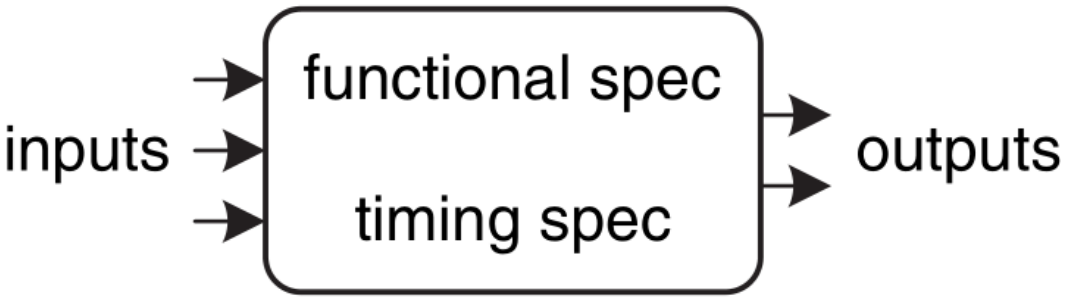
Digital Logic Design

Lecture 2

Combinational Logic Design



Any digital circuit has

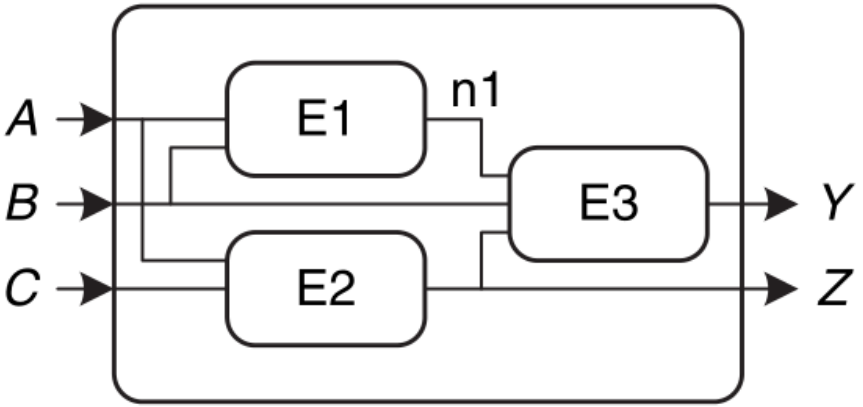
one or more discrete-valued **input terminals**

one or more discrete-valued **output terminals**

a **functional specification** describing the relationship between inputs and outputs

a **timing specification** describing the delay between inputs changing and outputs responding

Circuits are composed of nodes and elements.



An **element** is itself a circuit with inputs, outputs, and a specification.

A **node** is a wire, whose voltage conveys a discrete-valued variable.

Nodes are classified as **input**, **output**, or **internal**

- **Inputs** receive values from the external world
- **Outputs** deliver values to the external world
- Wires that are not inputs or outputs are called **internal nodes**

Verilog modules

Module - a block of hardware with inputs and outputs

```
module sillyfunction (input a, b, c,
                      output y);
    assign y = ~a & ~b & ~c |
               a & ~b & ~c |
               a & ~b & c;
endmodule
```

Two styles for describing module functionality:

Behavioral models describe what a module does

Structural models describe how a module is built from simpler pieces

Simulation and synthesis

The two major purposes of hardware description languages are logic **simulation** and **synthesis**.

During **synthesis**, the textual description of a module is transformed into logic gates.

During **simulation**, inputs are applied to a module, and the outputs are checked to verify that the module operates correctly.

Synthesizable modules and testbenches

Not all Verilog commands can be synthesized into hardware.

Example: a command to print results on the screen during simulation does not translate into hardware.

The **synthesizable modules** describe the hardware.

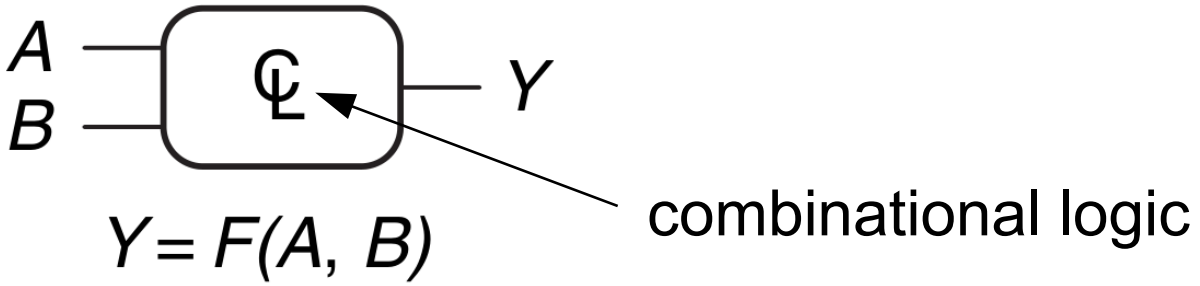
The **testbench** contains code to apply inputs to a module, check whether the output results are correct, and print discrepancies between expected and actual outputs.

Testbench code is intended only for simulation and cannot be synthesized.

Two types of digital circuits

Combinational circuits: the outputs of the circuit depend only on the current inputs.

Combinational circuits do not need to know the history of past inputs, and therefore, do not require any memory elements.



Sequential circuits: outputs depend on not only the current inputs, but also on all of the past inputs.

Sequential circuits must contain memory elements in order to remember the history of past input values.

Any elementary logic gate (BUF, NOT, AND, NAND, OR, NOR, XOR, XNOR) is a combinational circuit.

A “large” digital circuit may contain both combinational circuits and sequential circuits.

Both combinational and sequential circuits consist of the same building blocks – the AND, OR, and NOT gates, or just NAND gates.

What makes them different is in the way the gates are connected.

Logic gates in Verilog

```
module gates (input [3:0] a, b,
              output [3:0] y1, y2, y3, y4, y5);

    assign y1 = a & b; // AND
    assign y2 = a | b; // OR
    assign y3 = a ^ b; // XOR
    assign y4 = ~(a & b); // NAND
    assign y5 = ~(a | b); // NOR
endmodule
```

`[3:0]` – a 4-bit bus in the **little-endian** order

= continuous assignment, describes combinational logic

Anytime the inputs on the right side of the = in a continuous assignment statement change, the output on the left side is recomputed.

The rules of combinational composition

A circuit is combinational if it consists of interconnected circuit elements such that

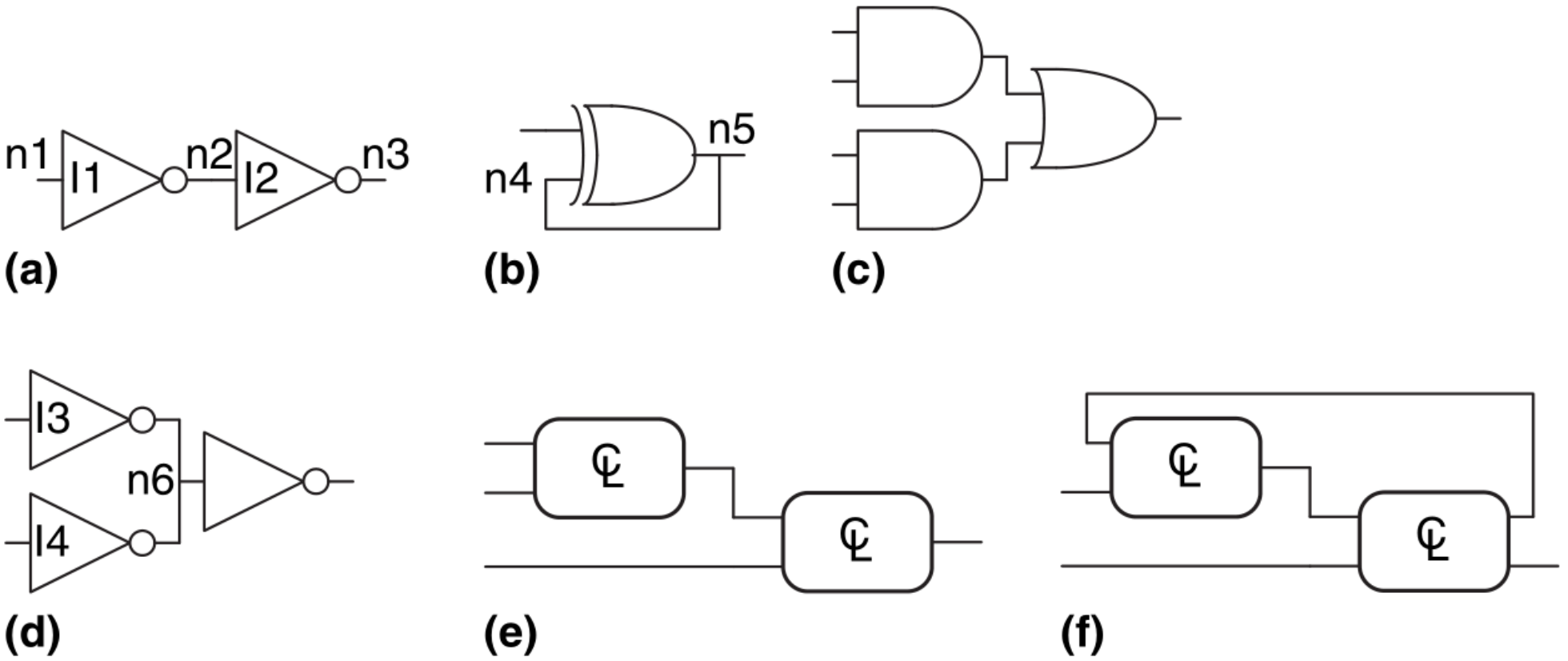
– Every circuit element is itself combinational

– Every node of the circuit is either designated as an input to the circuit or connects to exactly one output terminal of a circuit element

– The circuit contains no cyclic paths: every path through the circuit visits each circuit node at most once.

The rules of combinational composition are sufficient but not strictly necessary.

Certain circuits that disobey these rules are still combinational, so long as the outputs depend only on the current values of the inputs.

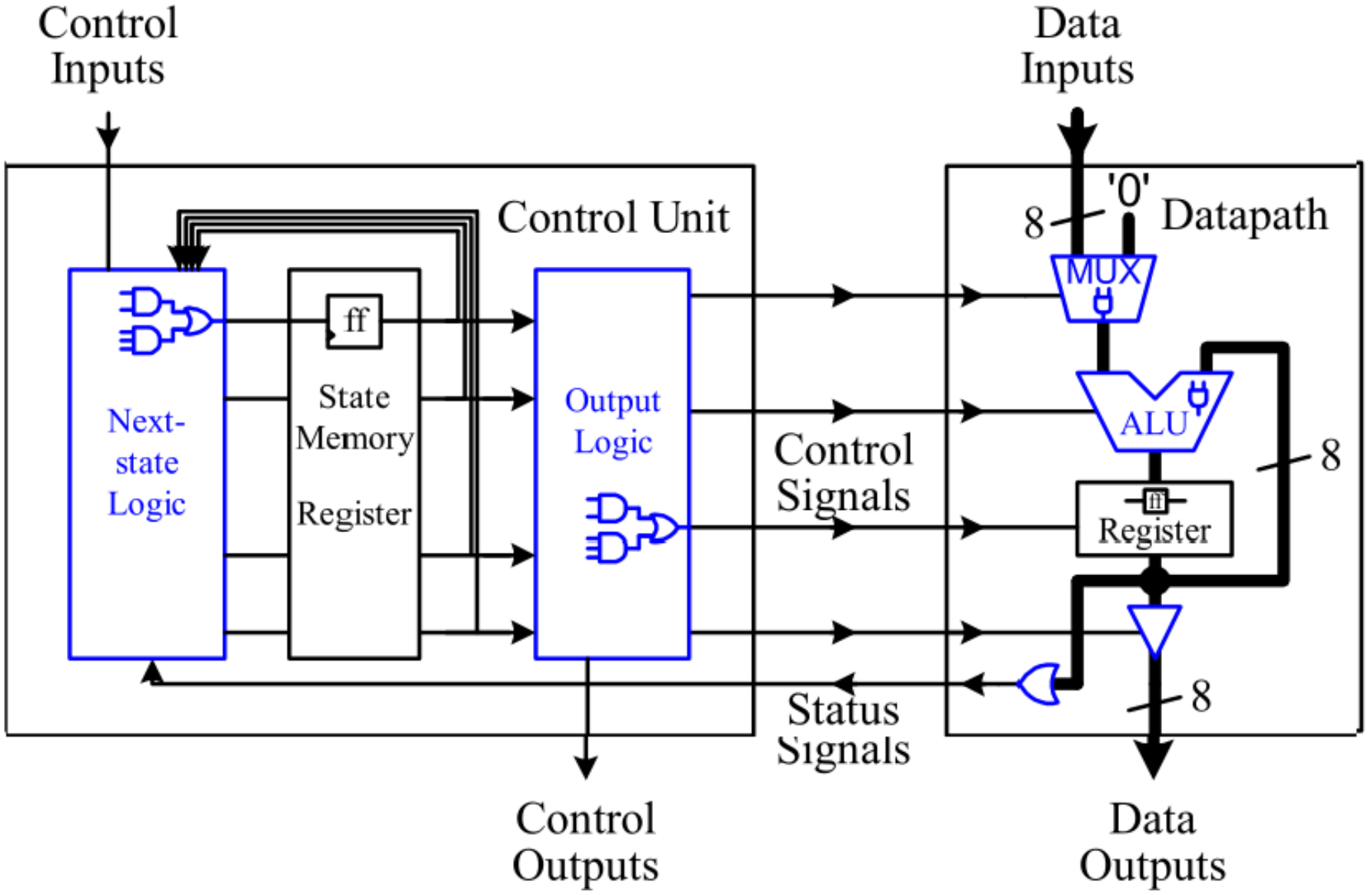


a, c, e are combinational

b, d are not combinational

f may or may not be combinational

Combinational circuits in the microprocessor



Analysis of combinational circuits

The **analysis** of combinational circuits:

Given: a combinational circuit

Task: to derive a precise description of the operation of the circuit

The functional specification of a combinational circuit is usually expressed as a truth table or a Boolean equation.

Synthesis of combinational circuits

It is the reverse procedure of the analysis of combinational circuits.

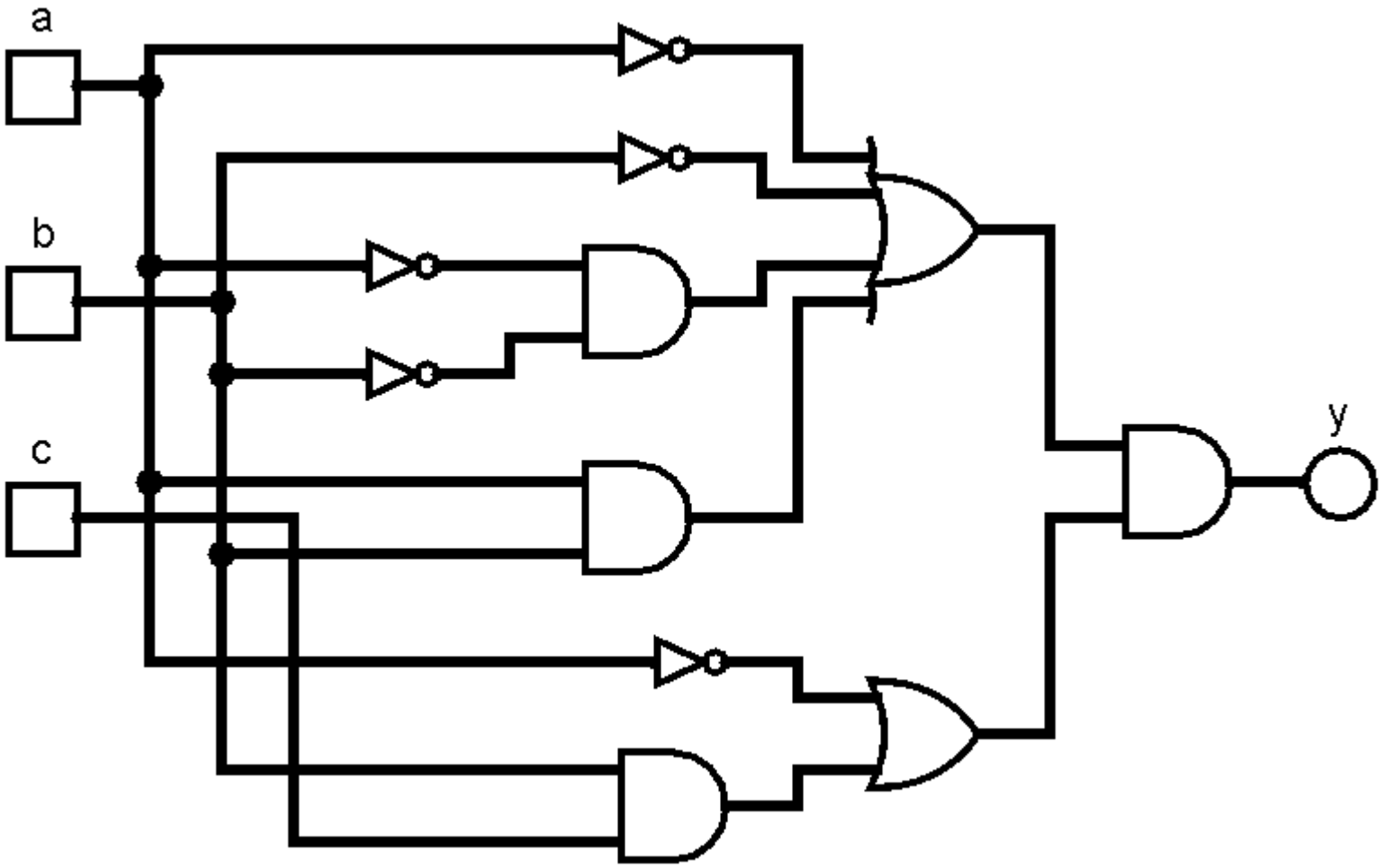
In **synthesis**, we start with a description of the operation of the circuit.

From this description, we derive either the truth table or the Boolean logical function that describes the operation of the circuit.

Once we have either the truth table or the logical function, we can translate that into a circuit diagram.

Boolean algebra is used to simplify Boolean equations, and therefore to minimise the circuit.

Example: $y=(\bar{a}+\bar{b}+\bar{a}\,\bar{b}+ab)(\bar{a}+bc)$



$$y=(\bar{a}+\bar{b}+\bar{a}\,\bar{b}+ab)(\bar{a}+bc)=$$

$$(\bar{a}\cdot 1+\bar{b}\cdot 1+\bar{a}\,\bar{b}+ab)(\bar{a}+bc)=$$

$$(\bar{a}(b+\bar{b})+\bar{b}(a+\bar{a})+\bar{a}\,\bar{b}+ab)(\bar{a}+bc)=$$

$$(\bar{a}\,b+\bar{a}\,\bar{b}+\bar{b}\,a+\bar{b}\,\bar{a}+\bar{a}\,\bar{b}+ab)(\bar{a}+bc)=$$

$$(\bar{a}\,b+\bar{a}\,\bar{b}+\bar{b}\,a+ab)(\bar{a}+bc)=$$

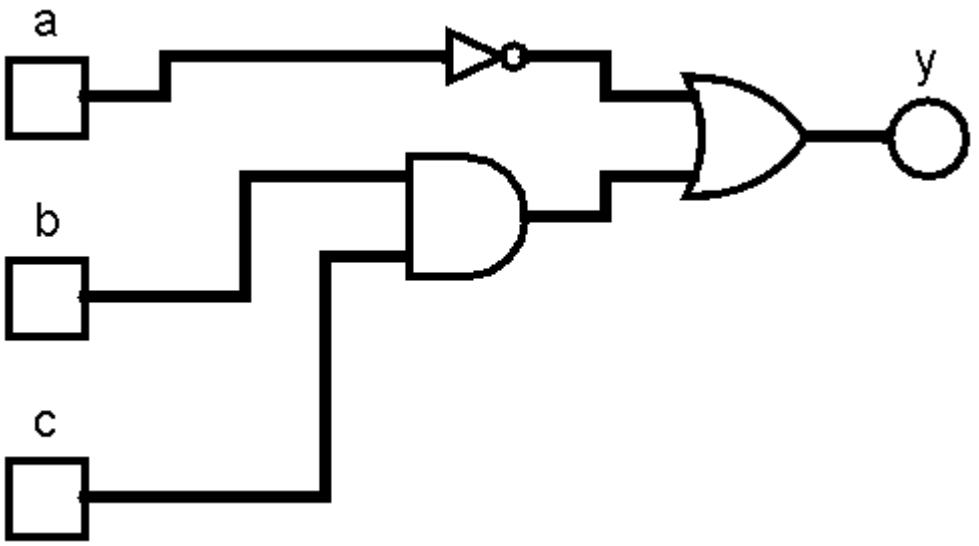
$$(\bar{a}(b+\bar{b})+a(\bar{b}+b))(\bar{a}+bc)=$$

$$(\bar{a}\cdot 1+a\cdot 1)(\bar{a}+bc)=$$

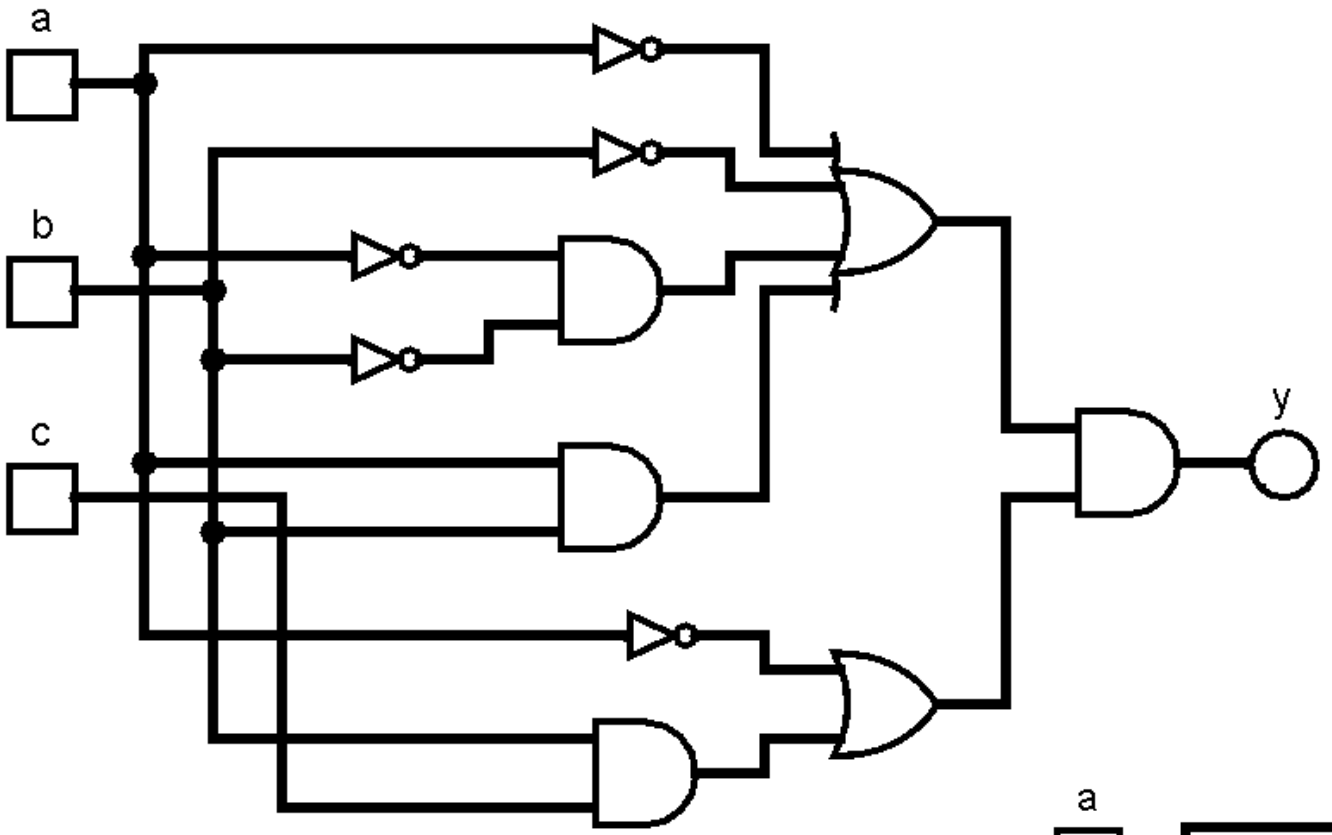
$$(\bar{a}+a)(\bar{a}+bc)=$$

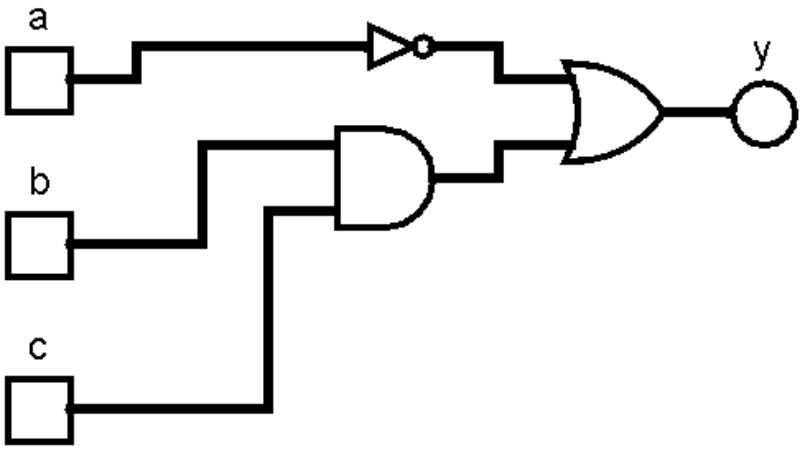
$$1\cdot(\bar{a}+bc)=$$

$$\bar{a}+bc$$



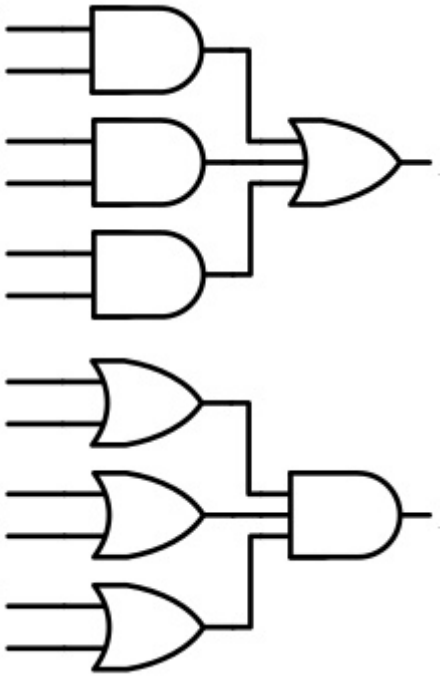
Different timing





Multilevel combinational logic

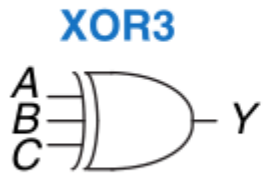
Logic in sum-of-products (or product-of-sums) form is called **two-level logic** because it consists of literals connected to a level of AND gates connected to a level of OR gates (or to a level of OR gates connected to a level of AND gates)



Designers often build circuits with more than two levels of logic gates.

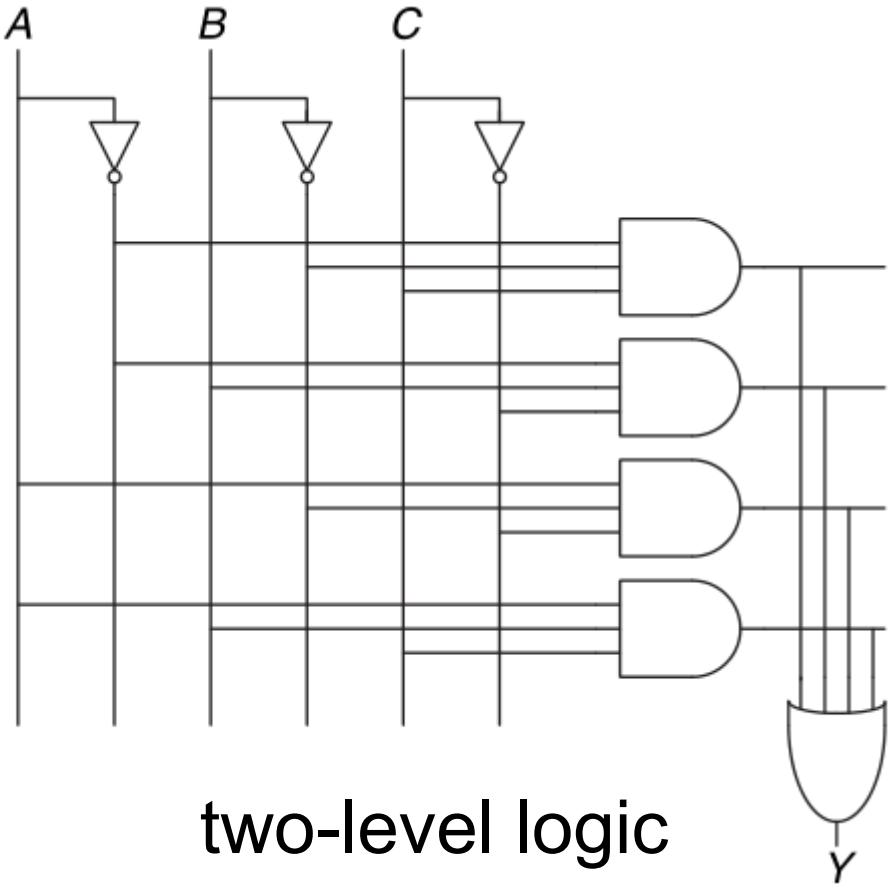
These multilevel combinational circuits may use less hardware than their two-level counterparts.

Example: three-input XOR



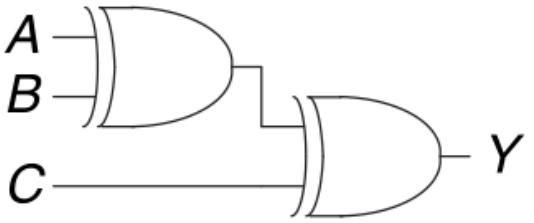
$Y = A \oplus B \oplus C$

A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



two-level logic

using two-input XORs



Ambiguity in the definition of the XOR function of multiple variables

There are two possible interpretations as to what a multi-input XOR should do.

One interpretation is that the output is high if ONLY 1 input is high (an exclusive or).

Another interpretation of XOR is that it is effectively a series of XOR gates chained together.

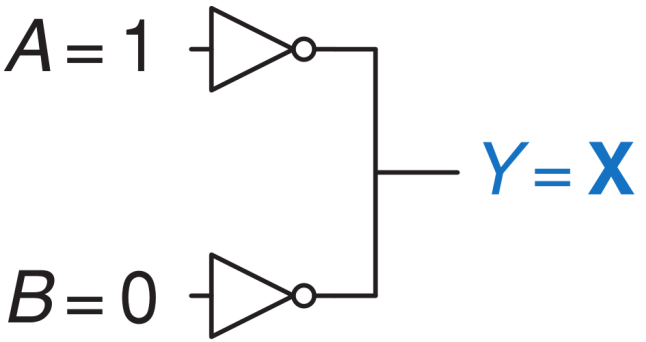
In this case, if the number of high inputs is even (including 0), the output is 0. If the number of high inputs is odd, the output is 1.

Logisim uses the first interpretation.

Illegal Value: X

The symbol X indicates that the circuit node has an **unknown** or **illegal** value.

This happens if it is being driven to both 0 and 1 at the same time.



This is called **contention** which is an error and must be avoided.

Contention also can cause **large amounts of power** to flow between the fighting gates, resulting in the circuit getting **hot** and possibly **damaged**.

X values are also sometimes used by circuit simulators to indicate an uninitialized value.

A "don’t care" value

Digital designers use the symbol X to indicate **don’t care values** in truth tables.

This means that the value of the variable in the truth table is unimportant (can be either 0 or 1).

Floating Value: Z

The symbol Z indicates that a node is being driven neither HIGH nor LOW.

The node is said to be **floating**, high impedance, or high Z

A floating node might be 0, might be 1, or might be at some voltage in between, depending on the history of the system.

Z may be produced if you forget to connect a voltage to a circuit input, or assume that an unconnected input is the same as an input with the value of 0.

This mistake may cause the circuit to behave erratically as the floating input randomly changes from 0 to 1.

Z's and X's in Verilog

Verilog signal values are 0, 1, z, and x

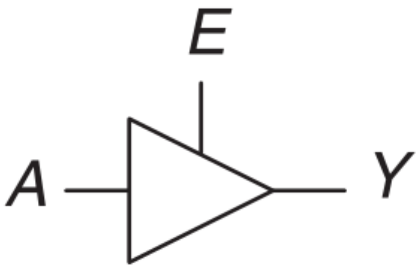
Verilog constants starting with z or x are padded with leading z's or x's (instead of 0's) to reach their full length when necessary.

Verilog AND gate:

&		A			
		0	1	z	x
B	0	0	0	0	0
	1	0	1	x	x
	z	0	x	x	x
	x	0	x	x	x

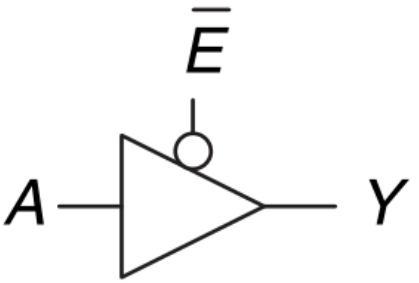
Tristate buffer

active high enable



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

active low enable



$\overline{E}$	A	Y
0	0	0
0	1	1
1	0	Z
1	1	Z

3 possible output states: 0, 1, Z

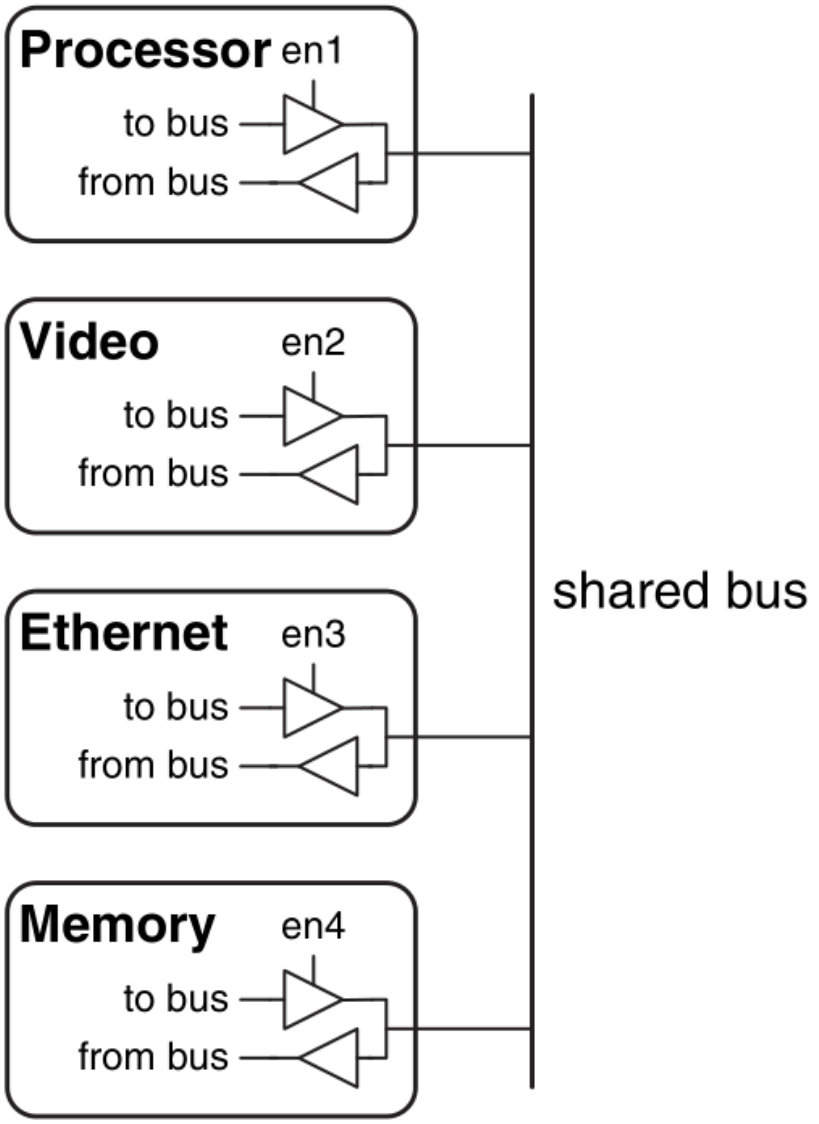
Either buffer or Z depending on E

Tristate bus connecting multiple chips

Only one chip at a time is allowed to assert its enable signal to drive a value onto the bus.

The other chips must produce Z so that they do not cause contention with the chip talking to the memory.

Any chip can read the information from the shared bus at any time.

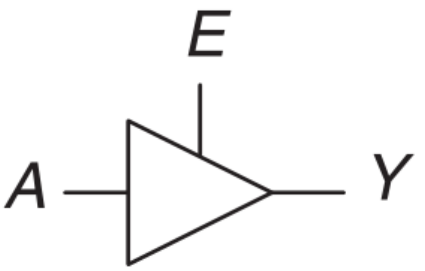


If a bus is simultaneously driven to 0 and 1 by two enabled tristate buffers (or other gates), the result is x (contention).

If all the tristate buffers driving a bus are simultaneously OFF, the bus will float, indicated by z

Tristate buffer in Verilog

```
module tristate (input [3:0] a,
                 input en,
                 output [3:0] y);
    assign y = en ? a : 4'bz;
endmodule
```

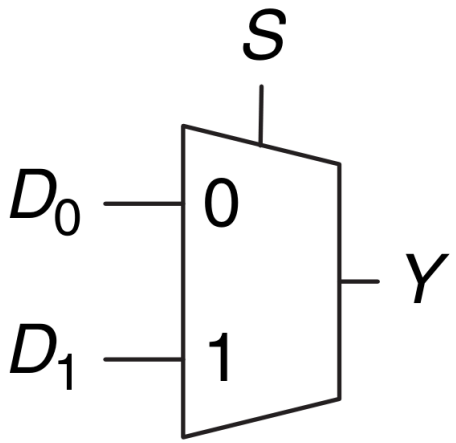


E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Multiplexers

Multiplexers choose an output from among several possible inputs based on the value of a **select** signal.

2:1 multiplexer



S is also called a **control signal**

S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Behavioral model in Verilog:

```
module mux2 (input [3:0] d0, d1,
             input s,
             output [3:0] y);
    assign y = s ? d1 : d0;
endmodule
```

If s is 1, then y = d1 . If s is 0, then y = d0

```
module mux2_b (input d0, d1, s,
               output reg y);
    always @ (*)
     case (s)
        1'b0: y = d0;
        1'b1: y = d1;
      endcase
endmodule
```

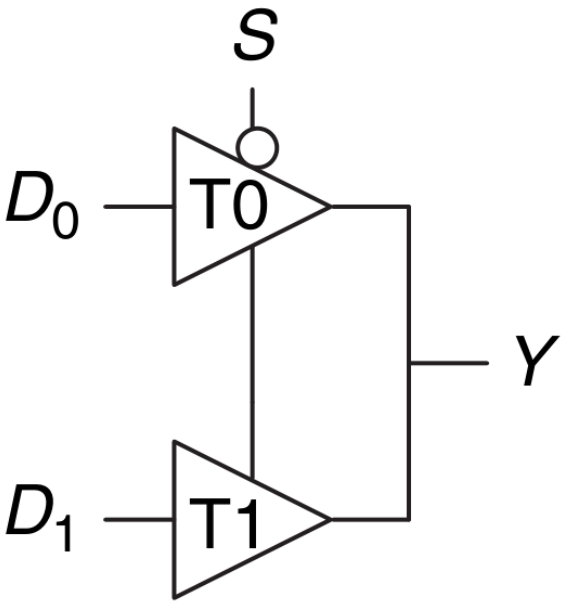
```
module mux2_b_tb ();
  reg d0, d1, s;
  wire y;

  mux2_b dut (d0, d1, s, y);

  initial begin
   $monitor(
   "s=%b d0=%b d1=%b y=%b",
   s, d0, d1, y);
   s = 0; d0 = 0; d1 = 0; #1;
   s = 0; d0 = 0; d1 = 1; #1;
   s = 0; d0 = 1; d1 = 0; #1;
   s = 0; d0 = 1; d1 = 1; #1;
   s = 1; d0 = 0; d1 = 0; #1;
   s = 1; d0 = 0; d1 = 1; #1;
   s = 1; d0 = 1; d1 = 0; #1;
   s = 1; d0 = 1; d1 = 1; #1;
   $finish;
  end
endmodule
```

```
C:\iverilog\samples>vvp mux2_b
s=0 d0=0 d1=0 y=0
s=0 d0=0 d1=1 y=0
s=0 d0=1 d1=0 y=1
s=0 d0=1 d1=1 y=1
s=1 d0=0 d1=0 y=0
s=1 d0=0 d1=1 y=1
s=1 d0=1 d1=0 y=0
s=1 d0=1 d1=1 y=1
```

Multiplexers can be built from tristate buffers



Structural model in Verilog

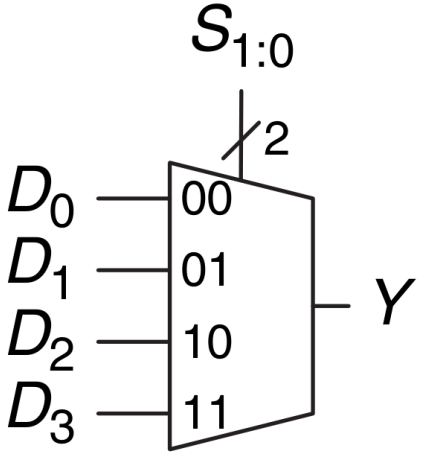
```
module mux2 (input [3:0] d0, d1,
             input s,
             output [3:0] y);
    tristate t0 (d0, ~s, y);
    tristate t1 (d1, s, y);
endmodule
```

Shorting together the outputs of multiple gates violates the rules for combinational composition.

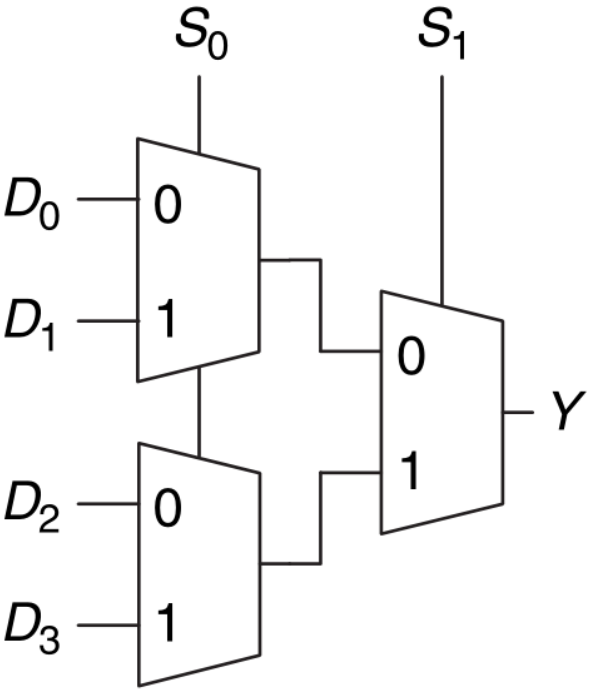
But because exactly one of the outputs is driven at any time, the resulting circuit is still combinational.

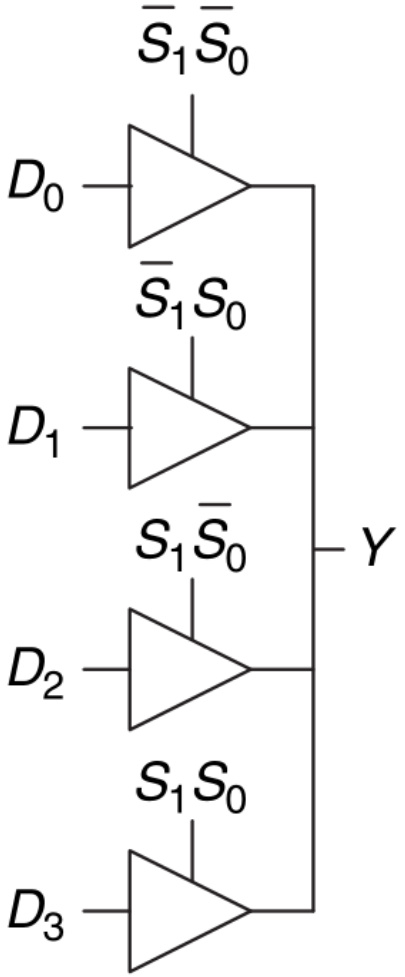
4:1 multiplexer

two select signals



The 4:1 multiplexer can be built using tristates, or multiple 2:1 multiplexers.



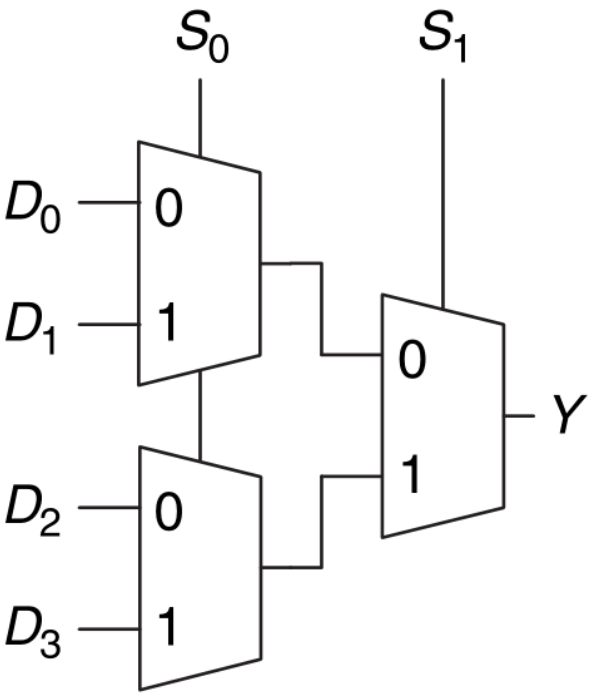


N:1 multiplexer needs $\log_2 N$ select lines

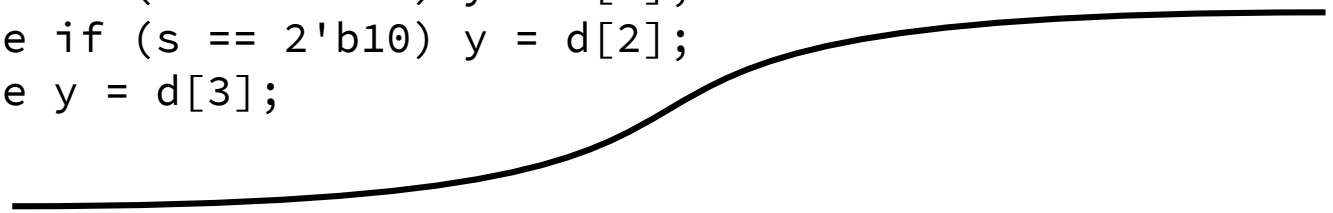
4:1 multiplexer

Behavioral model:

```
module mux4 (input [3:0] d0, d1, d2, d3,
             input [1:0] s,
             output [3:0] y);
    assign y = s[1] ? (s[0] ? d3 : d2)
                    : (s[0] ? d1 : d0);
endmodule
```



```
module mux4 (input [3:0] d,
             input [1:0] s,
             output reg y);
    always @ (*)
      if (s == 2'b00) y = d[0];
      else if (s == 2'b01) y = d[1];
      else if (s == 2'b10) y = d[2];
      else y = d[3];
endmodule
```



```
module mux4_tb ();
  reg [3:0] d;
  reg [1:0] s;
  wire y;
  integer i,j;
  mux4 dut(d, s, y);
  initial begin
    for (i = 0; i < 4; i = i+1) begin
      s = i;
      for (j = 0; j < 16; j = j+1) begin
        d = j;
        #1; $display("s=%b d=%b y=%b", s, d, y);
      end
    end
    $finish;
  end
endmodule
```

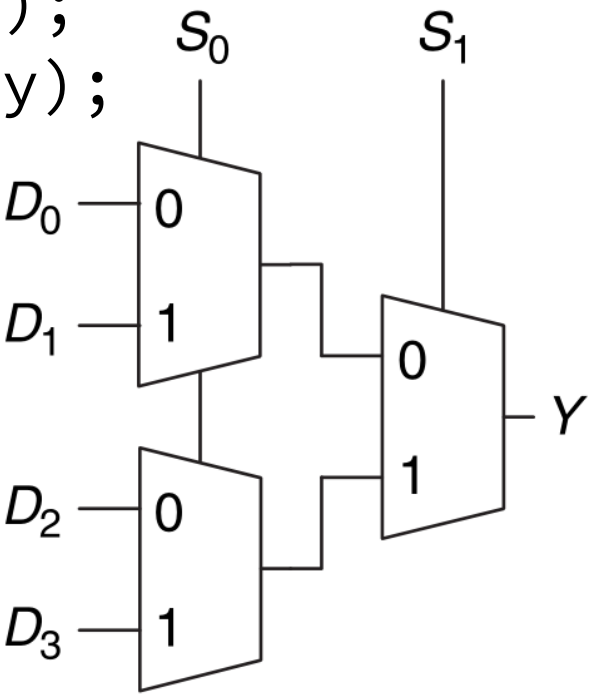
4:1 multiplexer

Structural model:

```
module mux4 (input [3:0] d0, d1, d2, d3,
             input [1:0] s,
             output [3:0] y);

    wire [3:0] low, high;

    mux2 lowmux (d0, d1, s[0], low);
    mux2 highmux (d2, d3, s[0], high);
    mux2 finalmux (low, high, s[1], y);
endmodule
```



`wire` – internal variables that are neither inputs nor outputs but only internal to the module

Complex systems are designed hierarchically.

The overall system is described structurally by instantiating its major components.

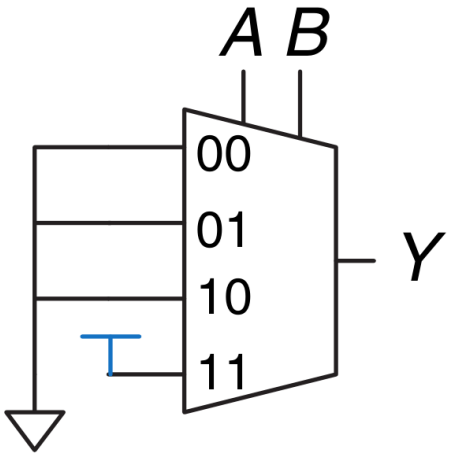
Each of these components is described structurally from its building blocks, and so forth recursively until the pieces are simple enough to describe behaviorally.

It is good style to avoid (or at least to minimize) mixing structural and behavioral descriptions within a single module.

Multiplexer logic

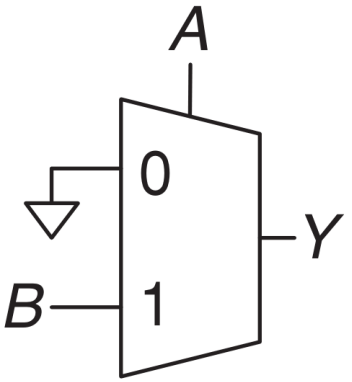
A 2^N-input multiplexer can be programmed to perform any *N*-input logic function.

Example: two-input AND



One can cut the multiplexer size in half, using only a 2^{N-1}-input multiplexer to perform any *N*-input logic function

Example: two-input AND



An example of multiplexer logic

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

$$Y = A\overline{B} + \overline{B}\overline{C} + \overline{A}BC$$