HDL Design Homework: SystemC vs SystemVerilog

Yousef Awad
EEL 4783 - HDL in Digital Systems

March 1, 2026

Part 1: SystemC

1.1: Counter Module

a) The completed module skeleton

```
#include <systemc>

using namespace sc_core;
using namespace sc_dt;

SC_MODULE(Counter4) {
  sc_in<bool> clk, reset;
  sc_out<sc_uint<4>> count;
  sc_uint<4> current_val; // Internal state

  void count_proc() {
    // Synchronous active-high reset
    if (reset.read() == true) {
      current_val = 0;
    } else {
      // sc_uint<4> automatically wraps at 15
      current_val = current_val + 1;
    }
    count.write(current_val);
  }

  SC_CTOR(Counter4) {
    SC_METHOD(count_proc);
    sensitive << clk.pos(); // Trigger on rising edge of clock
  }
};
```

b) Why is SC_METHOD (not SC_THREAD) appropriate here?

`SC_METHOD` is ideal because it executes its entire block of code and returns control to the simulator without needing to be suspended. Since the requirements explicitly state not to use `wait()`, `SC_METHOD` is the correct choice over `SC_THREAD` (which requires `wait()` to suspend and resume execution).

1.2: Testbench Snippet

```
#include "counter.cpp"
#include <systemc>

using namespace sc_core;
using namespace sc_dt;

int sc_main(int argc, char *argv[]) {
  // 1. Instantiate signals and clock (10ns period, 50% duty cycle)
  sc_clock clk("clk", 10, SC_NS, 0.5);
  sc_signal<bool> reset;
  sc_signal<sc_uint<4>> count;

  // 2. Instantiate Counter4
  Counter4 dut("dut");
  dut.clk(clk);
  dut.reset(reset);
  dut.count(count);

  // 3. Enable VCD tracing
  sc_trace_file *wf = sc_create_vcd_trace_file("counter_trace");
  sc_trace(wf, clk, "clk");
  sc_trace(wf, reset, "reset");
  sc_trace(wf, count, "count");

  // 4. Stimulus and Simulation Control
  std::cout << "Starting simulation..." << std::endl;

  reset.write(true);   // Assert reset
  sc_start(15, SC_NS); // For first 15ns

  reset.write(false); // Release reset

  // Run for the remainder of the 200ns total simulation time
  sc_start(185, SC_NS);

  // 5. Clean up
  std::cout << "Simulation finished. Trace saved to counter_trace.vcd"
            << std::endl;
  sc_close_vcd_trace_file(wf);

  return 0;
}
```

Part 2: SystemVerilog

2.1: Shift Register RTL

```
`timescale 1ns / 10ps

module shift_reg #(
    parameter DATA_W = 8
) (
    input logic clk,
    reset,
    shift_en,
    input logic [DATA_W-1:0] data_in,
    output logic [DATA_W-1:0] data_out
);

  // Array of registers for internal stages as hinted
  logic [DATA_W-1:0] stages[3:0];

  always_ff @(posedge clk) begin
    if (reset) begin
      stages[0] <= '0;
      stages[1] <= '0;
      stages[2] <= '0;
      stages[3] <= '0;
    end else if (shift_en) begin
      stages[0] <= data_in;
      stages[1] <= stages[0];
      stages[2] <= stages[1];
      stages[3] <= stages[2];
    end
    // If shift_en == 0, state is naturally held in always_ff
  end

  // Continuous assignment to the output port
  assign data_out = stages[3];

endmodule
```

2.2: Testbench Logic

```
`timescale 1ns / 10ps

module shift_reg_tb;

  // Inputs to the DUT are declared as 'reg'
  reg clk;
  reg reset;
  reg shift_en;
  reg [7:0] data_in;

  // Outputs from the DUT are declared as 'wire'
  wire [7:0] data_out;  // Assumed 8-bit output, adjust if your module differs

  // match your actual Verilog design file.
  shift_reg dut (
      .clk(clk),
      .reset(reset),
      .shift_en(shift_en),
```

```
      .data_in(data_in),
      .data_out(data_out)
  );

  // Generates a clock with a 10ns period, matching the #10 step delays in your
    stimulus
  initial begin
    clk = 0;
    forever #5 clk = ~clk;
  end

  // 4. Your Stimulus Block
  initial begin
    // Dumpfile setup for "shift.vcd"
    $dumpfile("shift.vcd");
    // Adjusted to match the testbench module name 'shift_reg_tb'
    $dumpvars(0, shift_reg_tb);

    // Reset sequence
    reset = 1;
    shift_en = 0;
    data_in = 8'hAA;
    #15;
    reset = 0;

    // Cycles 1-4: shift_en = 1, incrementing data_in
    shift_en = 1;
    data_in = 8'h11;
    #10;
    data_in = 8'h22;
    #10;
    data_in = 8'h33;
    #10;
    data_in = 8'h44;
    #10;

    // Cycles 5-6: shift_en = 0 (hold state)
    shift_en = 0;
    #20;

    // Cycle 7: shift_en = 1, data_in = 8'hFF
    shift_en = 1;
    data_in  = 8'hFF;
    #10;

    // Wait remaining time to hit 100ns total simulation time
    // Time elapsed: 15 (reset) + 40 (c1-4) + 20 (c5-6) + 10 (c7) = 85ns
    #15;

    $finish;
  end

endmodule
```

2.3: Concept Check

a) Why does SystemC use sc_signal instead of plain C++ variables for ports?
`sc_signal` implements the evaluate-update paradigm of hardware simulation, ensuring that updates are deferred until the next simulation delta cycle to prevent race conditions that plain C++ variables

would cause.

b) What critical error occurs if you declare data_out as wire instead of logic in Problem 2.1?

If you attempt to assign to `data_out` procedurally (inside the `always_ff` block) while it is declared as a `wire`, the SystemVerilog compiler will throw an error, because `wire` types can only be driven by continuous assignments (`assign`), whereas `logic` can be driven procedurally.