Outline
1) Tip: Tinman
2) Testbench Discussion
3) Problem Set #1 Questions
4) ModelSim Tutorial Questions

Tip: Tinman
- A way to run Windows on a UNIX machine
- Type “sunpci” at the command prompt

General Description of a Testbench
1) Generates stimulus (i.e. test vectors)
   - Common sources: forced assignment, counters, shift registers
2) Monitors response (optional)
   - Compare result against known good result

Module/Testbench Parallelism

<table>
<thead>
<tr>
<th>Module</th>
<th>Testbench</th>
</tr>
</thead>
<tbody>
<tr>
<td>module counter (count, clk, rst); input clk, rst; output [7:0] count; reg [7:0] count;</td>
<td>module test_counter; reg clk, rst; wire [7:0] count; counter_dut(count,clk,rst);</td>
</tr>
</tbody>
</table>
**Issues Related to Testbenches**

*Denotes subjects that will be discussed later in the course.

- *Behavioral Verilog* (*initial, always, forever, etc.*)
  - Most common: *initial & always.*
  - Partitions code into group behavior (e.g. clock oscillation - see code below).

- Register variables (*reg, integer, real, time, realtime*)
  - Most common: *reg* (& *integer* for calculations).
  - Typically holds a single test vector.

- Continuous assignments (e.g. *assign a = b + c;*)
  - Alternative to behavioral statements.
  - Left-hand side must be net.

- *Blocking/non-blocking assignments* (*= / <=*)
  - In short, they’re the difference between serial and parallel assignment.
  - More complicated than this. Read the Cummings Paper.
  - Do NOT mix assignment types within a block of code.

- *Assignment delays* (e.g. *#5 a = 0;*)
  - Used to schedule assignments in simulation time.

- Module instantiation
  - Necessary for hierarchical/object-oriented design & testing designs.

- Port connectivity (by place or name)
  - Defines signal passing between modules and other structures, such as testbenches.

- System tasks and functions (*$monitor, $stime, $stop, $finish, etc.*)
  - Most common: *$stop.*
  - *$finish = $stop* + terminates ModelSim.

- *Compiler directives* (*`timescale, etc.*)
  - *`timescale* defines simulator’s time step and precision.
  - *`* is the stroke sharing the key with ~.

- Variable initialization
  - At *t = 0* in simulation, all variables equal to *x.*
  - Therefore, reset signal needed in design.

- Variable declarations (individual vs. listed)
  - “Listed” variables share all properties
• Keywords *begin & end*
  o Defines a block of code.
  o Equivalent to C’s curly brackets, { }.

• Design validation
  o One method is to test user’s design against known “good” design.
    ▪ This can within a testbench or using wave comparison.

• Equality
• Constants
Counter Module

module counter (count, clk, rst);
output [7:0] count;
input clk, rst;

reg [7:0] count;
parameter tpd_clk_to_count = 1; // Never used
parameter tpd_reset_to_count = 1; // Never used

// Never used
function [7:0] increment;
input [7:0] val;
reg [3:0] i;
reg carry;
begin
  increment = val;
carry = 1'b1;
  /*
  * Exit this loop when carry == zero, OR all bits processed
  */
  for (i = 4'b0; ((carry == 4'b1) || (i <= 7)); i = i + 4'b1)
    begin
      increment[i] = val[i] ^ carry;
carry = val[i] & carry;
    end
end
endfunction

/*****************************************************************
* The following always block was changed to make it synthesizable.
always @ (posedge clk or posedge rst)
  if (rst)
    count = #tpd_reset_to_count 8'h00;
  else
    count <= #tpd_clk_to_count increment(count);
******************************************************************/
always @ (posedge clk or posedge rst)
  if (rst)
    count = 8'h00;
  else
    count <= count + 8'h01;
endmodule
Testbench Module

module test_counter;
reg clk, rst;
wire [7:0] count;
counter #(5,10) dut (count,clk,rst);

initial // Clock generator
begin
    clk = 0;
    #10 forever #10 clk = !clk;
end

initial // Test stimulus
begin
    rst = 0;
    #5 rst = 1;
    #4 rst = 0;
    #50000 $stop;
end

initial
    $monitor($stime,, rst,, clk,,, count);
endmodule

Waveform Results