
     end

time_base_generator #(.CLK_CYCLES(1025)) inst_0 (
                      .clk(tb_local_clock),
                      .reset_n(tb_local_reset_n),
                      .q(tb_q)
                     );
logic tb_test_led;
always_ff@(posedge tb_local_clock)
    if(tb_local_reset_n == 1'b0)
       tb_test_led <= 0;
    else
       if(tb_q == 1) tb_test_led <= ~ tb_test_led;
endmodule
```



[SystemVerilog] Source Code: time_base_generation_v2_tb.sv

Basic Testbench Design using SystemVerilog HDL

Introduction to SystemVerilog

- More useful testbench features …

4-bit Timer with Integrated Overflow Indicator

1/1

```
module count_up_4bit_co(
  input logic clk, input logic reset_n,
  output logic [3:0] q,
  output logic q_co
  );
  always_ff@(posedge clk)
    if(reset_n == 1‘b0)
        q <= 4'b0000;
    else
        q <= q + 1'b1;
  logic reg_tmp;
  always_ff@(posedge clk)
    if(reset_n == 1‘b0)
        reg_tmp <= 1'b0;
    else
        reg_tmp <= q[0];

  assign q_co = (q == 4'b0000) ? reg_tmp : 1'b0;

endmodule
```

[SystemVerilog] Source Code: count_up_4bit_co.sv

Corresponding Testbench ...

1/2

```
`timescale 1ns/1ps
`define HALF_CLOCK_PERIOD   5
`define RESET_PERIOD    20
`define SIM_DURATION    1000

 module count_up_4bit_co_v2_tb();
   logic [3:0] tb_q_count; logic tb_q_co;
   logic tb_local_clock = 0;
   initial
      begin: clock_generation_process
         tb_local_clock = 0;
         forever #`HALF_CLOCK_PERIOD tb_local_clock = ~tb_local_clock;
      end
   logic tb_local_reset_n = 0;
   initial
   begin: reset_generation_process
     $display ("Simulation starts ...");
     #`RESET_PERIOD tb_local_reset_n = 1'b1;
     #`SIM_DURATION $stop();
   end
```

[SystemVerilog] Source Code: count_up_4bit_co_tb.sv

Corresponding Testbench ...

2/2

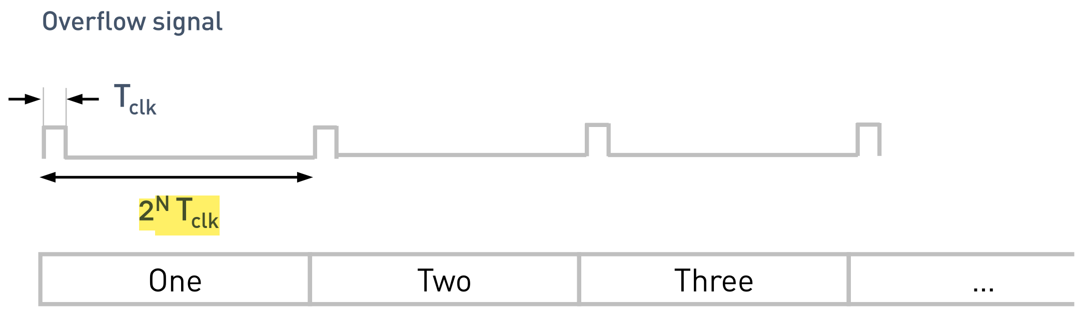
```
count_up_4bit_co inst_dut( .clk(tb_local_clock),
                           .reset_n(tb_local_reset_n),
                           .q(tb_q_count),
                           .q_co(tb_q_co));

logic [3:0] overflow_count = 0;
string state_string;                                          // ASCII string ...

always_ff@(posedge tb_q_co) begin
   $display("Overflow detected!\n");
   @(posedge tb_local_clock); @(posedge tb_local_clock); // Two clock delays ...
   overflow_count = overflow_count + 1'b1;
   case(overflow_count)
      4'b0000: state_string = "Zero";
      4'b0001: state_string = "One";
      4'b0010: state_string = "Two";
      4'b0011: state_string = "Three";
      4'b0100: state_string = "Four";
    endcase
end
endmodule
```

[SystemVerilog] Source Code: count_up_4bit_co_tb.sv

4-bit Timer with Integrated Overflow Indicator



Notes

- A simple way to improve the readability of a timing diagram ...
- In the ModelSim waveform viewer, make sure that the ASCII string is also formatted as an ASCII string ;-)