EE/CSE371 SystemVerilog Quick Reference Sheet

Max Arnold, Justin Hsia

Signal Basics

```
logic var1;         // single-bit wire/value
logic [7:0] var2;   // 8-bit packed array (bus)

4'b1101   // 4-bit constant (equivalently: 4'd13, 4'hD)

{expr1, ..., exprN}  // concatenation
{6{expr}}            // replication

parameter [1:0] S_idle = 2'd0;  // named parameter
enum {S_0, S_1} state;     // enumerated state variable
```

Basic Operators

```
~,  &,  |,  ^, ~&, ~|, ~^    // Boolean operators
!, &&, ||, ==    // logical operators

assign out = expr;  // continuous assignment

cond ? true_expr : false_expr  // ternary operator
```

"Control Flow"

```
begin      // start of code block, '{' in C/Java
   <code>
end        // end of code block, '}' in C/Java

if, else, for, while  // syntax similar to C/Java

case (state)          // case has no fallthrough
   2'd0:    <code>    // constant specified
   S_1:     <code>    // named value specified
   default: <code>    // catch-all case
endcase

repeat (num) <statement>;  // repeat num times
```

Procedural Blocks

```
always_comb begin           // for combinational logic
   var1 = var2 | var3;      // blocking assign (=)
end

always_ff @(posedge clk) begin  // for sequential logic
   var1 <= var2 | var3;     // non-blocking assign (<=)
end

initial           // starts at beginning of simulation
```

Module Definition

```
// separate port list and declarations
module calculate (in1, in2, out1);
   input logic in1;
   input logic [3:0] in2;
   output logic out1;
   <code>
endmodule

// combined port list and declarations
module calculate (input  logic in1,
                  input  logic [3:0] in2,
                  output logic out1);
   <code>
endmodule

module calculate_testbench ();  // no ports for
   <testbench code>             //   testbenches
endmodule
```

Module Instantiation

```
logic sig1, sig2;        // by position (discouraged)
calculate c1 (sig1, sig2);

logic in1, out1;         // implicitly by name
calculate c2 (.in1, .out1);

logic sig1, sig2;        // explicitly by name
calculate c3 (.in1(sig1), .out1(sig2);

logic in1, sig1, out1;   // match all remaining by name
calculate c4 (.*);
calculate c5 (.in1(sig1), .*);
```

Module Parameterization

```
module calc #(parameter width=4)  // parameter list
             ( <port list> );
   <code>
endmodule

calc #(8) c6 (.*);          // positional param
calc #(.width(8)) c7 (.*);  // explicit param
```

Testbench Timing Controls

```
// Note: oftentimes <statement> is empty

#10 <statement>;     // wait 10 time units
#(num) <statement>;  // wait num time units

// wait until next rising edge of clock
@(posedge clk) <statement>;
```

Testbench Simulated Clock

```
logic clk;
parameter CLOCK_PERIOD = 10;  // arbitrary choice
initial begin
   clk <= 0;
   forever #(CLOCK_PERIOD/2) clk <= ~clk;
end
```

System Tasks for Simulation

```
// outputs once when encountered (+newline)
$display(<format string>, …);

// outputs once when encountered (no newline)
$write(<format string>, …);

// outputs anytime one of its signal changes (+newline)
$monitor(<format string>, …);

// returns current time (in time format)
$time

// interrupts simulation (for examination)
$stop;
```

Format String Escape Sequences

```
"%0b %0h %0d"   // binary, hex, decimal

"%1.2f %1.2e"   // real (decimal), real (scientific)

"%c %s"         // ASCII character, string

"%0t"           // time format
```

Generate Statement

```
genvar i;
generate
   for (i = 0; i < 16; i++) begin : label1
      // structures to be generated go here, including
      // modules, always blocks, and assign statements.
      module1 m1 (.in(arr[i]), .out(outArr[i]));
      always_ff @(posedge clk)
         <code>
   end
endgenerate
```