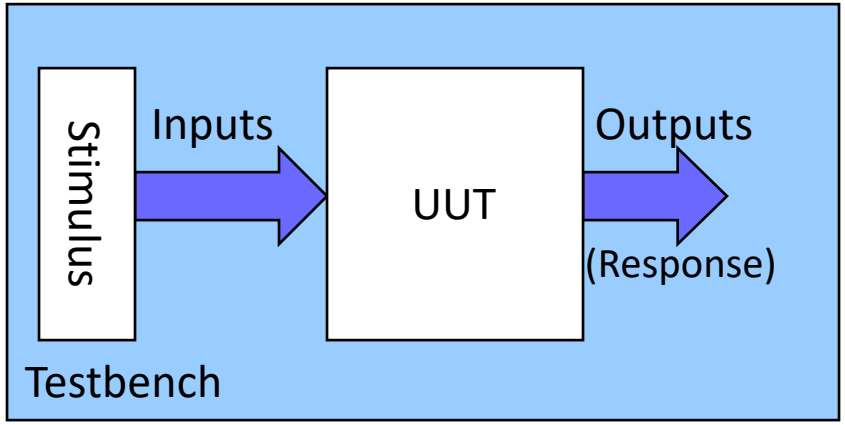
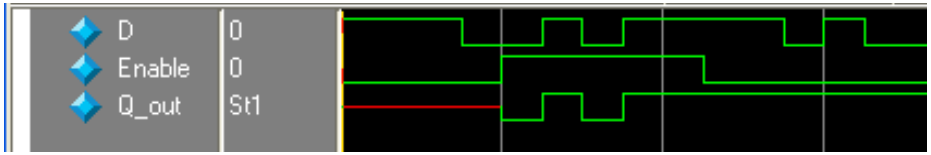
DIGITAL DESIGN WITH HDL

Chapter 3: Test bench for Simulation

Simulation

- A method to test a designed module to ensure its operations
  - Applying stimulus to inputs and checking outputs
- Use a test bench module
  - An instantiation of a unit under test
  - Verilog code to generate stimulus
  - Verilog code to monitor and display the response
    - Text
    - Waveform






```
#0 x1=1'b0; x2=1'b0; x3=1'b0;
#10 x1=1'b0; x2=1'b0; x3=1'b1;
#10 x1=1'b0; x2=1'b1; x3=1'b0;
```

Simulation vs. Synthesis

- Simulation
  - Can put delays to gates, wire, statements
  - Used to approximate "real" operation while simulating
- Synthesis
  - Ignores all delays
    - Technology-dependent
    - Result of physical properties
  - Used to create hardware netlist to implement a circuit with a particular technology
    - ASIC
    - FPGA

Delays

- Gate delays
  - Propagation delay: discuss in structural model
    - and #4 (z_out, x_in, y_in); // 4 unit propagation delay
    - assign #3 z_out = a & b; // 3 unit propagation delay
  - Inertial delay: amount of time inputs must be stable to generate correct outputs
    - Used in test bench
    - Syntax: #delay <statement>
- Transport delay
  - Time required for a signal traveling from source to destination
    - Usually very small
  - Technology-dependent
  - wire #3 Cnet; // 3 unit transport delay

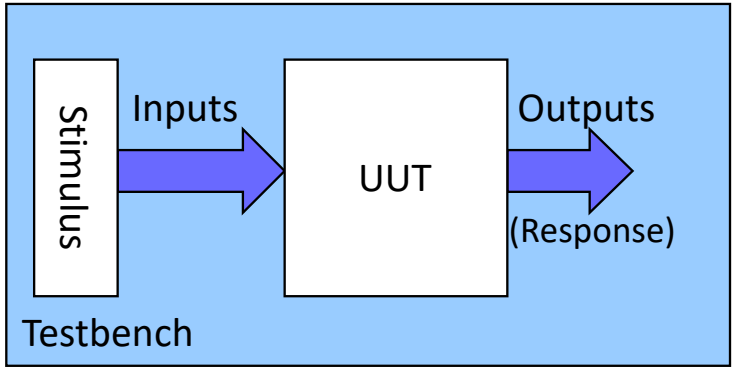
Delays in test bench

- Most common use in class

- Single test bench tests many possibilities
  - Need to examine each case separately
  - Spread them out over "time" (to view with waveform easier)
    - #10 input_a = 1; //assign 1 to input_a after 10 unit of time
    - #10 input_a = 0; //assign 0 to input_a after 10 unit of time

- Use to generate a clock signal
  - Example later in lecture

Verilog test bench



- Test bench is a special module
  - Used for simulation only
    - Do not have input or output ports
    - Containing some system tasks that are not synthesizable: $monitor, $display,…
  - Containing Verilog code for:
    - Instantiating the unit under test
    - Generating input stimulus
    - Monitoring or displaying output response (text-based like printf() in ANSI-C)
      - Can used waveform viewer provided by tools

Test bench example

```
module log_eqn_sop8_tb;
   reg x1, x2, x3, x4;
   wire z1;
```

```
   initial                                   Display output response
      $monitor ("x1=%b,  x2=%b, x3=%b, x4 =%b, z =%b", x1,
x2, x3, x4, z1);
```

```
   initial begin
      #0 x1=1'b0; x2=1'b0; x3=1'b0; x4=1'b0;
      #10 x1=1'b1; x2=1'b1; x3=1'b1; x4=1'b0;
      #10 x1=1'b0; x2=1'b1; x3=1'b0; x4=1'b1;
      #10 x1=1'b1; x2=1'b1; x3=1'b1; x4=1'b1;
      #10 x1=1'b0; x2=1'b0; x3=1'b0; x4=1'b0;
      #10 $finish;      Stimulus
   end
```

```
   log_eqn_sop8 inst1 (.x1(x1),.x2(x2),.x3(x3),.x4(x4),.z1(z1));
```

Instantiation of UUT

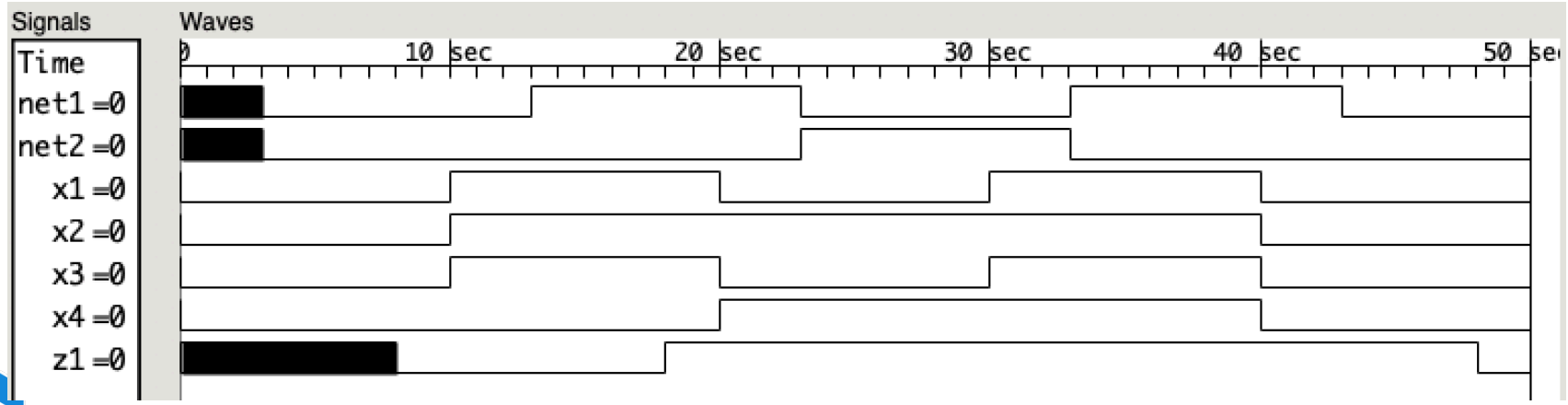
```
endmodule
```

```
module log_eqn_sop8 (x1, x2, x3, x4, z1);
   input x1, x2, x3, x4;
   output z1;
   and #3 inst1 (net1, x1, x2);
   and #3 inst2 (net2, ~x3, x4);
   or #5 inst3 (z1, net1, net2);
endmodule
```

Module under test

Output example

```
0 x1=0,  x2=0, x3=0, x4=0, z=x
8 x1=0,  x2=0, x3=0, x4=0, z=0
10 x1=1,  x2=1, x3=1, x4=0, z=0
18 x1=1,  x2=1, x3=1, x4=0, z=1
20 x1=0,  x2=1, x3=0, x4=1, z=1
30 x1=1,  x2=1, x3=1, x4=1, z=1
40 x1=0,  x2=0, x3=0, x4=0, z=1
48 x1=0,  x2=0, x3=0, x4=0, z=0
```



Test bench signals

```
reg x1, x2, x3, x4;
wire z1;
```

- Declare reg signals for inputs of the UUT
  - Can group all signals into a vector
  - E.g., reg [3:0] x; instead of x1, x2, x2, x4 in the previous example
- Declare wire signals for outputs of the UUT
  - E.g., wire z1;

Output test

```
$monitor ("x1=%b,  x2=%b, x3=%b, x4 =%b, z =%b", x1, x2, x3, x4, z1)
```

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

Format strings

- Formatting string
  - %h, %H: hex
  - %d, %D: decimal
  - %o, %O: octal
  - %b, %B: binary
  - %t: time (getting time by $time)

```
reg [3:0] x = 10; //only testbench
initial $monitor ("x = %b", x);
```

```
x = 1010
```

```
reg [3:0] x = 10; //only testbench
initial $monitor ("x = %h", x);
```

```
x = a
```

The initial block

- The initial block is one of the two behavioral blocks (along with always)
  - Mainly used in simulation (rarely used in design)
  - Execute statements **only one time** in sequential
    - Next statement must wait for the previous statement finished
  - All stimulus and system calls must be placed inside
  - Use begin… end if consisting multiple statements
    - The order of statements is important
- A module can contains unlimited number of blocks
  - Executed in parallel, starting from time 0

Stimulus

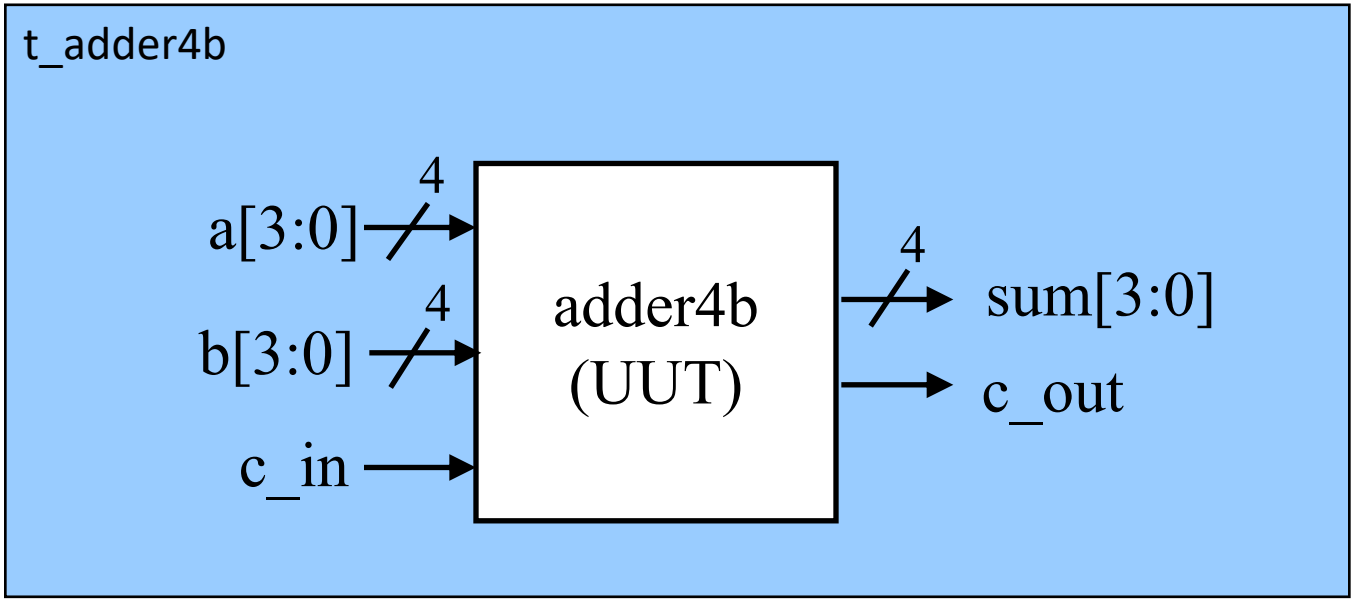
- Stimulus is the process of providing input signals to the UUT
- Can be done by one or more initial blocks
  - Except the clock signal, should use with an always block
  - All blocks are executed in parallel
- Blocks contain
  - Statements assign values to reg variables (input signals for the UUT)
  - Delay to spread assignments overtime
- Requirements
  - Combinational circuits: test **all possible combinations** of inputs
  - Sequential circuits: test **all possible states**

Values representation

- Format: <number of bits>'<base><value>
  - <base>: meaning of each digit in <value>
    - b: binary
    - h: hexadecimal
    - d: decimal (**default**)
    - o: octal
- Example: 4'b0000, 8'h10, 12'd2020
- Values assign rules:
  - 0 bits added to high-significant when assigning a small value to a large variable
    - reg [3:0] a = 2'b01; // only for simulation, a will store 4'b0001
  - Low-significant bits are selected when assigning a large value to a small variable
    - reg [3:0] a = 12'd2020; //only for simulation, a will store 4'b0100

Example

- Test a 4-bit full adder module (adder4b)
  - Don't care how the module is designed
  - Only need to check the correctness of the UUT



Example: stimulus version - 1

```
module t_adder4b;
  reg[8:0] stim;  // Combine all input signals into one vector
  wire[3:0] S;  // outputs of UUT are wires
  wire C4;

  adder4b a1(S, C4, stim[8:5], stim[4:1], stim[0]);
  initial $monitor("%t: %b %h %h %h %b", $time, C4, S, stim[8:5], stim[4:1], stim[0]);
  initial begin
      stim = 9'b000000000;  // at 0 ns
      #10 stim = 9'b111100001;  // at 10 ns
      #10 stim = 9'b000011111;  // at 20 ns
      #10 stim = 9'b111100010;  // at 30 ns
      #10 stim = 9'b000111110;  // at 40 ns
      #10 $stop;  // at 50 ns – stops simulation
  end
endmodule
```

Have not tested all combination yet

Example: stimulus version - 2

```
reg [3:0] a, b;          Have not tested all combination yet
reg c_in;
wire[3:0] S;  // outputs of UUT are wires
wire C4;

adder4b a1(S, C4, a, b, c_in);
initial begin
      a = 4'b0000; b = 4'b0000; c_in = 1'b0; // at 0 ns
      #10 a = 4'b1111; b = 4'b0000; c_in = 1'b1;  // at 10 ns
      #10 a = 4'b0000; b = 4'b1111; c_in = 1'b1;  // at 20 ns
      #10 a = 4'b1111; b = 4'b0001; c_in = 1'b0;  // at 30 ns
      #10 a = 4'b0001; b = 4'b1111; c_in = 1'b0;  // at 40 ns
      #10 $stop;  // at 50 ns – stops simulation
end
```

Example: stimulus version - 3

```
reg [3:0] a, b;
reg c_in;
wire[3:0] S;
wire C4;

adder4b a1(S, C4, a, b, cs);
initial begin
        a = 4'b0000;
        #10 a = 4'b1111;
        #10 a = 4'b0000;
        #10 a = 4'b1111;
        #10 a = 4'b0001;
        #10 $stop;
end
```

```
initial begin
        b = 4'b0000;
        #10 b = 4'b0000;
        #10 b = 4'b1111;
        #10 b = 4'b0001;
        #10 b = 4'b1111;
end
initial begin
        c_in = 1'b0;
        #10 c_in = 1'b0;
        #10 c_in = 1'b1;
        #10 c_in = 1'b0;
        #10 c_in = 1'b0;
end
```

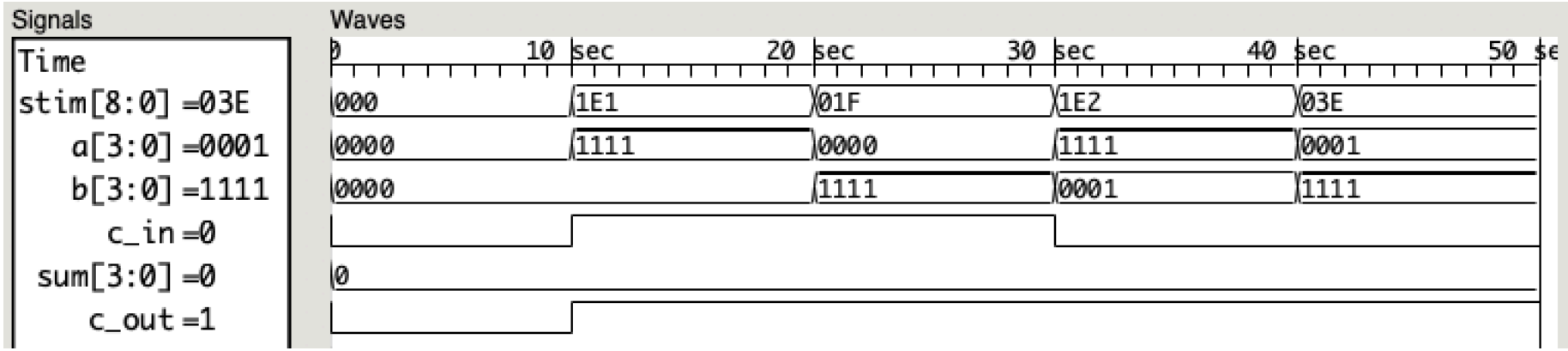
Advantages ???

Disadvantages ???

Oder of **initial** blocks are not important

Output

0: 0 0 0 0 0
10: 1 0 f 0 1
20: 1 0 0 f 1
30: 1 0 f 1 0
40: 1 0 1 f 0



Exhaustive testing

- For combinational designs w/ up to 8 or 9 bits input
  - Test ALL combinations of inputs to verify output
    - $2^9 = 512$ combinations
  - Could enumerate all test vectors, but don't…
- Generate them using a "for" loop!

```
reg [4:0] x;
initial begin
  for (x = 0; x < 16; x = x + 1)
    #5; // need a delay here!
end
```

Why do we need 5 bit for storing values up to 15?

- Should use "reg" data type for the loop index virable

Why extra bit added?

- Want to test all vectors 0000_2 to 1111_2

```
reg [3:0] x;
initial begin
  for (x = 0; x < 16; x = x + 1) #5;// need a delay here
end
```

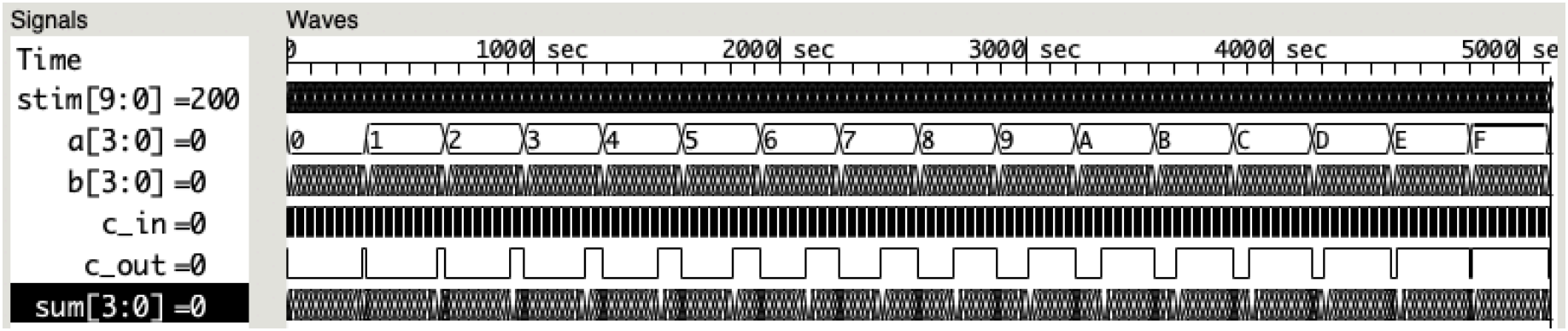
- If x is 4 bits, it only gets up to 1111_2 (15)
  - $1100_2 \Rightarrow 1101_2 \Rightarrow 1110_2 \Rightarrow 1111_2 \Rightarrow 0000_2$
- x is never >= 16… so loop goes forever!

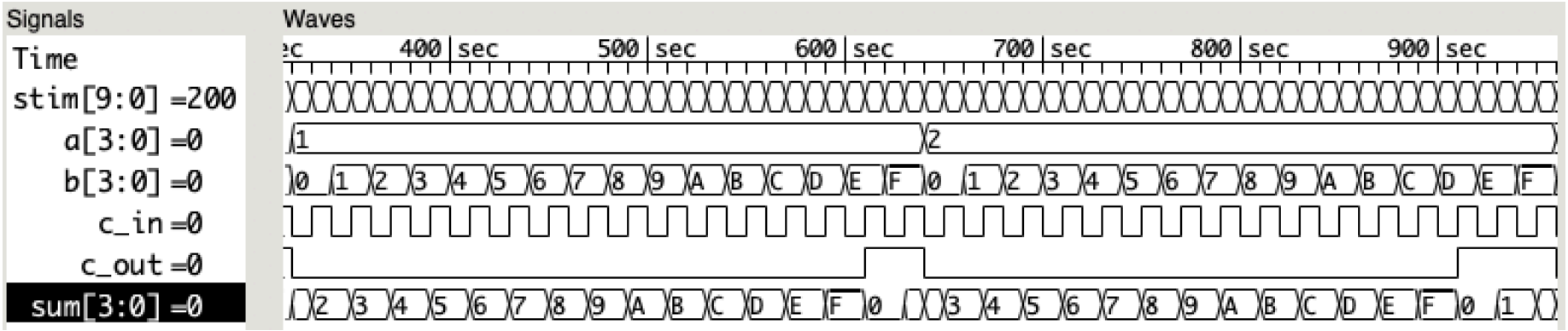
Example: stimulus version - 1

```
module t_adder4b;
  reg[9:0] stim;//10 bits instead of 9 bits
  wire[3:0] S;
  wire C4;

  adder4b a1(S, C4, stim[8:5], stim[4:1], stim[0]);
  initial $monitor("%t: %b %h %h %h %b", $time, C4, S, stim[8:5], stim[4:1], stim[0]);
  initial begin
      for (stim = 0; stim < 512; stim = stim + 1) #10;
      #10 $stop;
 end
endmodule
```

Output waveform

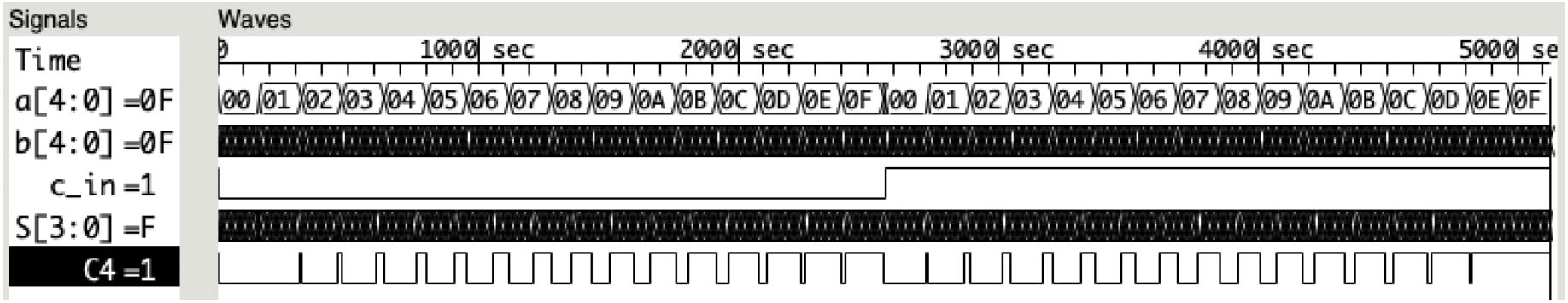


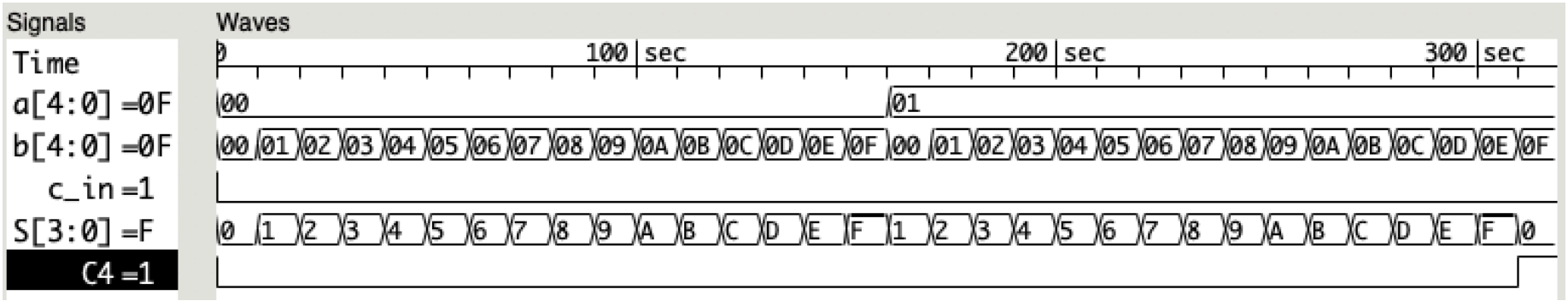


Example: stimulus version - 2

```
module t_adder4b;
  reg [4:0] a, b;
  reg c_in;
  wire[3:0] S;  // outputs of UUT are wires
  wire C4;
  adder4b a1(S, C4, a[3:0], b[3:0], c_in);
  initial begin
        c_in = 1'b0;
        for (a = 0; a < 16; a = a + 1)
              for (b = 0; b < 16; b = b + 1) #10;
        #10 c_in = 1'b1;
        for (a = 0; a < 16; a = a + 1)
              for (b = 0; b < 16; b = b + 1) #10;
        $stop;
  end
endmodule
```

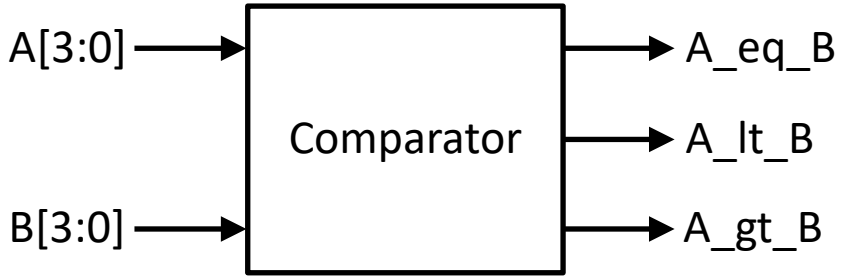
Output waveform





Exercise

- Write a test bench to verify a 4 bit comparator module




```
module Comp_4_str(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;
  // Code to compare A to B
  // and set A_gt_B, A_lt_B, A_eq_B accordingly
endmodule
```

Test bench for sequential circuits

- Sequential circuits need clock signals
  - Generating a clock signal

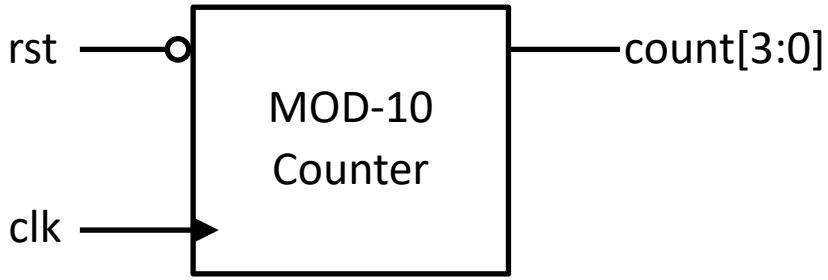
```
reg clk;
initial begin
   clk = 1'b0;
   forever #5 clk = ~clk
end
```

```
reg clk;
initial clk = 1'b0;
always #5 clk = ~clk;
```

- Test all possible states of the UUT
  - Leave enough time for generating all outputs each state

Example

- Write a test bench to test a MOD-10 counter




```
module mod10_counter(rst, clk, count);
output [3:0] count; //0->9->0
input   clk, rst;
  // Code to count
  // and set count back to 0 after 9
endmodule
```

Test bench

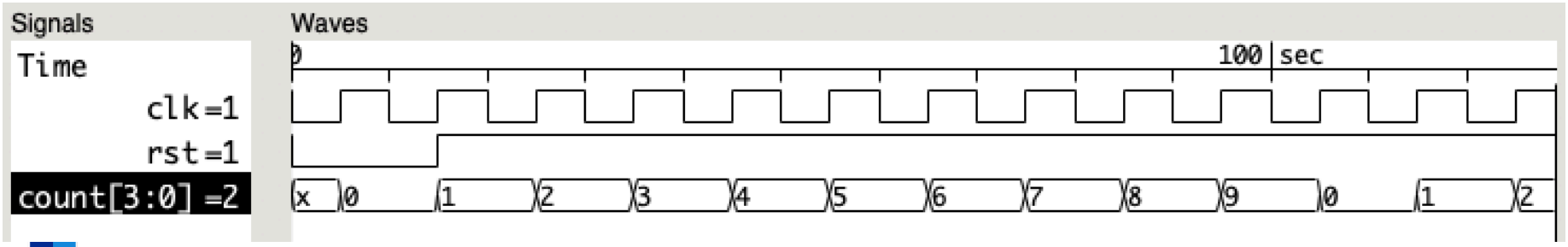
```
module t_mod10_counter;
   reg clk, rst;
   wire [3:0] count;

   mod10_counter M1(.clk(clk), .rst(rst), .count(count));

   initial $monitor("time = %t: rst = %b, count = %b", $time, rst, count);
   initial begin //generating the clock signal
      clk = 1'b0;
      forever #5 clk = ~clk;
   end
   initial begin
      rst = 1'b0; //test reset
      #15 rst = 1'b1; //leave rst active in at least a cycle
      #120 $stop; //let the counter count in at least 11 cycles
   end
endmodule
```

Output

```
time =                0: rst = 0, count = xxxx
time =                5: rst = 0, count = 0000
time =               15: rst = 1, count = 0001
time =               25: rst = 1, count = 0010
time =               35: rst = 1, count = 0011
time =               45: rst = 1, count = 0100
time =               55: rst = 1, count = 0101
time =               65: rst = 1, count = 0110
time =               75: rst = 1, count = 0111
time =               85: rst = 1, count = 1000
time =               95: rst = 1, count = 1001
time =              105: rst = 1, count = 0000
time =              115: rst = 1, count = 0001
time =              125: rst = 1, count = 0010
```



Force/release in test bench

- Syntax:

```
force <signal> = <value>;
force <instance>.<signal> = <value>;
```

```
release <signal>;
release <instance>.<signal>;
```

- Allows you to "override" values **for simulation**
- Doesn't do anything in "real life"
  - No fair saying "if 2+2 == 5, then force to 4", synthesizers won't allow force…release anyway
- Can be applied to nets as well as register
  - Useful for debugging connectivity errors

Example

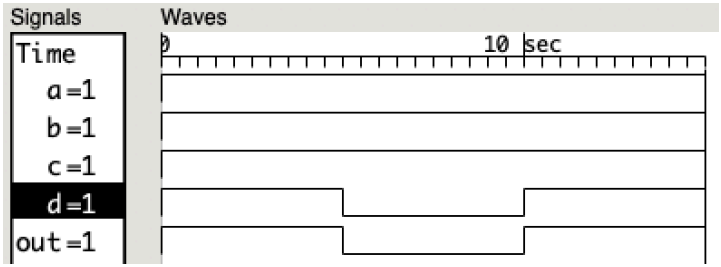
```
module t_force_release;
  reg a, b, c;
  wire out;

  force_release F1(a, b, c, out);
  initial begin
   $monitor ("a = %b, b = %b, c = %b, out = %b", a, b, c, out);
  end

  initial begin
    a = 1;
    b = 1;
    c = 1;
    #5 force F1.d = 0;
    #5 release F1.d;
    #5 $stop;
  end
endmodule // t_force_release
```

```
module force_release(a, b, c,
out);
  input a, b, c;
  output out;
  wire   d;
  and (d, a, b);
  and (out, d, c);
endmodule // force_release
```

```
a = 1, b = 1, c = 1, out = 1
a = 1, b = 1, c = 1, out = 0
a = 1, b = 1, c = 1, out = 1
```



Concluding remarks

- Test bench for simulating modules
  - Combinational circuits: test all possible input combinations
  - Sequential circuits: check all possible states
- Signals in test bench
  - `reg` for input signals of UUT
  - `wire` for output signals of UUT
  - Clk signals: use `forever` or `always` (copy/paste)
- Watch results
  - Text (`$monitor`, `$display`)
  - Waveform

The end