ELEC6044 Advanced Design 2008/09
Brief introduction to SystemVerilog

Tom J Kazmierski

tjk@ecs.soton.ac.uk, Zepler 4223, x 23520
What is SystemVerilog

- Hardware description language (HDL)
  - It is a new HDL (2005)
  - VHDL started in 1986, Verilog - 1995
- Hardware verification language
- It is an HDL that handles all aspects of the digital hardware design and verification flow:
  - design description
  - functional simulation
  - formal verification
- Unlike VHDL, SystemVerilog has been designed specifically for automated synthesis of digital system descriptions into hardware
module adder (  
// connections to the module:  
output logic Sum, Cout,  
input logic Cin, A, B);  
// logic operations  
assign Sum = Cin ^ A ^ B;  
assign Cout =  
    A & B | A & Cin | B & Cin;  
endmodule
module rslatch (input logic r, s, output logic q, qbar);

// assign statements are concurrent processes:
assign q = ~ (r | qbar);
assign qbar = ~ (s | q);

endmodule
D latch with enable

module dlatch (input logic d, en, output logic q);

// standard latched logic behaviour, synthesises as memory:
always_latch
  if (en)
    q <= d;

endmodule
D flip-flop – edge triggered

module dff (output logic q, qbar, input logic d, clk);
always_ff @ (posedge clk) // process with a sensitivity clause
begin
    q <= d ; // q changes when clk is rising
    qbar <= ~q;
end
endmodule
D flip-flop – edge triggered with asynchronous reset

Nb. the always_ff block imposes the restriction that only one event control is allowed.

module dff (output logic q, qbar, input logic d, clk);

always_ff @ (posedge clk iff reset==0 or posedge reset)
if (reset)
    q <= 1b'0;
else
    q <= d;
endmodule

alternative syntax:
q <= (reset? 1b'0:d);
Mult-bit D flip-flop registers

module dff (output logic[7:0] q, input logic [7:0] d, input logic clk, nReset);

always_ff @ (posedge clk iff nReset==1 or negedge nReset)
  if (~nReset)
    q <= 8b'0;
  else
    q <= d;
endmodule

module counter (output logic[15:0] q, input logic clk, nReset);

always_ff @ (posedge clk iff nReset==1 or negedge nReset)
  // arithmetic operations are allowed on logic objects:
  q <= (~nReset: 16b’0: q+1);
endmodule
State machines

At least two processes:

One to define the flip-flop operation:
always_ff @ (posedge clock)
present_state <= next_state

Another to define the combinational logic:
always // sensitivity is inferred automatically 

...
module parity (output logic state, input logic in, clk);
parameter even = 1'b0, odd = 1'b1;
logic state,next;

always_ff @ (posedge clk)
state <= next;

// combinational logic part:
always
  case (state)
    even: next <= (in ? odd : even);
    odd:   next <= (in ? even : odd);
  endcase

endmodule
Simulations in Modelsim and testbenches

module mux2 (input logic [1:0] A, input logic Sel, output logic Y) ;
    assign Y = (Sel ? A[0] : A[1]);
endmodule

// let us simulate a simple 2-input multiplexer
module test_mux ; // note the absence of external ports: this is the top module

// This file is the testbench for mux2.sv
// Provides stimuli for examining the multiplexer functionality

logic [1:0] A ; // declare local signals
logic  Sel, Y;

mux2 m (.*) ;   // Invoke module mux2; here local signals have the same types and names as mux2 ports,
//so they can be wired to mux ports automatically

initial // an initial process runs only once
  begin  // statements below specify signal changes at various times during simulation

    #10ns Sel = 1'b0 ;
    #20ns   A = 2'b0 ;
    #10ns   A = 2'b1 ;
    #10ns Sel = 1'b1;
    #20ns A[1] = 1'b1;
    #10ns Sel = 1'b0 ;

  end // initial

endmodule
Writing a testbench for sequential logic
Parity detector example

module parity (output logic state, input logic in, clk);
parameter even = 1'b0, odd = 1'b1;
logic state, next;

always_ff @ (posedge clk)
state <= next;

// combinational logic part:
always
  case (state)
    even: next <= (in ? odd : even);
    odd:   next <= (in ? even : odd);
  endcase
endmodule
// testbench for the parity detector
module test_parity;
// This is the testbench for parity.v
// Provides stimuli, i.e. the clock and input signal for verification of parity detector functionality

logic in, clk, reset, state;

parity pd (.*); // invoke an instance of the module parity;
   // this syntax maps local signals in,clk,state to corresponding ports
   // of module parity by the same names

initial // an initial process always runs once and is not triggered by events;
   // this is how a perpetual clock can be created
begin
clk = '0 ; // note new syntax for binary constant
forever #50ns clk = ~clk ; // perpetual 10MHz clock
end

...
A testbench for sequential logic

```verilog
initial // runs in parallel with the other initial process
    begin // this process provides stimuli for in and reset
        in = '0;
        reset = '1;

        #120ns reset = '0 ;
        #220ns  in = '1;
        #100ns in = '0;
        #65ns  in = '1;
        #225ns in = '0;
    end

endmodule
```