Day 1

Universal Verification Methodology

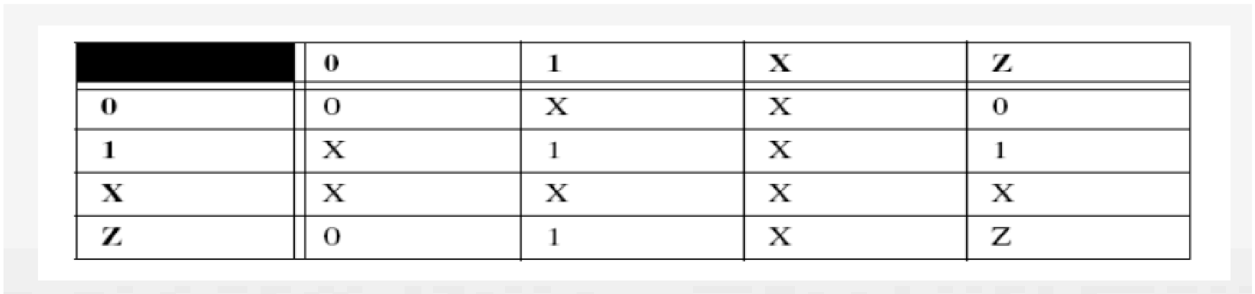
Intro

- Job of verification engineers
    - Reduce risk
        - Test chips before fab
    - Highest risk activity = debugging hardware
- How does UVM help
    - Create testbenches quick - architecture is provided by UVM code
- Agenda
    - SysVerilog for VHDL engineers
    - Object oriented programming
    - SysVerilog interfaces
    - Packages, includes, macros
    - UVM test objects
    - UVM environments
    - Transaction level testing
    - The analysis layer
    - UVM reporting
    - Functional coverage with covergroups
    - Introduction to sequences

SysVerilog for VHDL engineers

- Why SysVerilog
    - Test benches
        - OOP
        - Functional coverage
        - Randomization
        - Methodology libraries
    - RTL
        - Multithreaded
        - Hardware data types
        - Netlisting
- Diff in philosophy
    - VHDL - Contract
        - All terms defined
        - Many sections
        - Long, avoids ambiguity

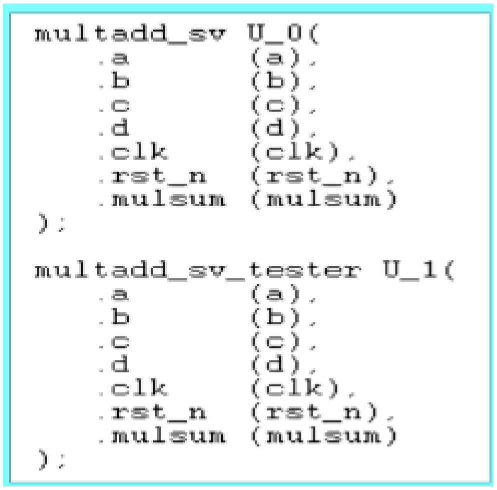
    - SysVerilog
        - Assumes common language
        - Paragraphs not sections
        - Short, quickly written
    - Gotta know what’s happening in the simulator for SysV
- SysV is case sensitive!
- Multithreaded behavior
    - process in VHDL is initial or always
        - Initial block runs once and disappears
        - Always runs infinitely
- Delays
    - wait(expr);
    - #100ns;
    - @signal;
    - All valid ways to implement waits
- Waiting on an edge
    - @(posedge sig)
    - @(negedge sig)
- SysV data values
    - 4-state values built into language (no libraries)
        - 1
        - 0
        - X
        - Z



	0	1	X	Z
0	0	X	X	0
1	X	1	X	1
X	X	X	X	X
Z	0	1	X	Z

Conflict Resolution Table

- Constants are not strings
    - Syntax for variables
        - <width>’<radix><numerals>
- Four State Types - hold 0, 1, X, Z
    - logic
    - reg (same as logic)
    - integer (32-bit)
    - time (64-bit)
    - **Unsigned by default**
- Two State Types - hold 0, 1
    - int (32-bit)

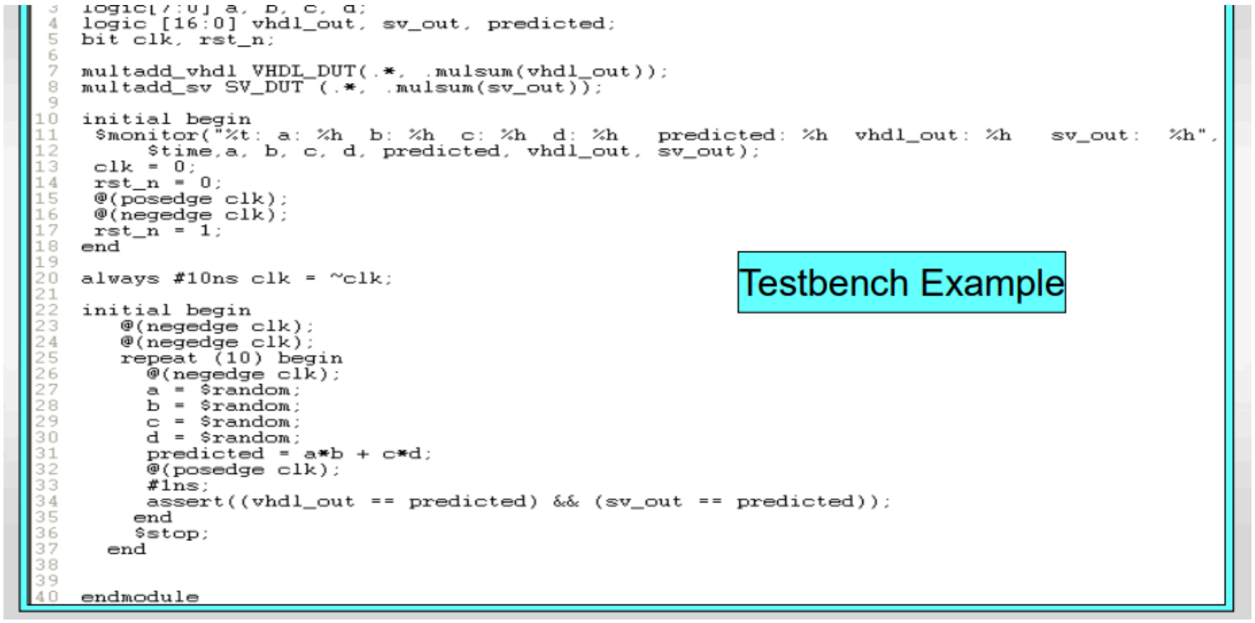
    - shortint (16-bit)
    - longint (64-bit)
    - byte (8-bit)
    - bit (1 bit)
    - **Signed by default**
- SysV converts between types automatically
    - When going from 4-state to 2-state types
        - X, Z → 0
    - 2-state to 4-state, 0, 1 are defined for both types
- Declaring registers and memories
    - logic [3:0] halfbyte;
    - logic [1:8] reversbits_mem[7:0];
        - Array
- Assigning diff widths
    - Truncation
        - 101011001010 → 11001010 (cut off the bits that were too big)
    - Signed extension
        - 11001010 → 111111001010 (signed bit was extended to target size)
    - Unsigned extension
        - 11001010 → 000011001010 (0s were added to the front)
- Concurrent assignment
    - VHDL
        - signal <= equation;
    - Verilog
        - assign signal = equation;
- Blocking and non-blocking assignment
    - = operator for blocking
    - <= operator for non-blocking
- Instantiation
    - 

```
multadd_sv U_0(
    .a      (a),
    .b      (b),
    .c      (c),
    .d      (d),
    .clk    (clk),
    .rst_n  (rst_n),
    .mulsum (mulsum)
);

multadd_sv_tester U_1(
    .a      (a),
    .b      (b),
    .c      (c),
    .d      (d),
    .clk    (clk),
    .rst_n  (rst_n),
    .mulsum (mulsum)
);
```

    - SystemVerilog Instantiation Shortcuts:

- Assuming the signals at the top level have the same names as the ports in the instantiated module:
    - <portname> matches the top level signal to the port of the same name.
    - * matches all top level signals to ports of the same name



```
logic[7:0] a, b, c, d;
logic [16:0] vhdl_out, sv_out, predicted;
bit clk, rst_n;

multadd_vhdl VHDL_DUT(.*, .mulsum(vhdl_out));
multadd_sv SV_DUT (.*, .mulsum(sv_out));

initial begin
 $monitor("%t: a: %h  b: %h  c: %h  d: %h   predicted: %h  vhdl_out: %h   sv_out:  %h",
     $time,a, b, c, d, predicted, vhdl_out, sv_out);
 clk = 0;
 rst_n = 0;
 @(posedge clk);
 @(negedge clk);
 rst_n = 1;
end

always #10ns clk = ~clk;

initial begin
   @(negedge clk);
   @(negedge clk);
   repeat (10) begin
     @(negedge clk);
     a = $random;
     b = $random;
     c = $random;
     d = $random;
     predicted = a*b + c*d;
     @(posedge clk);
     #1ns;
     assert((vhdl_out == predicted) && (sv_out == predicted));
    end
    $stop;
  end


endmodule
```

Testbench Example