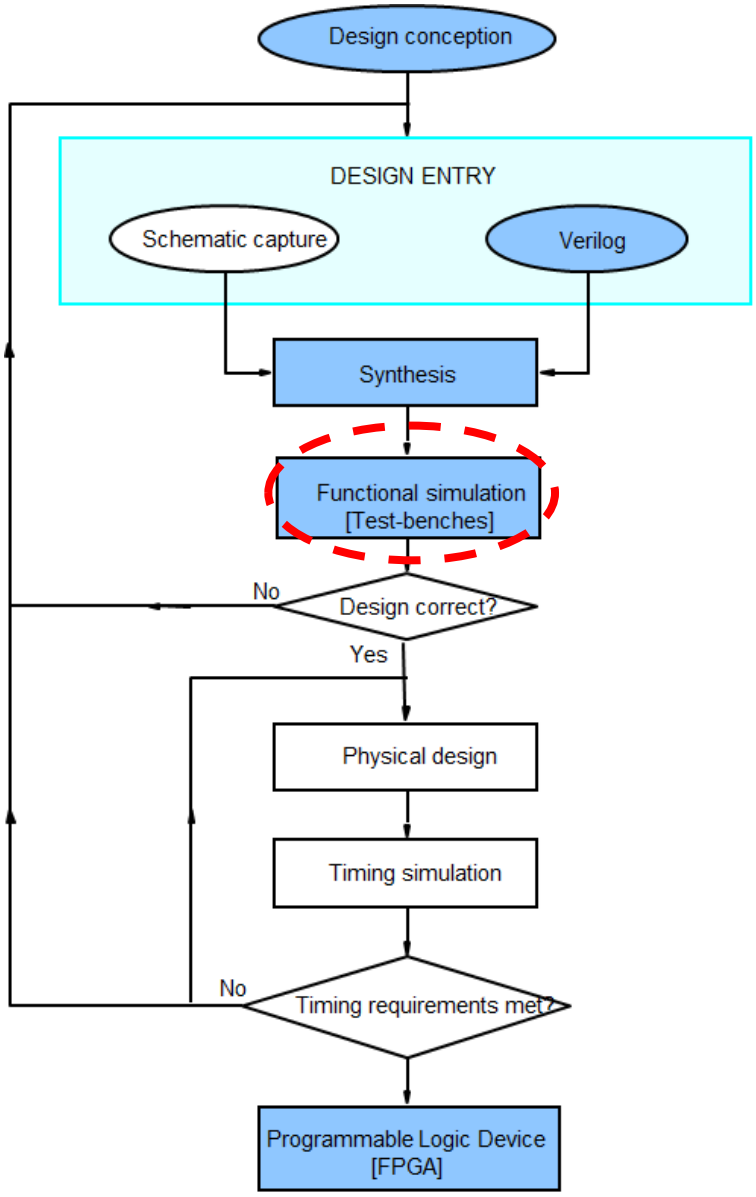
EE-421: Digital System Design

Verilog for Verification: Test-Bench

Instructor: Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

Where are we Heading?



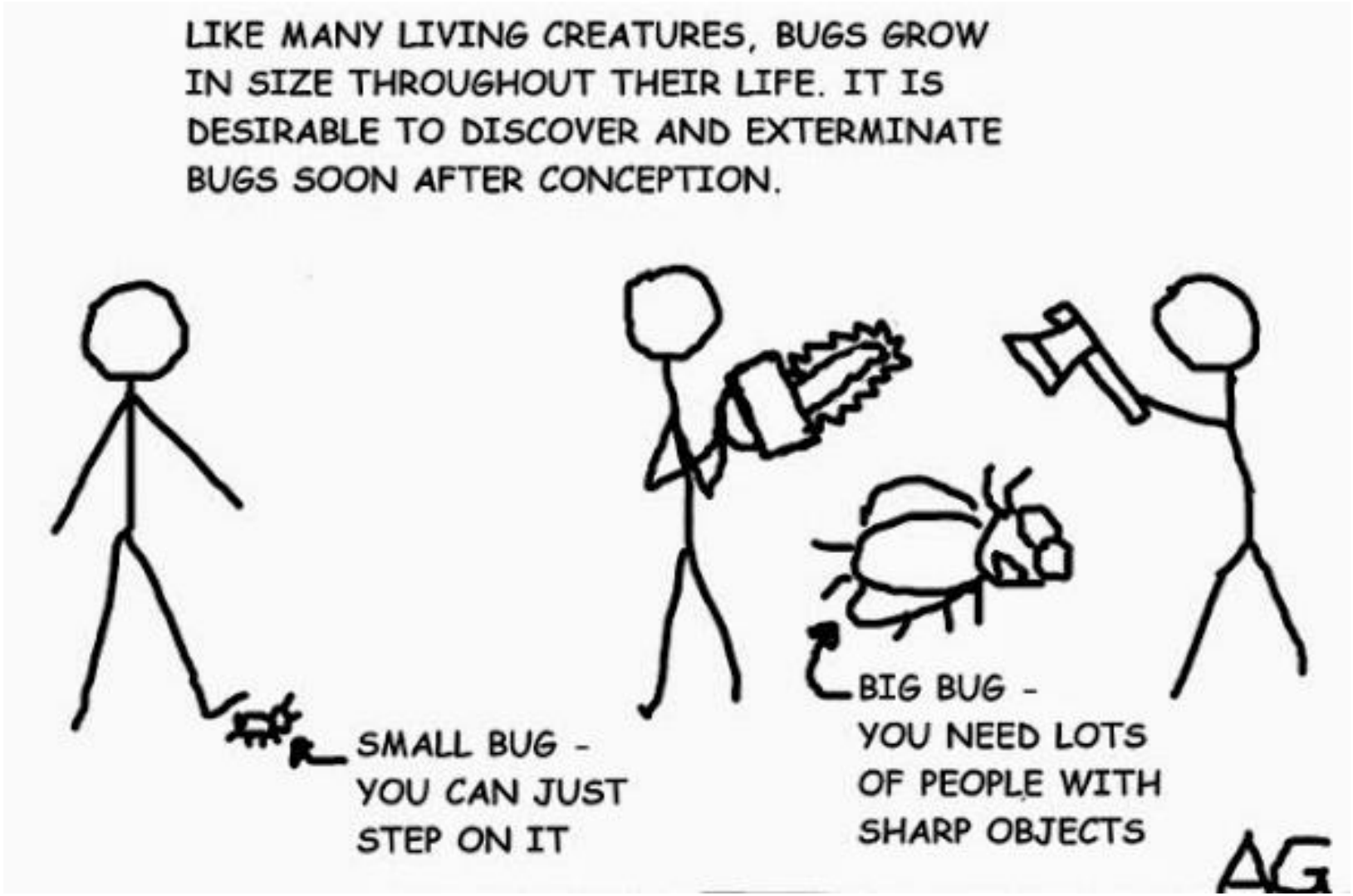
How Do You Know That A Circuit Work

- You have written the Verilog code of a circuit
  - Does it work correctly?
  - Even if the syntax is correct, it might do what you want?
  - What exactly it is that you want anyway?

- Trial and error can be costly
  - You need to 'test' your circuit in advance

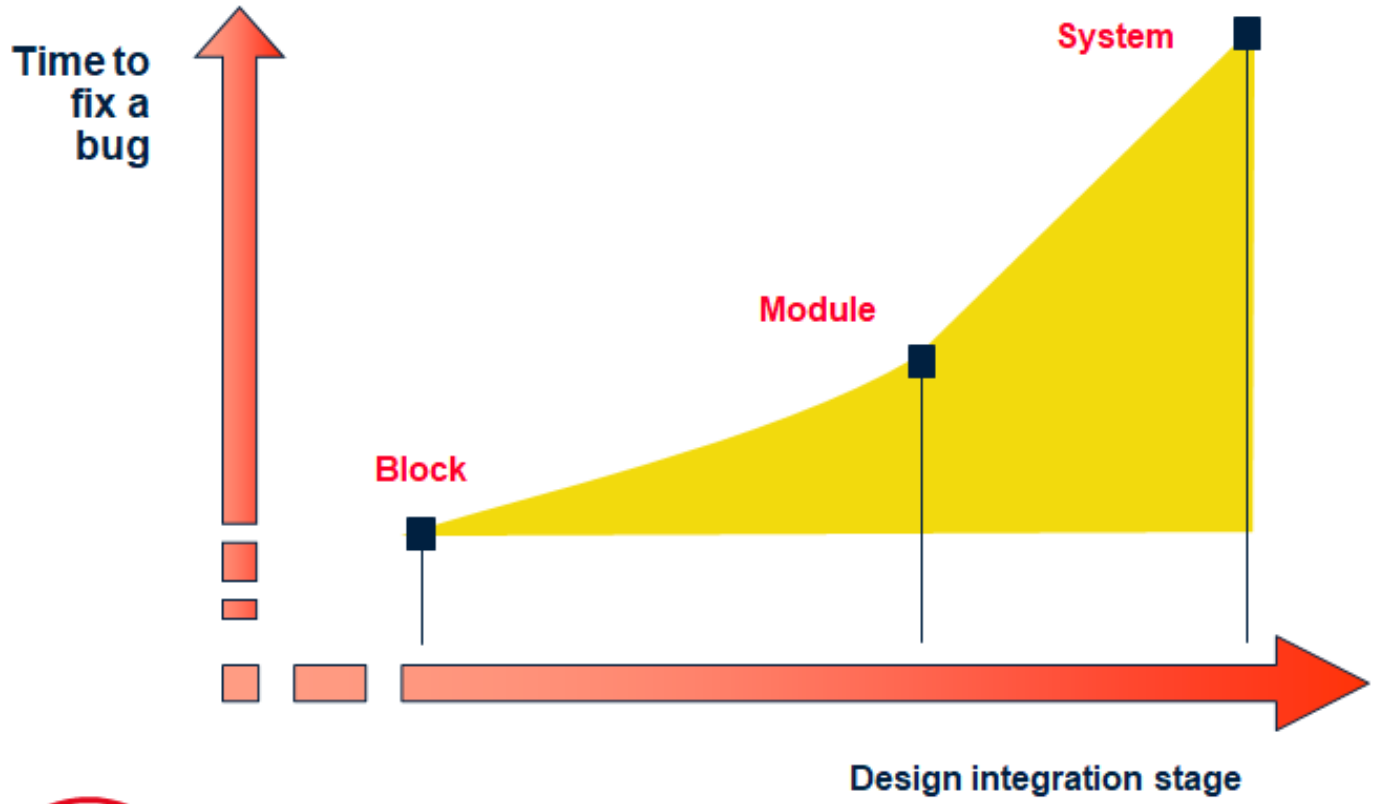
- In modern digital designs, functional verification is the most time-consuming design stage:
  - You want your design to be "bugs" free
    - Bugs have a cost!

Why Should We "FIX" Bugs ASAP?



Credit: Andy Glover -> cartoontester.blogspot.com

Why Should We "FIX" Bugs ASAP?

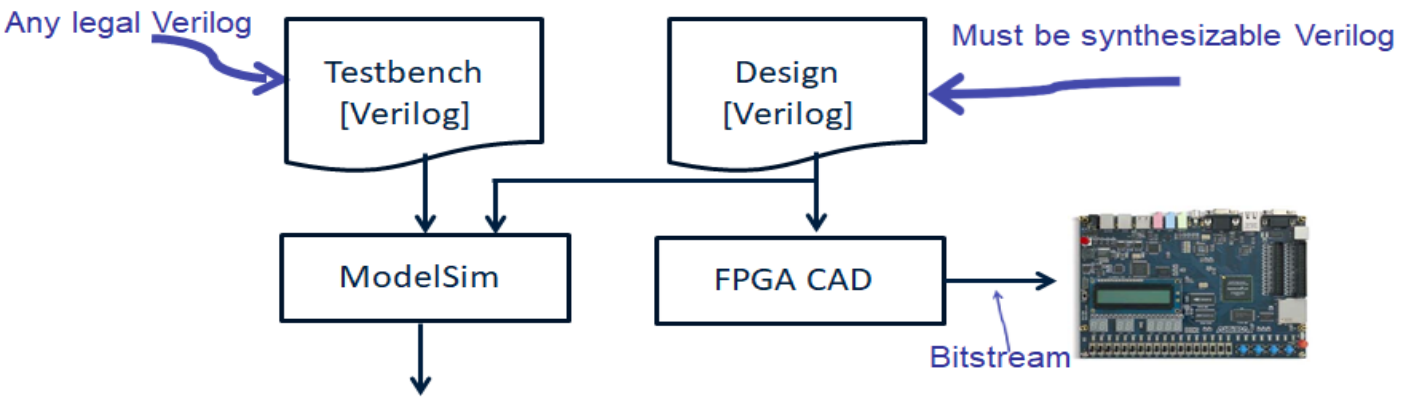


- Increase in chip NREs make respins an unaffordable proposition
- Average ASIC NRE ~$122,000
- SoC NREs range from $300,000 to $1,000,000

NRE=non-recurring engineering

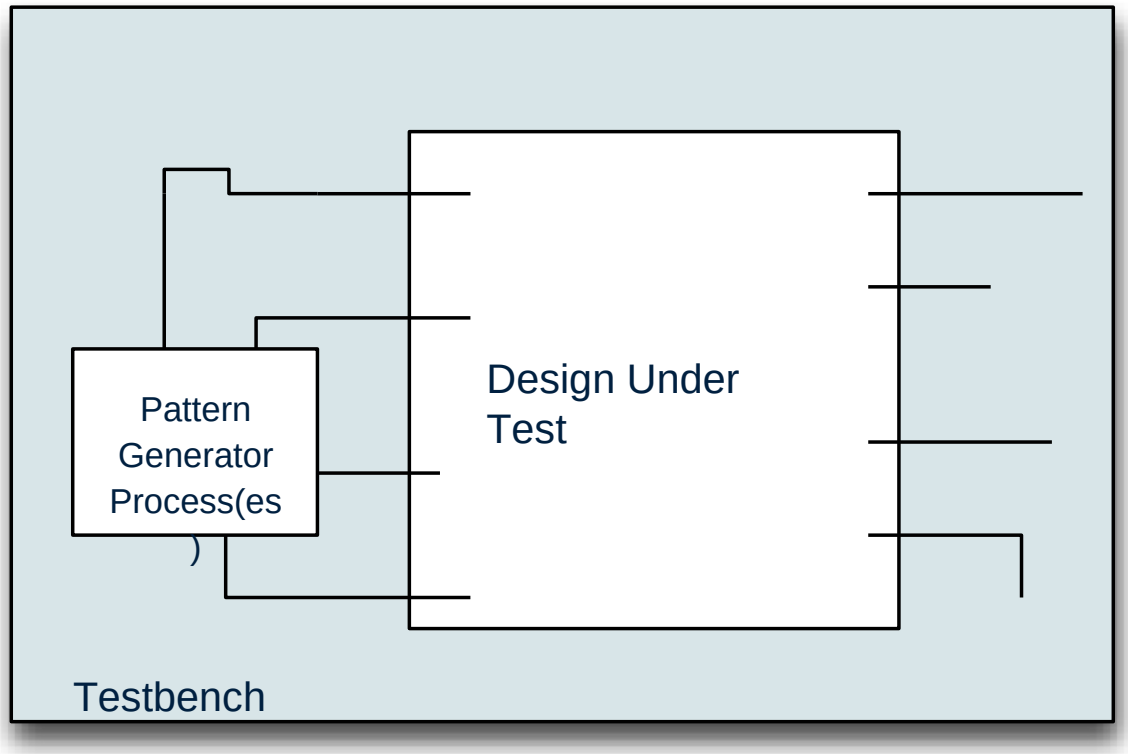
The Idea Behind a Testbench

- Using a computer simulator to test your circuit:
  - You instantiate your design under test (DUT) inside a testbench
    - A testbench is a Verilog module that provides inputs and possibly monitors outputs, used during testing
    - **Never becomes part of your design**, therefore does **not need** to be synthesizable
    - Allows the behavior of the circuit to be verified without actually manufacturing the circuit
  - Supply the circuit with some inputs:
    - See what it does
    - Does it return the "correct" outputs?



Design Under Test [DUT]

- The test bench applies stimulus to the DUT,
  - *Therefore must be instantiated in the test bench.*
  - *A testbench typically does not have inputs or outputs!*



Recipe for Expressing Testbench in Verilog

1. Declare tb module with no interface,
   - i.e module tb (); ….. end module

2. Reg and Wire Declaration inside tb module:
   - Inputs to the DUT are REG type
   - Outputs from the DUT are wire type

3. DUT Instantiation

4. Initialization:
   - Signals are undefined at startup.
   - Initialize any reg types in the design to a known value.
     - Use initialize block
   - Clock Generation:
     - Use always block to generate clk

Example: Testbench

- Write Verilog code to implement the following function in hardware:

$$y = \overline{(b \cdot c)} + (a \cdot \overline{b})$$

```
module myblock (input a, b, c,
                output y);

    assign y = ~b & ~c | a & ~b;
endmodule
```

Example: Testbench

```
module testbench();       // 1. Testbench has no inputs and outputs
      reg a, b, c;        // 2. inputs to the DUT are reg type
      wire y;             // 3. outputs from the DUT are wire type

      // 4. instantiate device under test [note argument passing by
            position]
      myblock dut (.a(a), .b(b), .c(c), .y(y) );

     // 5. apply inputs one at a time
     initial begin        // sequential block
     a = 0; b = 0; c = 0; #10; // apply inputs, wait 10ns
     c = 1; #10;          // apply inputs, wait 10ns
     b = 1; c = 0; #10;   // etc .. etc..
     c = 1; #10;
     a = 1; b = 0; c = 0; #10;
     end
endmodule
```

Example: Testbench

- Simple testbench instantiates the design under test
- It applies a series of inputs
- The outputs have to be observed and compared using a simulator program.
- This type of testbench does not help with the outputs
- Initial statement is similar to always, it just starts once at the beginning, and does not repeat.
- The statements have to be blocking.

Initial and Always blocks in TB

- Initial and Always blocks inside tb module:
  - Always and initial blocks are two sequential control blocks that operate on *reg* types in a Verilog simulation.
  - Each initial and always block executes concurrently in every module at the start of simulation.
  - Initial blocks start executing sequentially starting with the first line:
    - between the "begin end pair" each line executes from top to bottom until a delay is reached.

Other Helpful TB Constructs

- Generating clocks:

```
// generate clock
always      // no sensitivity list, so it always executes
  begin
    clk = 1; #5; clk = 0; #5;      // 10ns period
  end
```

- $display
  - is used to print to a line, and enter a carriage return at the end.
  - $display($time, "<< count = %d - Turning OFF count enable >>",cnt_out);

Recommended Reading

- A Verilog HDL Test Bench Primer Application Note
  - Uploaded on LMS

THANK YOU