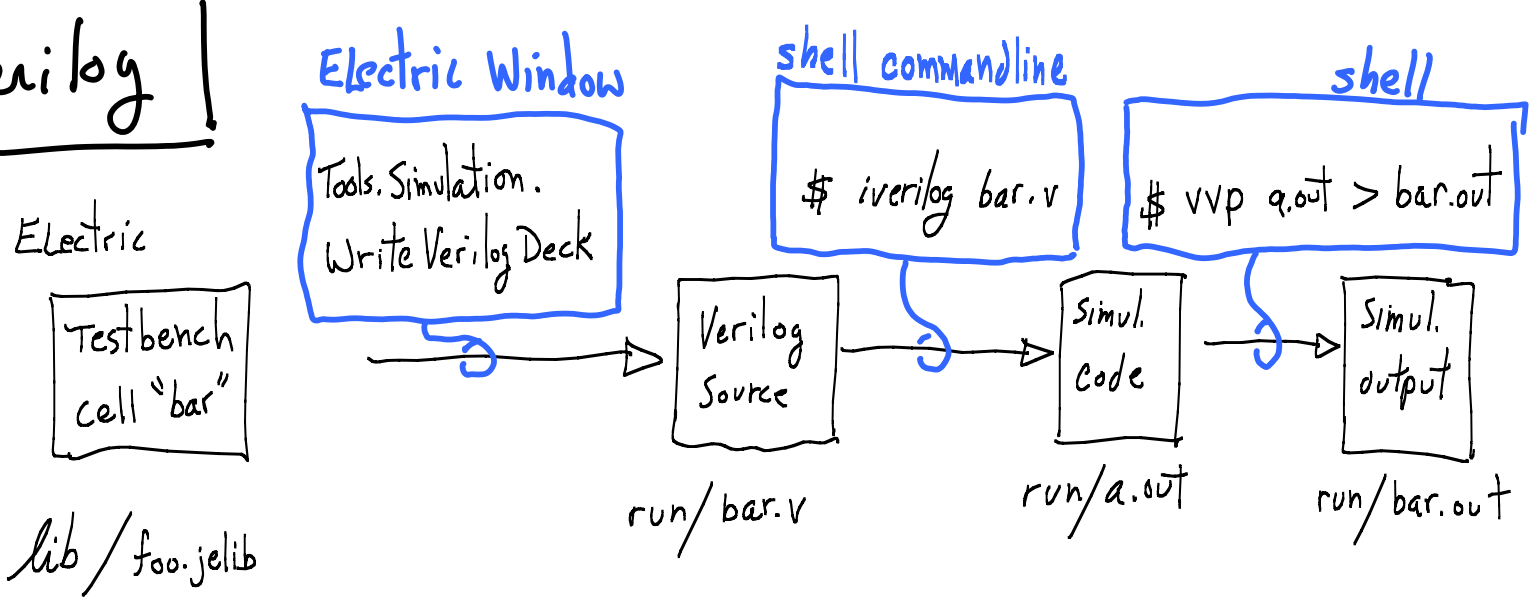
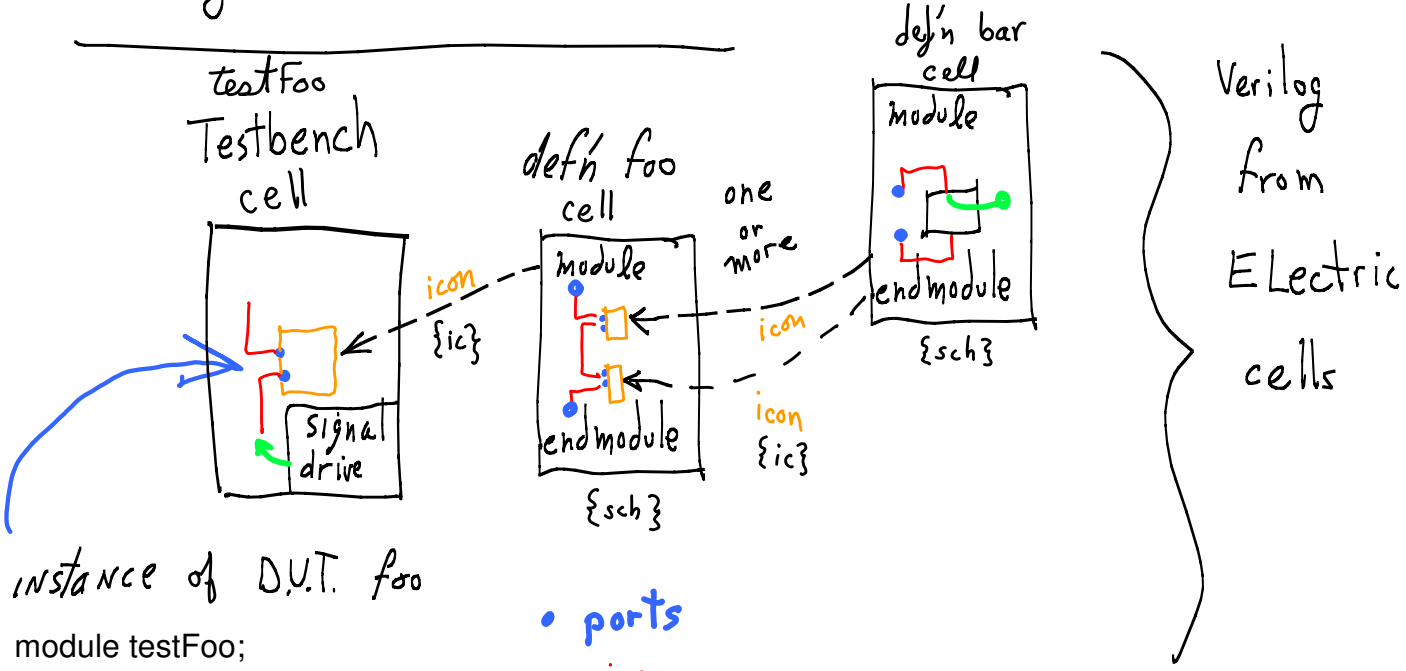
Verilog !



Verilog Code Structure



instance of D.U.T. foo

- ports
- wires
- regs

```
module testFoo;

   wire res;
   reg sig;

   lib__foo( res, sig );

   initial begin
      sig = 0;
      #100 $finish;
   end

   always begin
      #1 sig = ~sig;
   end

end module
```

```
module foo( out, in);

   input in; wire in;
   output out; wire out;
   wire x;


   lib__bar( out, x);
   lib__fubar( x, in);

endmodule;
```

```
module bar( a, b);

   input b; wire b;
   output a, reg a;

   always @( b ) begin
      a = ~b;
   end
   initial begin
      a = 0;
   end

endmodule
```

Structural vs. Behavioral

Structural ⟷ wire, gates, devices: wire, reg, AND, or, ...

Behavioral ⟷ if( ) then( ), while, wait, ..., case, ...

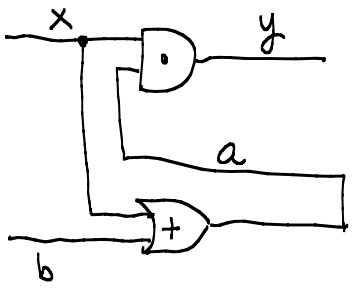
"Helper" Language
integer, ..., while, ...

Gate-level

STRUCTURAL

via "primitives"

```
module foo ( y, x, b );

  input x, b;
  output y;
  wire y, x, a, b;

  and   and_0( y, x, a );
  or    or_0( a, b, x);

endmodule
```



OR

via "continuous assignment"

```
module foo ( y, x, b );
  input x, b;
  output y;
  wire y, x, a, b;

  assign y = (x & (x | b));

endmodule
```

Delays

```
initial begin

    #1 a = 0;
    #1 b = 0;
    #1 c = 0;

end
```

When (what simulation time) does

a become 0?
b become 0?
c become 0?

Signal values
---

x == don't know
z == disconnected
0
1

(x & 0) = ?
(x | 0) = ?

```
$display("time = %0d  a = %b", $time, a );
...      "        b );
...      "        c );
```

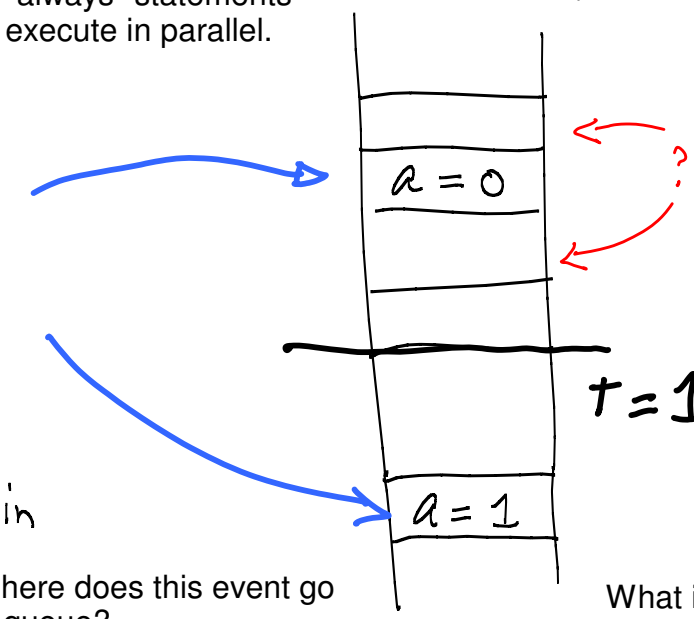
what are the values of a, b, c for every tic?

Event Queue

All "initial" and "always" statements execute in parallel.

```
initial begin
  a = 0;
  #1 a = ~a;
end
```

```
always @(a) begin
  c = ~a;
end
```




```
initial begin
  b = 1;
end
```

```
initial begin
  $display(... a)
end
```

Where does this event go in queue?

What is the value printed by $display("a=%d", a)?

What timestamp does the event $display() have?

Events cause other events: signal "a" changes, creates event that changes "c".

Events placed in queue, pulled from queue for current step,new events added, until no events left in this time step.

Discrete Event Simulation

(versus Discrete Time Simulation)

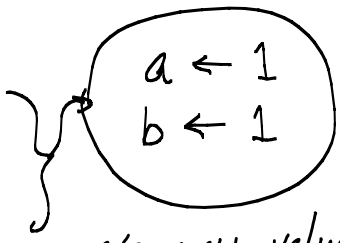
← no ordering of events *

Ordering of Events*

c = 0
a = 0
event: c ← 1

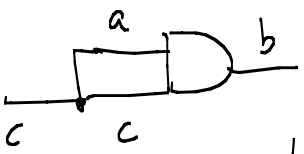
```
input c;
wire c;

output a, b;
reg a, b;

always @(c) begin
   a = c;
   b = a & c;
end
```

blocking assignment



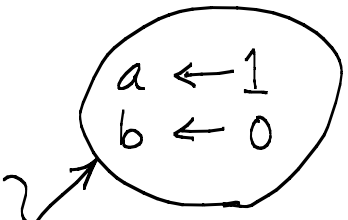
a's new value used for b.

RHS evaluated in order, after preceding LHS assignment




```
input c;
wire c;

output a, b;
reg a, b;

always @(c) begin
   a <= c;
   b <= a & c;
end
```

non-blocking assignment



a's old value used for b

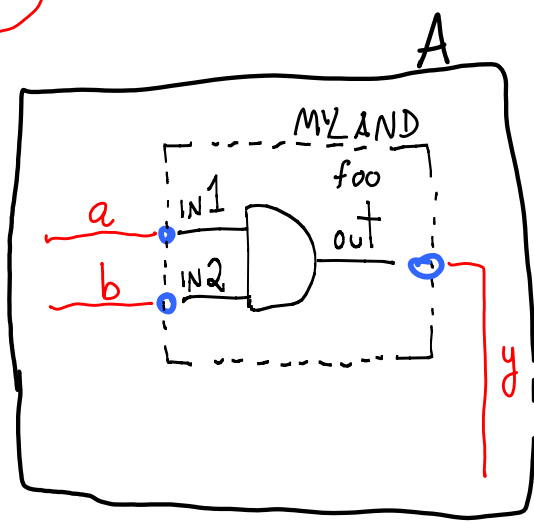
RHS evaluated in parallel, globally, before LHS assignment

Ports by name

Should def'n of MY_AND be inside def'n of A?

How deep can nesting be?

```
module A;
  wire a, b, y;
  MY_AND foo( .in1( a ), .in2( b ), .out( y ) );
endmodule

module MY_AND( out, in1, in2 );
  out <= #1 (in1 & in2);
endmodule
```



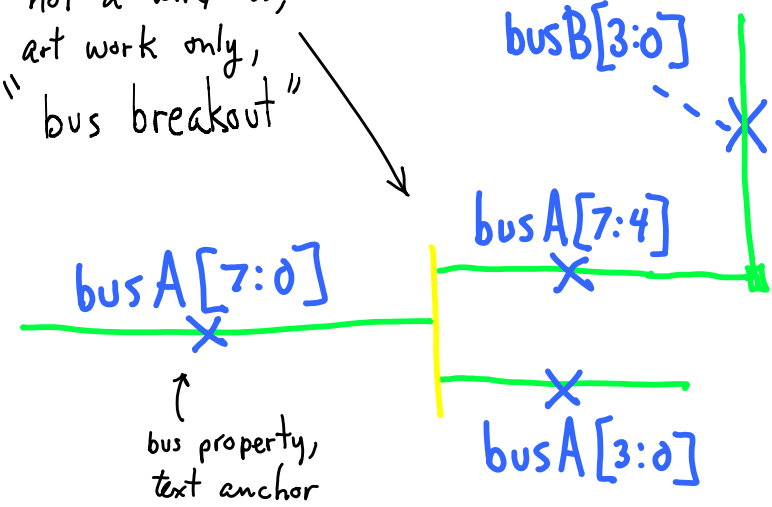
OR

MYAND foo(y, a, b)

Busses, Arrays

```
reg [7:0] busA;
wire [3:0] busB;

assign busB = busA[4:7];

initial begin
   busA = 8'd255;
end

//-- busA = 8'hff;
//-- busA = 8'b11111111;
```

not a wire/bus, art work only, "bus breakout"



(See, Edit.SelectObject)

nums

size(# bits)	format	number
8	d	255
	h	FF
	b	1111 1111

(d = decimal format; h = hex format; b = binary format)

Combining

```
input  [3:0]  a, b;
output [15:0] y;

assign y = { 3 { a[3:0] },  b[3:2],  2'b00  };
```

what is connected to:

y[0]? → 0
y[3]? → b[3]
y[4]? → a[0]
y[12]? → a[0]
y[15]? → a[3]

```
`define busWidth 16
`include foo.vh
reg [(`busWidth-1):0] busA;
```

```
always @(posedge clk) begin
    a = b;
    @(negedge clk)
    a = ~b;
    @(c or clk)
    a = 0;
end
```

What's in "a" if c doesn't change?

ARRAYS

```
reg [7:0] regWords [15:0];
```

type id array size

```
regWords[1] = 8'b01010000;
regWords[1][0] = 1'b1;
```

What's in RegWords[1] ?

→ 01010001

Tasks = methods

```
module memory(...);
  ...

  reg [7:0] regWords [15:0];
  integer i;

  task clear;
      begin
        for ( i = 0; i < 15; i = i + 1) begin
            regWords[i] = 8'd0;
        end
      end
  endtask

endmodule
```

Task

invoke task

```
module top ();

  memory mem;

  initial begin
      mem.clear;
      mem.regWords[0] = 8'b000111;
  end

endmodule
```

Names from above

in Top : instance.instance.thing

Pre-defined procedures | VPI

- $display(" ", ...); has eoln
- $write(" ", ...); no eoln
- $time sim. time step
- $monitor(" ", x); } like $display, but w/ always @(x)
- $strobe } (broken?)
- $fwrite
- $fopen (?)
- ⋮

- $readmemb("file", dataArray); file contain binary notation
- $readmemh("file", dataArray); file contains hex notation

- $dumpl( ) — use w/ GTKwave
  dumps every signal @ every change

- $stop goes into interactive mode

^C sends "kill" signal to process, interactive mode, use $finish