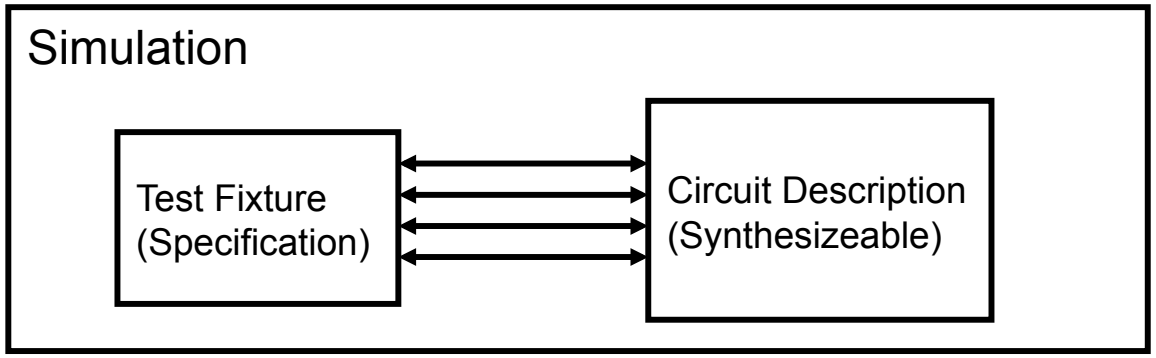
Test Fixtures

- Provides clock
- Provides test vectors/checks results
  - test vectors and results are precomputed
  - usually read vectors from file
- Models system environment
  - Complex program that simulates external environment
- Test fixture can all the language features
  - initial, delays, read/write files, etc.



Test Fixtures

Verilog programs used to drive the simulation
- Much easier than drawing waveforms

Test fixtures are usually non-synthesizeable
- Anything goes! Really is like writing a C program
- Not really Verilog!

Self-checking text fixtures are required for regression testing
- You can write an arbitrary program in Verilog

Initial Blocks

- Like always blocks
  - execute once at the very beginning of simulation
  - not synthesizeable
    - use reset instead

Verilog Clock Generator

```
module clockGenerator (CLK);
  parameter period = 10;
  parameter howlong = 100;
  output         reg CLK;

  initial begin
    CLK = 0;
    #(period/2);
    repeat (howlong) begin
      CLK = 1;
      #(period-period/2);
      CLK = 0;
      #(period/2);
    end
    $finish;
  end

endmodule
```

Finishes the simulation
Cannot be restarted

Another Clock Generator

```
module clock_gen (masterclk);

    `define PERIOD = 10;

    output masterclk;
    reg    masterclk;

    initial masterclk = 0;

    always begin
        #`PERIOD/2
        masterclk = ~masterclk;
    end

endmodule
```

use `define to make constants easier to find and change

use of initial and always blocks

Simple Test Fixture

```
module stimulus
  (output a, b, c);
  parameter    delay = 10;
  reg    [2:0] cnt;

  initial begin
    cnt = 0;
    repeat (8) begin
      #delay cnt=cnt+1;
    end
    #delay $finish;
  end

  assign {c, a, b} = cnt;
endmodule
```

```
module full_addr1 (A, B, Cin, S, Cout);
  input     A, B, Cin;
  output    S, Cout;

  assign {Cout, S} = A + B + Cin;
endmodule
```

```
module driver;      // Structural Verilog connects test-fixture to full adder
  wire       a, b, cin, sum, cout;
  stimulus   stim (a, b, cin);
  full_addr1 fa1 (a, b, cin, sum, cout);

  initial begin
    $monitor ("@ time=%0d cin=%b, a=%b, b=%b, cout=%d, sum=%d",
                $time, cin, a, b, cout, sum);
  end
endmodule
```

$monitor prints when any printed signal changes

$time is simulation time

Text Fixture Example

```
module what_tf
   (output [8:0] data,
    input [3:0] count);

    integer     i;
    assign data = i;
    initial begin
      for (i = 0; i <= 511; i = i + 1) begin
         #10 $display("Data = %x, Count = %d", data, count);
      end
      $stop;
   end
endmodule
```

$display is like printf

$stop just pauses the simulation – can be restarted

Self-Checking Text Fixture

```
module countLeadingZeros_tf(
  output reg [15:0] in,
  input [4:0] out);
  integer i;
  initial begin
     in = 'h8000;
     for (i = 0; i <= 16; i = i + 1) begin
       #10 if (out !== i) begin
          $display("***ERROR***");
          $stop;
       end
      in = (in >> 1);
     end
    $stop;
  end
endmodule
```

Test Vectors

```
module testData(clk, reset, data);
  input clk;
  output reset, data;
  reg [1:0] testVector [100:0];
  reg reset, data;
  integer count;

  initial begin
    $readmemb("data.dat", testVector);
    count = 0;
    { reset, data } = testVector[0];
  end

  always @(posedge clk) begin
    count <= count + 1;
    #1 { reset, data } <= testVector[count];
  end
endmodule
```

Homework Text Fixture #4

```
module convert_tf
  (input [7:0] R, G, B,
   output reg [7:0] Y, CR, CB);
   reg [47:0] mem [0:1023];
   reg [7:0]  Rs, Gs, Bs;
   integer    i;
   integer    errors;
   function absdiff;
      input   a, b;
      begin
      if (a > b) absdiff = a - b;
      else absdiff = b - a;
      end
   endfunction
   initial begin
      $readmemh("data.txt", mem);
      errors = 0;
      for (i=0; i < 1024; i = i + 1) begin
      { Y, CR, CB, Rs, Gs, Bs } = mem[i];
      #10
        if (absdiff(R, Rs)>1 || absdiff(G, Gs)>1 || absdiff(B, Bs)>1) begin
           $display("Error: (YCrCb)=%d,%d,%d (R,G,B)=%d,%d,%d should be %d,%d,%d", Y,CR,CB,R,G,B,Rs,Gs,Bs);
           errors = errors + 1;
      end
      end
      $display("End of Simulation: %d errors found", errors);
      $stop;
   end // initial begin
endmodule // convert_tf
```

Verilog Simulation

- Interpreted vs. compiled simulation
  - performance of the simulation
- Level of simulation
  - accuracy of the model
- Relationship to synthesis
  - can all that can be simulated be synthesized?

Intepreted vs. Compiled Simulation

- Interpreted
  - data structures constructed from input file
  - simulator walks data structures and decided when something occurs
  - basic algorithm:
    - take an event from queue, evaluate all modules sensitive to that event, place new events on queue, repeat

Intepreted vs. Compiled Simulation

- Compiled
  - input file is translated into code that is compiled/linked with kernel
  - basic algorithm:
    - same as above
    - except that now functions associated with elements are simply executed and directly place events on queue
    - overhead of compilation must be amortized over total simulation time and its harder to make changes – need dynamic linking

Simulation Level

- Electrical
  - solve differential equations for all devices simultaneously to determine precise analog shape of waveforms
- Transistor
  - model individual transistors as switches - this can be close to electrical simulation if restricted to digital circuits
- Gate
  - use abstraction of Boolean algebra to view gates as black-boxes if only interested in digital values and delay
- Cycle or register-transfer
  - determine correct values only at clock edges, ignore gate delays if interested only in proper logical behavior of detailed implementation
- Functional (or behavioral) level
  - no interest in internal details of circuit implementation (just a program)

Simulation Time and Event Queues

- Event queue
  - changes in signal values are "events" placed on the queue
  - queue is a list of changes to propagate
  - priority queue of pending events based on time of occurrence
  - multiple events on same signal can be on queue
- Time
  - advanced whenever an event is taken off the queue
    - advance to time of event
  - parallel activities are implicitly interleaved
  - what do we do about events with zero delay?

Verilog Time

- All computations happen in zero time unless there are explicit delays or waits in the code
  - #delay - blocks execution until that much time has passed
    - event placed on queue to wake up block at that time
  - @ or wait - waits for an event, e.g., @(posedge clk) and wait (x==0)
    - nothing happens until that event is taken off the queue
- When an event is removed from the queue, all the blocks sensitive to it are evaluated in parallel and advance to their next blocking point (delay/wait)
- Time advances as long as there are events to process
  - infinite loops are easy to write
  - use explicit $finish
  - use specified number of clock periods