Announcements

• Homework #2 due Thurs.
  - Drop box in 240 Cory

• Next Tues. [9-20] lecture will be taped ahead
  - 12:30-2pm, 306 Soda
  - Please do attend if you can
EECS150 Design Methodology

Hierarchically define structure and/or behavior of circuit.

HDL Specification

Simulation
- Functional verification.

Synthesis
- Maps specification to resources of implementation platform (FPGA for us).

Let’s look at the other branch.

Design Verification

- Industrial design teams spend a large percentage of the design time on design verification:
  - Removing functional bugs, messaging the design to meet performance, cost, and power constraints.
- Particularly important for IC design, less so for FPGAs.
- A variety of tools and strategies are employed.
  - **Simulation**: software that interprets the design description and mimics signal behavior and timing (and power consumption).
    - Simulation provides better controllability and observability over real hardware. Saves on wasted development time and money.
  - **Emulation**: hardware platform (usually FPGAs) are used to mimic behavior of another system. “Fast simulation.”
  - **Static Analysis**: tools examines circuit structure and reports on expected performance, power, or compares alternative design representations looking for differences.
Simulation Testing Strategies

- **Unit Testing**: Large systems are often too complex to test all at once, so an bottom-up hierarchical approach. Sub-modules are tested in isolation.

- **Combinational Logic blocks**: when practical, exhaustive testing. Otherwise a combination of random and directed tests.

- **Finite state machines**: test every possible transition and output.

- **Processors**: use software to expose bugs.

- In all cases, the simulated output values are checked against the expected values. Expected values are derived through a variety of means:
  - behavior model running along side the design under test
  - precomputed inputs and outputs (vectors)
  - co-simulation. Ex: C-language model runs along side ModelSim

Simulation

- Verilog/VHDL simulators use 4 signal values: 
  0, 1, X (unknown), Z (undriven)

- **Simulation algorithm typically ”discrete event simulation”**
Discrete Event Simulation Engine

- A time-ordered list of events is maintained
  
  Event: a value-change scheduled to occur at a given time

- All events for a given time are kept together
  
  The scheduler removes events for a given time ...

... propagates values, executes models, and creates new events ...

---

Testbench

Top-level modules written specifically to test other modules.

```verilog
module testmux;

  reg a, b, s;
  wire f;
  reg expected;

  mux2 myMux (.select(s), .in0(a), .in1(b), .out(f));

initial begin
  s=0; a=0; b=1; expected=0;
  #10 a=1; b=0; expected=1;
  #10 s=1; a=0; b=1; expected=1;
end

initial $monitor(
  "select=%b in0=%b in1=%b out=%b, expected out=%b time=%d",
  s, a, b, f, expected, $time);
endmodule // testmux
```

- Instantiation of DUT (device under test).
- Initial block similar to “always” block without a trigger. It triggers once automatically at the beginning of simulation. (Also supported on FPGAs).
- “#n” used to advance time in simulation. Delays some action by a number of simulation time units.
- Note use of blocking assignments.
- Note multiple initial blocks.
- A variety of other “system functions exist for displaying output and controlling the simulation.

---
Mux4 Testbench

module testmux4;
  reg [5:0] count = 6'b000000;
  reg [1:0] S;
  wire f;
  mux4 myMux (.select(S), .in0(a), .in1(b), .in2(c), .in3(d), .out(f));
  initial
  begin
    repeat(64)
    begin
      {S, d, c, b, a} = count[5:0];
      case (S)
        2'b00: expected = a;
        2'b01: expected = b;
        2'b10: expected = c;
        2'b11: expected = d;
      endcase // case(S)
      $strobe( "select=%b in0=%b in1=%b in2=%b in3=%b out=%b, expected=%b time=%d", S, a, b, c, d, f, expected, $time);
      #8 count = count + 1'b1;
    end
  end
endmodule

Alternative to $strobe in this case, #8 if (f != expected) $display("Mismatch: ...");

How To Create Clocks
How To Create Clocks (II)

module testFSM;
reg in;
wire out;
reg clk=0, rst;
reg expect;
FSM1 myFSM (.out(out), .in(in), .clk(clk), .rst(rst));
always #5 clk = ˜clk;
initial
begin
rst=1;
#10 in=0; rst=0; expect=0;
#10 in=1; rst=0; expect=0;
#10 in=0; rst=0; expect=0;
#10 in=1; rst=0; expect=0;
#10 in=1; rst=0; expect=1;
#10 in=1; rst=0; expect=1;
#10 in=0; rst=0; expect=0;
#10 $stop;
end
always
begin
strobe($time," in=%b, rst=%b, expect=%b out=%b", in, rst, expect, out);
end
endmodule

FSM Testbench Example

Test all arcs.

DUT instantiation

100MHz clk signal

Self-loop

start in IDLE

transition to S0

transition to IDLE

transition to S0

transition to S1

transition to IDLE

Note: Input changes are forced to occur on negative edge of clock.

Strobe output occurs 1ns before rising edge of clock.

Debug is easier if you have access to state value also.

Either 1) bring out to ports, or 2) use waveform viewer.
**Final Words (for now) on Simulation**

Testing is not always fun, but you should view it as part of the design process. Untested potentially buggy designs are a dime-a-dozen. Verified designs have real value.

Devising a test strategy is an integral part of the design process. It shows that you have your head around the design. It should not be an afterthought.