"If you don’t test it, it isn’t going to work"
- Mark Horowitz

Testing

- Logic blocks (*.v)
  - Will be synthesized into hardware circuits
- Testing blocks (*.vt)
  - Pretty much anything goes
- Instantiate logic blocks inside testing blocks; drive inputs and check outputs there
- Examples of things that are ok in *.vt but not ok in *.v modules in 281 unless you are told otherwise
  - “#” delay statements are essential in testing modules and should never be in hardware except D FFs for “clock to Q” delays
  - “signed” regs and wires are extremely useful for printing 2’s complement signals
  - “for” loops

B. Baas, EEC 281
Example test module

• Basic flow Approach 1
  – All signals are generated by test code in the test module
  – Easier and quicker to set up

Example test module

• Basic flow Approach 2
  – Both “test” block and “hardware” block are coordinated by the same clock signal which is generated by an independent clock oscillator module in the test module
  – Better for more complex systems
Example test module

• Basic flow Approach 2
  – A very simple clock oscillator could look like this:

```verilog
initial begin
  clock = 1'b0; // must initialize...
  #10000;       // main simulation
  $finish;      // stop simulation
end
// osc inverts clock value every #100
always begin
  #100;         // cycle time = #200
  clock = ~clock;
end
```

  – A slightly better design would use a clock reset signal

```verilog
initial begin
  reset    = 1'b1;
  data     = 8'h00;
  clock    = 1'b0; // must initialize
  #100;       // main simulation
  $finish;      // stop simulation
end
// osc inverts clock value every #100
always begin
  #100;         // cycle time = #200
  clock = ~clock;
end
```

Example test module

• Basic flow Approach 2
  – Here is an example of how code in the test generator could look:

```verilog
initial begin
  reset    = 1'b1;
  data     = 8'h00;
  clock    = 1'b0; // must initialize
  #100;       // main simulation
  $finish;      // stop simulation
end
// osc inverts clock value every #100
always begin
  #100;         // cycle time = #200
  clock = ~clock;
end
```
Example Given To You

- Example test bench and modules on web page found with the link: [Notes on running verilog](#)
- The example is not the most realistic partitioning (e.g., we would normally never put D FFs in their own modules), but it is a good working example to get you started
- Other files
  - `tbench.vf`
    - Contains files needed for simulator
  - `Makefile`
    - Very handy method for typing commands

Verification

- A number of ways to verify designs
  1) Eyeball text printouts
    - Quickest and easiest
  2) Eyeball waveforms
    - Quick and easy
  3) Write reference code and some other code to see if the two are the same. Make sure you temporarily force an error to test your setup.
    - This is the most robust and is what is required for non-trivial designs
Verifying Correctness

- As designs become more complex, verifying it is correct becomes more difficult
- “Golden reference” approach
  - Write an easy to understand simple model in a higher-level language
    - C or matlab commonly
      - Matlab is a natural choice for DSP applications
    - Must be written in a different way from the verilog implementation to avoid repeating the same bugs
  - Designers agree this is the correct function (imagine a detailed design review)
  - Many high-level tests run on golden reference
    - Model should be fast

Verifying Correctness (continued)

- Two major approaches to comparing with the Golden Reference
  1) Hardware and Reference must be “close”
    - Possibly compare SNR of hardware vs. reference
    - Inadequate for control hardware which must be a perfect match
    - Possibly more testing is needed than approach #2
  2) Hardware and Reference must be “Bit-accurate”
    - Hardware must match golden reference exactly, bit for bit
    - Far easier to automate testing
    - Possibly a smaller amount of testing will be needed than approach #1
    - Matlab must now do awkward operations such as rounding and saturation that match hardware
      - For example, \( \text{floor}(\text{in} + 0.5) \) for rounding
    - Input data may be generated from either matlab or verilog. I think it's a little easier to generate input data in verilog and then print both input and output to a matlab-readable file and test and compare in matlab. You may find it handy to declare variables as signed in verilog and print them using $fwrite so both positive and negative numbers print correctly.