SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

19ECB302–VLSI DESIGN

III YEAR/ V SEMESTER

1

UNIT 5-SPECIFICATION USING VERILOG HDL

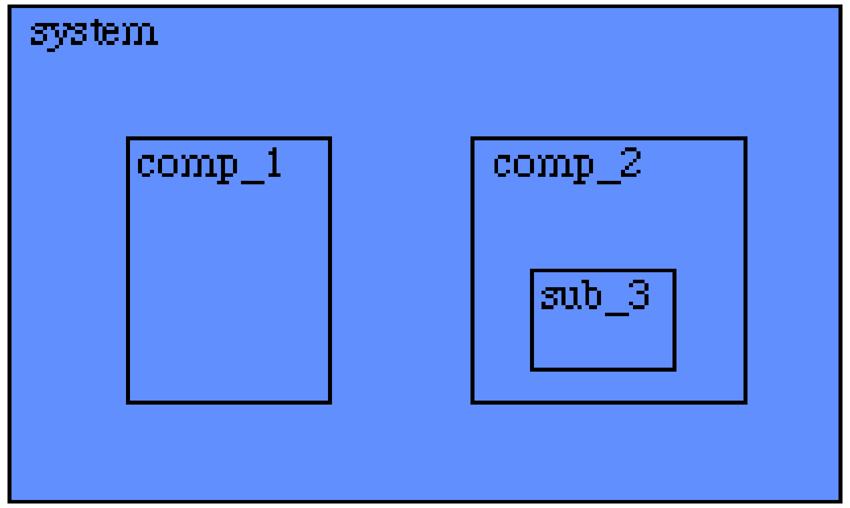
TOPIC 9,10–DESIGN HIERARCHIES, BEHAVIORAL AND RTL MODELING

OUTLINE

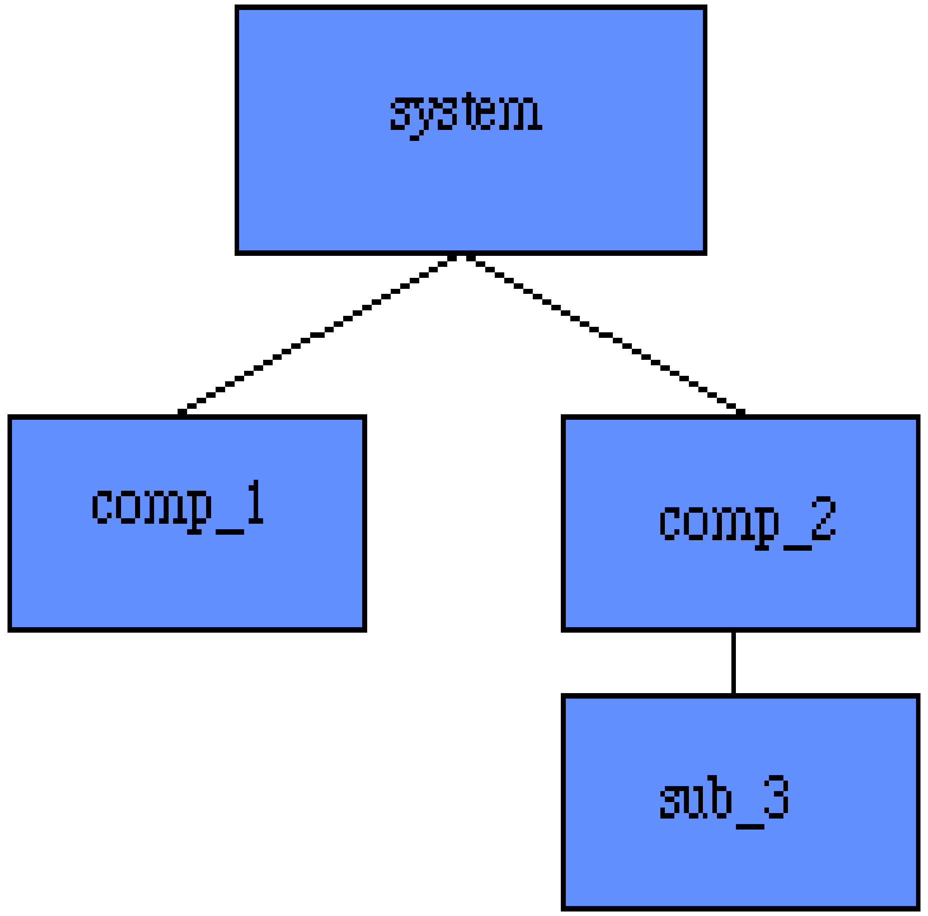
- DESIGN HIERARCHIES
- WIRE AND VECTOR ASSIGNMENT
- VECTORS OF WIRES
- SIGNAL AND SIGNAL EDGE SENSITIVITY
- TWO ROLES OF HDL AND RELATED TOOLS
- SYNTHESIS VS SIMULATION
- ACTIVITY
- STRUCTURAL VS BEHAVIORAL HDL CONSTRUCTS
- THREE MODULE COMPONENTS-DATA FLOW,BEHAVIOURAL(RTL),STRUCTURAL
- MIXED MODELING STYLE
- ASSESSMENT
- SUMMARY

DESIGN HIERARCHIES

- Represent the hierarchy of a design



  - modules
    - the basic building blocks
  - ports
    - the I/O pins in hardware
    - input, output or inout



DESIGN HIERARCHIES

- The module is the basic building block in Verilog
  - Modules can be interconnected to describe the structure of your digital system
  - Modules start with keyword module and end with keyword endmodule

```
Module AND <port list>

    •
    •
    •

endmodule
```

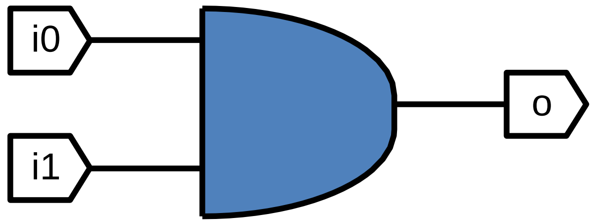
```
Module CPU <port list>

    •
    •
    •

endmodule
```

  - Modules have ports for interconnection with other modules

- Module Ports
  - Similar to pins on a chip
  - Provide a way to communicate with outside world
  - Ports can be input, output or inout




```
Module AND (i0, i1, o);
   input  i0, i1;
   output o;

endmodule
```

DESIGN HIERARCHIES

- Can (should) specify module connections by name
  - Helps keep the bugs away
  - Example

```
mux2to1 mux1 (.A (A[1])
              .B (B[1]),
              .O (O[1]),
              .S (Sel)   );
```

  - Verilog won't complain about the order (but it is still poor practice to mix them up):

WIRE AND VECTOR ASSIGNMENT

- Wire assignment: "continuous assignment"
  - Connect combinational logic block or other wire to wire input
  - **Order of statements not important to Verilog**, executed totally in parallel
  - But order of statements can be important to clarity of thought!
  - When right-hand-side changes, it immediately flows through to left
  - Designated by the keyword **assign**

```
wire c;
assign c = a | b;
wire c = a | b;     // same thing
```

VECTORS OF WIRES

- Wire vectors:

  wire [7:0] W1;   // 8 bits, w1[7] is MSB

  – Also called “buses”
- Operations
  – Bit select: W1[3]

  – Range select: W1[3:2]

  – Concatenate:

  vec = {x, y, z};

  {carry, sum} = vec[0:1];

  – e.g., swap high and low-order bytes of 16-bit vector

  wire [15:0] w1, w2;

  assign w2 = {w1[7:0], w1[15:8]}

SIGNAL AND SIGNAL EDGE SENSITIVITY

- Signal sensitivity: evaluate block on any signal change

  always @(CLK)

- Edge sensitivity: evaluate block on particular signal change

  always @(posedge CLK)

TWO ROLES OF HDL AND RELATED TOOLS

- #1: Specifying digital logic
  – Specify the logic that appears in final design
  – Either
    • Translated automatically (called *synthesis*) or
    • Optimized manually (automatically checked for equivalence)
- #2: Simulating and testing a design
  – High-speed simulation is crucial for large designs
  – Many HDL *interpreters* optimized for speed
  – Testbench: code to test design, but not part of final design

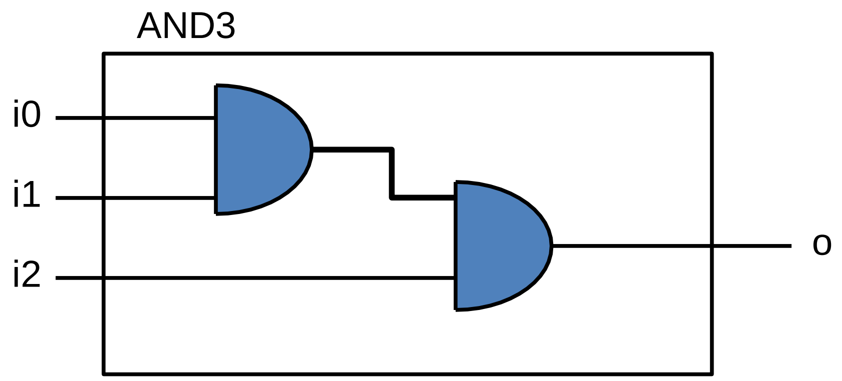
DESIGN HIERARCHIES

- Build up more complex modules using simpler modules
- Example: 4-bit wide mux from four 1-bit muxes
  - Again, just "drawing" boxes and wires

```
module mux2to1_4(
    input [3:0] A,
  input [3:0] B,
    input Sel,
    output [3:0] O );

    mux2to1 mux0 (Sel, A[0], B[0], O[0]);
    mux2to1 mux1 (Sel, A[1], B[1], O[1]);
    mux2to1 mux2 (Sel, A[2], B[2], O[2]);
    mux2to1 mux3 (Sel, A[3], B[3], O[3]);
endmodule
```

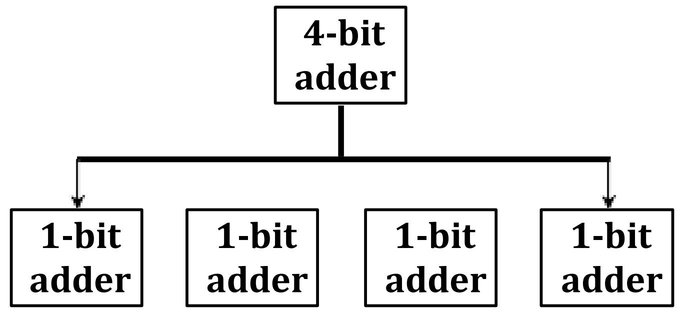
- Module instances
  - Verilog models consist of a hierarchy of module *instances*
  - In C++ speak: modules are classes and instances are objects




```
Module AND3 (i0, i1, i2, o);
   input  i0, i1, i2;
   output 0;
   wire temp;
   AND a0 (.i0(i0), .i1(i1), .o(temp));
   AND a1 (.i0(i2), .i1(temp), .o(0));
endmodule
```

DESIGN HIERARCHIES

- **Top-Down Design Methodology**




```
module CPA4b(Cout, Sum, a,b,Cin);

output        Cout;
output [3:0]  Sum;
 input [3:0]
 Input    a,b;  Cin;  c;

 wire [2:0]
 adder       fa0(c[0], Sum[0], a[0], b[0], Cin);              //by position mapping
 adder       fa1(.a(a[1]), .b(b[1]), .cin(c[0]), .carry(c[1]), .sum(Sum[1]));        //by name mapping
 adder        fa2(c[2], Sum[2], a[2], b[2], c[1]);
 adder       fa3(Cout, Sum[3], a[3], b[3], c[2]);
 endmodule
```

```
module adder (carry, sum, a, b, cin);
output carry, sum;
input a, b, cin;
assign {carry, sum} = a + b + cin;
endmodule
```

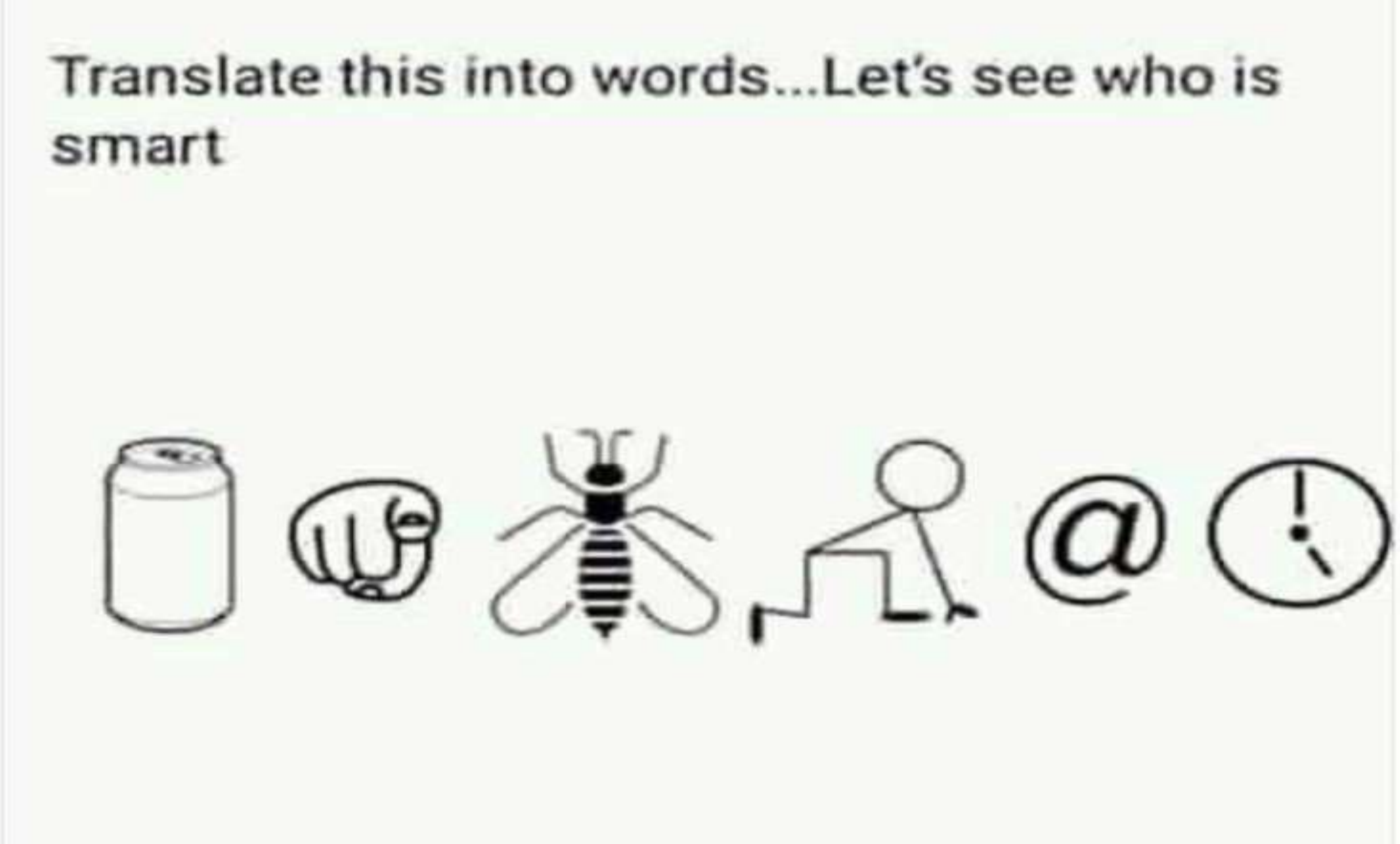
SYNTHESIS VS SIMULATION

- HDLs have features for *both* synthesis and simulation
  - E.g., simulation-only operations for error messages, reading files
  - Obviously, these can be simulated, but not synthesized into circuits
  - Also has constructs such as for-loops, while-loops, etc.
    - These are either un-synthesizable or (worse) synthesize poorly
  - **You need procedural code for testbench and *only* for testbench**

- Trends: a moving target
  - Good: better synthesis tools for higher-level constructs
  - Bad: harder than ever to know what is synthesizable or not

- Important distinction: What is a "higher-level" construct and what is "procedural code"?

ACTIVITY



Ref.: https://puzzlersworld.com

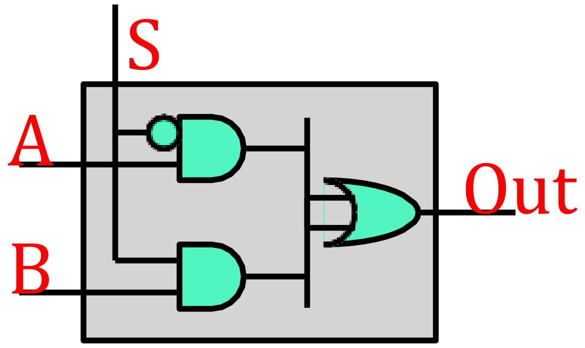
STRUCTURAL VS BEHAVIORAL HDL CONSTRUCTS

- Structural constructs specify actual hardware structures
  – Low-level, direct correspondence to hardware
    - Primitive gates (e.g., and, or, not)
    - Hierarchical structures via modules
  – Analogous to programming software in assembly
- Behavioral constructs specify an operation on bits
  – High-level, more abstract
    - Specified via equations, e.g., out = (a & b) | c
- Not all behavioral constructs are synthesizable
  – We've already talked about the pitfalls of trying to "program"
  – But even some combinational logic won't synthesize well
  – out = a % b   // modulo operation – what does this synthesize to?
  – We will not use: + - * / % > >= < <= >> <<

VERILOG STRUCTURAL VS BEHAVIORAL EXAMPLE

Structural

```
module mux2to1(
  input S, A, B,
  output Out );
  wire S_, AnS_, BnS;
  not (S_, S); Y
  and (AnS_, A, S_);
  and (BnS, B, S);
  or (Out, AnS_, BnS);
endmodule
```



Better:

```
assign Out = S? B:A;
```

Behavioral

```
module mux2to1(
  input S, A, B,
  output Out );
  assign Out = (~S & A) | (S & B);
endmodule
```

THREE MODULE COMPONENTS

- Interface specification – new style (Verilog 2001)

```
module mux2to1(
   input S, A, B,
   output O );
```

  - Can also have **inout**: bidirectional wire (we will not need or use)
- Declarations
  - Internal wires, i.e., wires that remain within this module
  - Wires also known as “nets” or “signals”

```
wire S_, AnS_, BnS;
```

- Implementation: primitive and module instantiations

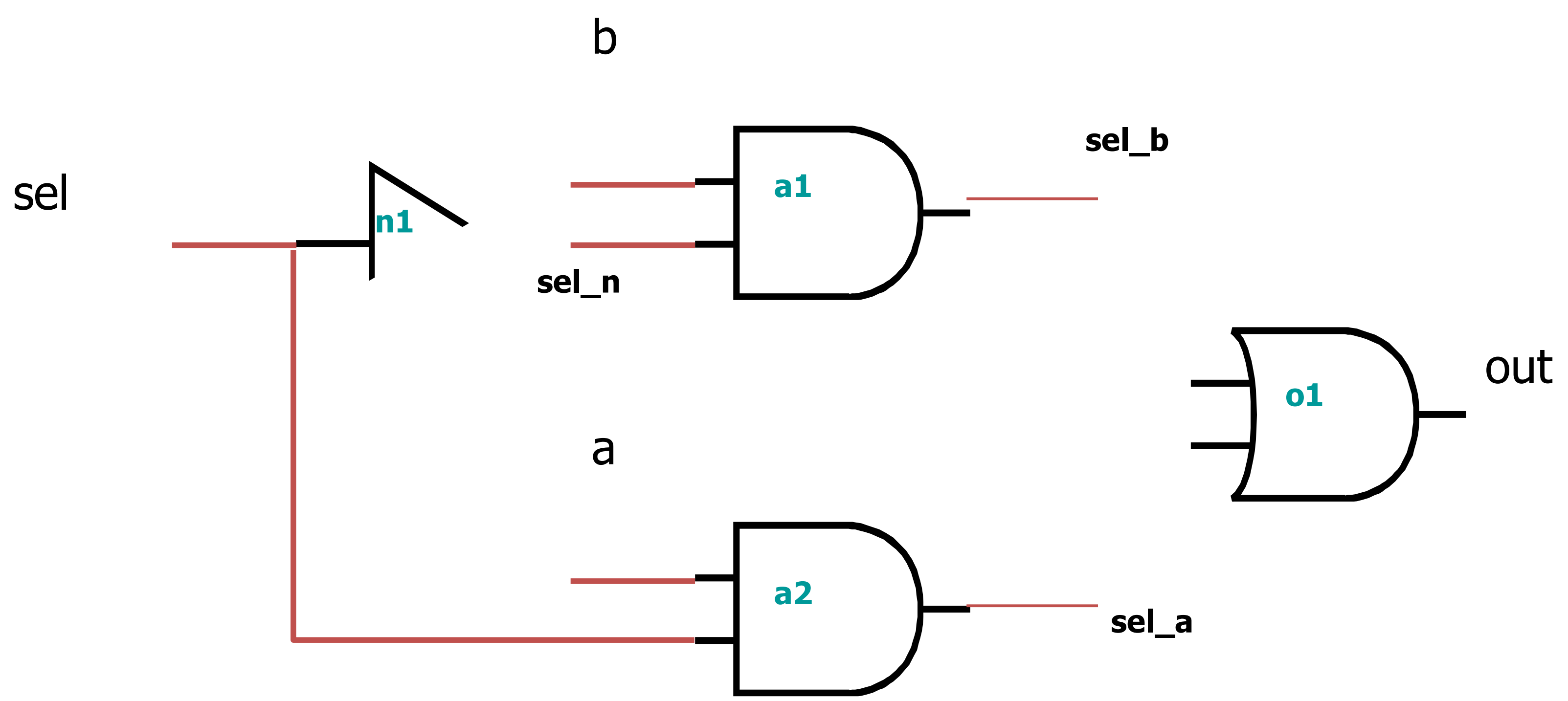
```
and (AnS_, A, S_);
```

THREE MODULE COMPONENTS-1

- **Structural**: Logic is described in terms of Verilog gate primitives

- Example:

```
not  n1(sel_n, sel);
and a1(sel_b, b, sel_b);
and a2(sel_a, a, sel);
or    o1(out, sel_b, sel_a);
```

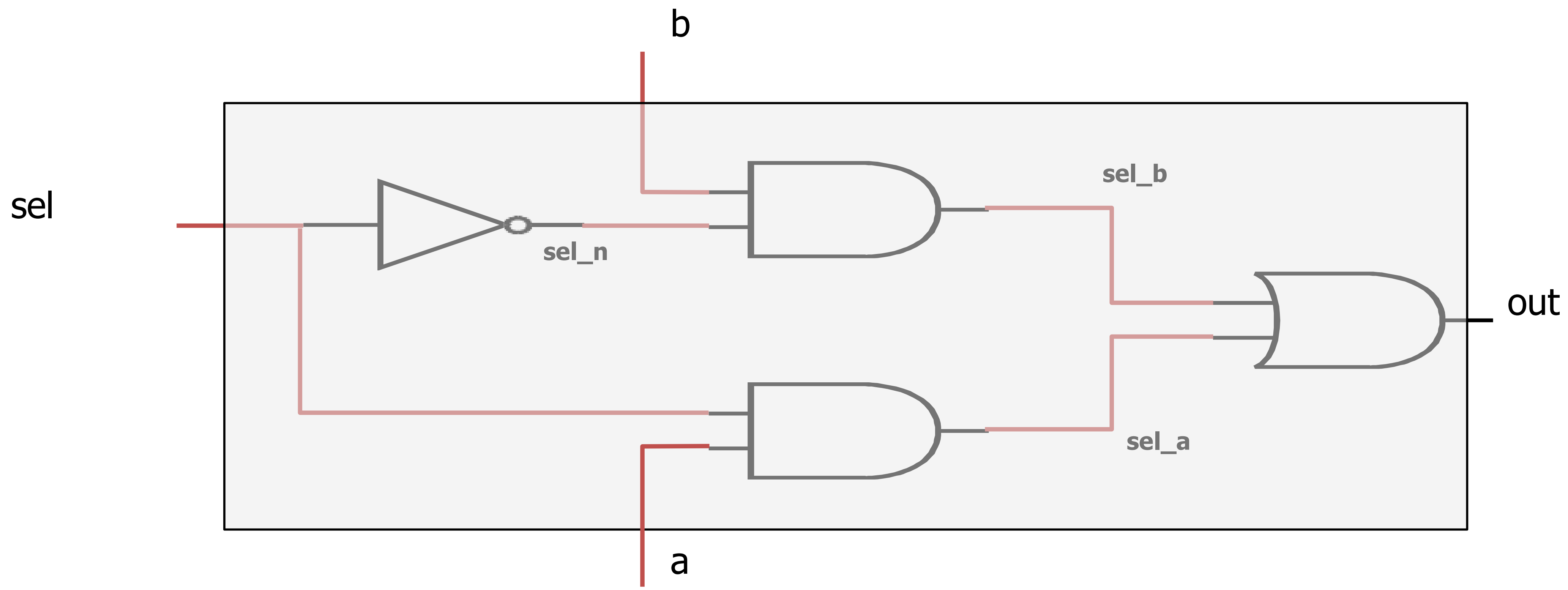


THREE MODULE COMPONENTS-2

- **Dataflow**: Specify output signals in terms of input signals

- Example:

  assign out = (sel & a) | (~sel & b);

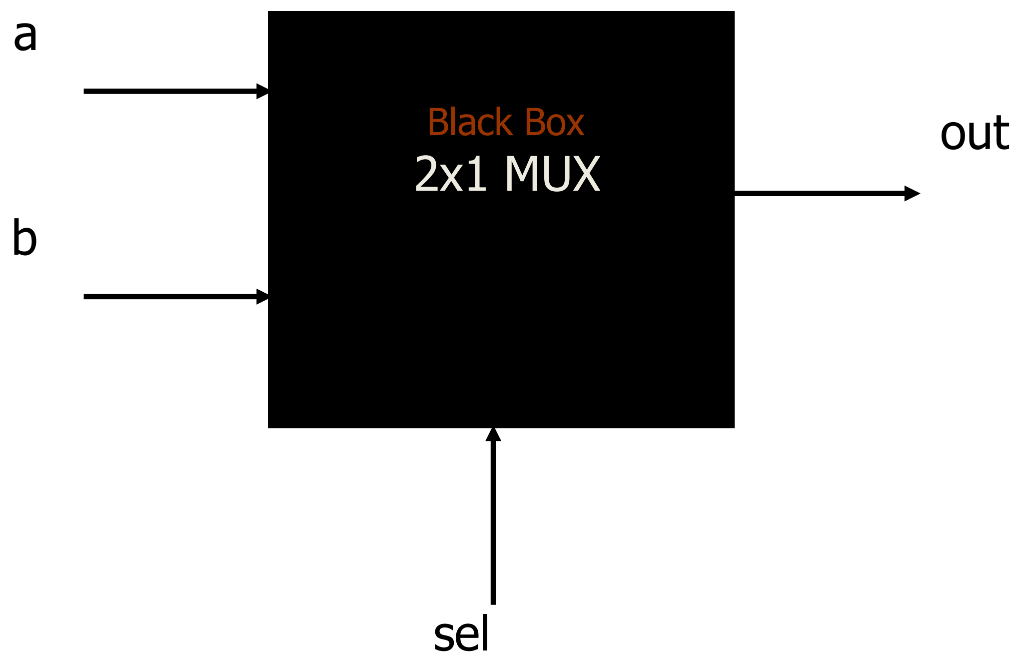


THREE MODULE COMPONENTS-3

- **Behavioral**: Algorithmically specify the behavior of the design

- Example:

```
if (select == 0) begin
      out = b;
end
else if (select == 1) begin
      out = a;
end
```



BEHAVIORAL STATEMENTS

```
module mux2to1_4(A, B, Sel, O);
  input [3:0] A;
  input [3:0] B;
  input Sel;
  output [3:0] O;

  mux2to1 mux0 (Sel, A[0], B[0], O[0]);
  mux2to1 mux1 (Sel, A[1], B[1], O[1]);
  mux2to1 mux2 (Sel, A[2], B[2], O[2]);
  mux2to1 mux3 (Sel, A[3], B[3], O[3]);
endmodule
```

Like in C, but use **begin-end** instead of **{-}** to group

if (<expr>) <stmt> else if <stmt>

for (<stmt>;<expr>;<stmt>) <stmt>

Careful: No ++ operator in Verilog

BEHAVIOR INVOCATION: ALWAYS

```
always @(<sensitivity><or sensitivity>*)
    begin
      <stmt>*
    end
```

- Defines reaction of module to changes in input
  - sensitivity list: signals or signal edges that trigger change
  - Keyword **or**: disjunction of multiple sensitivity elements
  - Multiple **always** sections are allowed
    - Careful: don't know order in which signals arrive
    - Best to use one

THREE MODULE COMPONENTS

Structural Modeling

- Execution: Concurrent
- Format (Primitive Gates):

  and G2(Carry, A, B);
- First parameter (Carry) – Output
- Other Inputs (A, B) - Inputs

Dataflow Modeling

- Uses continuous assignment statement
  - Format: assign [delay] net = expression;
  - Example: assign sum = a ^ b;
- **Delay**: Time duration between assignment from RHS to LHS
- All continuous assignment statements execute concurrently
- Order of the statement does not impact the design

THREE MODULE COMPONENTS

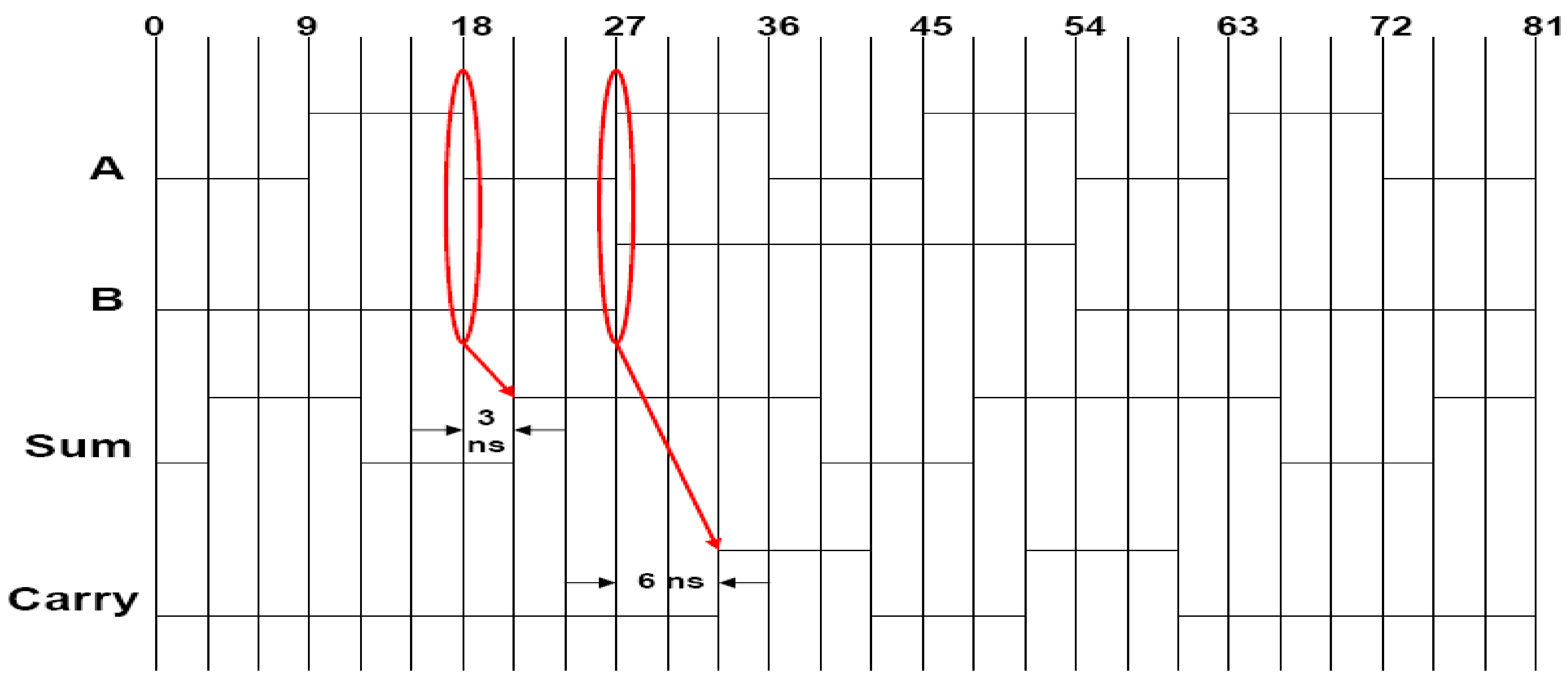
Dataflow Modeling

- Delay can be introduced
  - Example: assign #2 sum = a ^ b;
  - "#2" indicates 2 time-units
  - No delay specified : 0 (default)
- Associate time-unit with physical time
  - `timescale time-unit/time-precision
  - Example: `timescale 1ns/100 ps
- Timescale

  `timescale 1ns/100ps
  - 1 Time unit = 1 ns
  - Time precision is 100ps (0.1 ns)
  - 10.512ns is interpreted as 10.5ns

- Example:

```
`timescale 1ns/100ps
module HalfAdder (A, B, Sum, Carry);
  input A, B;
  output Sum, Carry;
  assign #3 Sum = A ^ B;
  assign #6 Carry = A & B;
endmodule
```



THREE MODULE COMPONENTS

- **Example:**

```
module mux_2x1(a, b, sel,
   out);
   input a, a, sel;
   output out;
   always @(a or b or sel)
   begin
     if (sel == 1)
        out = a;
    else out = b;
   end
endmodule
```

Sensitivity List

VERILOG MODULE EXAMPLE & RTL VS STRUCTURAL

```
module Full_Adder_Behavioral_Verilog(
 input X1, X2, Cin,
 output S, Cout
 );
  reg[1:0] temp;
 always @(*)
 begin
 temp = {1'b0,X1} + {1'b0,X2}+{1'b0,Cin};
 end
 assign S = temp[0];
 assign Cout = temp[1];
endmodule
```

STRUCTURAL MODEL EXAMPLE

```
Module Full_Adder_Structural_Verilog ( input X1, X2, Cin, output S, Cout );
   wire a1, a2, a3;
   xor u1(a1,X1,X2);
and  u2(a2,X1,X2);
and  u3(a3,a1,Cin);
or u4(Cout,a2,a3);
    xor u5(S,a1,Cin);
 endmodule
```

MIXED MODELING STYLE

```
//mixed-design full adder
module full_adder_mixed (a, b, cin, sum, cout);
//list inputs and outputs
 input a, b, cin;
output sum, cout;
 //define reg and wires
reg cout;
wire a, b, cin;
wire sum;
wire net1;
//built-in primitive
xor (net1, a, b);
//behavioral
always @ (a or b or cin)
 begin
      cout = cin & (a ^ b) | (a & b);
end
//dataflow
assign sum = net1 ^ cin;        endmodule
```

ASSESSMENT

1.Quiz: what's the difference?

```
always @(D or CLK) if (CLK) Q <= D;
always @(posedge CLK) Q <= D;
```

2.Fill up the blanks

```
module mux_2x1(a, b, sel, out);
  input a, b, sel;
  output out;
  always @(---------------------)
  begin
    if (sel == 1)
      out = a;
   else out = -----------;
  end
endmodule
```

3.List out the three modeling styles name

4.Write the Verilog HDL code for mixed modeling

SUMMARY & THANK YOU