Digital Design with the Verilog HDL
Chapter 3: Hierarchy & Simulation

Binh Tran-Thanh

Department of Computer Engineering
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology

May 26, 2023

Module Port List Declaration (Multiple ways)

```
module Add_half(c_out, sum, a, b);
   output sum, c_out;
   input a, b;
   ...
endmodule
//*********************************************//
module Add_half(output c_out, sum, input a, b);
   ...
endmodule
//*********************************************//
module xor_8bit(out, a, b);
   output[7:0] out;
   input[7:0] a, b;
   ...
endmodule
//*********************************************//
module xor_8bit(output[7:0] out, input[7:0] a, b);
   ...
endmodule
```

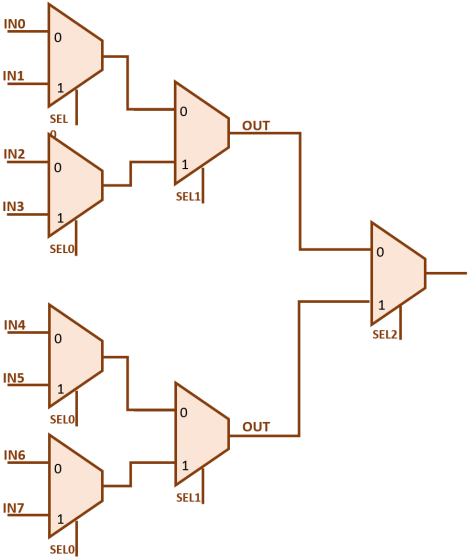
Structural Design Tip

- If a design is complex, draw a block diagram!
- Label the signals connecting the blocks
- Label ports on blocks if not primitives/obvious.
- Easier to double-check your code!
- Don't bother with *300-gate design* ...
- But if that **big**, probably should use hierarchy!

Example: Hierarchy Multiplexer

```
module mux_8to1(output out,
           input in0, in1, in2, in3, in4, in5, in6, in7,
           input[2:0] select );

   ...
endmodule

//*********[mux2to1 as submudule]*************//
module mux_2to1( output out, input in0, in1, select);
   wire n0, n1, n2;

   ...
endmodule
```

8to1 Mux from 2to1 Muxs Structure



source: https://vlsiuniverse.blogspot.com

Interface: Hierarchical Multiplexer

```
module mux_8to1(output out,
            input in0, in1, in2, in3, in4, in5, in6, in7,
            input[2:0] select);
   wire n0, n1, n2, n3, n4, n5;

   //**** [level 1: 4 MUX2to1] ****//
   mux_2to1 M1_L1 (n0, in0, in1, select[0]),
            M2_L1 (n1, in2, in3, select[0]),
            M3_L1 (n2, in4, in5, select[0]),
            M4_l2 (n3, in6, in7, select[0]);
   //**** [level 2: 2 MUX2to1] ****//
   mux_2to1 M1_L2 (n4,  n0,  n1, select[1]),
            M2_L2 (n5,  n2,  n3, select[1]);
   //**** [level 3: 1 MUX2to1] ****//
   mux_2to1 M1_L3 (out, n4,  n5, select[2]);
endmodule
```

Timing Controls For Simulation

- Can put “delays” in a Verilog design
  - Gates, wires, even behavioral statements!
- SIMULATION
  - Used to approximate “real” operation while simulating.
  - Used to control testbench
- SYNTHESIS
  - Synthesis tool IGNORES these timing controls
    - Cannot tell a gate to wait 1.5 nanoseconds!
    - Delay is a result of physical properties!
  - Only timing (easily) controlled is on **clock-cycle** basis
    - Can tell synthesizer to attempt to meet cycle-time restriction

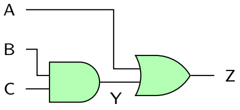
Zero Delay vs. Unit Delay

- When no timing controls specified: zero delay
  - Unrealistic –even electrons take time to move
  - OUT is updated same time A and/or B change:
    `and (OUT, A, B)`
- Unit delay often used
  - Not accurate either, but closer...
  - "Depth" of circuit does affect speed!
  - Easier to see how changes propagate through circuit
  - OUT is updated 1 "unit" after A and/or B change:
    `and #1 A0(OUT, A, B);`

Zero/Unit Delay Example

Zero Delay					
T	**A**	**B**	**C**	**Y**	**Z**
0	0	0	0	0	0
1	0	0	1	0	0
2	0	1	0	0	0
3	0	1	1	1	1
4	1	0	0	0	1
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1
8	0	0	0	0	0
9	0	0	1	0	0
10	0	1	0	0	0
11	0	1	1	1	1
12	1	0	0	0	1
13	1	0	1	0	1
14	1	1	0	0	1
15	1	1	1	0	1

Unit Delay					
T	**A**	**B**	**C**	**Y**	**Z**
0	0	1	0	x	x
1	0	1	0	0	x
2	0	1	0	0	0
3	0	1	1	0	0
4	0	1	1	1	0
5	0	1	1	1	1
6	1	0	0	1	1
7	1	0	0	0	1
8	1	1	1	0	1
9	1	1	1	1	1
10	1	0	0	1	1
11	1	0	0	0	1
12	0	1	0	0	1
13	0	1	0	0	0
14	0	1	1	0	0
15	0	1	1	1	0
16	0	1	1	1	1



Zero Delay: Y and Z change at same "time" as A, B, and C!

Unit Delay: Y changes 1 unit after B, C

Unit Delay: Z changes 1 unit after A, Y

Types Of Delays

- Inertial Delay (Gates)
  - Suppresses pulses shorter than delay amount
  - In reality, gates need to have inputs held a certain time before output is accurate
  - This models that behavior
- Transport Delay (Nets)
  - "Time of flight" from source to sink
  - Short pulses transmitted
- Not critical for most of class
  - May need to know when debugging
  - Good to know for building very accurate simulation

Delay Examples

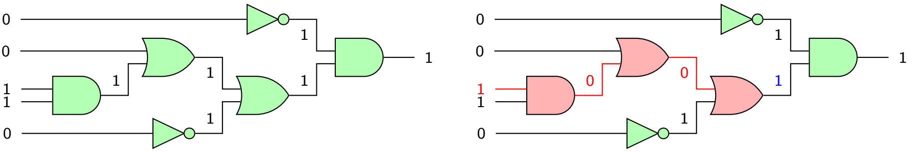
```
wire #5 net_1;                 // 5 units transport delay

and #4 (z_out, x_in, y_in); // 4 units inertial delay

assign #3 z_out= a & b;        // 3 units inertial delay

wire #2 z_out;                 // 2 units transport delay
and #3 (z_out, x_in, y_in); // 3 for gate, 2 for wire

wire #3 c;                     // 3 units transport delay
assign #5 c = a & b;           // 5 for assign, 3 for wire
```

Delays In Testbenches

- Most common use in class
- Single testbench tests many possibilities
  - Need to examine each case separately
  - Spread them out over “time”
- Use to generate a clock signal
  - Example later in lecture

Simulation

Update only if changed

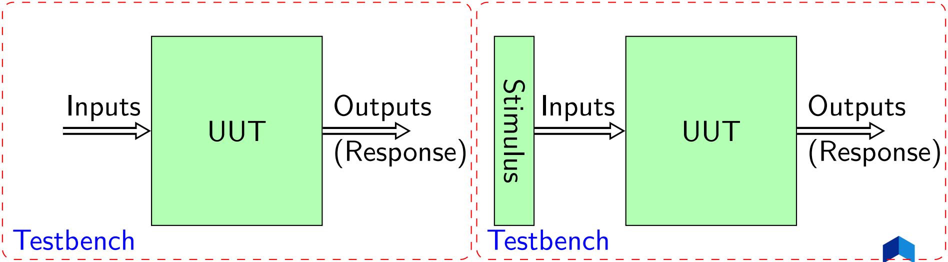


Some circuits are very large

- Updating every signal ⇒ very slow simulation
- Event-driven simulation is much faster!

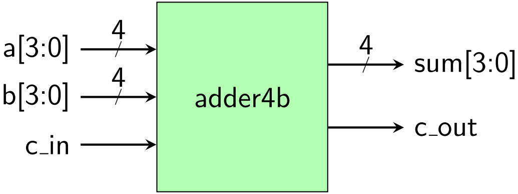
Simulation of Verilog

- Need to verify your design
  - "Unit Under Test" (UUT)
- Use a "testbench"!
  - Special Verilog module with no ports
  - Generates or routes inputs to the UUT
  - Outputs information about the results



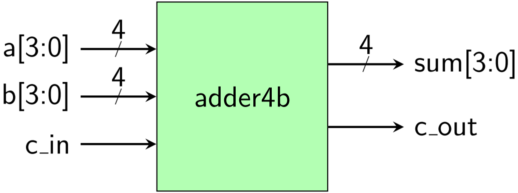
Simulation [Functionality] Example

```
module adder4b (sum, c_out, a, b, c_in);
  input [3:0] a, b;
  input c_in;
  output [3:0] sum;
  output c_out;

  assign {c_out, sum} = a + b + c_in;

endmodule
```

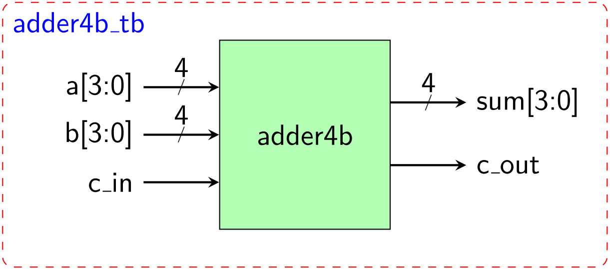


Simulation [Timing and Functionality] Example

```verilog
`timescale 1ns /1ns        // time_unit/time_precision
module adder4b_delay (sum, c_out, a, b, c_in);
  input [3:0] a, b;
  input c_in;
  output [3:0] sum;
  output c_out;

  assign #5 {c_out, sum} = a + b + c_in;

endmodule
```



Simulation Example



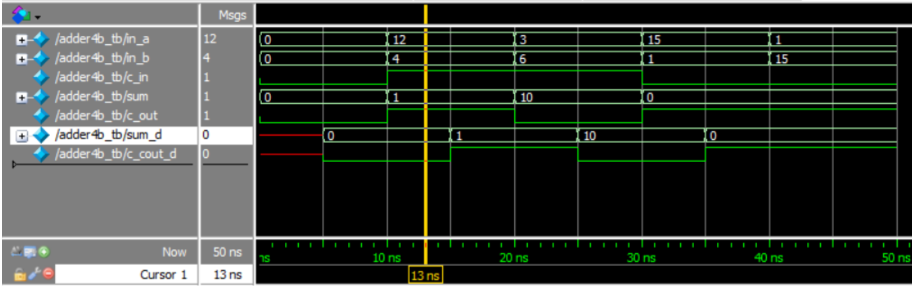
Testbenches frequently named (should NOT mix style)

- `<UUT name>_tb.v (recommend)`
- `<UUT name>_t.v`
- `tb_<UUT name>.v`
- `t_<UUT name>.v`

Testbench Example

```
`timescale 1ns /1ns        // time_unit/time_precision
module adder4b_tb;
   reg [3:0] in_a, in_b;  // inputs to UUT are regs
   reg c_in;              // inputs to UUT are regs
   wire [3:0] sum, sum_d; // outputs of UUT are wires
   wire c_out, c_cout_d;  // outputs of UUT are wires
   // instantiate UUT
   adder4b       A1 (sum, c_out, in_a, in_b, c_in);
   adder4b_delay A2 (sum_d, c_cout_d, in_a, in_b, c_in);
   // stimulus generation
   initial begin
          {in_a, in_b, c_in} = 9'b0000_0000_0; // at 0 ns
      #10 {in_a, in_b, c_in} = 9'b1100_0100_1; // at 10 ns
      #10 {in_a, in_b, c_in} = 9'b0011_0110_1; // at 20 ns
      #10 {in_a, in_b, c_in} = 9'b1111_0001_0; // at 30 ns
      #10 {in_a, in_b, c_in} = 9'b0001_1111_0; // at 40 ns
      #10 $stop; // at 50 ns, stops simulation
   end
endmodule
```

Testbench Waveform



Testbench Requirements

- Instantiate the unit being tested (UUT)
- Provide input to that unit
  - Usually a number of different input combinations!
- Watch the “results” (outputs of UUT)
  - Can watch ModelSimWave window...
  - Can print out information to the screen or to a file

Output Test Info

- Several different system calls to output info
  - `$monitor`
    - Output the given values whenever one changes
    - Can use when simulating Structural, RTL, and/or Behavioral
  - `$display`, `$strobe`
    - Output specific information as if printf or coutin a program
    - Used in Behavioral Verilog
- Can use formatting strings with these commands
- Only means anything in simulation
- Ignored by synthesizer

Output Format Strings

- Formatting string

```
%h, %H hex
%d, %D decimal
%o, %O octal
%b, %B binary
%t     time
```

- `$monitor("%t: %b %h %h %h %b\n", $time, c_out, sum, a, b, c_in);`
- Can get more details from Verilog standard

Output Example

```
‘timescale 1ns /1ns        // time_unit/time_precision
module adder4b_tb;
   reg [3:0] in_a, in_b; // inputs to UUT are regs
   reg c_in;             // inputs to UUT are regs
   wire [3:0] sum;       // outputs of UUT are wires
   wire c_out;           // outputs of UUT are wires
   // instantiate UUT
   adder4b UUT(sum, c_out, in_a, in_b, c_in);
   // monitor statement
   initial $monitor("time %t: cout=%b,sum=%h, in_a=%h, in_b
    =%h, cin=%b\n", $time, c_out, sum, in_a, in_b, c_in);
   // stimulus generation
   initial begin
          {in_a, in_b, c_in} = 9’b0000_0000_0; // at 0 ns
      #10 {in_a, in_b, c_in} = 9’b1100_0100_1; // at 10 ns
      #10 {in_a, in_b, c_in} = 9’b0011_0110_1; // at 20 ns
      #10 {in_a, in_b, c_in} = 9’b1111_0001_0; // at 30 ns
      #10 {in_a, in_b, c_in} = 9’b0001_1111_0; // at 40 ns
      #10 $stop; // at 50 ns, stops simulation
   end
endmodule
```

Output Example Output [Text View]

Executed at
https://www.tutorialspoint.com/compile_verilog_online.php

```
time  0: cout=0, sum=0, in_a=0, in_b=0, cin=0
time 10: cout=1, sum=1, in_a=c, in_b=4, cin=1
time 20: cout=0, sum=a, in_a=3, in_b=6, cin=1
time 30: cout=1, sum=0, in_a=f, in_b=1, cin=0
time 40: cout=1, sum=0, in_a=1, in_b=f, cin=0
```

Testbench (Read data input from file) Example

```
'timescale 1ns /1ns        // time_unit/time_precision
module adder4b_read_file_tb();
 reg [3:0] in_a, in_b; // inputs to UUT are regs
 reg c_in;             // inputs to UUT are regs
 wire [3:0] sum;       // outputs of UUT are wires
 wire c_out;           // outputs of UUT are wires
 integer fd;           // file descriptors
 // instantiate UUT
 adder4b A1 (sum, c_out, in_a, in_b, c_in);
 // monitor statement
 initial $monitor("time %t: cout=%b,sum=%d, in_a=%d, in_b=%
   d, cin=%b", $time, c_out, sum, in_a, in_b, c_in);
 // stimulus generation
 initial begin
  fd = $fopen ("data.in", "r"); ␣
  if (fd) begin
   while ($fscanf (fd, "%h %h %b", in_a, in_b, c_in) != -1)
     begin
    #5; end
  end
  $fclose(fd); // close file handler
  $stop;      // finish simulation
 end // end initial
endmodule
```

DataIn file Example

data.in file

```
1 5 1
2 6 1
3 7 1
4 8 1
5 9 1
6 10 0
7 11 0
8 12 0
9 13 0
10 14 1
11 15 1
```

Testbench (Read input file, write output file) Example

```
'timescale 1ns /1ns       // time_unit/time_precision
module adder4b_read_file_write_output_tb();
 reg [3:0] in_a, in_b; // inputs to UUT are regs
 reg c_in;             // inputs to UUT are regs
 wire [3:0] sum;       // outputs of UUT are wires
 wire c_out;           // outputs of UUT are wires
 integer read_fd, write_fd;    // file descriptors
 // instantiate UUT
 adder4b A1 (sum, c_out, in_a, in_b, c_in);
 // monitor statement
 initial $monitor("time %t: cout=%b, sum=%d, in_a=%d, in_b
   =%d, cin=%b", $time, c_out, sum, in_a, in_b, c_in);
 initial #100 $stop;
 // stimulus generation
 initial begin
  read_fd = $fopen ("data.in", "r");
  write_fd = $fopen ("data.out", "w");
  if (write_fd ==0 && read_fd ==0) begin
     $display("File was NOT opened successfully");
     $stop;    // stop
  end

  while ($fscanf (read_fd, "%d %d %b", in_a, in_b, c_in) !=
     -1) begin
    $fdisplay (write_fd, "time %t: cout=%b, sum=%d, in_a=%d
    , in_b=%d, cin=%b", $time, c_out, sum, in_a, in_b, c_in)
    ;
    #5;
  end

  $fclose(read_fd); // close read file handler
  $fclose(write_fd); // close write file handler

 end // end initial
endmodule
```

Exhaustive Testing

- For combinational designs w/ up to 8 or 9 inputs
  - Test ALL combinations of inputs to verify output
  - Could enumerate all test vectors, but don't ...
  - Generate them using a "for" loop!

```
reg[4:0] x;
initial begin
// Remember to check infinite loop
// This example uses 5-bit counter for 16 samples
for(x = 0; x < 16; x = x + 1)
   #5 ;         // need a delay here!
end
```

- Need to use "reg" type for loop variable?

Example: UUT

```
module Comparator4b(A_gt_B, A_lt_B, A_eq_B, A, B);
  output A_gt_B, A_lt_B, A_eq_B;
  input [3:0] A, B;

  // RTL Styles
  assign A_eq_B = (A == B)? 1 : 0;
  assign A_gt_B = (A > B) ? 1 : 0;
  assign A_lt_B = (A < B) ? 1 : 0;

endmodule
```

Example: Testbench

```
module Comparator4b_tb;
   wire A_gt_B, A_lt_B, A_eq_B;
   reg [4:0] A, B; // 5-bit to prevent loop wrap around
   // UUT
   Comparator4b M1(A_gt_B, A_lt_B, A_eq_B, A[3:0], B[3:0]);
   initial $monitor("%t: A=%h, B=%h, AgtB=%b, AltB=%b, AeqB
   =%b", $time, A[3:0], B[3:0], A_gt_B, A_lt_B, A_eq_B);
   initial #2000 $finish; // end simulation, quit program
   initial begin
      #5;
      /** After #5, exhaustive test of valid inputs **/
      for (A = 0; A < 16; A = A + 1) begin
         for (B = 0; B < 16; B = B + 1) begin
            #5; // every 5 time unit, A, B will be updated
         end // first for
      end // second for
   end // initial
endmodule
```

Example: Testbench Output [Text view]

Executed at
`https://www.tutorialspoint.com/compile_verilog_online.php`

```
   0: A=x, B=x, AgtB=x, AltB=x, AeqB=x
   5: A=0, B=0, AgtB=0, AltB=0, AeqB=1
  10: A=0, B=1, AgtB=0, AltB=1, AeqB=0
  15: A=0, B=2, AgtB=0, AltB=1, AeqB=0
.....................................
  75: A=0, B=e, AgtB=0, AltB=1, AeqB=0
  80: A=0, B=f, AgtB=0, AltB=1, AeqB=0
  85: A=1, B=0, AgtB=1, AltB=0, AeqB=0
  90: A=1, B=1, AgtB=0, AltB=0, AeqB=1
  95: A=1, B=2, AgtB=0, AltB=1, AeqB=0
.....................................
1275: A=f, B=e, AgtB=1, AltB=0, AeqB=0
1280: A=f, B=f, AgtB=0, AltB=0, AeqB=1
1285: A=0, B=0, AgtB=0, AltB=0, AeqB=1
```

Combinational Testbench

```
module comb(output d, e, input a, b, c);
  and(d, a, b);
  nor(e, a, b, c);
endmodule

module comb_tb;
  wire d, e;
  reg [3:0] abc;
  comb CMD(d, e, abc[2], abc[1], abc[0]); // UUT
  initial $monitor("%t: a=%b, b=%b, c=%b, d=%b, e=%b",
   $time, abc[2], abc[1], abc[0], d, e);
  initial #2000 $finish;// end simulation, quit program
  // exhaustive test of valid inputs
  initial begin
    for(abc= 0; abc< 8; abc= abc+ 1) begin #5; end// for
  end// initial
endmodule
```

Generating Clocks

- Wrong way:

```
initial begin
  #5 clk = 0;
  #5 clk = 1;
  #5 clk = 0;
  ...   //(repeat hundreds of times)
end
```

- Right way:

```
initial
    clk = 0;
    always @ (clk)
       #5 clk = ~clk;
```

```
initial begin
  clk = 0;
  forever  #5 clk = ~clk;
end
```

- LESS TYPING
- Easier to read, harder to make mistake

FSM Testing

- Response to input vector depends on state
- For each state:
  - Check all transitions
  - For Moore, check output at each state
  - For Mealy, check output for each transition
  - This includes any transitions back to same state!
- Can be time consuming to traverse FSM repeatedly...

Example : 3-bit Counter

- Write a testbench to test the 3-bit counter.

```
module Counter3b(output reg [2:0] counter_out, input
    clk, rst);

   // Structural style
   always @(posedge clk) begin
      if (rst) begin counter_out <= 3'b000; end
      else begin counter_out <= counter_out + 1'b1; end
   end

endmodule
```

- Initially reset the counter and then test all states, but do not test reset in each state.

3-bit Counter Testbench

```
module Counter3b_tb;
  wire [2:0] out;
  reg clk, rst;

  Counter3b counter(out, clk, rst); // UUT

  initial $monitor("%t: out=%b, rst=%b, clk=%b", $time, out
    , rst, clk);
  initial #100 $finish; // end simulation, quit program
  initial begin
    clk= 0;
    forever #5 clk= ~clk; // What is the clock period?
  end
  initial begin
        rst= 1;
    #10 rst= 0;
  end // end initial
endmodule
```

3-bit Counter Testbench Output [Text View]

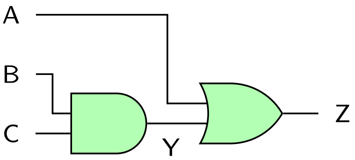
```
Time   0: out=xxx, rst=1, clk=0
Time   5: out=000, rst=1, clk=1
Time  10: out=000, rst=0, clk=0
Time  15: out=001, rst=0, clk=1
Time  20: out=001, rst=0, clk=0
Time  25: out=010, rst=0, clk=1
Time  30: out=010, rst=0, clk=0
Time  35: out=011, rst=0, clk=1
Time  40: out=011, rst=0, clk=0
Time  45: out=100, rst=0, clk=1
Time  50: out=100, rst=0, clk=0
Time  55: out=101, rst=0, clk=1
Time  60: out=101, rst=0, clk=0
Time  65: out=110, rst=0, clk=1
Time  70: out=110, rst=0, clk=0
Time  75: out=111, rst=0, clk=1
Time  80: out=111, rst=0, clk=0
Time  85: out=000, rst=0, clk=1
Time  90: out=000, rst=0, clk=0
Time  95: out=001, rst=0, clk=1
Time 100: out=001, rst=0, clk=0
```

Force/Release In Testbenches

- Allows you to "override" value FOR SIMULATION
- Doesn't do anything in "real life"
  - No fair saying "if 2+2 == 5, then force to 4" Synthesizer won't allow force...release anyway
- How does this help testing?
  - Can help to pinpoint bug
  - Can use with FSMs to override state
- Force to a state
- Test all edges/outputs for that state
- Force the next state to be tested, and repeat
- Can also use simulator force functionality

Force/Release Example

```
assign y = a & b;
assign z = y | c;
initial begin
  a = 0; b = 0; c = 0;
  #5 a = 0; b = 1; c = 0;
  #5 force y = 1;
  #5 b = 0;
  #5 release y;
  #5 $stop;
end
```



Time	a	b	c	y	z
0	0	0	0	0	0
5	0	1	0	0	0
10	0	1	0	1	1
15	0	0	0	1	1
20	0	0	0	0	0