ECE 4514
Digital Design II
Spring 2008

Lecture 9:
Review of Key Ideas, System Commands and Testbenches

A *Language* Lecture

Patrick Schaumont

Iterating the Key Ideas

- Verilog is a *modeling* language. It cannot express hardware directly. It describes the activities of a hardware implementation.

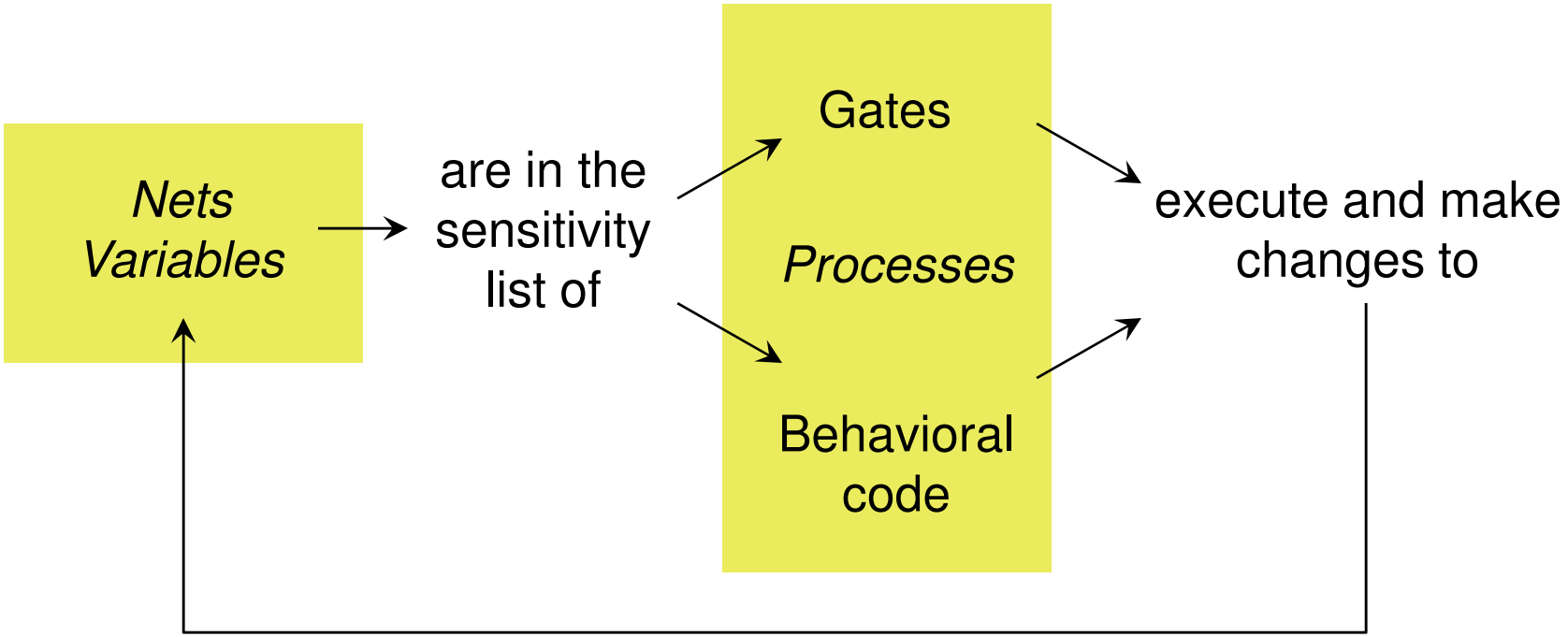
```
module a_module(q, rst, clk, d);
  output q;
  input rst;
  input clk;
  input d;
  wire next_r1;
  reg  r1;

  always @(posedge clk or negedge rst)
    if (rst)
      r1 = next_r1;
    else
      r1 = 0;

  assign next_r1 = d;
  assign q = r1;
endmodule
```

This fragment describes the behavior of a flip-flop but does not describe the flip-flop itself!

It describes the flip-flop using variables (reg), nets (wire), and processes (always, assign).

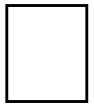
VERILOG = 'VERIfy LOGic'

Verilog Simulation Cycle



Iterating the Key Ideas

Simulation Time and Variables/Nets in Verilog



variable (reg)

- once assigned, will remember what is stored into it
- startup value = X
- value at time T defined by the previous assignment on the variable

wire (net)

- can hold a value only at the current time (time = T)
- startup value = X
- value at time T defined by the variable that drives the wire

Always and Assign

always blocks

- within block statement (begin .. end), executes sequentially
- will only stop when waiting for an event (wait, @, #)
- assignment (=) only on variables (reg) ('procedural assignment')
- intra-assignment delay possible

assign statement

- assignment (=) only on nets (wire) ('continuous assignment')
- no intra-assignment delay possible

Modeling Styles

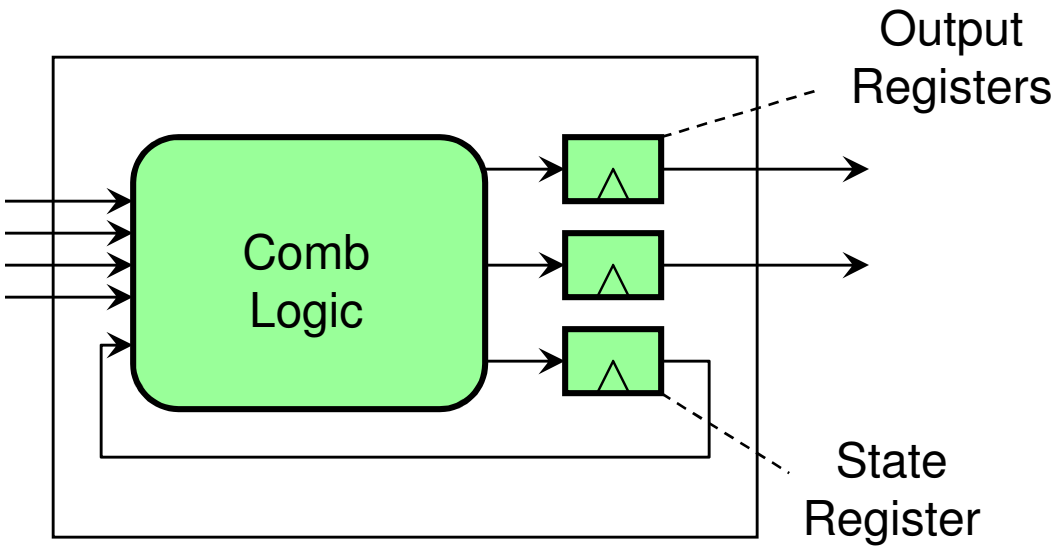
gate-level

- model hardware with structure, by interconnecting primitives

multiplexed data-path

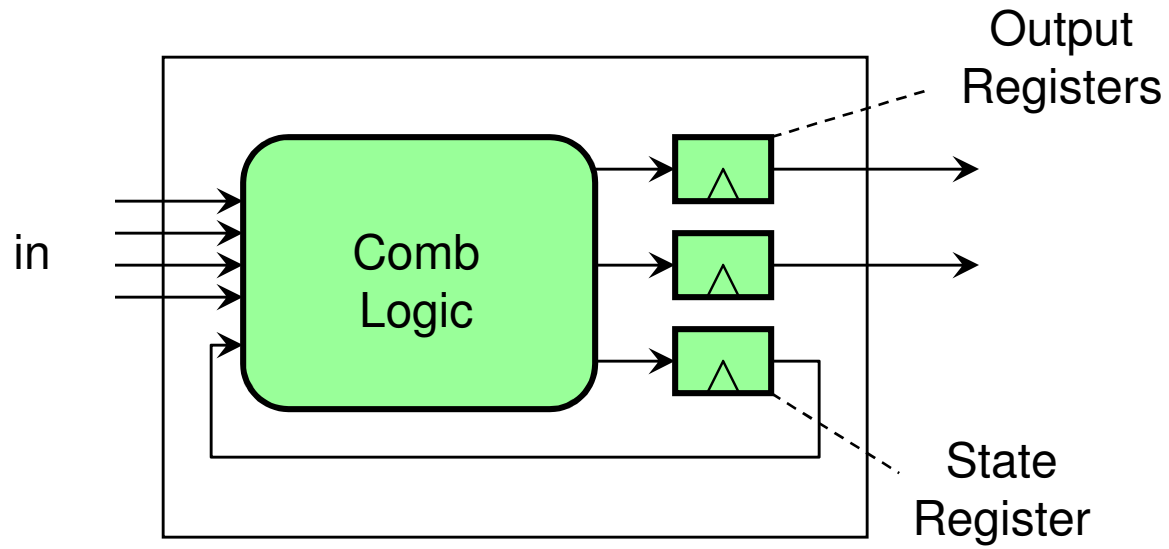
- model combinational logic with assign expressions, and flip-flops with wire/reg combination.
- update registers with always block

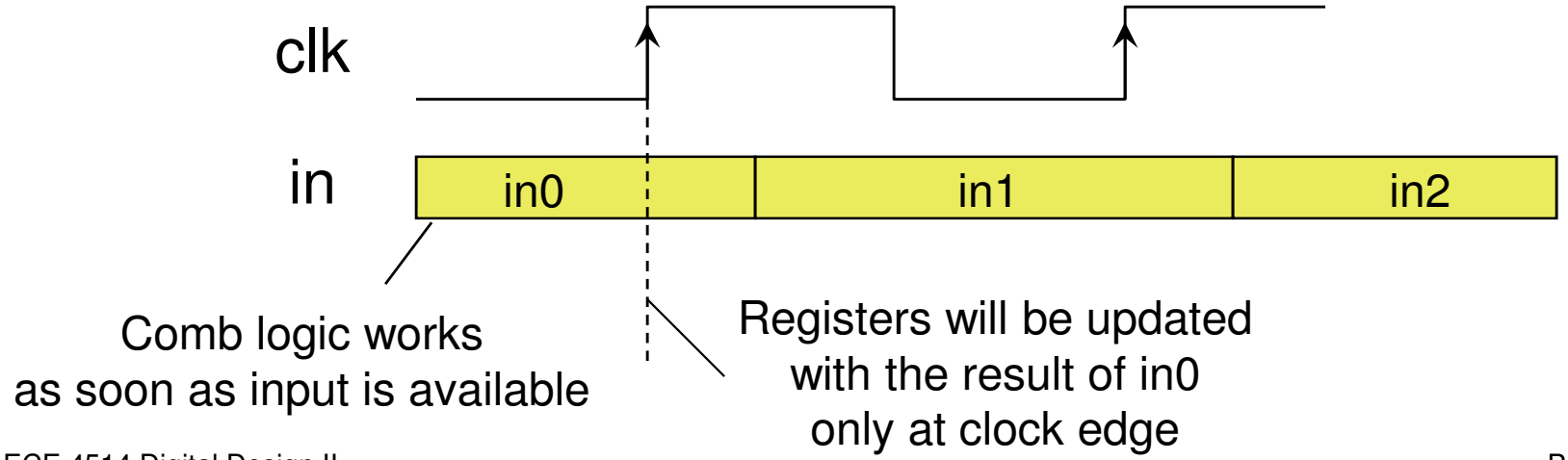
Inputs, Outputs, Registers, ..



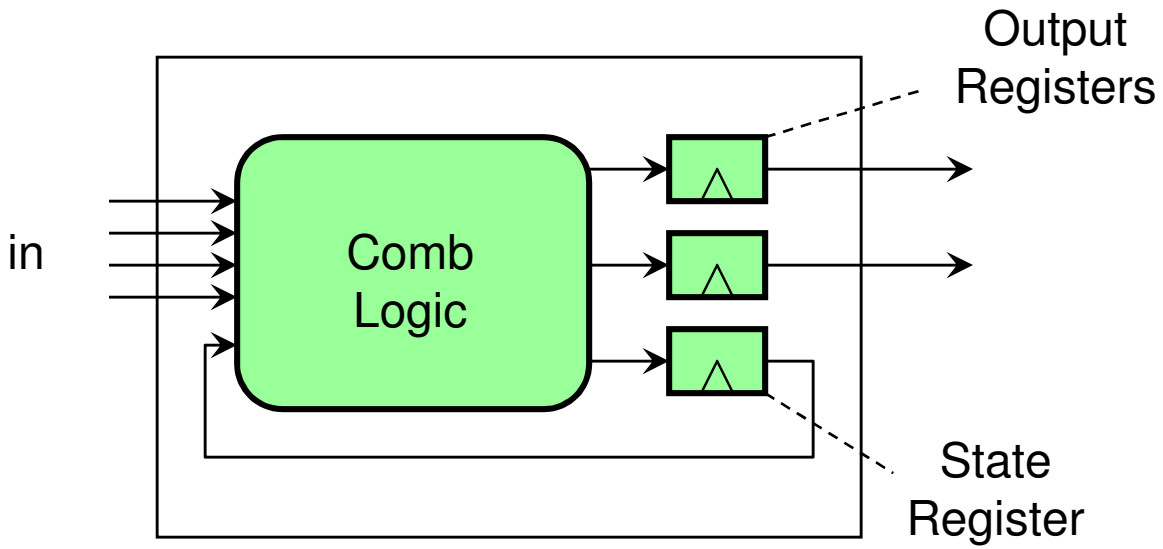
- Default structure for a single-clock module
  - Registers at the output should prevent a combinational path from input to the output.
  - *Registers at the output* does not mean that an output always MUST be a register. Exceptions do exist, and designers may have good arguments for it.

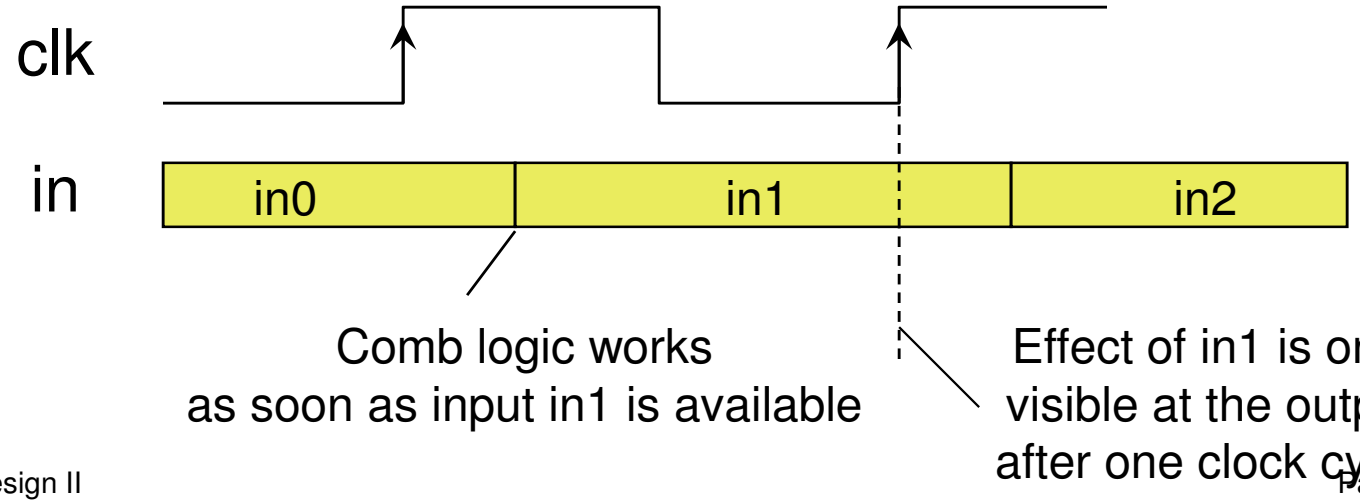
Inputs, Outputs, Registers, ..



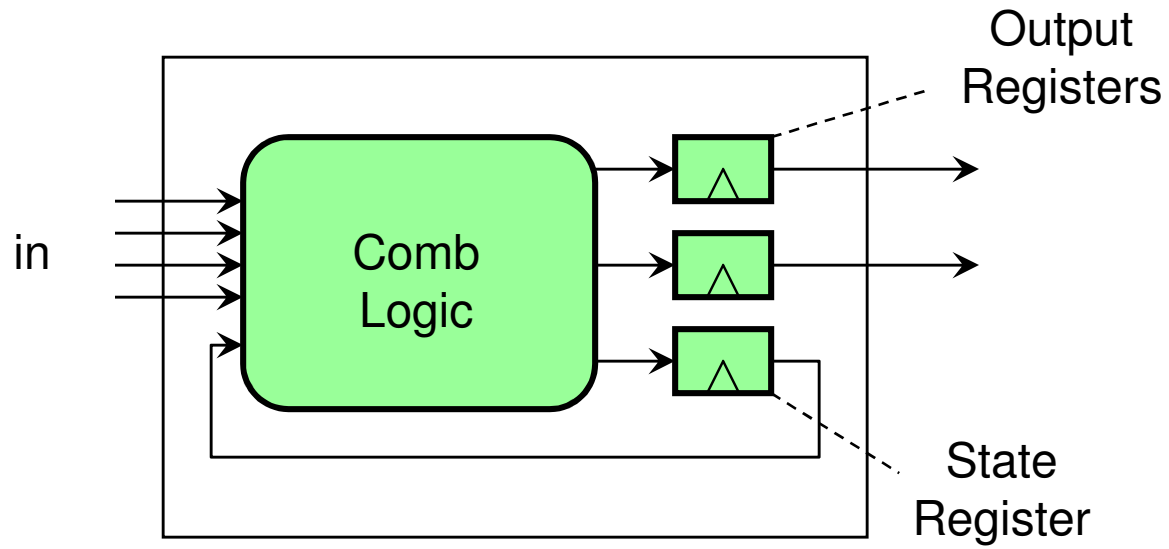


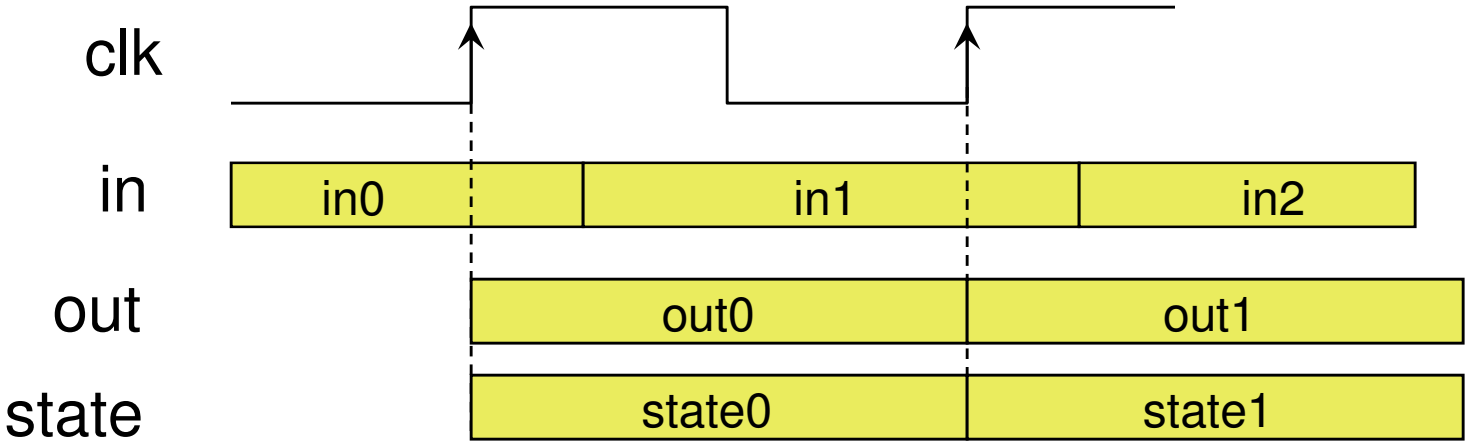
Inputs, Outputs, Registers, ..



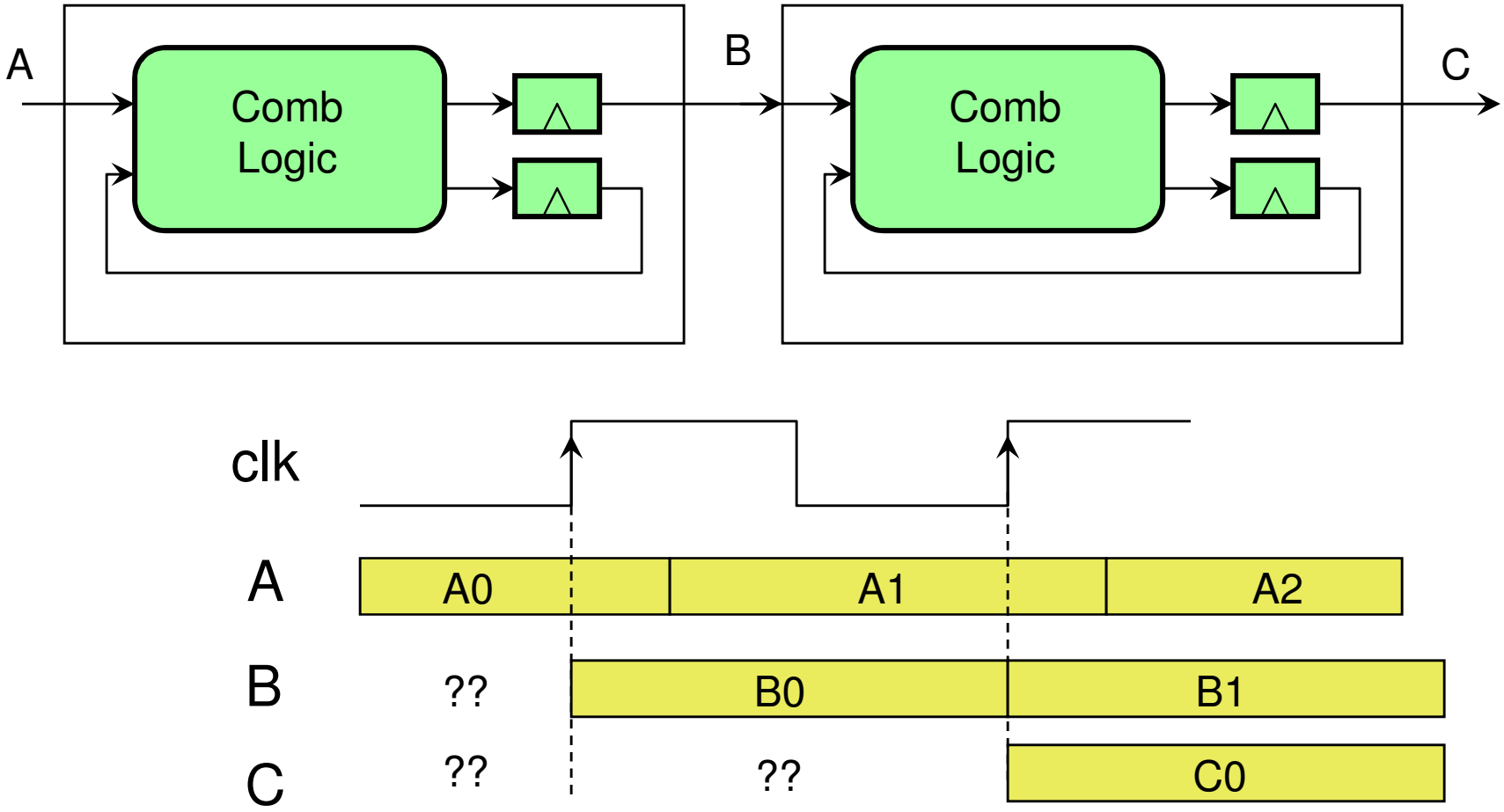


Inputs, Outputs, Registers, ..





Inputs, Outputs, Registers, ..



Testbench Design and System Commands

System Commands and Testbenches

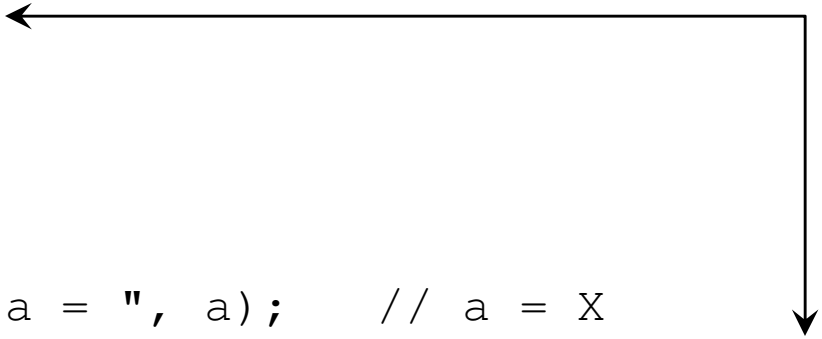
- Printing
- File IO
- VCD Files

Printing: $display

```
module helloworld;

initial begin
  $display("Hello World");
end

endmodule
```

$display for values

```
module helloworld;

reg [7:0] a;

initial begin
  #10 a = 30;
  #20 a = 40;
end;

initial begin
    $display("a = ", a);    // a = X
#10 $display("a = ", a);    // a = 30 indeterminate
#10 $display("a = ", a);    // a = 30
#10 $display("a = ", a);    // a = 40;
end

endmodule
```



Formatted $display

```
module helloworld;

reg [7:0] a;

initial begin
   a = 43;
   $display("a = %b", a);
   $display("a = %x", a);
   $display("a = %c", a);
   $display("a = %o", a);
end

endmodule
```

a = 00101011
a = 2b
a = +
a = 053

Formatted $display with multiple variables

```
module helloworld;

reg [7:0] a, b;

initial begin
   a = 43;
   b = 68;
   $display("a = %b and b = %d", a, b);
end

endmodule
```

```
a = 00101011 and b =  68
```

Special Sequences: %m, %l, $time

```
module helloworld;

initial begin
#10 $display("Module: %m");
    $display("Time: %d", $time);
    $display("Library: %l");
end

endmodule
```

```
Module: helloworld
Time:                    10
Library: work.helloworld
```

Printing whatever changes: $monitor

```
module helloworld;

reg a;

initial begin
  a = 0;
  # 35 a = 1'bx;
end

always #10 a = ~a;

initial #15 $display("%d d1 ", $time, a);

always  @(a) $display("%d d2 ", $time, a);

initial $monitor("%d d3 ", $time, a);

endmodule
```

```
0 d2 0
0 d3 0
10 d2 1
10 d3 1
15 d1 1
20 d2 0
20 d3 0
30 d2 1
30 d3 1
35 d2 x
35 d3 x
```

Files: $fopen, $fdisplay, $fclose

```
module helloworld;

reg [2:0] a;
integer thefile;

initial thefile = $fopen("theoutput.out");

initial a = 0;
always #10 a = a + 1;

initial begin
   #105 $fclose(thefile);
        $finish;
end

always  @(a) $fdisplay(thefile, "%b", a);

endmodule
```

File Descriptor

Contents:

000
001
010
011
100
101
110
111
000
001
010

Reading from a file: $fscanf

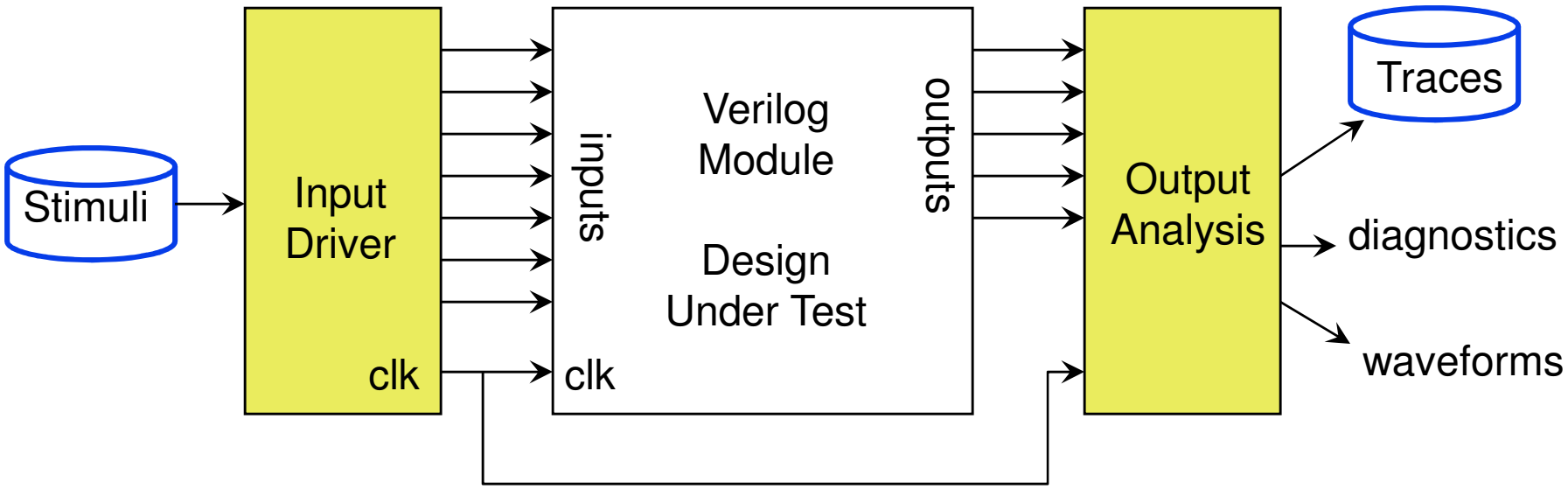
```
module helloworld;

reg [5:0] a, b;
integer thefile;

initial thefile = $fopen("theinput.txt","r");

always  #10 begin
   $display("Read %d %b", a, b);
   $fscanf(thefile, "%d %b", a, b);
   end

endmodule
```

theinput.txt

```
23  00000
14  0011
144 01011
12  0111111
```

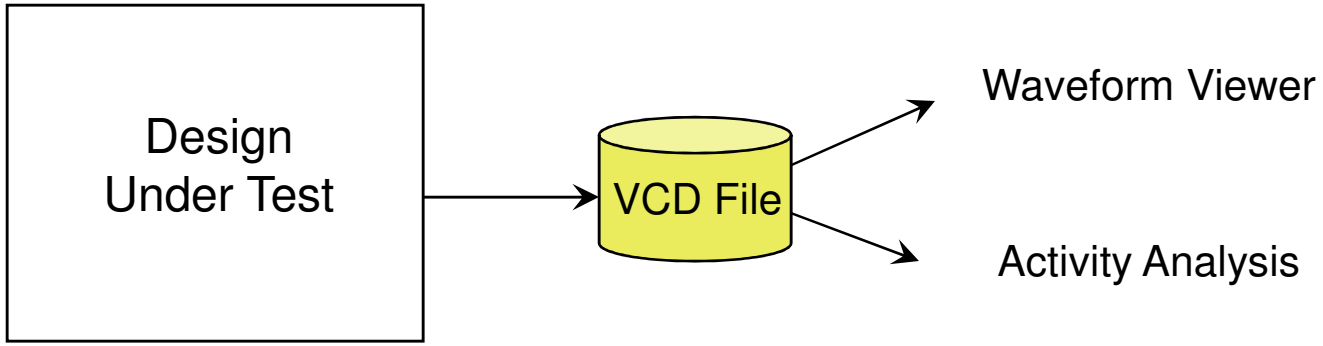
```
Read  x xxxxxx
Read 23 000000
Read 14 000011
Read 16 001011
Read 12 111111
Read 12 111111
...
```

Typical Application: Testbenches



VCD Files

- VCD = Value Change Dump
- A textual format that stores waveform data for a design
- Used by post-processing tools
  - Waveform Viewer
  - Activity Analyzer (Power estimator)
- Can be generated through Verilog System Tasks or through Modelsim commands



VCD Files - Example

```
module tsttoggle;

  wire q;
  reg clk, rst;

  initial $dumpfile("toggletst.vcd");
  initial $dumpvars(1, tsttoggle);

  toggle T(q, clk, rst);

  initial clk = 0;
  initial begin rst = 1;
            #10 rst = 0;
            #10 rst = 1;
          end
  always #50 clk = ~clk;

endmodule
```

Create VCD File toggletst.vcd
Dump all signals from module tsttoggle
(1 means: only current level of hierarchy)

Design Under Test
(Toggle Flip-flop)

toggletst.vcd

Declarations

```
$date
       Sun Feb 10 12:11:25 2008
$end
$version
       ModelSim Version 6.3c
$end
$timescale
       1ns
$end
$scope module tsttoggle $end
$var wire 1 ! q $end
$var reg 1 " clk $end
$var reg 1 # rst $end
$upscope $end
$enddefinitions $end
```

Waveform Data

```
#0
$dumpvars
0"
1#
x!
$end
#10
0#
0!
#20
1#
#50
1"
1!
#100
0"
```

toggletst.vcd

```
$date
        Sun Feb 10 12:11:25 2008
$end
$version
        ModelSim Version 6.3c
$end
$timescale
        1ns
$end
$scope module tsttoggle $end
$var wire 1 ! q $end
$var reg 1 " clk $end
$var reg 1 # rst $end
$upscope $end
$enddefinitions $end
#0
$dumpvars
0"
1#
x!
$end
#10
0#
0!
#20
1#
#50
1"
1!
#100
0"
```

q is a 1-bit wire, has symbol !

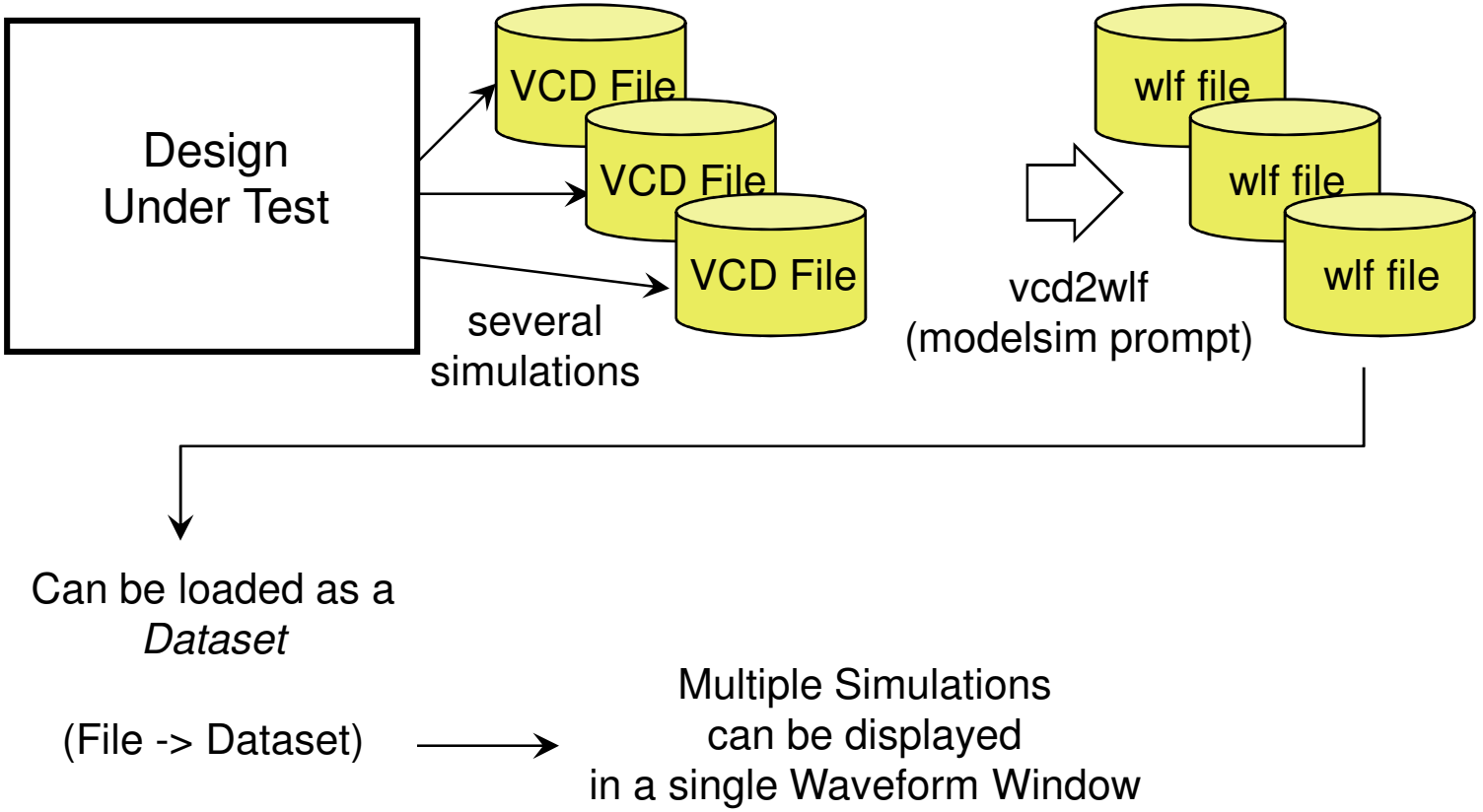
At 0, q = x

At 10, q = 0

At 50, q = 0

Using VCD Files in Modelsim

- Show data from multiple simulations



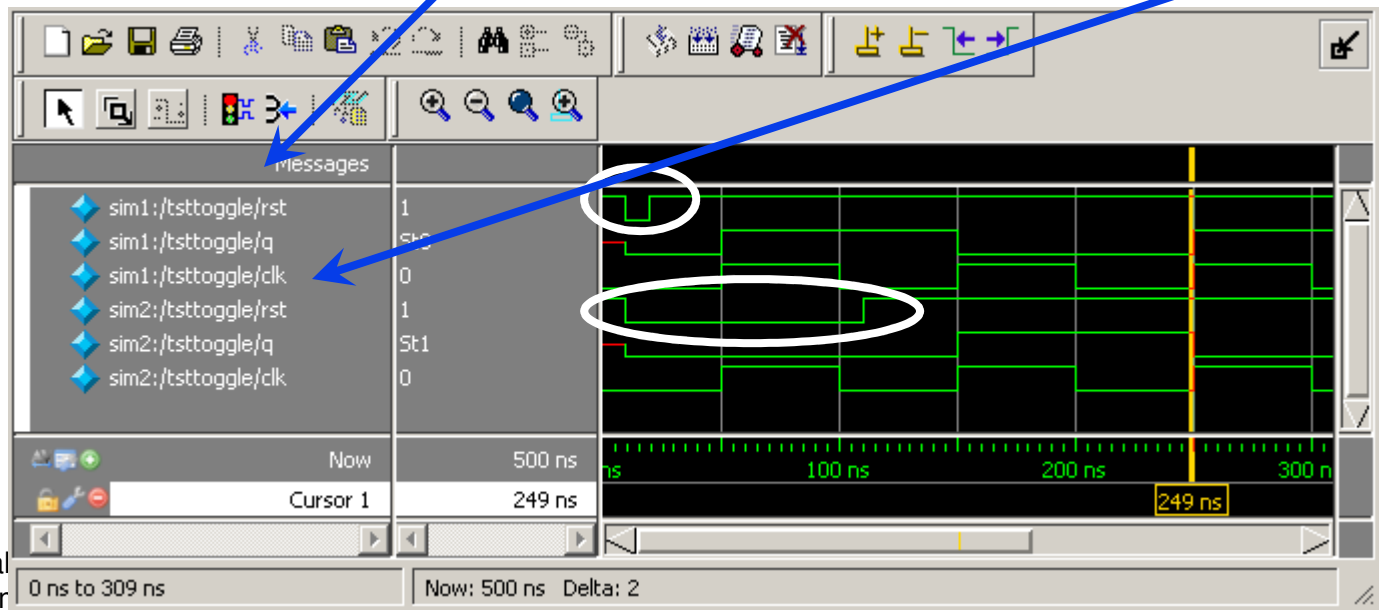
Example

```
module tsttoggle;
  wire q;
  reg clk, rst;
  initial $dumpfile("toggletst.vcd");
  initial $dumpvars(1, tsttoggle);
  toggle T(q, clk, rst);
  initial clk = 0;
  initial begin rst = 1;
            #10 rst = 0;
            #10 rst = 1;
          end
  always #50 clk = ~clk;
endmodule
```

sim1 dataset

```
module tsttoggle;
  wire q;
  reg clk, rst;
  initial $dumpfile("toggletst.vcd");
  initial $dumpvars(1, tsttoggle);
  toggle T(q, clk, rst);
  initial clk = 0;
  initial begin rst = 1;
            #100 rst = 0;
            #10 rst = 1;
          end
  always #50 clk = ~clk;
endmodule
```

sim2 dataset



Summary

- Key Ideas of Verilog Modeling
  - wire and reg; always-blocks and assign
- System tasks
  - $display, $monitor
  - $fopen, $fclose, $fdisplay, $fscanf
  - Testbenches based on file IO
- Value Change Dump
  - Record signal activity in a file