HW2: Adder

- 1101 + 0110 = ?

```
    1 1 0 1
+   0 1 1 0
-----------
```

Binary Adder

- 1101 + 0110 = ?
- Calculation by hand…

```
    1 1 0 1
+   0 1 1 0
-----------
  1 0 0 1 1
```

Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = ?
- C= ?
- Please write down your answer

A	B
0	0
0	1
1	0
1	1

Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = ?
- C= ?
- Please write down your answer

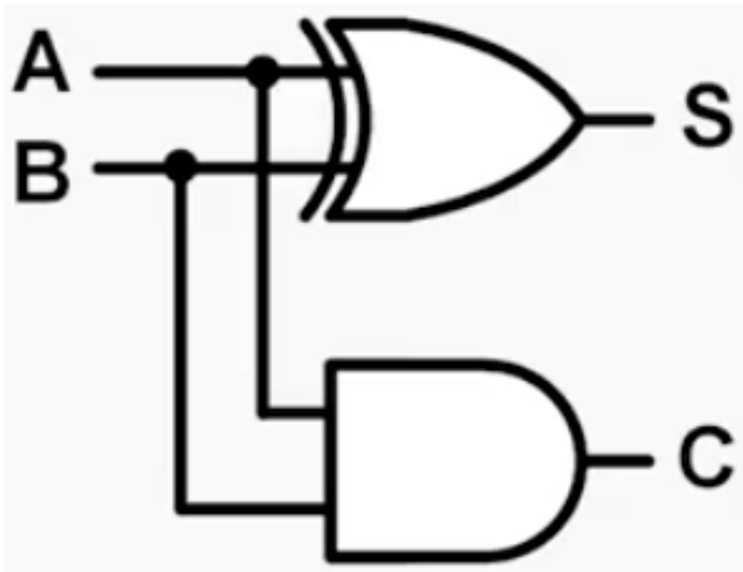
A	B
0	0
0	1
1	0
1	1

C	S
0	0
0	1
0	1
1	0

Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = ?
- C= ?
- Please write down your answer

A	B
0	0
0	1
1	0
1	1



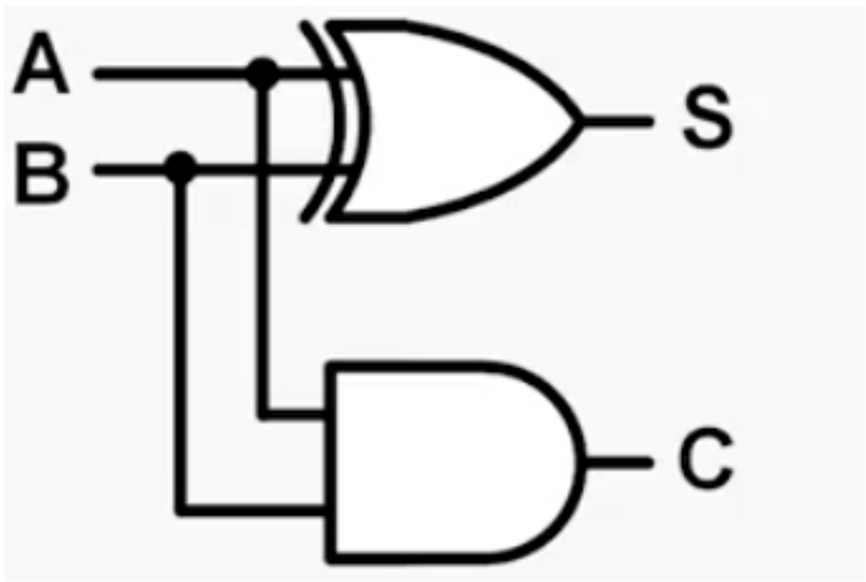
C	S
0	0
0	1
0	1
1	0

Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = A XOR B
- C=A AND B

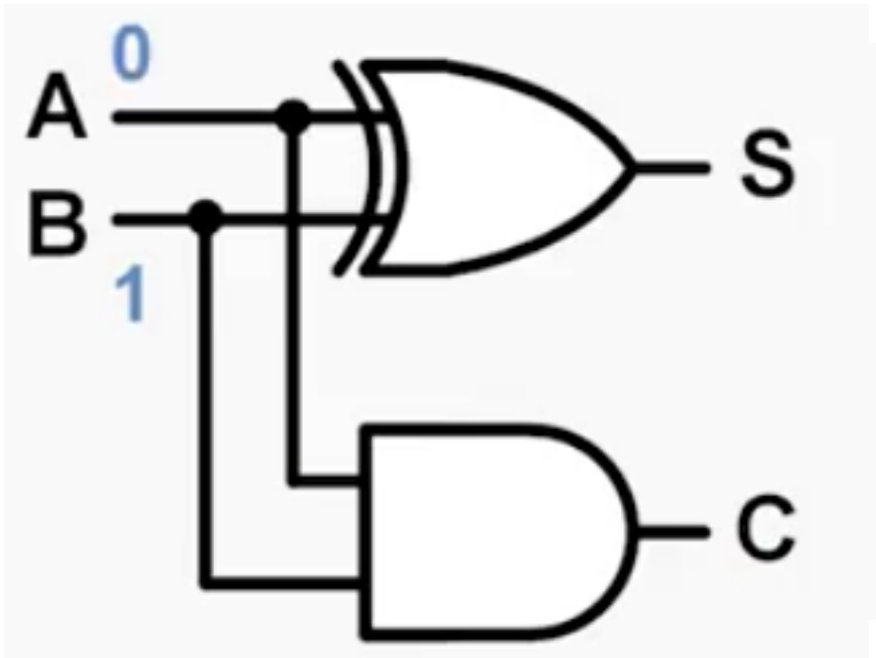
Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = A XOR B
- C=A AND B



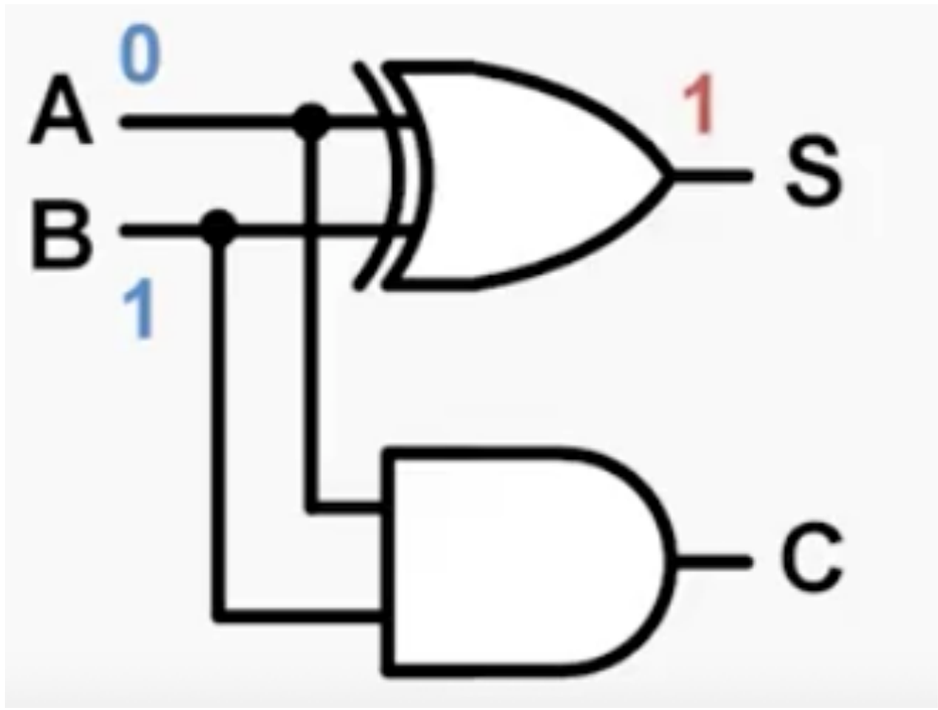
Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = A XOR B
- C=A AND B



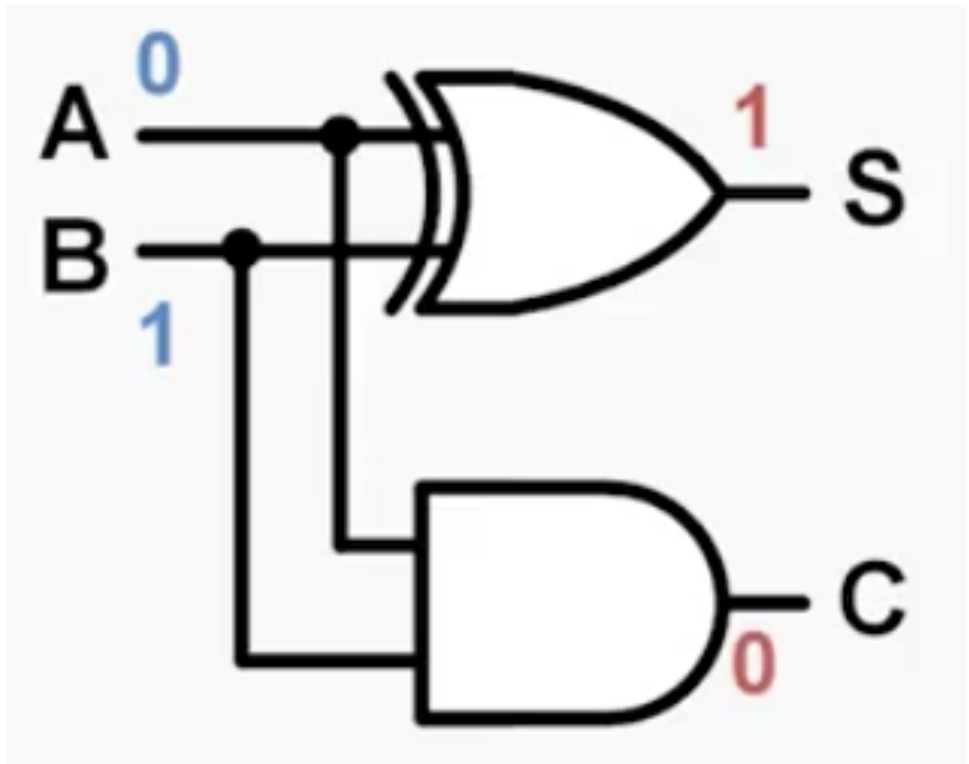
Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = A XOR B
- C=A AND B



Half Adder

- What is a half adder?
- Two 1-bit numbers are being added
  - Two inputs A and B
  - Two outputs S(um) and C(arry out)
- No carry in
- S = A XOR B
- C=A AND B



Full Adder

- What is a full adder?

Full Adder

- What is a full adder?
  - Has 3-bit input

Full Adder

- What is a full adder?
  - Has 3-bit input
  - Has a Cin (carry in) bit

Full Adder

- What is a full adder?
  - Has 3-bit input
  - Has a Cin (carry in) bit
  - Has two outputs: S(um) and Cout

Full Adder

- What is a full adder?
  - Has 3-bit input
  - Has a Cin (carry in) bit
  - Has two outputs: S(um) and Cout

Full Adder

- What is a full adder?
  - Has 3-bit input
  - Has a Cin (carry in) bit
  - Has two outputs: S(um) and Cout

A	B	C_{in}
0	0	0
0	1	0
1	0	0
1	1	0
0	0	1
0	1	1
1	0	1
1	1	1

Full Adder

- What is a full adder?
  - Has 3-bit input
  - Has a Cin (carry in) bit
  - Has two outputs: S(um) and Cout

A	B	C_{in}
0	0	0
0	1	0
1	0	0
1	1	0
0	0	1
0	1	1
1	0	1
1	1	1

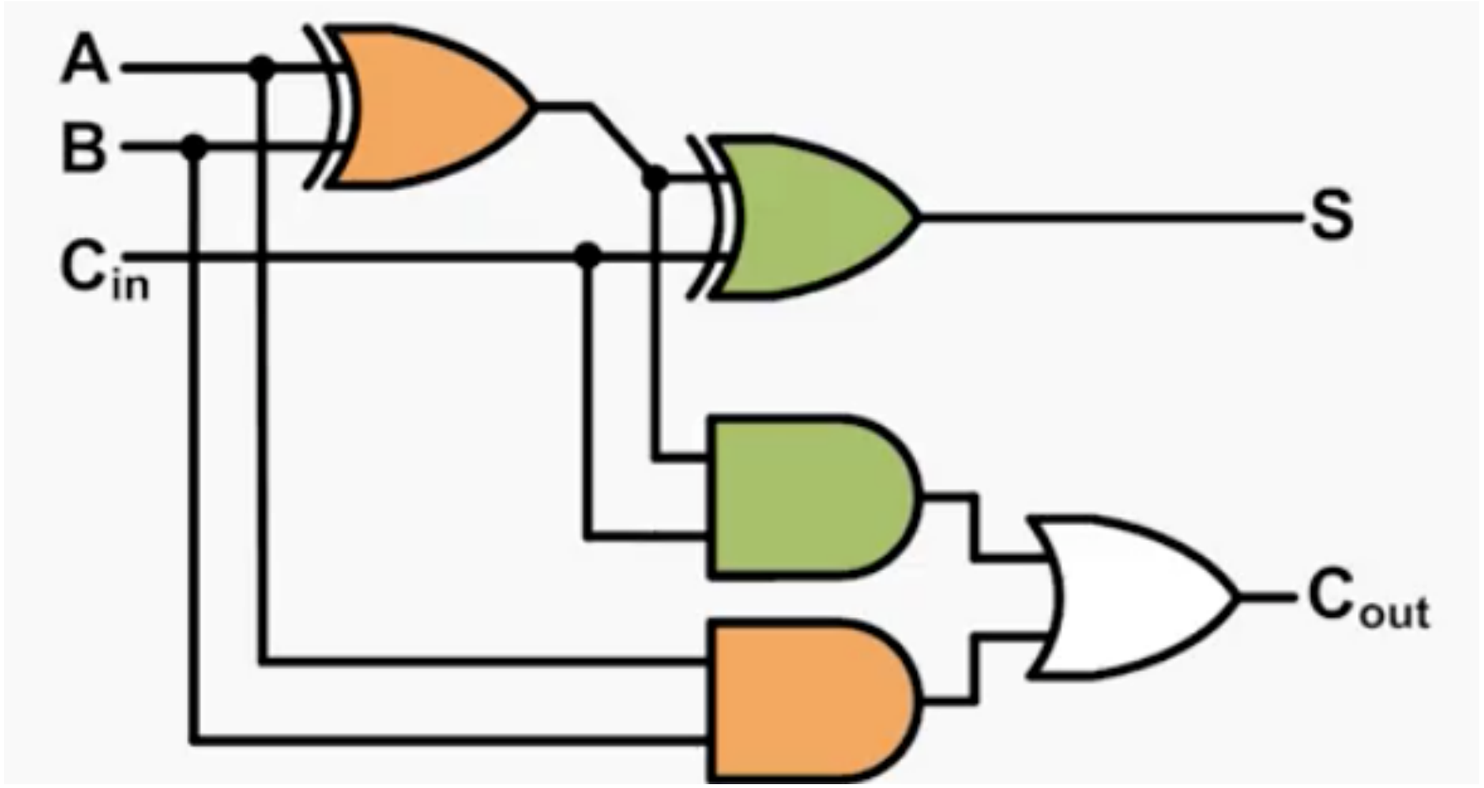
- Write down the truth table for Cout and S

Full Adder

- What is a full adder?
  - Has 3-bit input
  - Has a Cin (carry in) bit
  - Has two outputs: S(um) and Cout
- Write down the truth table for (

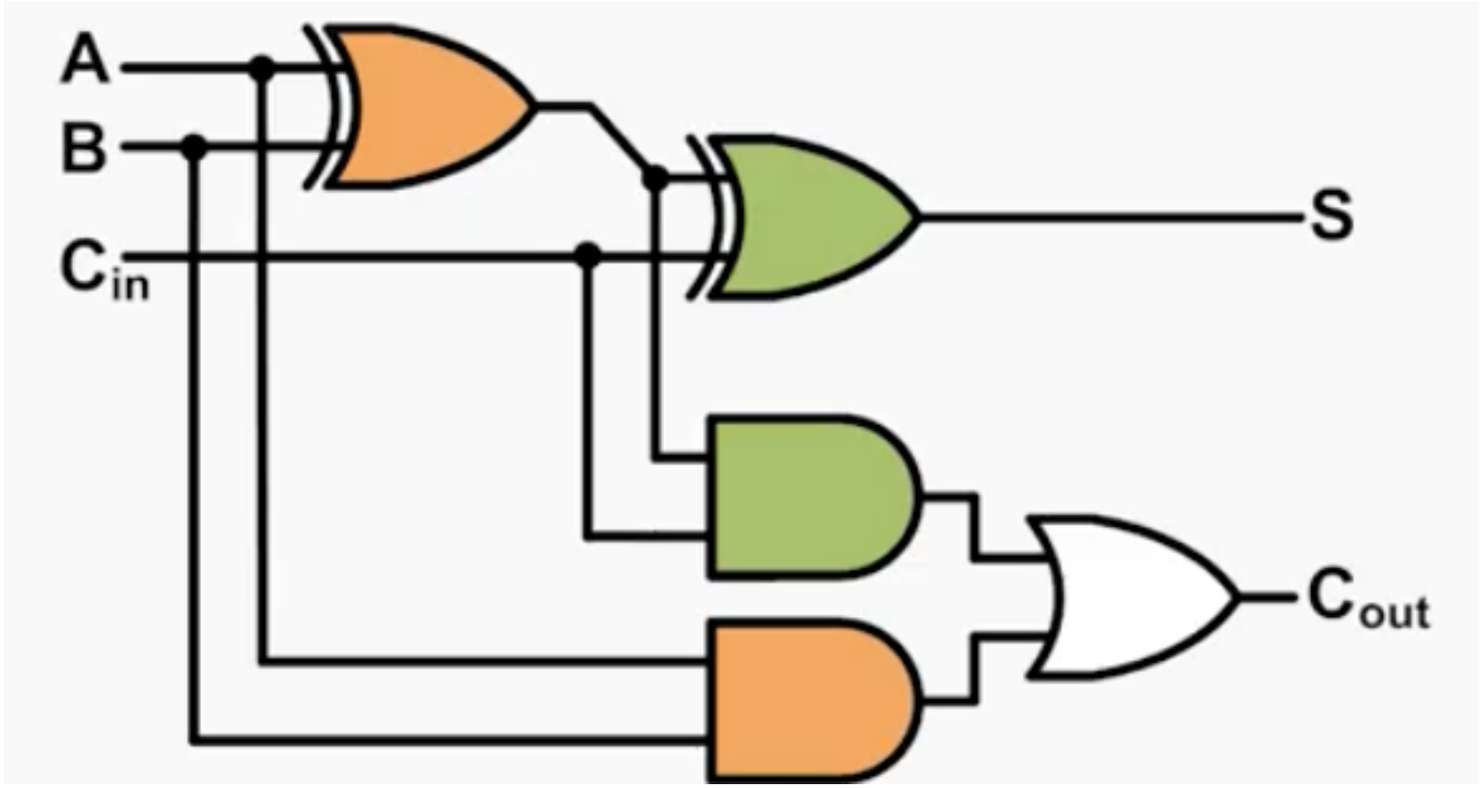
A	B	C_{in}	C_{out}	S
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

Full Adder



Full Adder

- Composition
  - Two half-adders: colored differently!



4-bit Full Adder

- A=1101
- B=0110

4-bit Full Adder

- A=1101
- B=0110

```
  1 1 0 1
+ 0 1 1 0
---------
1 0 0 1 1
```

4-bit Full Adder

- A=1101
- B=0110

```
  1 1 0 1
+ 0 1 1 0
---------
1 0 0 1 1
```

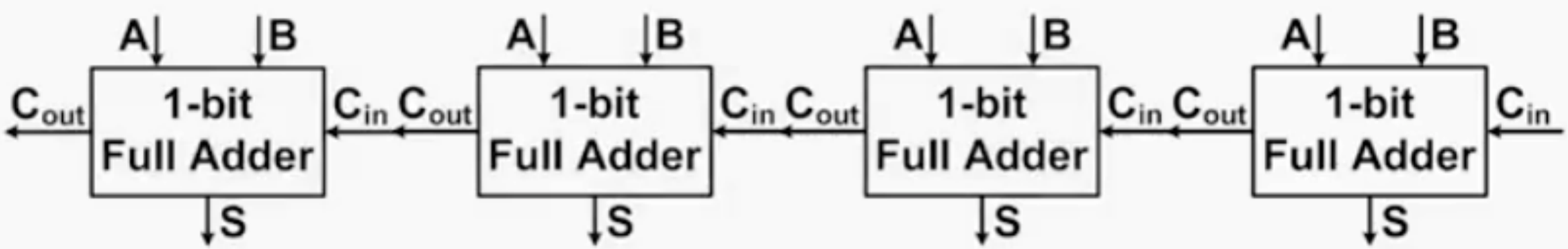
- How to build a 4-bit full adder?

4-bit Full Adder

- A=1101
- B=0110

```
  1 1 0 1
+ 0 1 1 0
---------
1 0 0 1 1
```

- How to build a 4-bit full adder?



EI

4-bit Full Adder

- How it works?

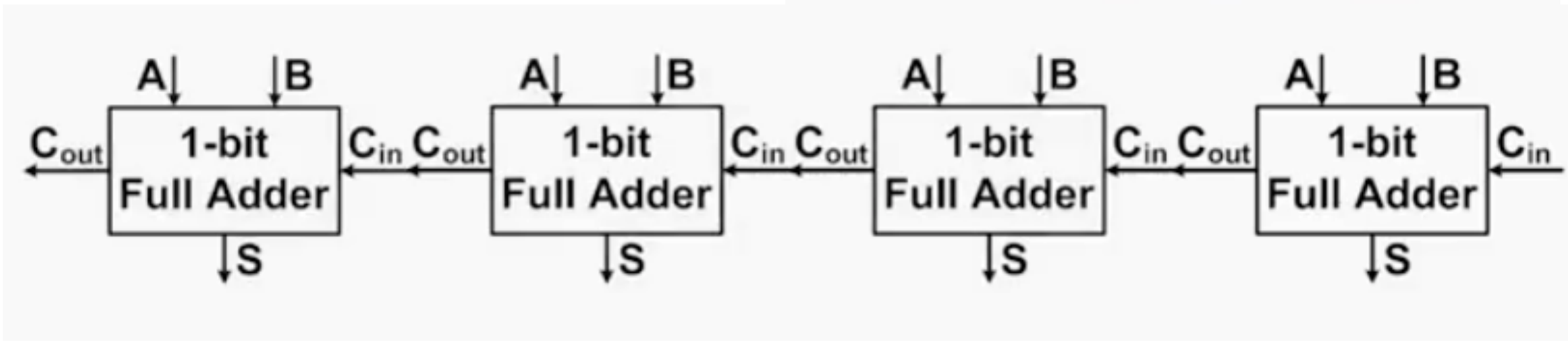
4-bit Full Adder

- How it works?

4-bit Full Adder

- How it works?

```
  1 1 0 1
+ 0 1 1 0
---------
1 0 0 1 1
```

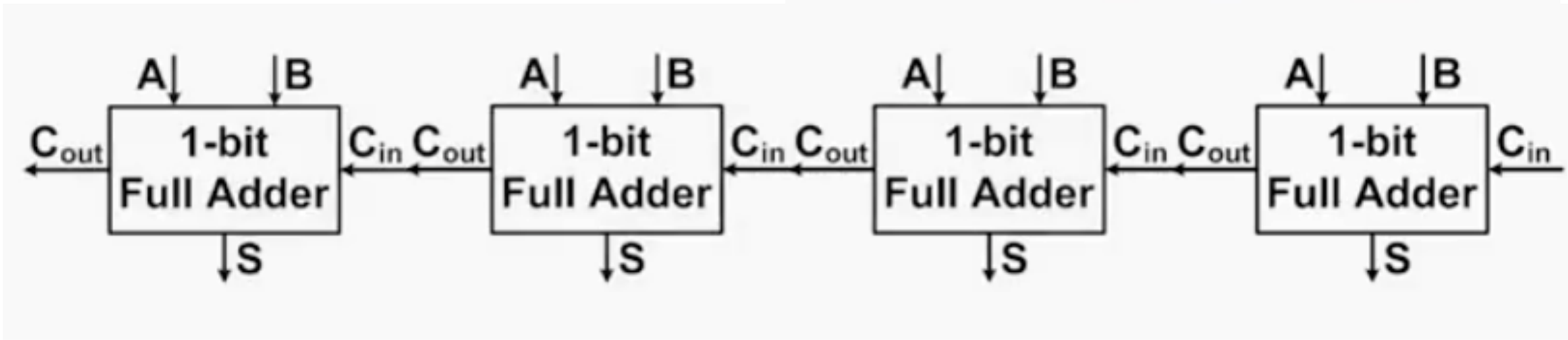


4-bit Full Adder

- How it works?

- Each bit assigned!

```
  1 1 0 1
+ 0 1 1 0
---------
1 0 0 1 1
```

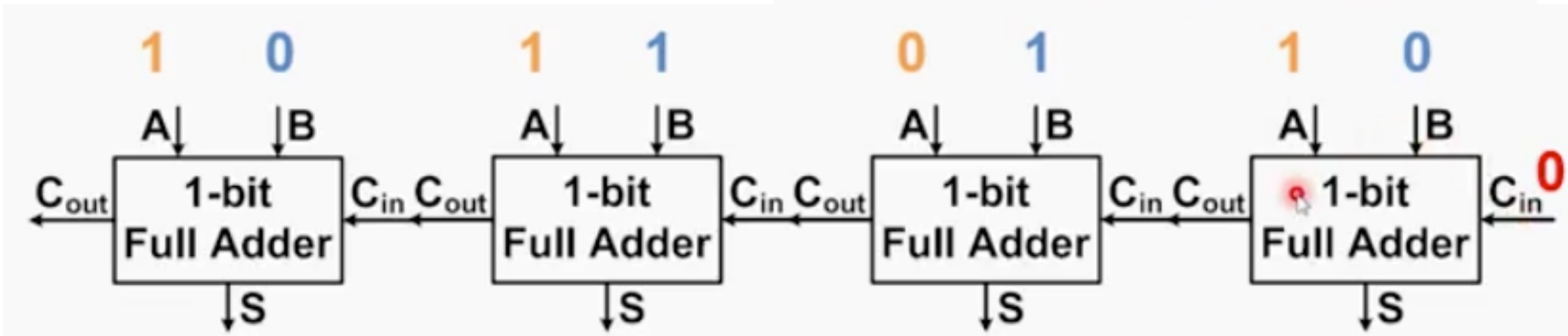


4-bit Full Adder

- How it works?

- Each bit assigned!

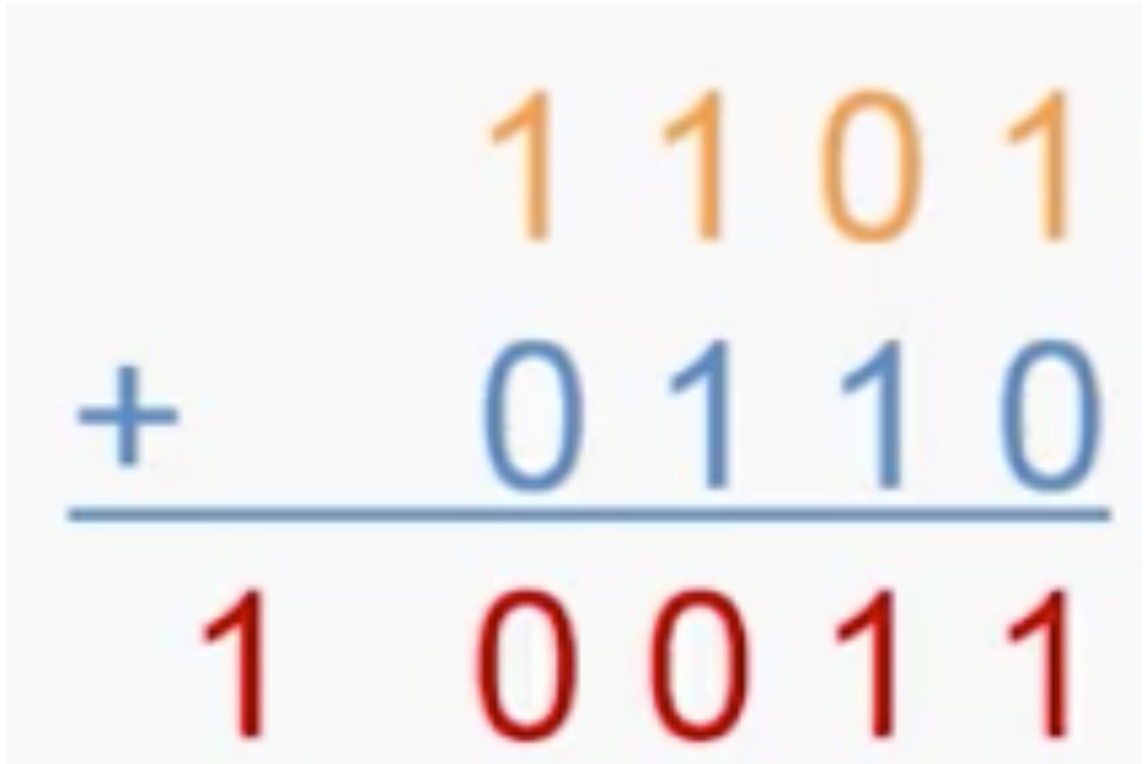
```
  1 1 0 1
+ 0 1 1 0
---------
1 0 0 1 1
```

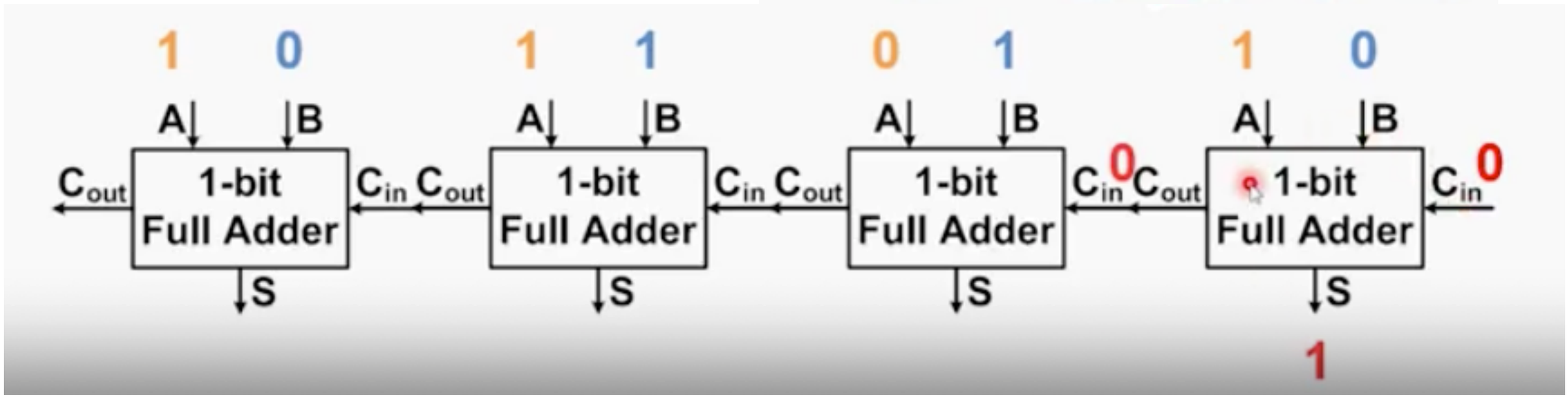


4-bit Full Adder

- How it works?

- Each bit assigned!

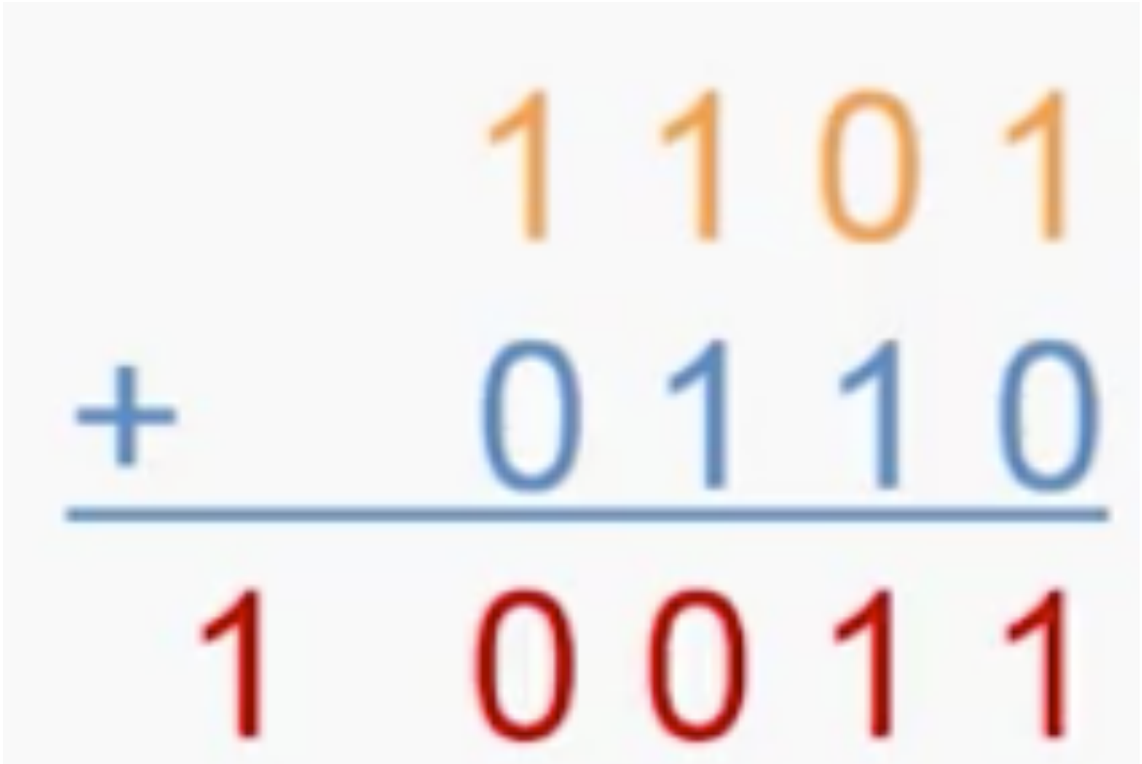


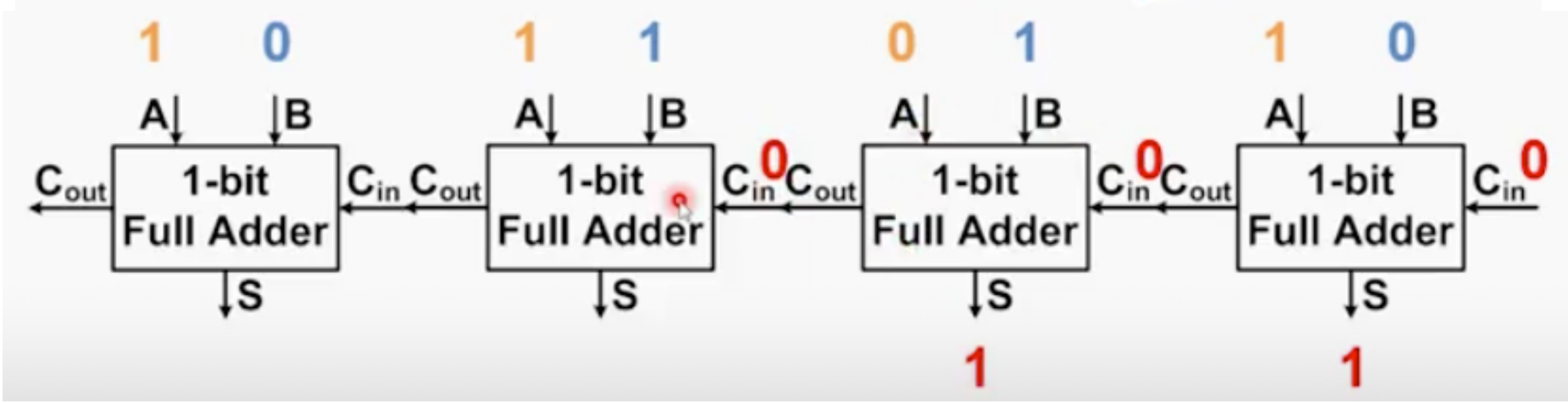


4-bit Full Adder

- How it works?

- Each bit assigned!

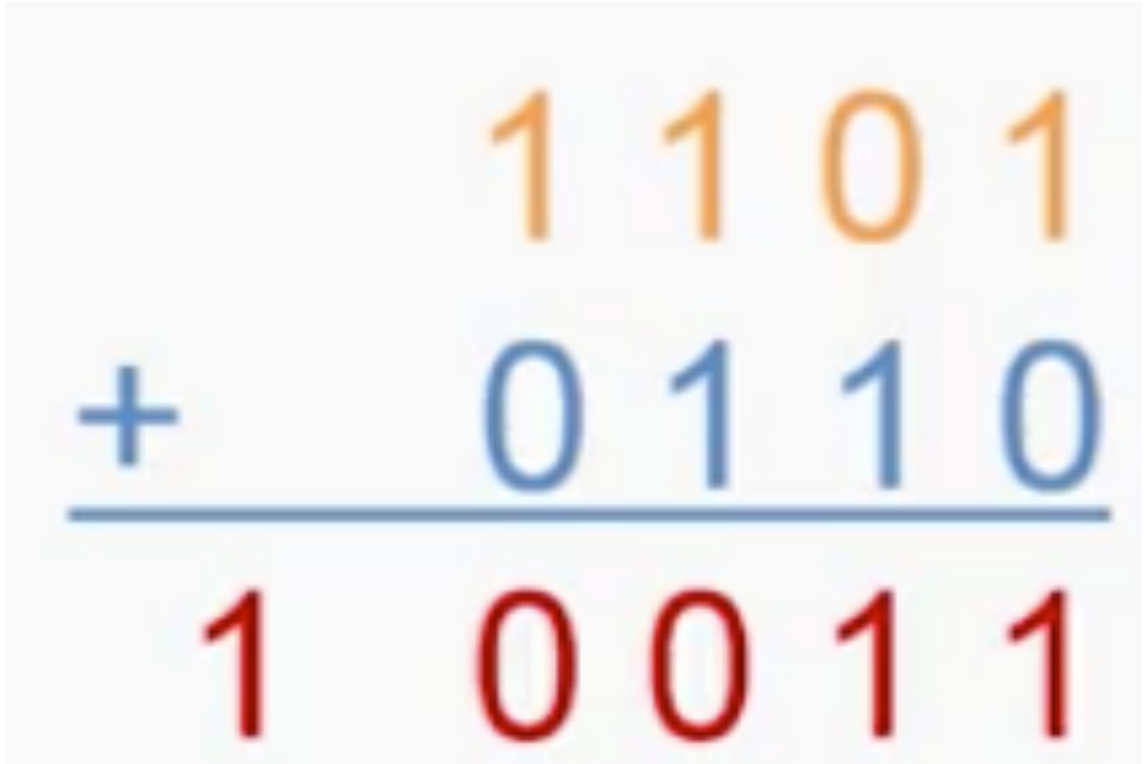


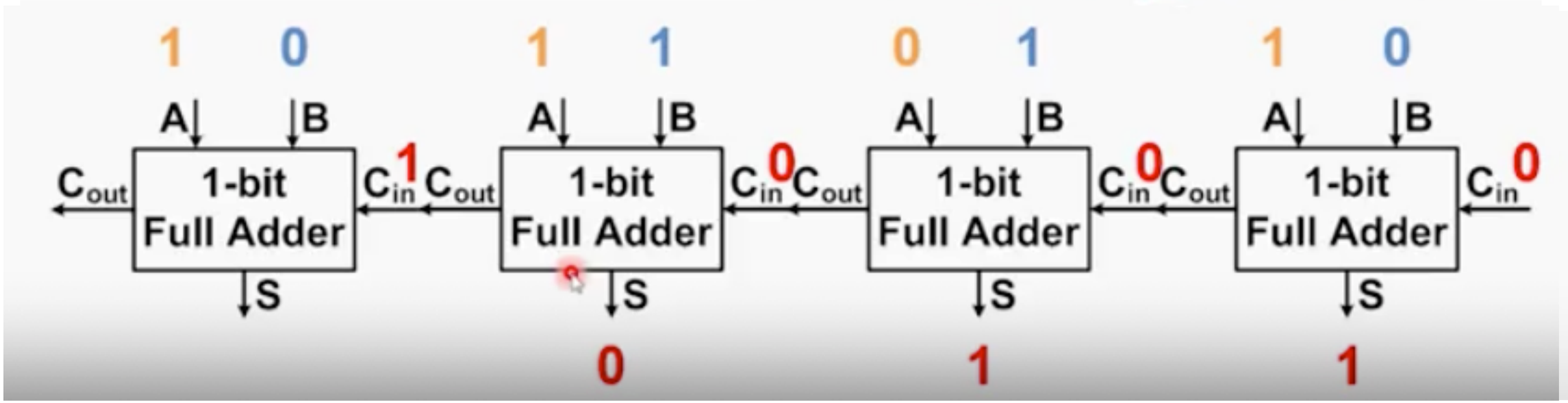


4-bit Full Adder

- How it works?

- Each bit assigned!

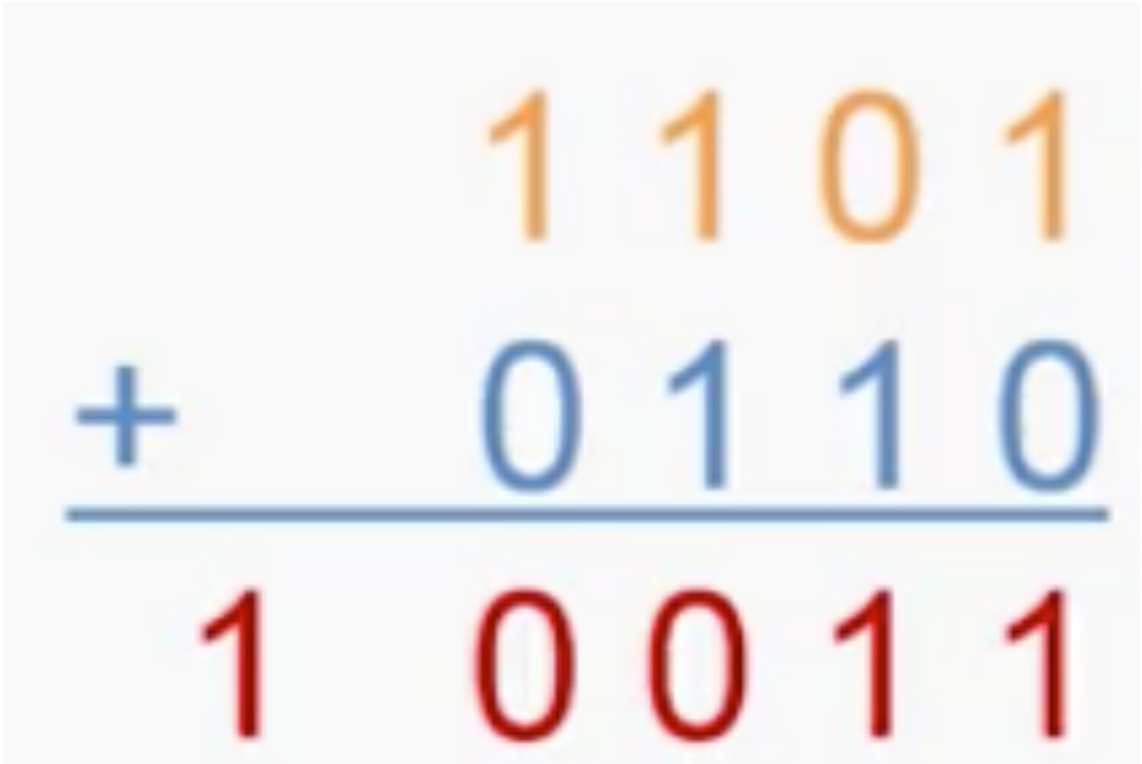


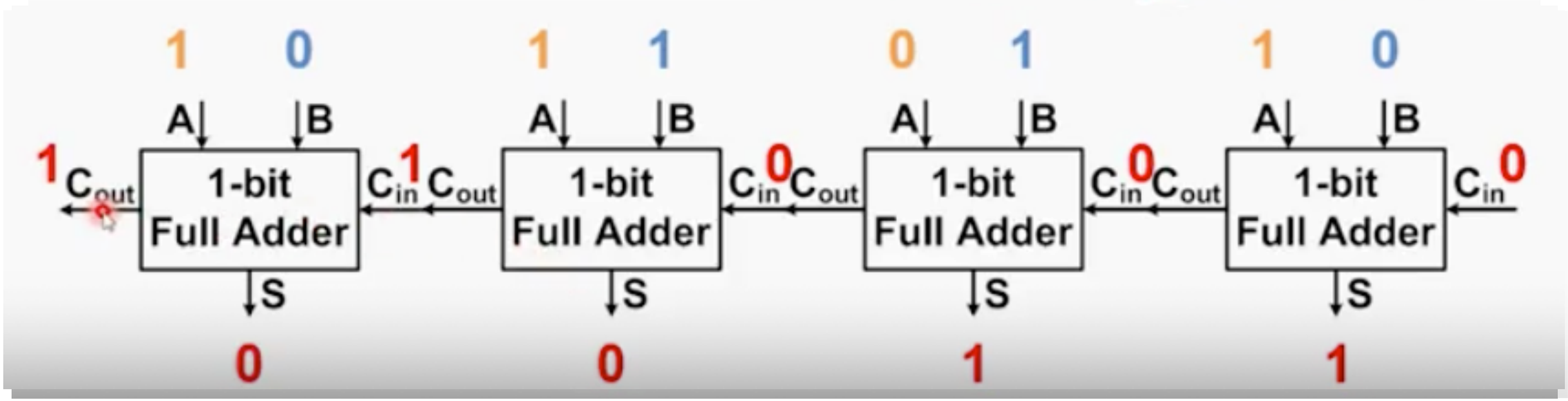


4-bit Full Adder

- How it works?

- Each bit assigned!





More bits Added

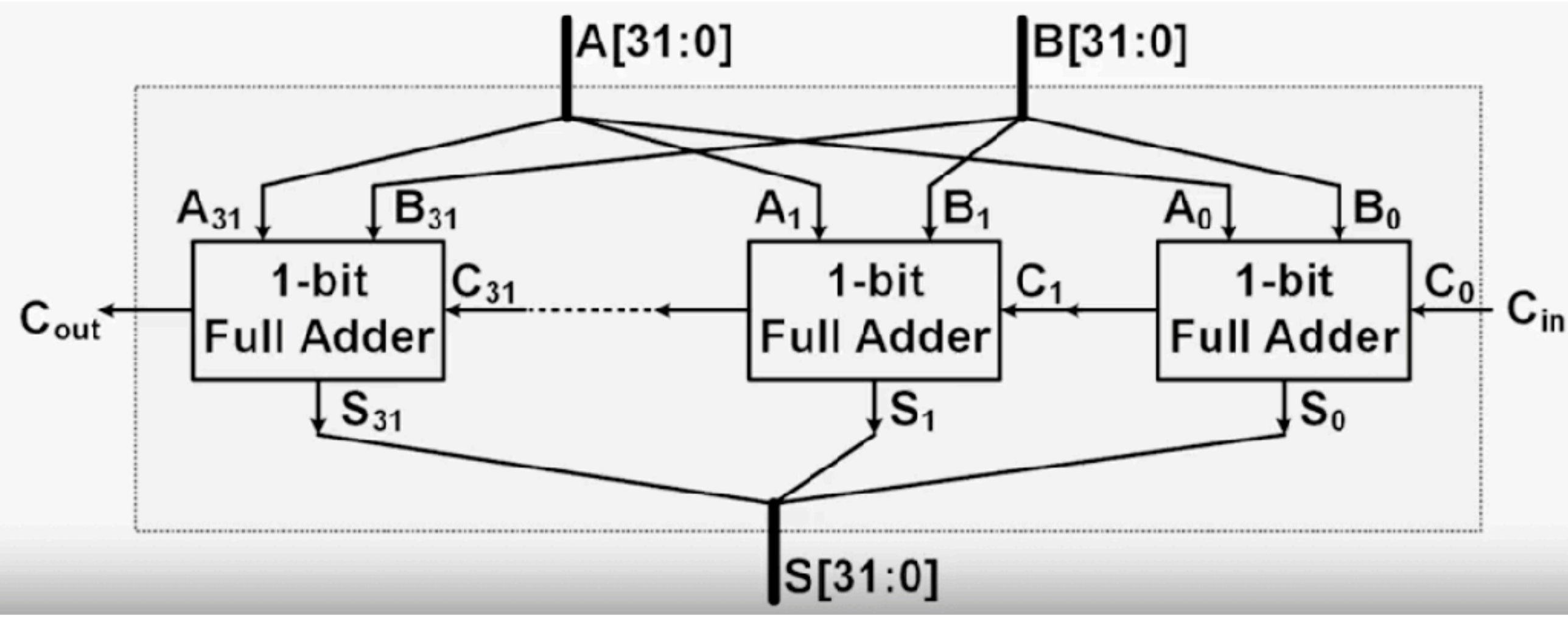
- 32-bit full adder

More bits Added

- 32-bit full adder
  - How to design?

More bits Added

- 32-bit full adder
  - How to design?



EECE 2322: Fundamentals of Digital Design and Computer Organization

Lecture 3_3: MUX, Decoder, and Encoder

Xiaolin Xu
Department of ECE
Northeastern University

Real-World Digital Applications

Select Input Source:
Press the Projector Source button

Select Input Source:
Press the desired source button

To Use Computer:
Button will show Computer

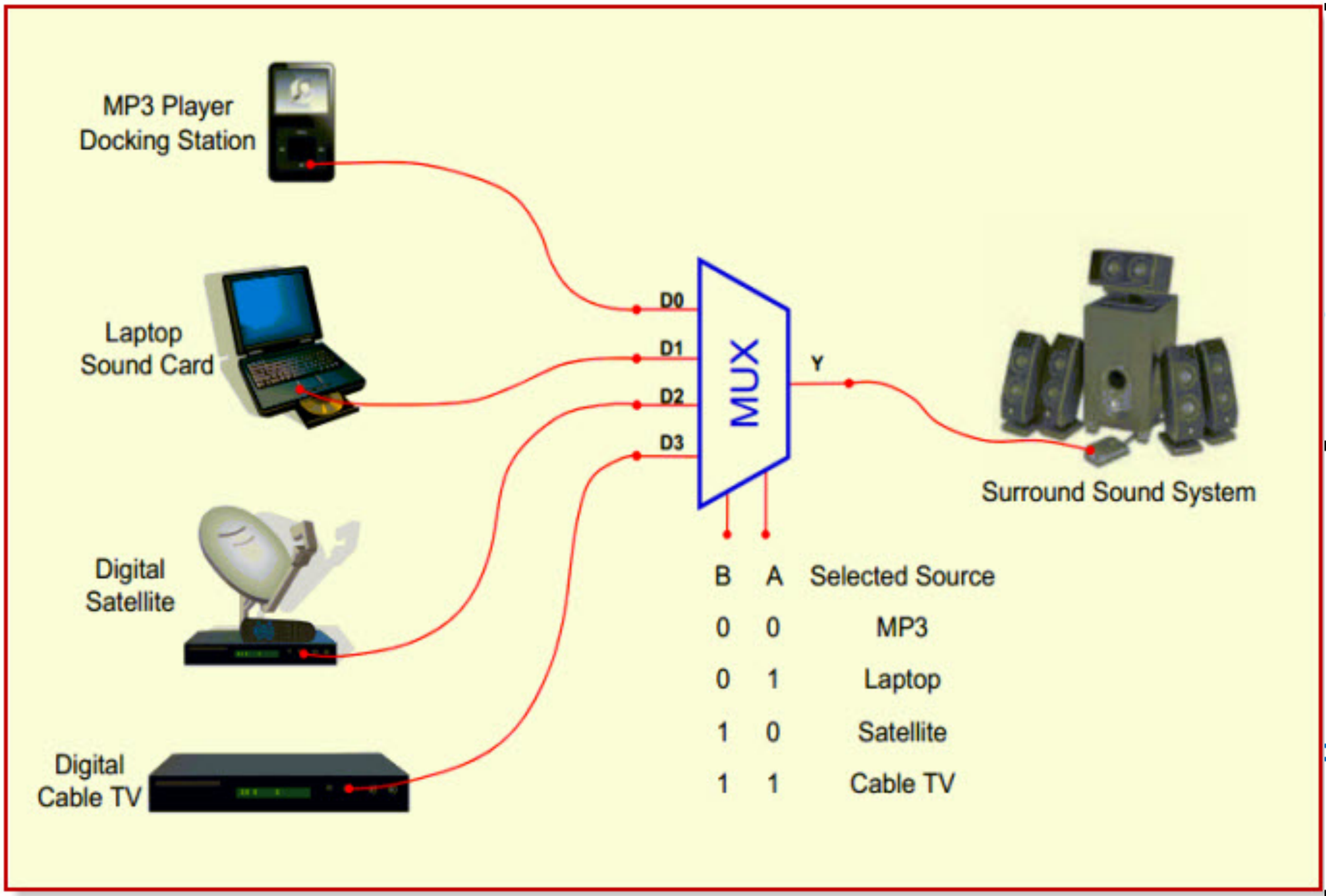
To Use Doc Cam:
Button will show Doc Cam

To Use Laptop VGA:
Button will show Laptop VGA

To Use Laptop HDMI:
Button will show Laptop HDMI

To Use AirMedia:
Button will show AirMedia

Real-World Digital Applications



Revisit Multiplexers (MUX)

- Data selector.
- Selects binary information from one of many input lines to a single output line.
- Selection inputs control which particular input line to select.
- Normally, there are 2n input lines, n selection inputs, and one output

MUX

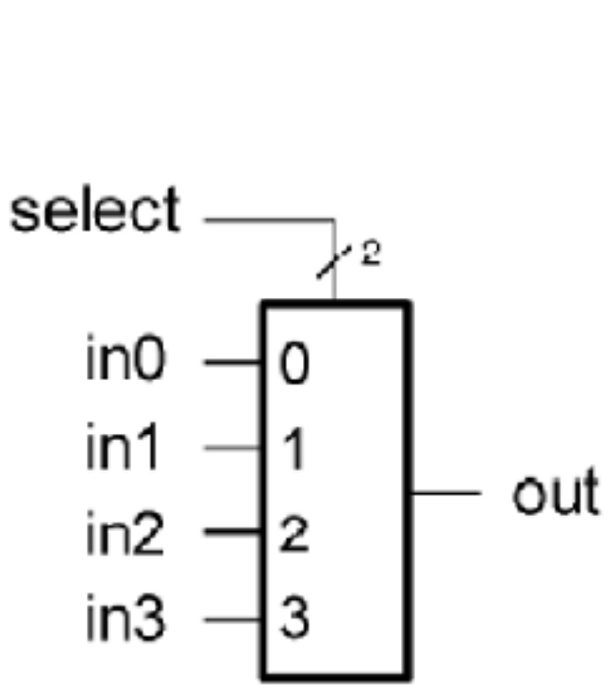
- A simple multiplexer has a single bit output
- Common Multiplexers are
- 2-to-1 2 inputs _?_ select lines
- 4-to-1 4 inputs _?_ select lines
- 8-to-1 8 inputs _?_ select lines
- 2n-to-1 2n inputs _?_select lines

2-1 MUX in Verilog

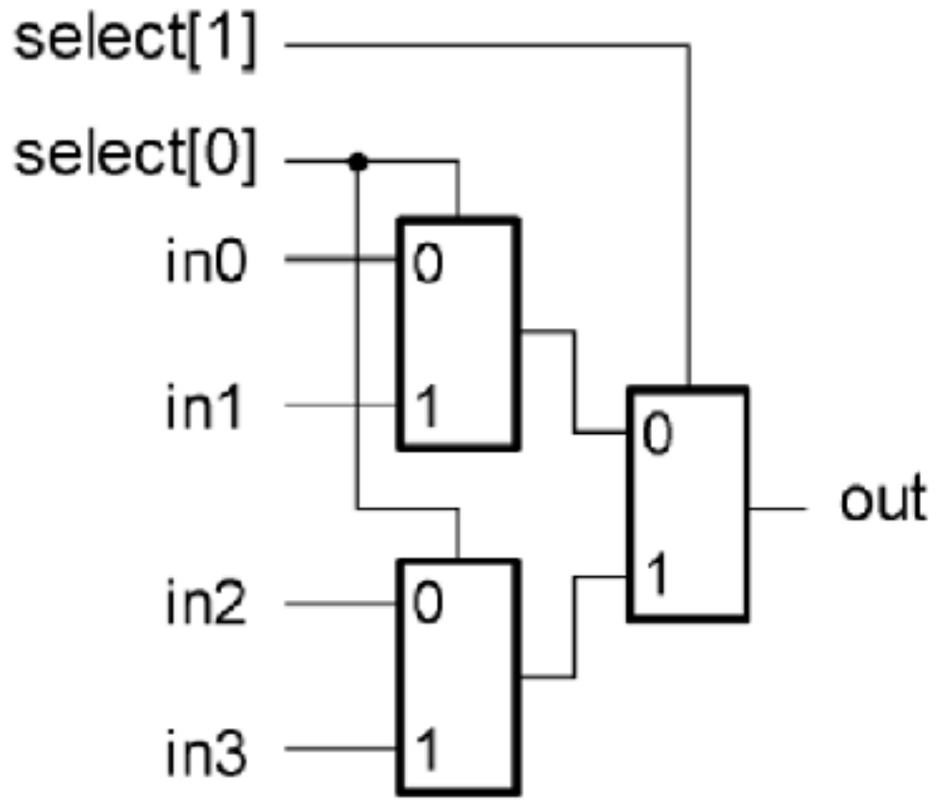
```
module mux2 (in0,in1,select,out);
    input in0,in1,select;
    output out;
    reg out;
    always @ (in0 or in1 or select) // sensitivity list
      if (select) out=in1;
      else out=in0;
endmodule // mux2
```

Larger MUX Design

- 4-to-1 mux out of 2-to-1 muxes



a) 4-input mux symbol



b) 4-input mux implemented with 2-input muxes

Verilog Code

- 4-to-1 mux out of 2-to-1 muxes

```
module mux4(in0, in1, in2, in3, select, out);
    input [1:0] select;
    input in0, in1, in2, in3;
    output out;
    wire w0, w1;

    mux2
     m0 (.select(select[0]), .in0(in0), .in1(in1), .out(w0)),
     m1 (.select(select[0]), .in0(in2), .in1(in3), .out(w1)),
     m2 (.select(select[1]), .in0(w0), .in1(w1), .out(out));
endmodule // mux4
```

Testbench for 4-to-1 MUX

```
module testmux4;
   reg [5:0] count = 6'b000000;
   reg a,b,c,d;
   reg [1:0] s;
   reg expected;
   wire f;

mux4 myMux(.select(s), .in0(a), .in1(b), .in2(c), .in3(d), .out(f));
initial
 begin
  repeat(64)
    begin
      a = count[0];
      b = count[1];
      c = count[2];
      d = count[3];
      s = count[5:4];
```

Testbench for 4-to-1 MUX (2)

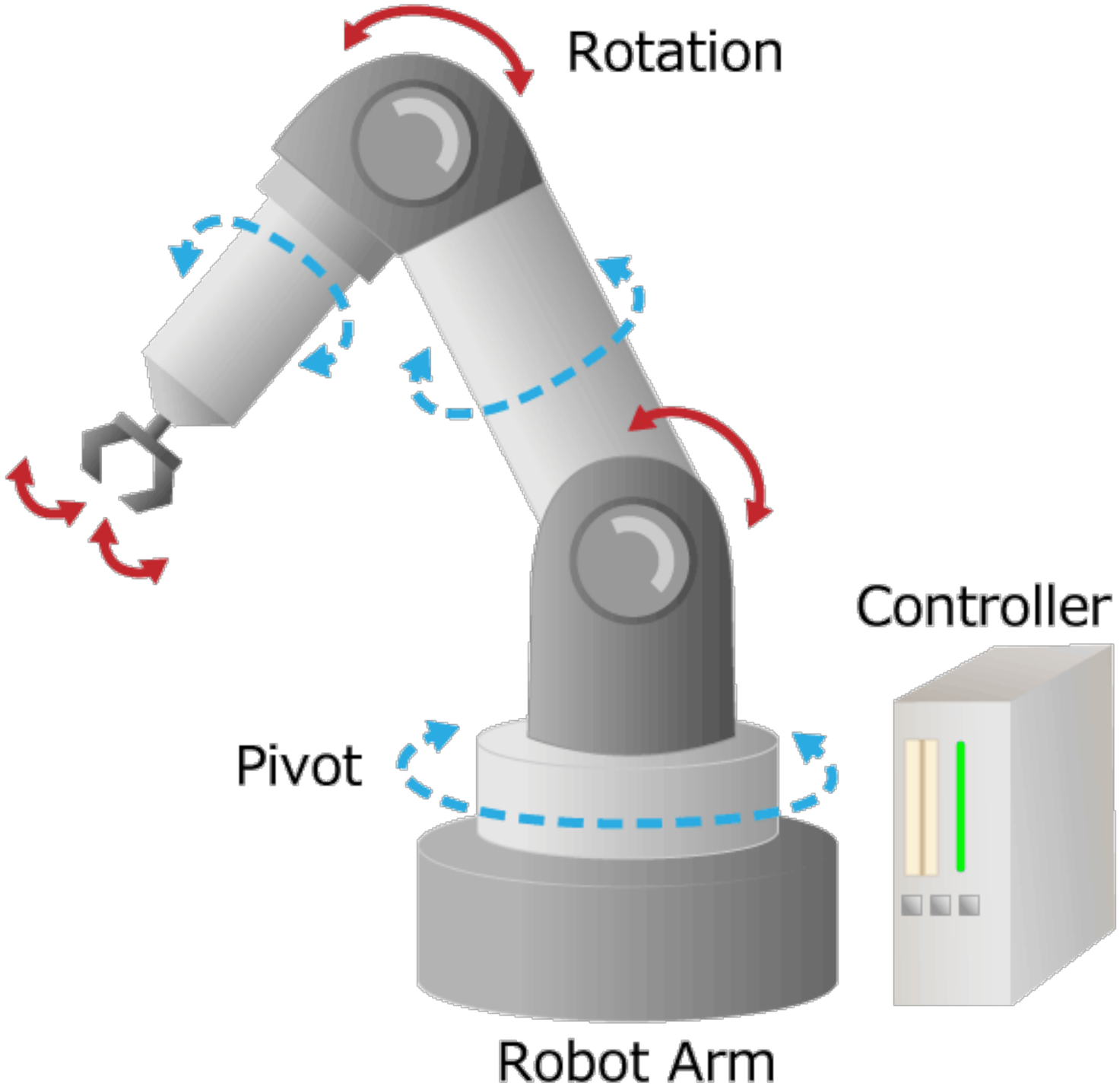
```
case (s)
        2'b00:  expected = a;
        2'b01:  expected = b;
        2'b10:  expected = c;
        2'b11:  expected = d;
    endcase
    #5
  $strobe("select=%b in0=%b in1=%b in2=%b in3= %b out = %b,
        expected = %b time =%d", s, a, b, c, d, f, expected, $time);
    #5 count = count + 1'b1;
  end
$stop;
end
endmodule //testbench
```

case in Verilog

- Checks if the given expression matches one of the other expressions in the list and branches accordingly
- **Mostly used in building MUXes (see why later)**

```
case (<expression>)
    case_item1 :    <single statement>
    case_item2,
    case_item3 :    <single statement>
    case_item4 :    begin
                        <multiple statements>
                        end
    default        : <statement>
endcase
```

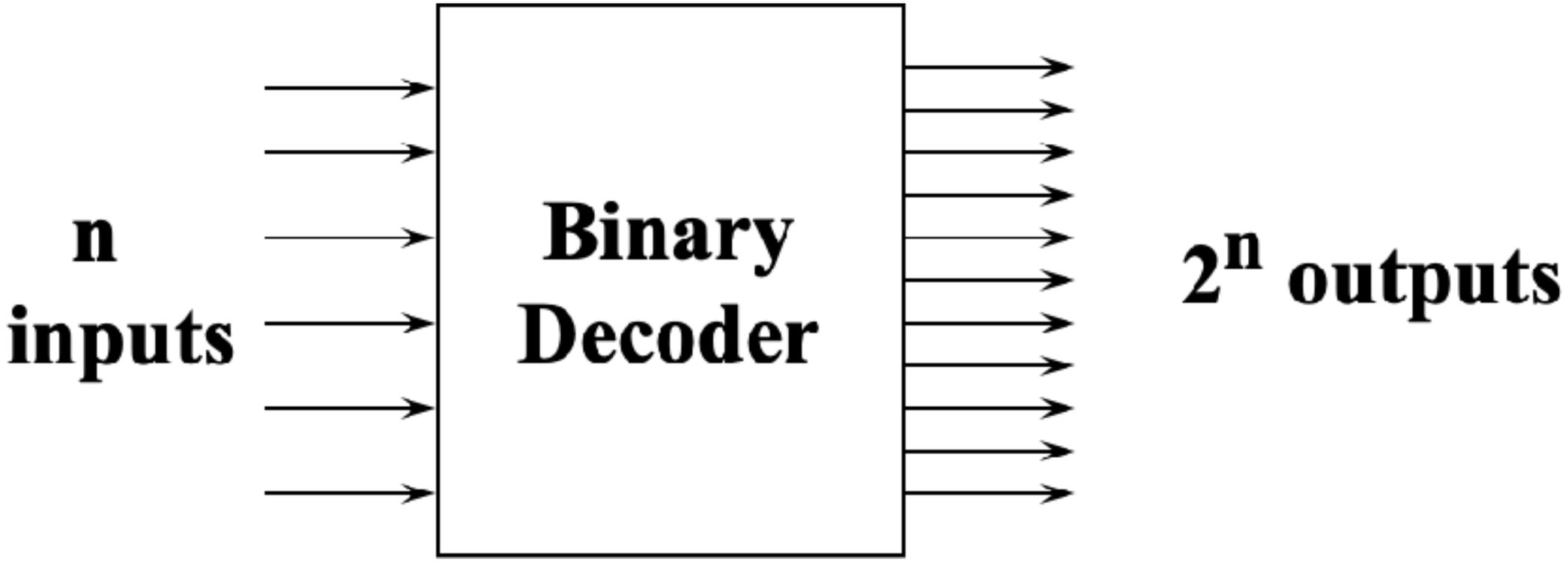
Real-World Digital Applications



Real-World Digital Applications

Decoder

- Logic with n input lines and 2n output lines
- Only one output is a 1 for any given input
-



Decoder Example

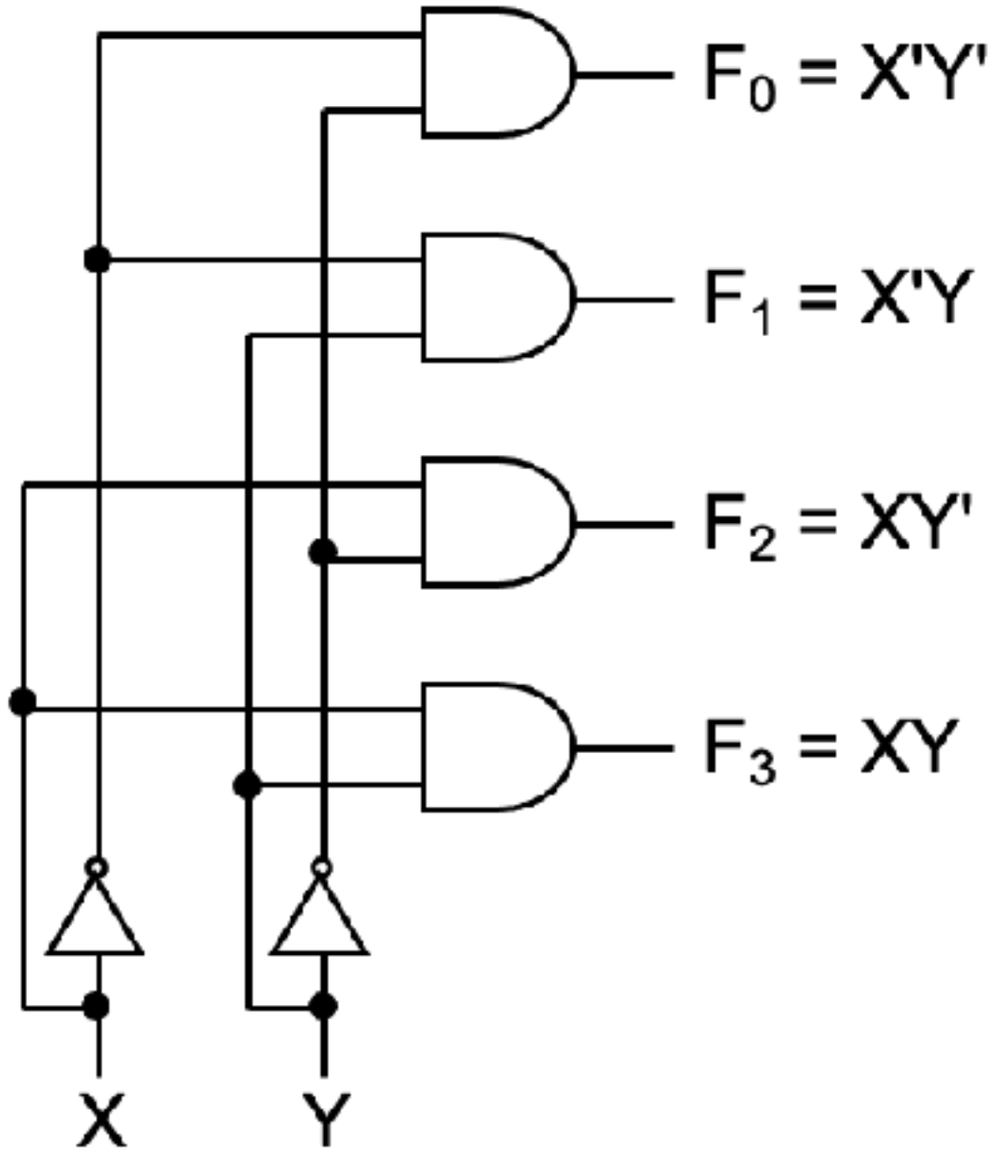
- 2-to-4 decoder

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Decoder Example

- 2-to-4 decoder

X	Y	F_0	F_1	F_2	F_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

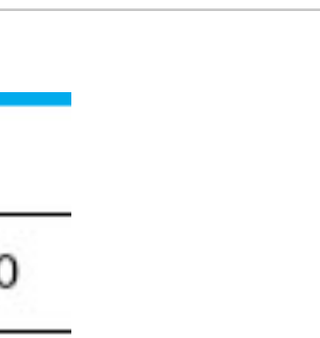


Binary Decoder with ENable

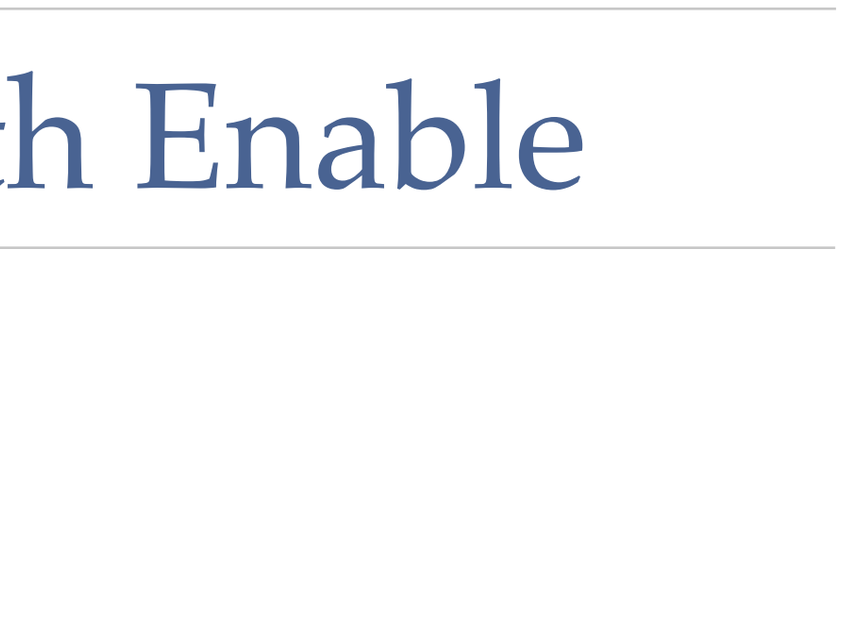
Inputs			*Outputs*			
EN	A1	A0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Binary Decoder with Enable

Inputs			*Outputs*			
EN	A1	A0	Y3	Y2	Y1	Y0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0



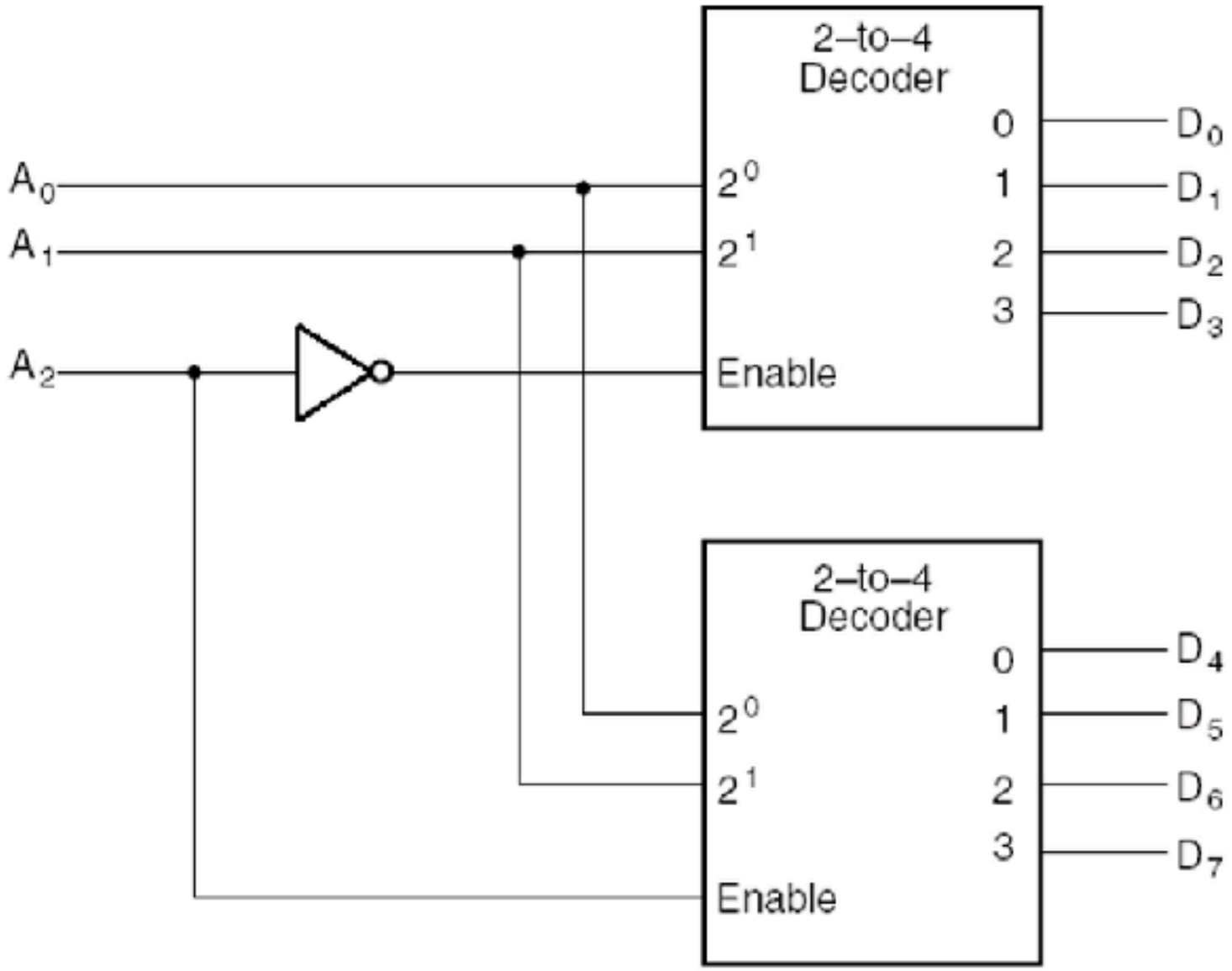
(a)



(b)

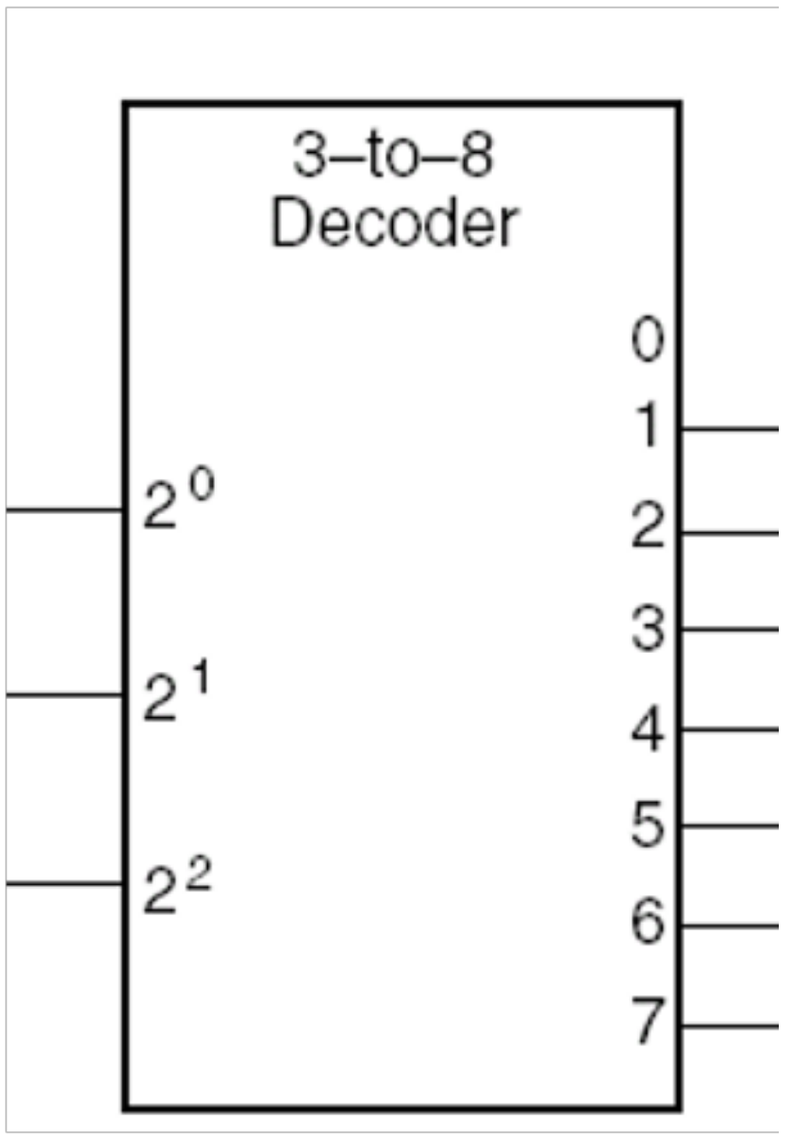
Building Larger Decoder

- Build larger decoders out of two or more smaller decoders
-



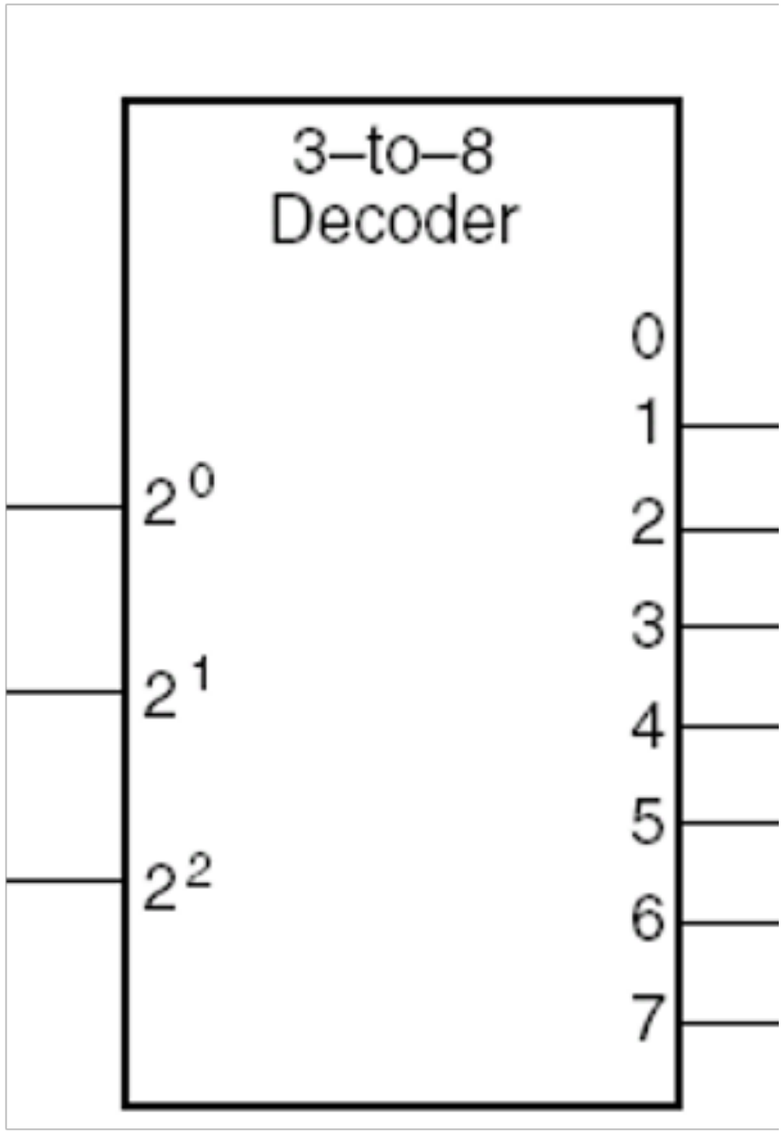
Decoder Example

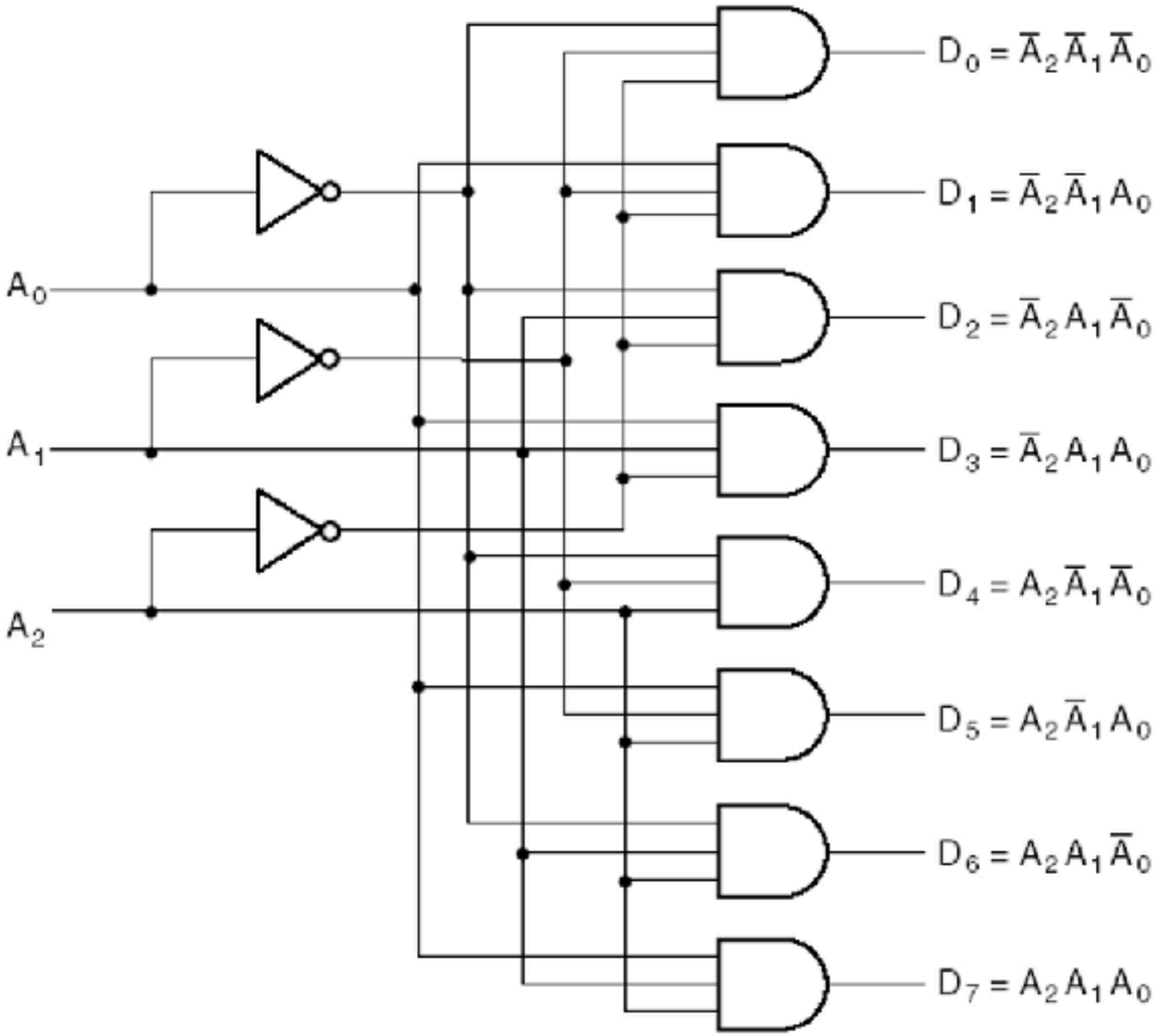
- 3-to-8 decoder



Decoder Example

- 3-to-8 decoder





5-To-32 Decoder

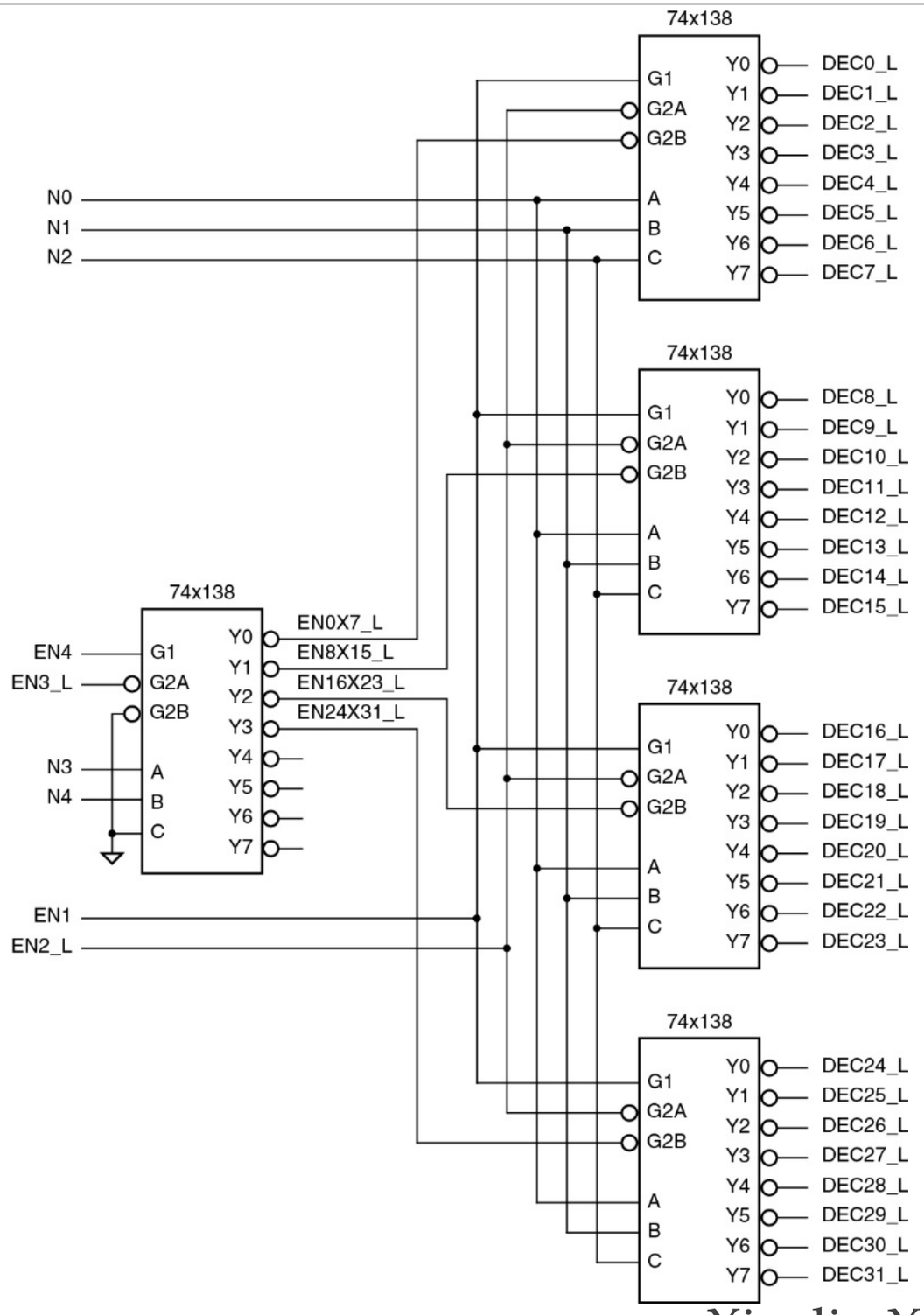
- How?

5-To-32 Decoder

- How?
- Cascading 3-To-8 decoders

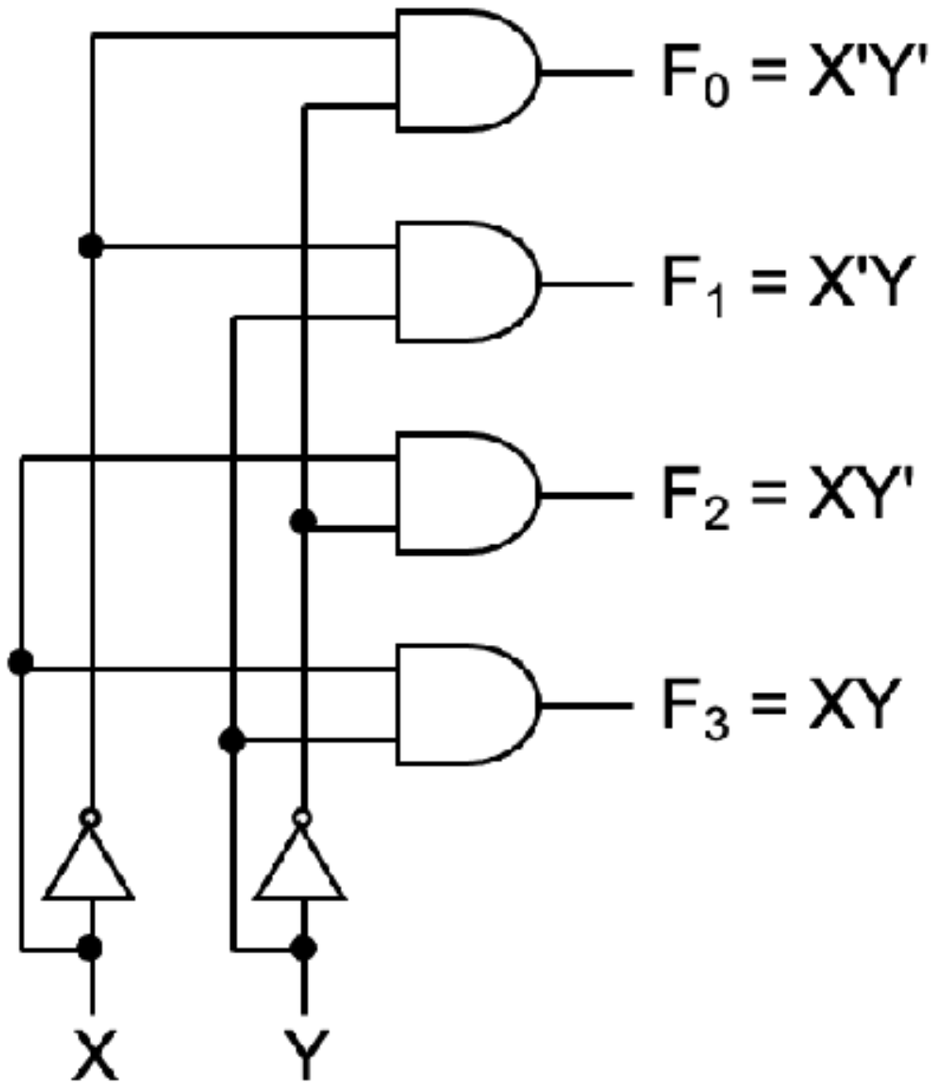
5-To-32 Decoder

- How?
- Cascading 3-To-8 decoders



2-to-4 Decoder in Verilog

- Think about how?



- Structural-style Verilog Module

2-to-4 Decoder in Verilog

- Think about how?

```
module Vr2to4dec(I0, I1, EN, Y0, Y1, Y2, Y3);
    input I0, I1, EN;
    output Y0, Y1, Y2, Y3;
    wire NOTI0, NOTI1;

    not U1 (NOTI0, I0);
    not U2 (NOTI1, I1);
    and U3 (Y0, NOTI0, NOTI1, EN);
    and U4 (Y1, I0, NOTI1, EN);
    and U5 (Y2, NOTI0, I1, EN);
    and U6 (Y3, I0, I1, EN);
endmodule
```

- Structural-style Verilog Module

2-to-4 Decoder in Verilog

```
module dec2to4 (Y, W, En);
   input [1:0] W;
   input En;
   output [3:0] Y;
   reg [3:0] Y;

   always@ (W or En)
      case ({En, W})    // concatenation of En and W
        3'b100: Y = 4'b0001;
        3'b101: Y = 4'b0010;
        3'b110: Y = 4'b0100;
        3'b111: Y = 4'b1000;
        default: Y = 4'b0000;
      endcase
endmodule
```

A Decoder with ENable in Verilog

- Think about how?

- Data flow style Verilog module

A Decoder with ENable in Verilog

- Think about how?

```
module dec24_dat(
output [3:0] y,
input [1:0] a,
input en);
assign y = en ? (4'b0001 << a) : 0;
endmodule
```

- Data flow style Verilog module