Version 1.0

mirafra TECHNOLOGIES

Digital Design – Combinational Logic II

Dr. Ipsita Biswas Mahapatra

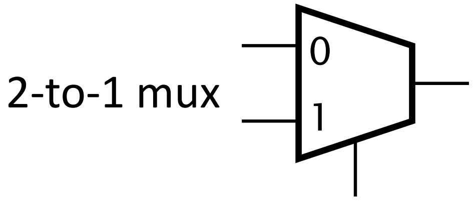
Member of Technical Staff

Goals & objectives

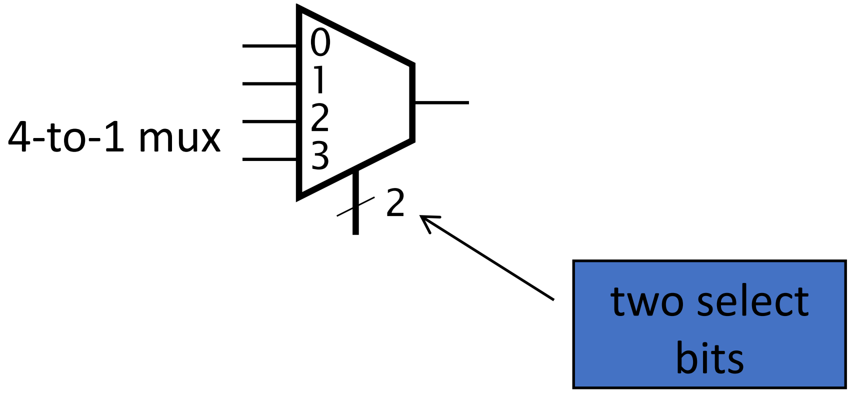
- Multiplexers
- De-multiplexers
- Encoders
- Decoders
- Priority encoders
- Comparator
- Half adder, Full adder
- Half subtractor, Full subtractor

Multiplexers

- It has two data inputs, one data output, and a select input that determines which input value is used for the output value.
- We can expand on this simple multiplexer along two dimensions:
  - First, we can add more data inputs, which also requires adding further select inputs to encode the choice of input to drive the output.
  - Second, we can use multiplexers in parallel to select between two sources of multibit encoded data.



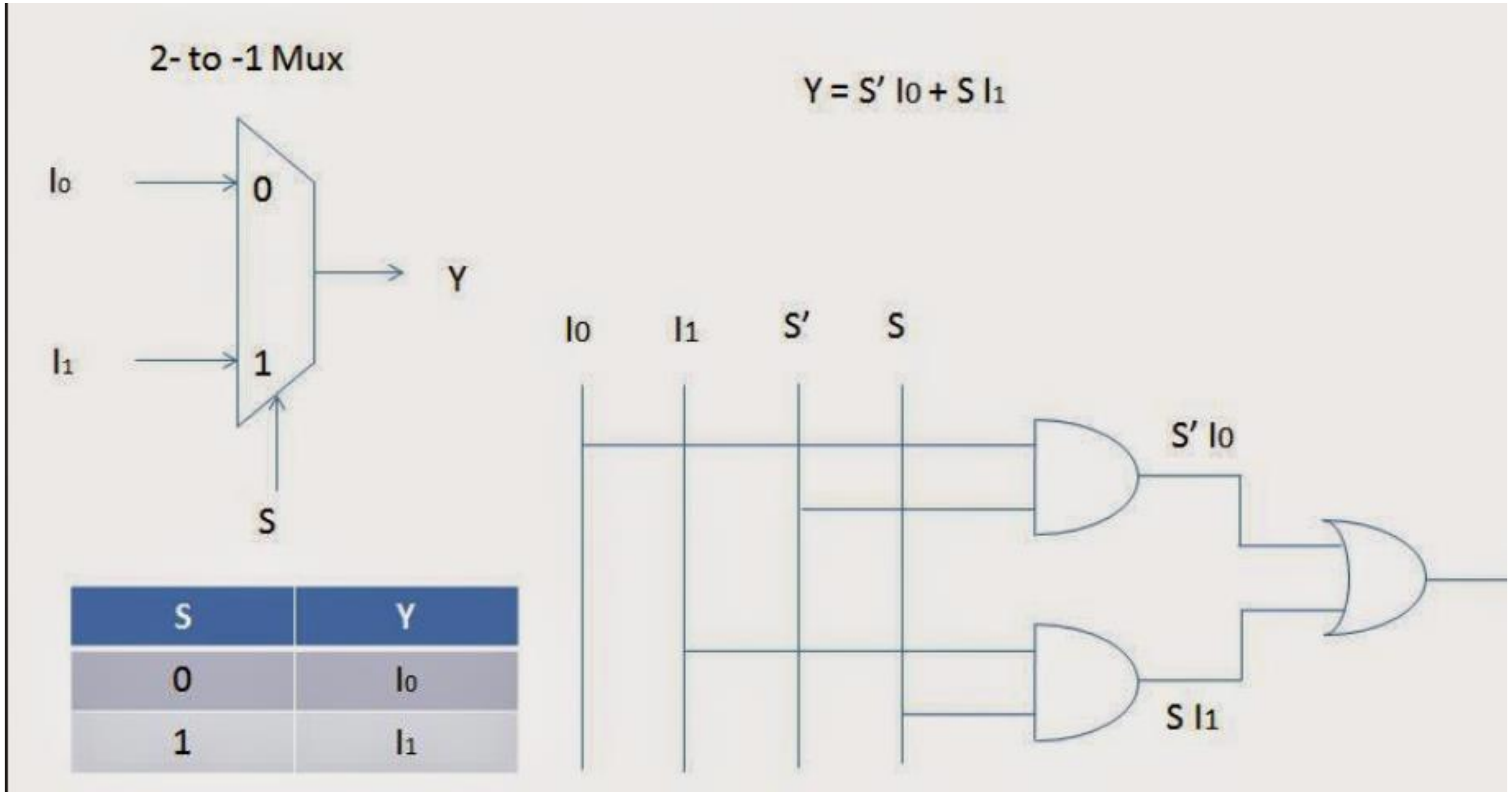
sel	z
0	a_0
1	a_1



sel	z
00	a_0
01	a_1
10	a_2
11	a_3

- N-to-1 multiplexer needs [$\log_2 N$] select bits

Multiplexer as data selector

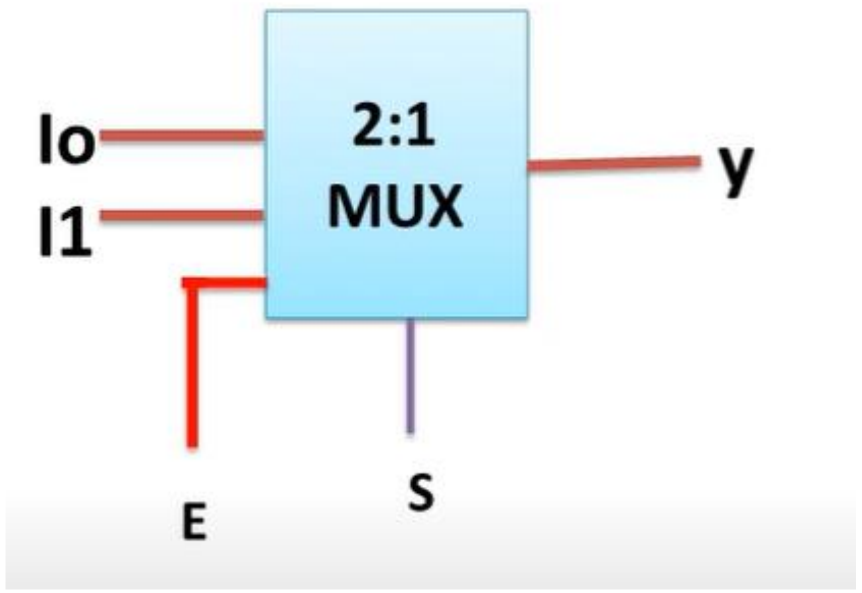


$Y = S' I_0 + S I_1$

S	Y
0	I_0
1	I_1

mirafra
TECHNOLOGIES

Multiplexer – with 2 control signals

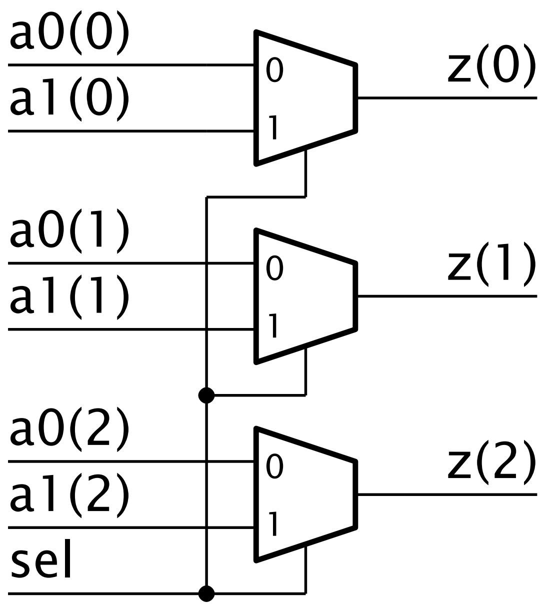


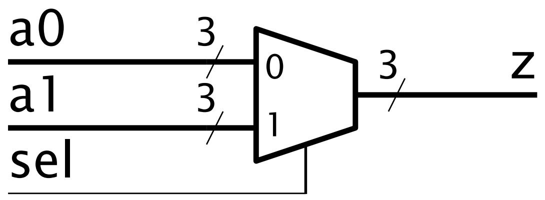
E	S	Y
0	X	0
1	0	I0
1	1	I1

$$Y = E.S'.I_0 + E.S.I_1$$

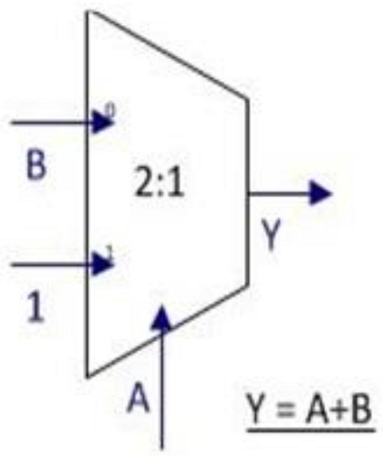
Multi-bit Multiplexers

- To select between *N* *m*-bit codeword inputs
  - Connect *m* *N*-input multiplexers in parallel
- Abstraction
  - Treat this as a component

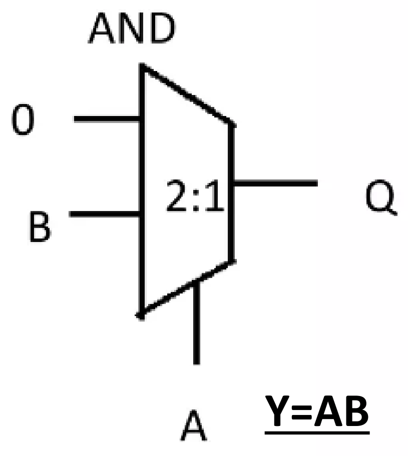




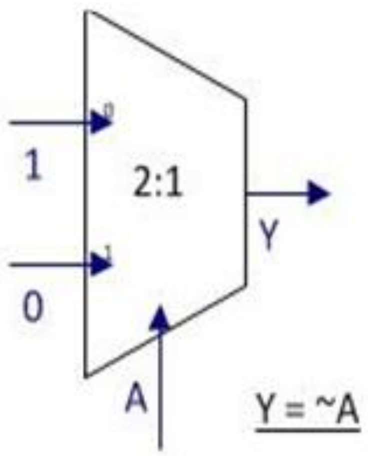
Basic gates using 2:1 Mux



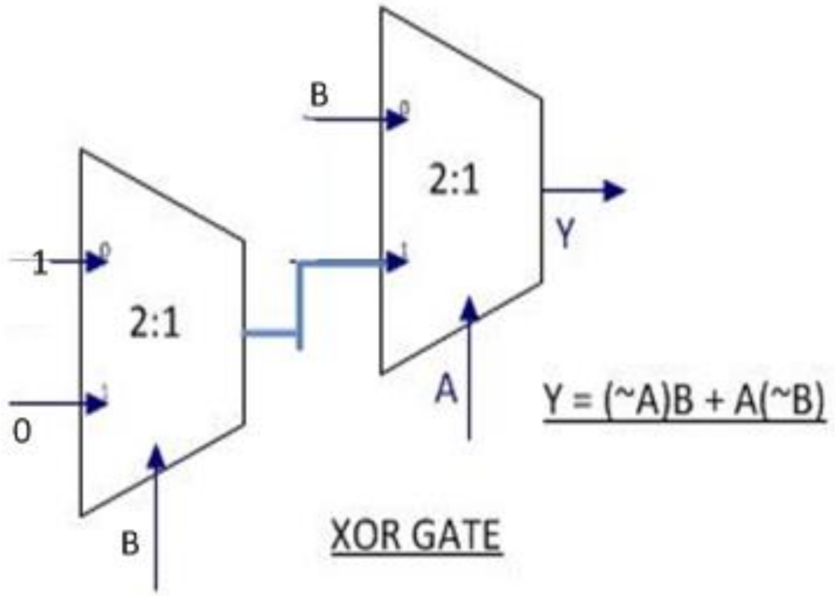
OR GATE



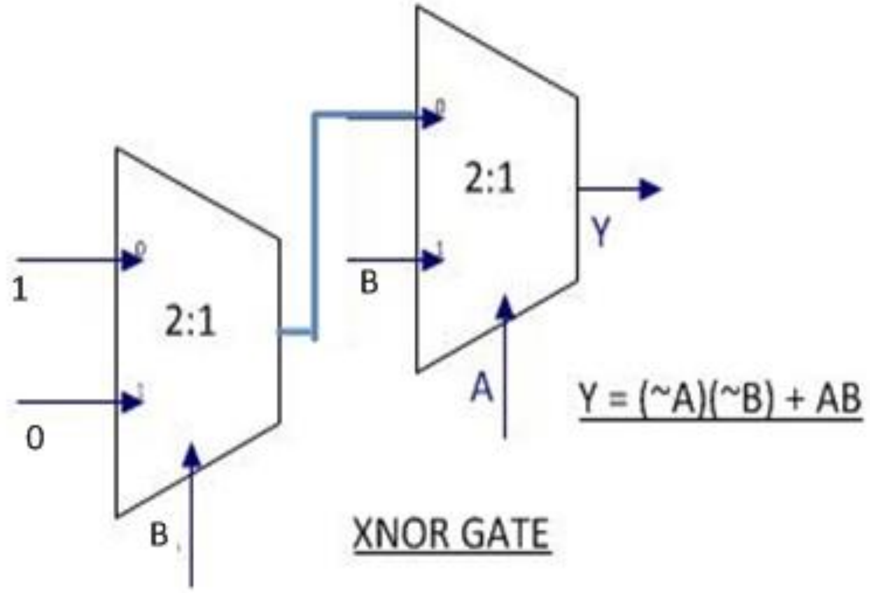
AND GATE



NOT GATE



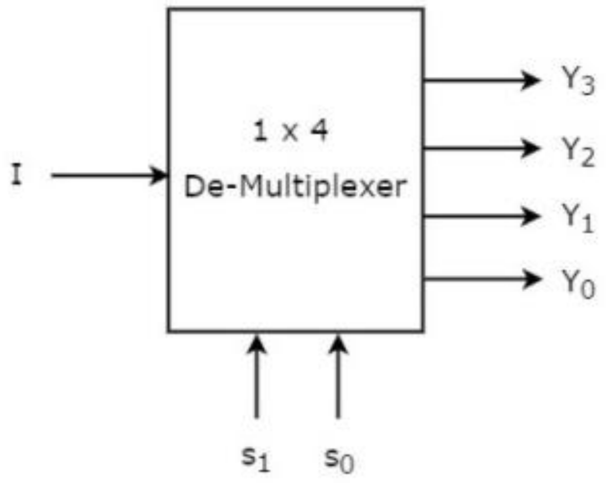
XOR GATE

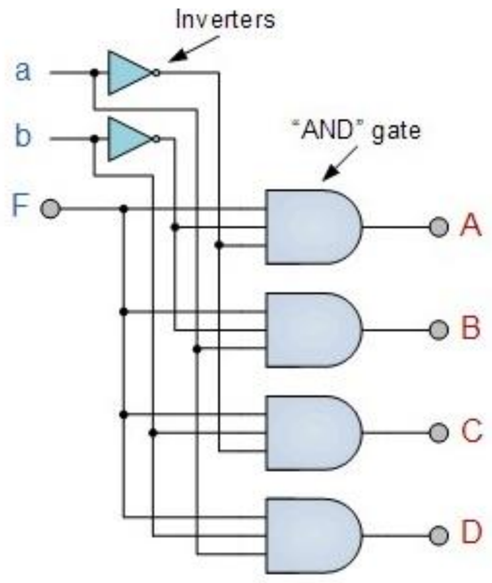


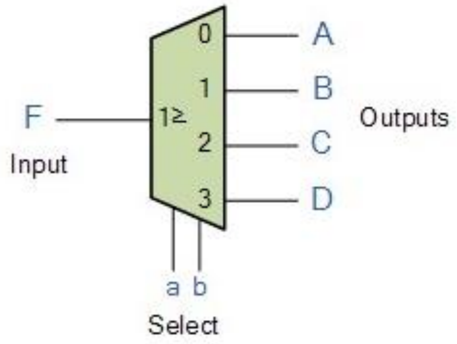
XNOR GATE

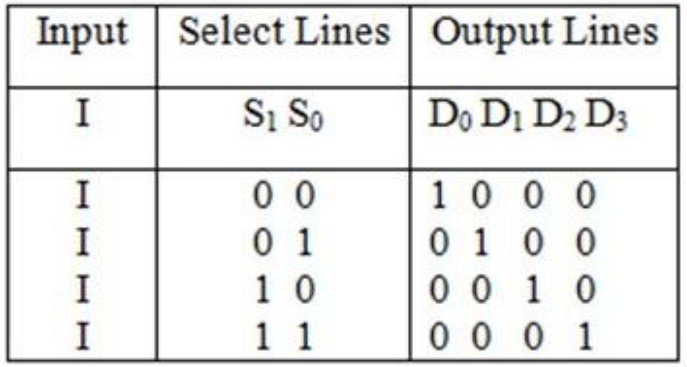
De-multiplexer

- Used to connect a single source to multiple destinations.
  - Example – Drive same signal to various outputs based on the selection.



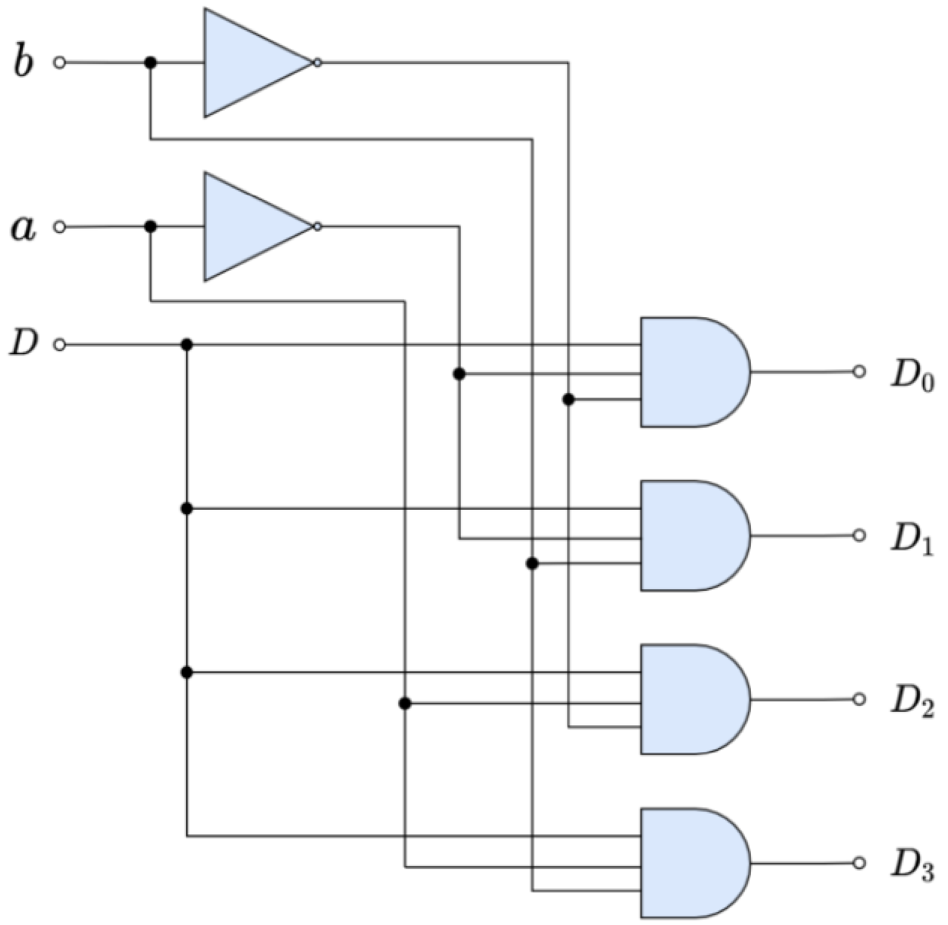






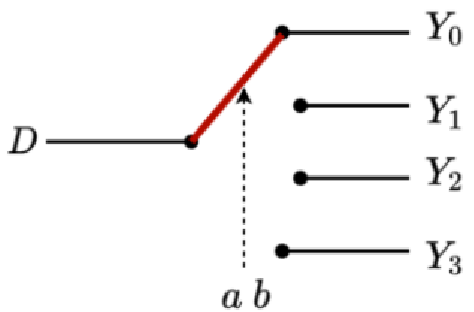
Input	Select Lines	Output Lines
I	S_1 S_0	D_0 D_1 D_2 D_3
I	0 0	1 0 0 0
I	0 1	0 1 0 0
I	1 0	0 0 1 0
I	1 1	0 0 0 1

De-multiplexer



Select Inputs		Input	Data Outputs			
b	a	D	Y_3	Y_2	Y_1	Y_0
0	0	1	0	0	0	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	1	0	0	0

$$D = \overline{a}\overline{b}Y_0 + a\overline{b}Y_1 + \overline{a}bY_2 + abY_3$$



Switch Equivalent (SPQT)

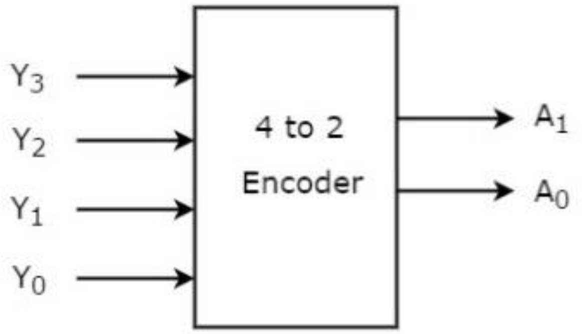
Summary: Mux and De-mux

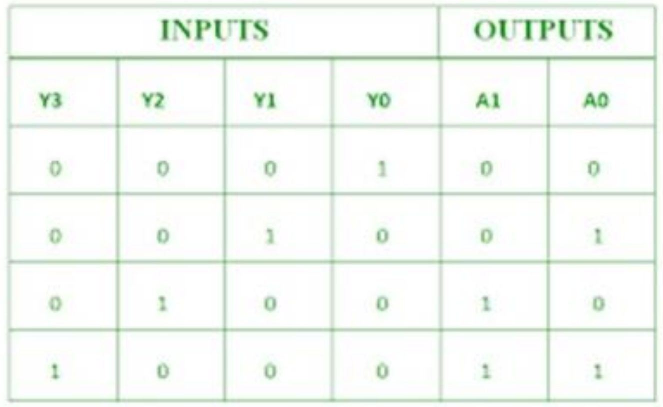
- Mux
  - It is a combinational circuit that selects binary information's from one of many input lines and direct it to output line
  - It is simply a data selector
- De-mux
  - One input many outputs
  - Data distributors
  - Reverse operation of mux

mirafra TECHNOLOGIES

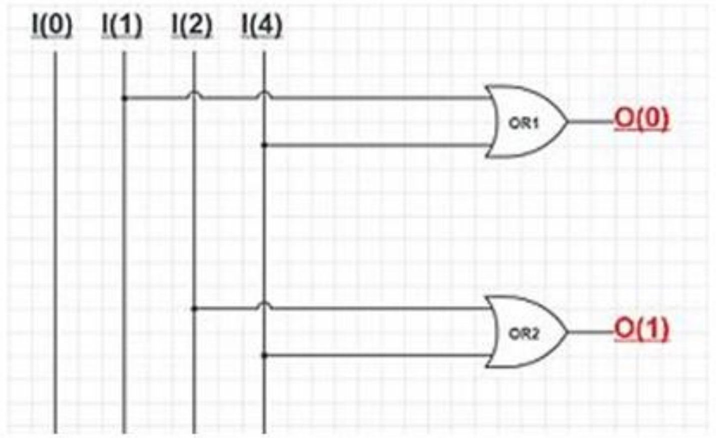
Encoder

- Applications: Decimal to Binary encoding in calculators
  - Required only one bit of all the input bits to be 1
  - If this is violated; then the output is 0





INPUTS				OUTPUTS	
Y3	Y2	Y1	Y0	A1	A0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1



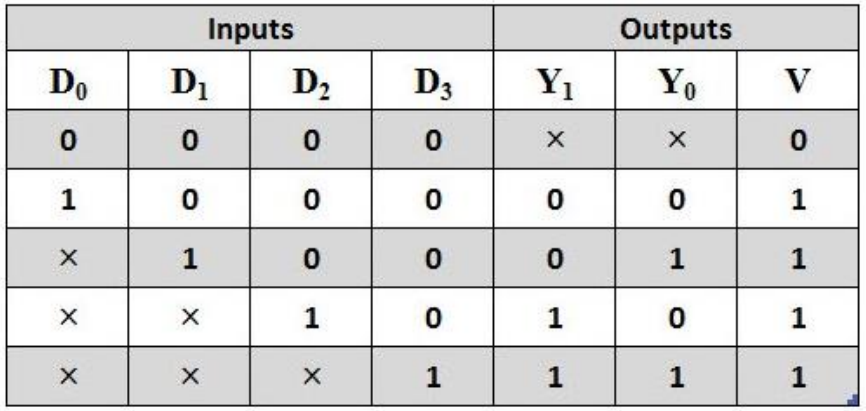
Priority Encoders

- Now let’s consider the possibility of more than one input to an encoder being 1 at a time. The design we described above would produce an incorrect output, possibly an invalid code word.
- The solution is to assign priorities to the inputs, so that if multiple inputs are 1, the encoder outputs the code word corresponding to the input with highest priority.
- Such an encoder is called, not surprisingly, a priority encoder .
- One application of priority encoders is to prioritize interrupts in embedded systems.

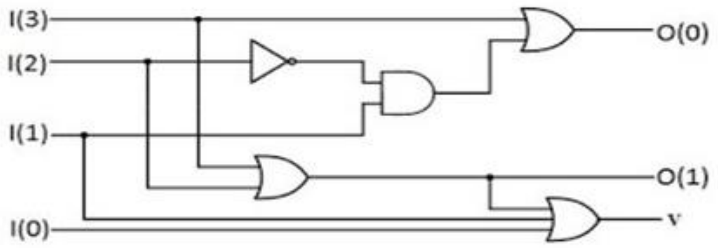
mirafra
TECHNOLOGIES

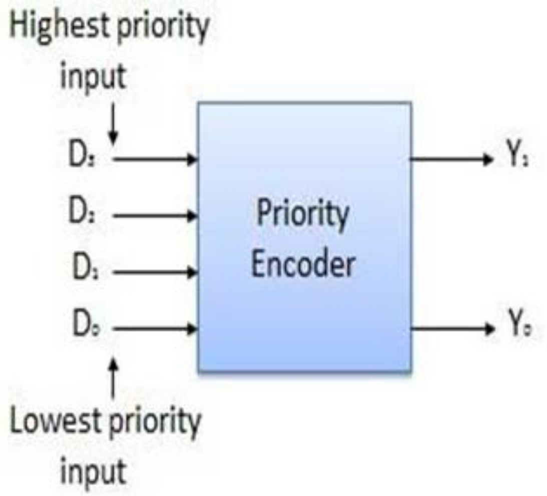
Priority encoder

- Applications: Decimal to binary encoding in calculators
    - Special type of encoders whose output corresponds to the currently active input, which has the highest priority. So, when an input with a higher priority is present, all other input with lower priority will be ignored.



Inputs				Outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	V
0	0	0	0	×	×	0
1	0	0	0	0	0	1
×	1	0	0	0	1	1
×	×	1	0	1	0	1
×	×	×	1	1	1	1





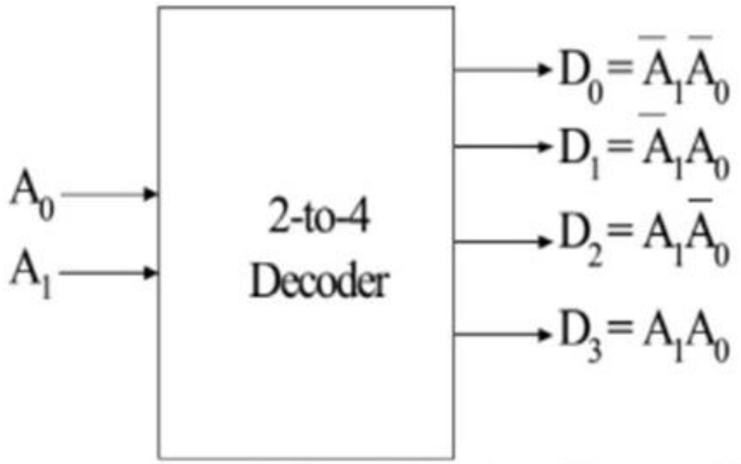
Decoders

mirafra TECHNOLOGIES

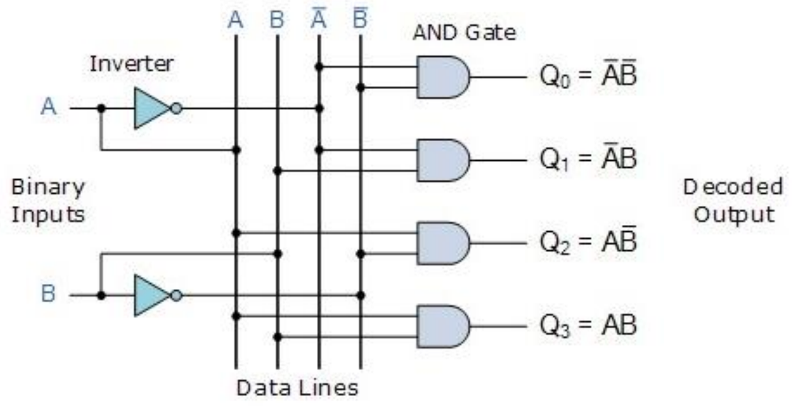
- Information's are mostly binary coded.
- In many designs, we need to derive a number of control signals from a binary coded signal, with one control signal corresponding to each valid code word.
- When the encoded signal takes on a given code value, the corresponding control signal is activated. We call a circuit that derives the control signals in this way a decoder.
- For an n-bit code, if every code word is valid, the decoder will have 2^n outputs.

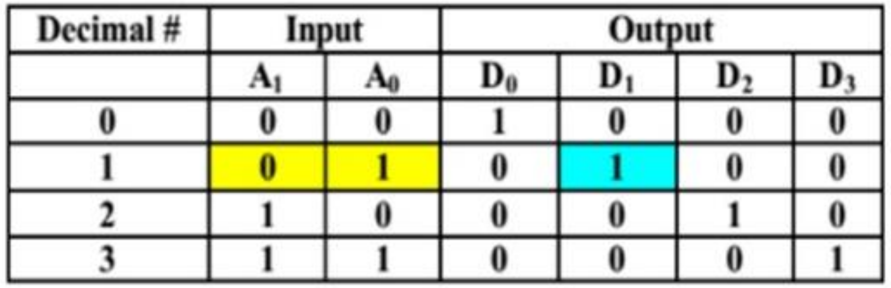
Decoder

- Usage example: Traffic light controller



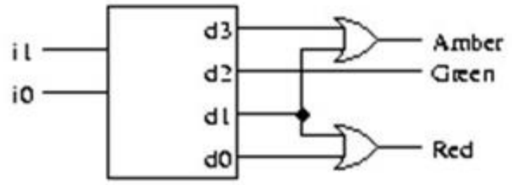
A 2-to-4 decoder without enable





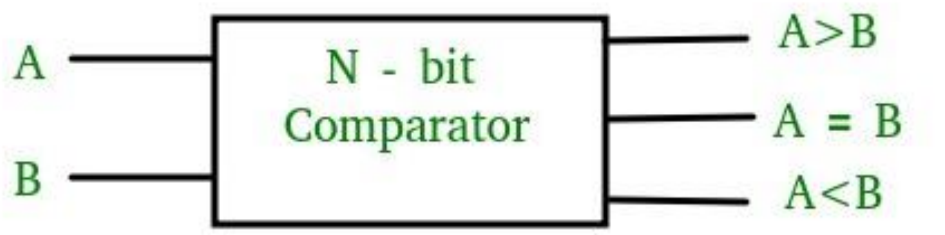
Decimal #	Input		Output			
	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	1	0	0	0	1	0
3	1	1	0	0	0	1

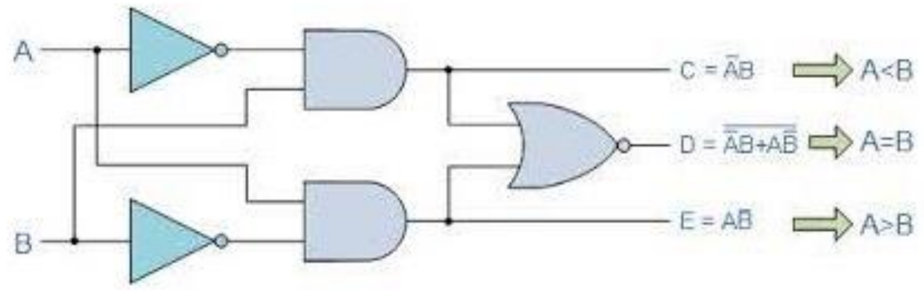
Truth table for 2-to-4 decoder

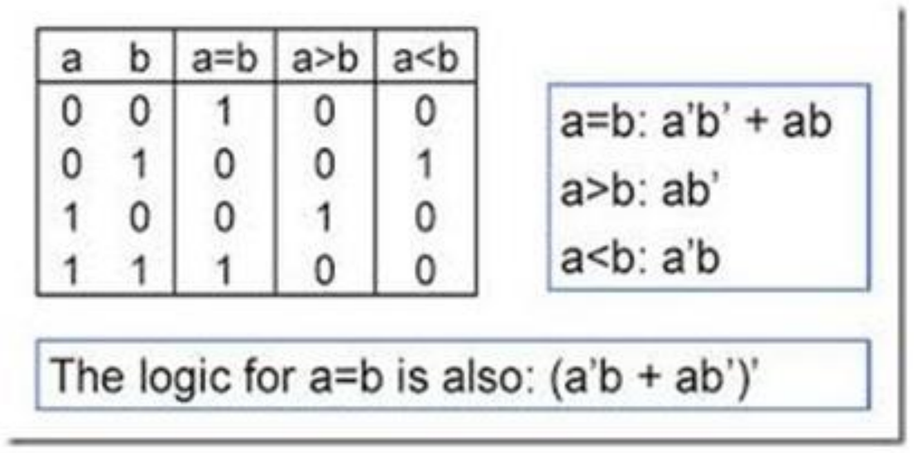


mirafra TECHNOLOGIES

Comparator







a	b	a=b	a>b	a<b
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

a=b: a'b' + ab

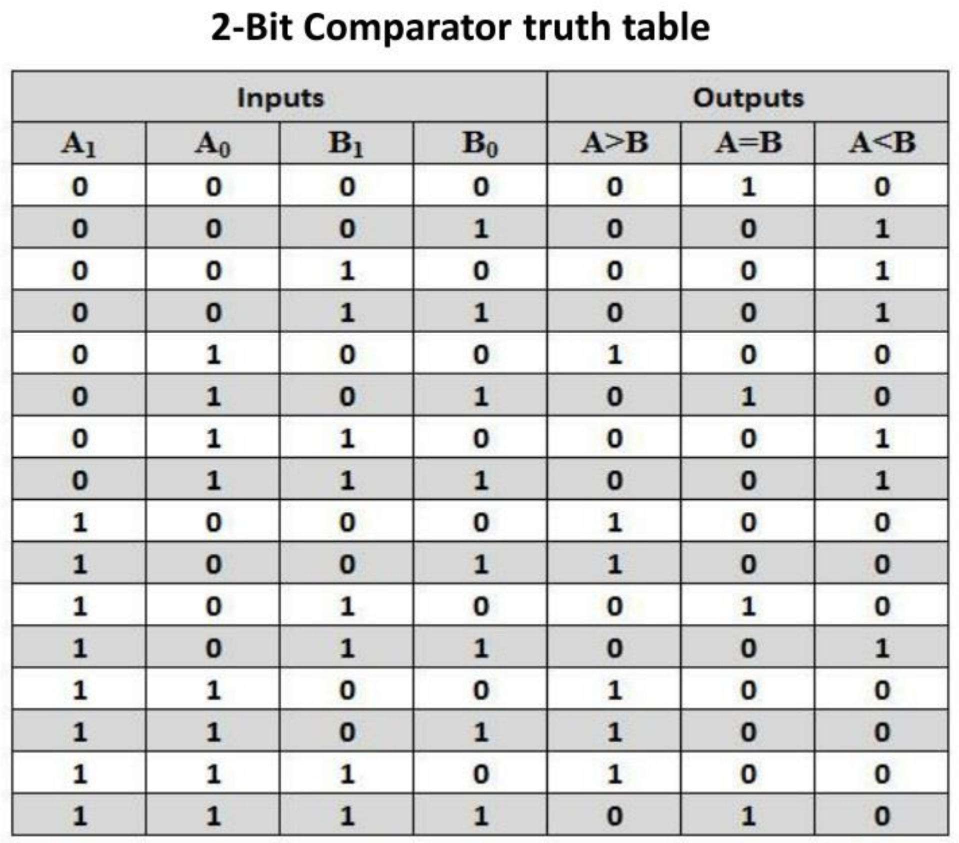
a>b: ab'

a<b: a'b

The logic for a=b is also: (a'b + ab')'

mirafra
TECHNOLOGIES

2 bit comparator

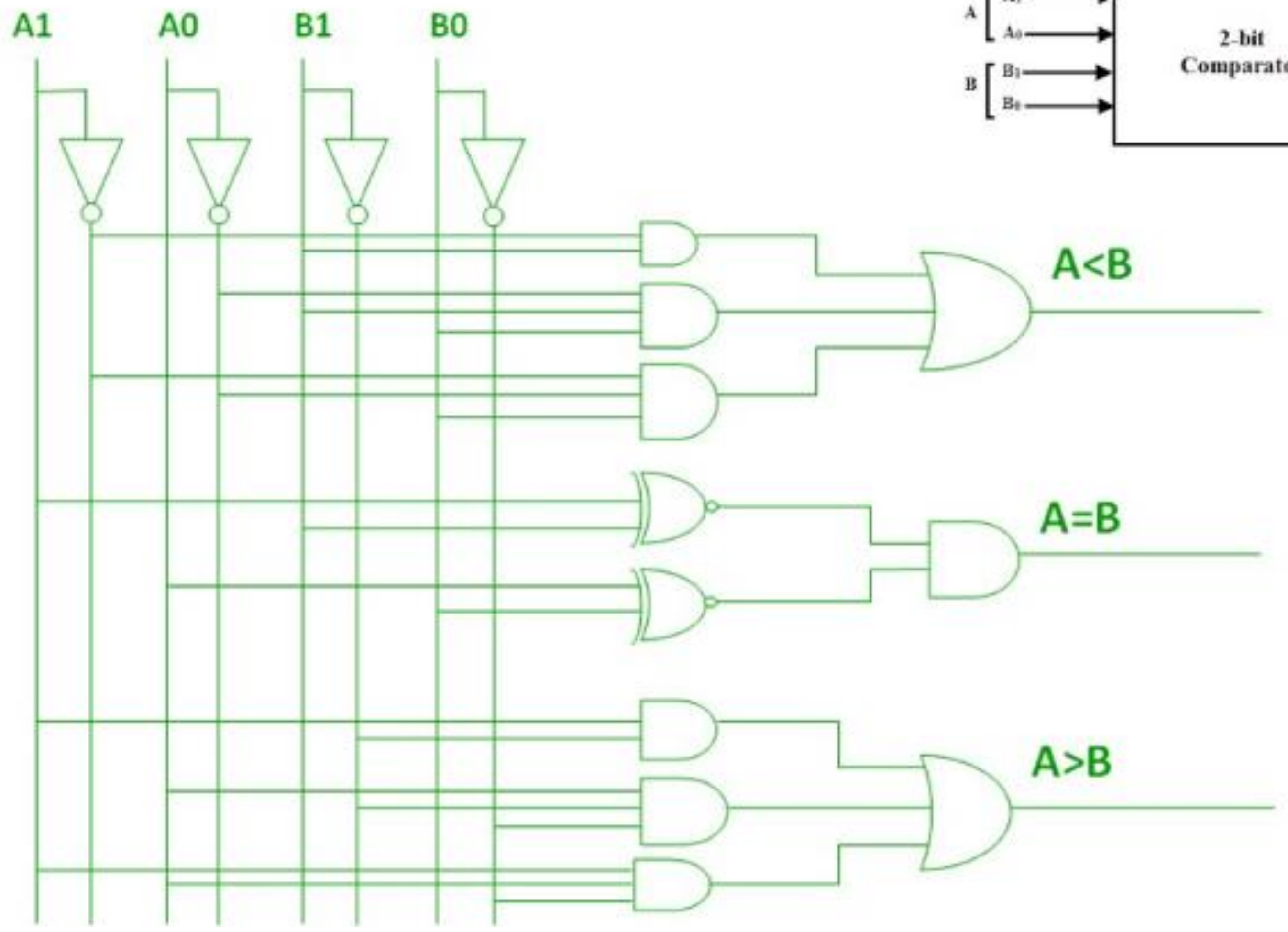


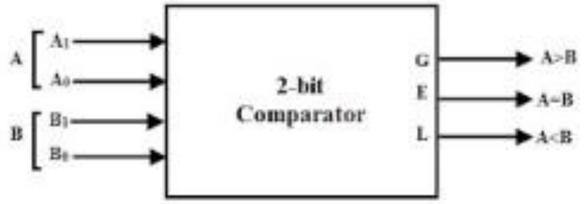
2-Bit Comparator truth table

Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

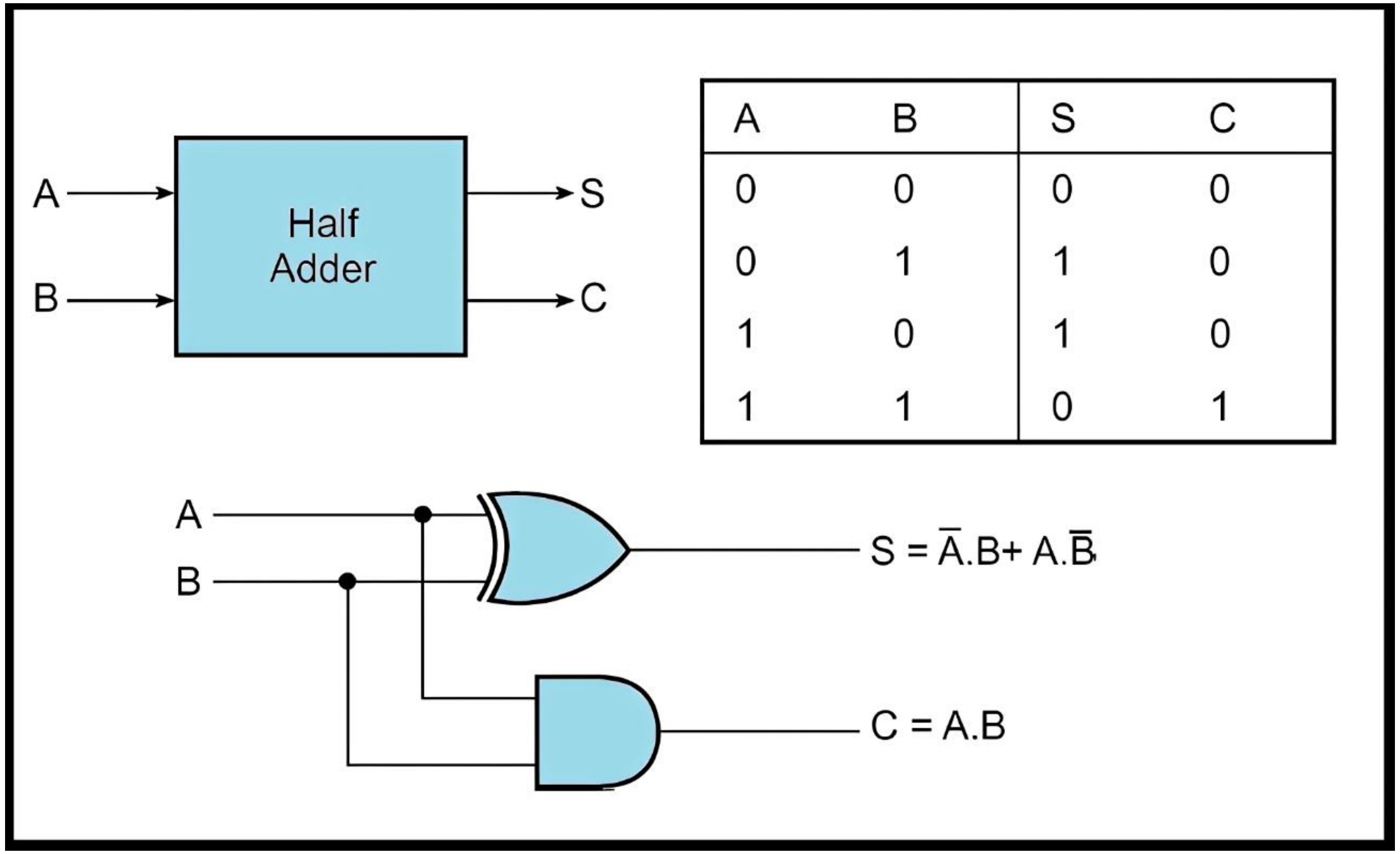
mirafra TECHNOLOGIES

2-bit comparator



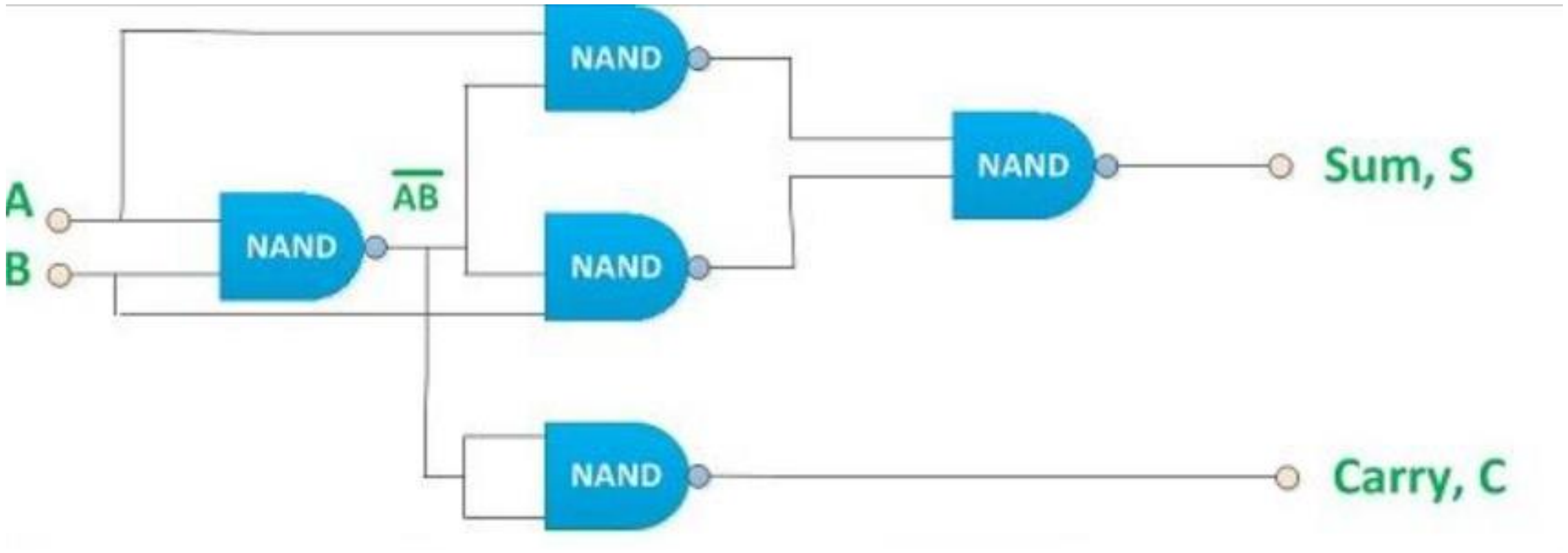


Half adder



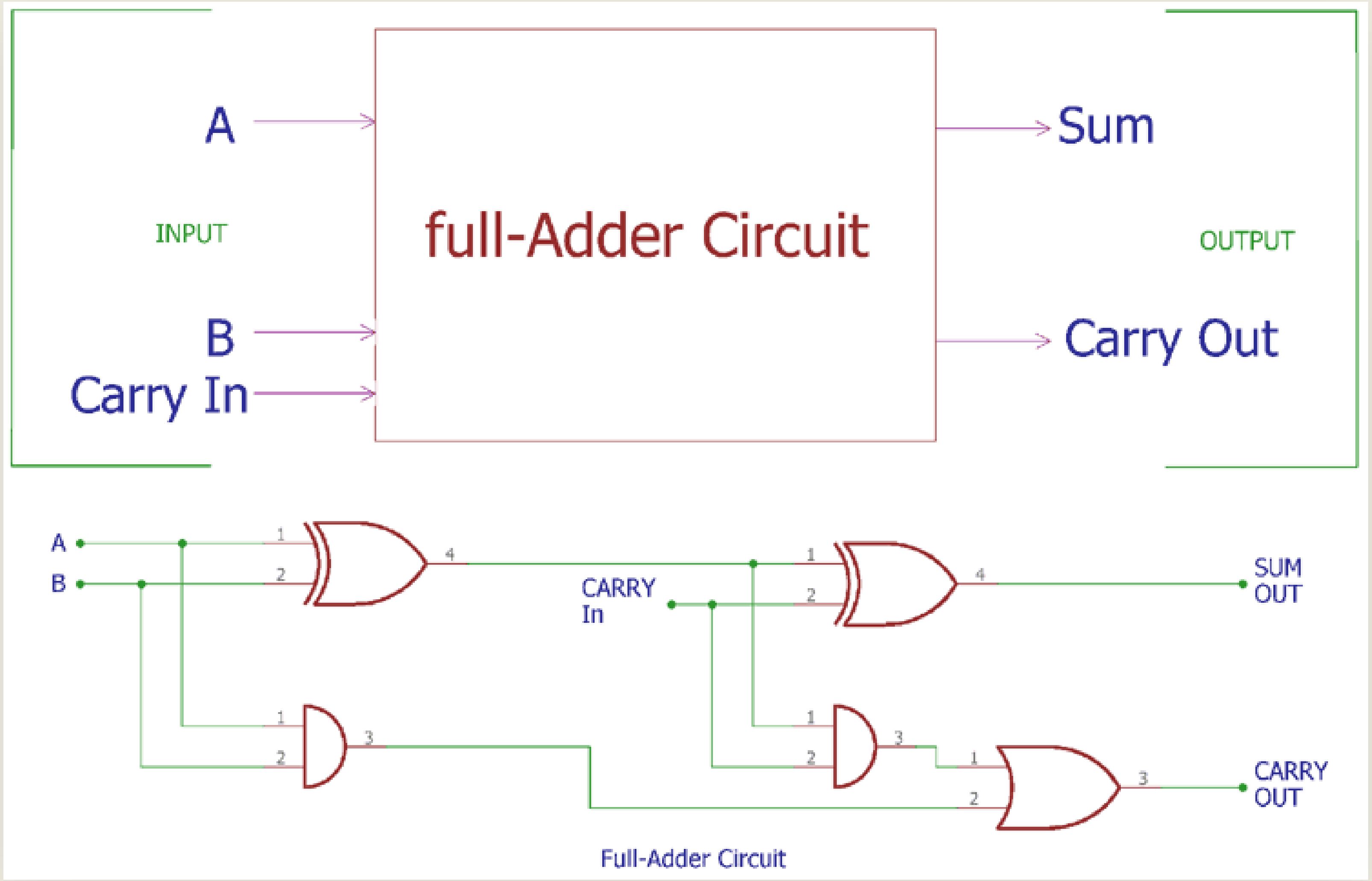
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

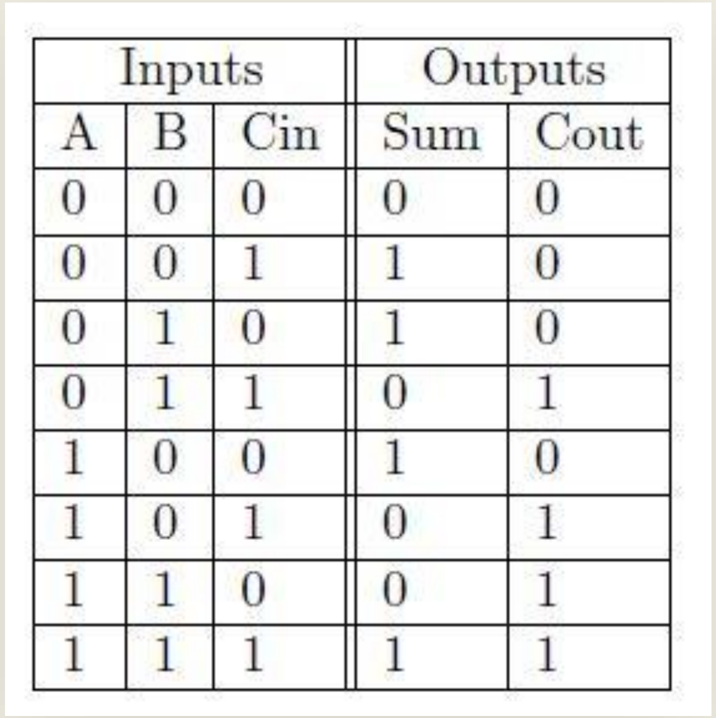
Half adder – using NAND gates



Full adder

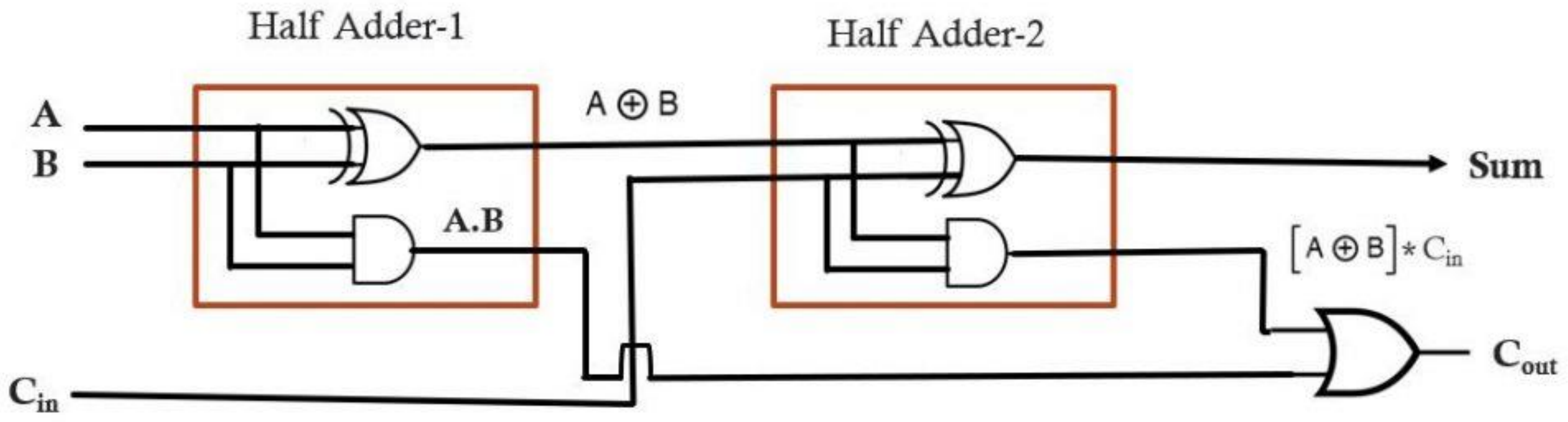
mirafra TECHNOLOGIES



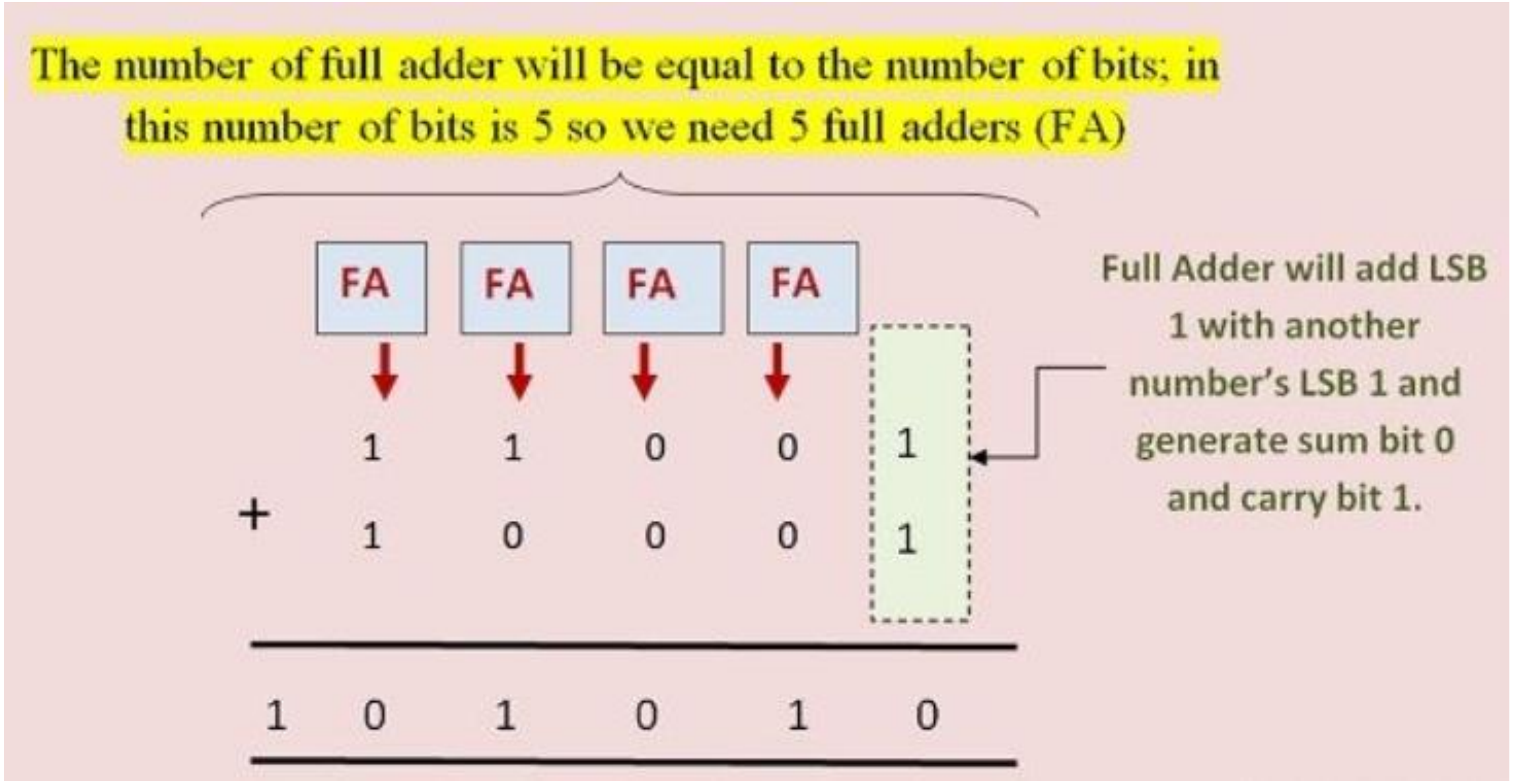


Inputs			Outputs	
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full-adder using Half-adder



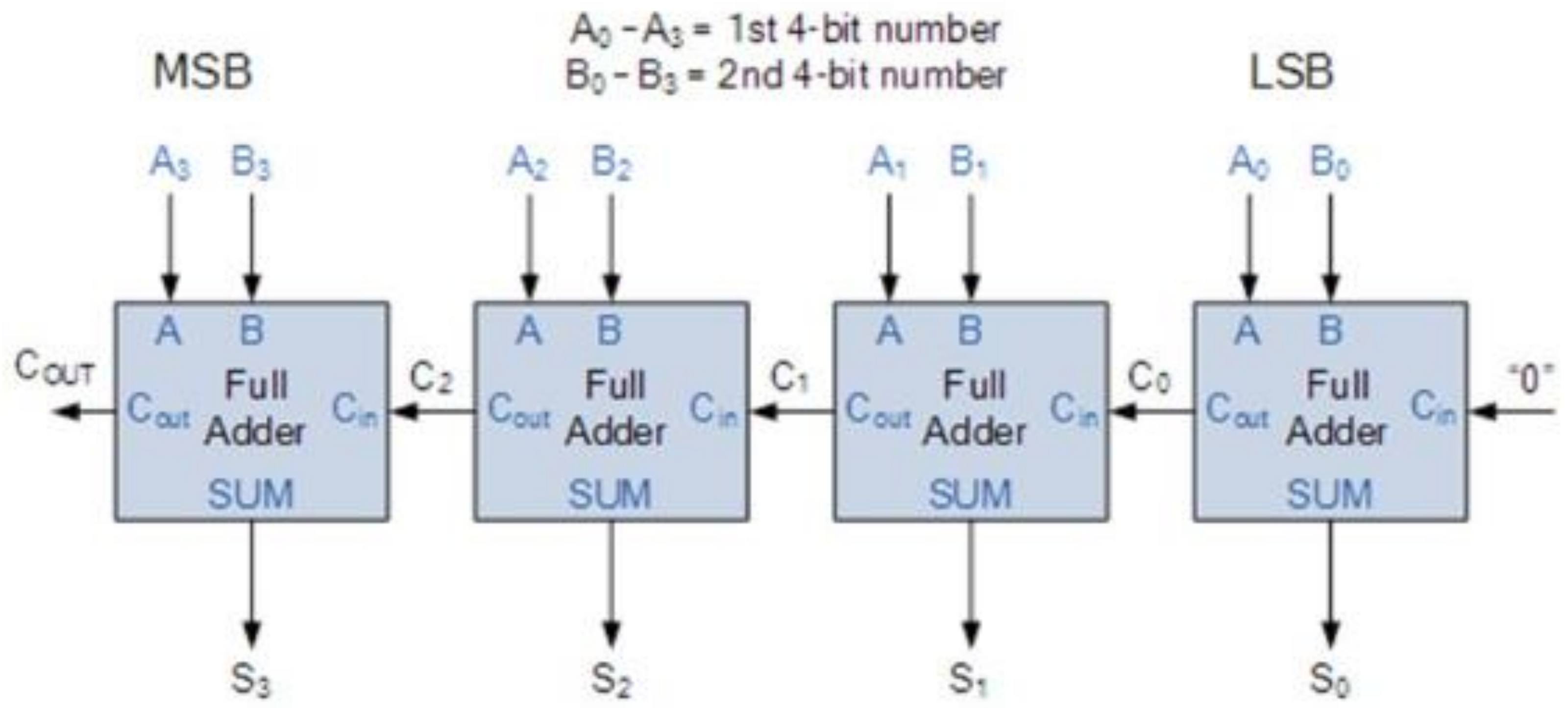
4-bit full-adder using 1-bit full-adder



The number of full adder will be equal to the number of bits; in this number of bits is 5 so we need 5 full adders (FA)

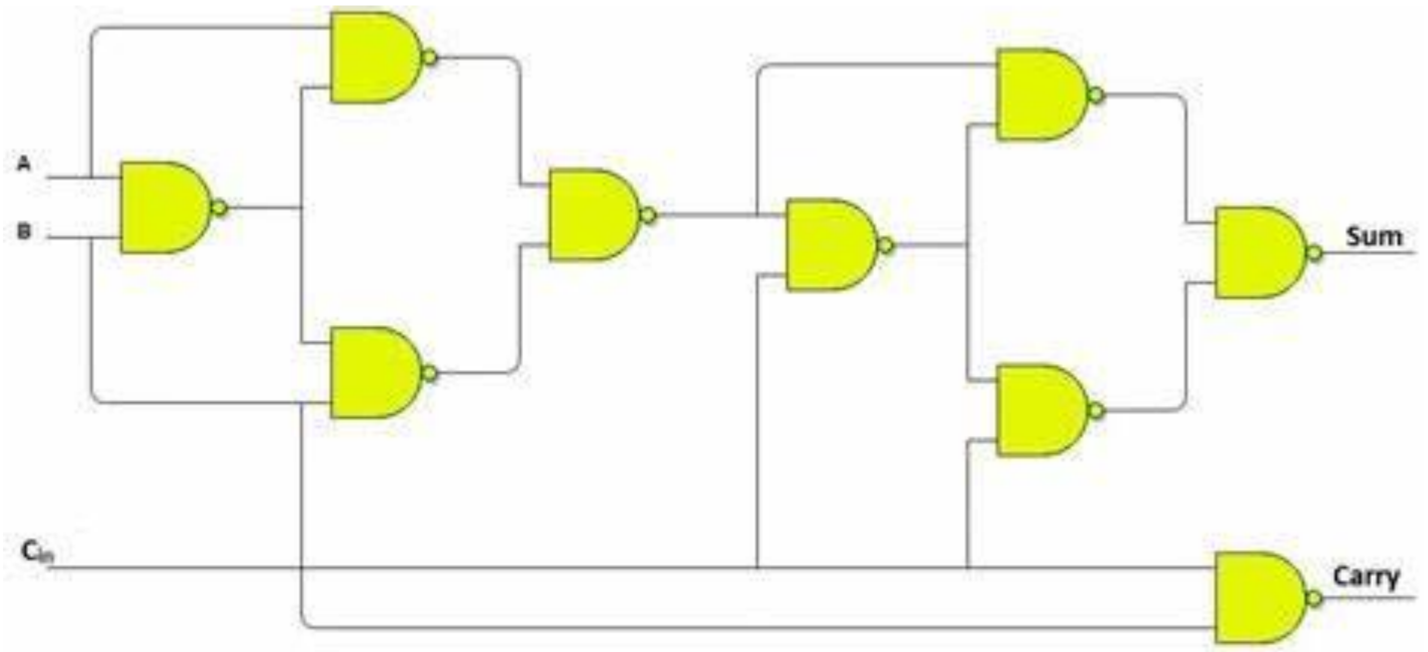
FA
FA
FA
FA
1 1 0 0 1
+ 1 0 0 0 1
1 0 1 0 1 0
Full Adder will add LSB 1 with another number's LSB 1 and generate sum bit 0 and carry bit 1.

4-bit full-adder using 1-bit full-adder



Full-adder using NAND gate

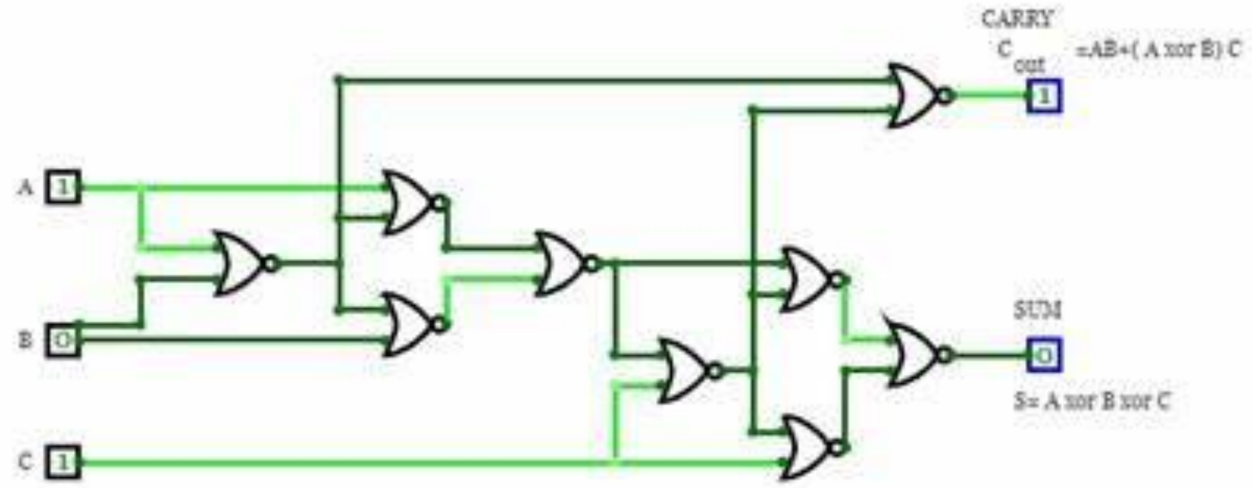
Full-adder using NAND gate



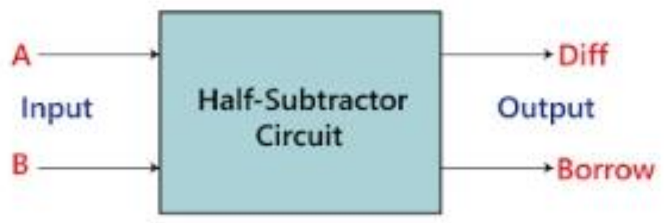
Full-adder using NOR gate

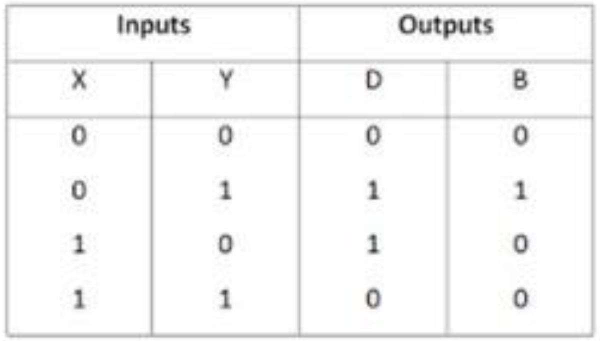
mirafra TECHNOLOGIES

Full-adder using NOR gate

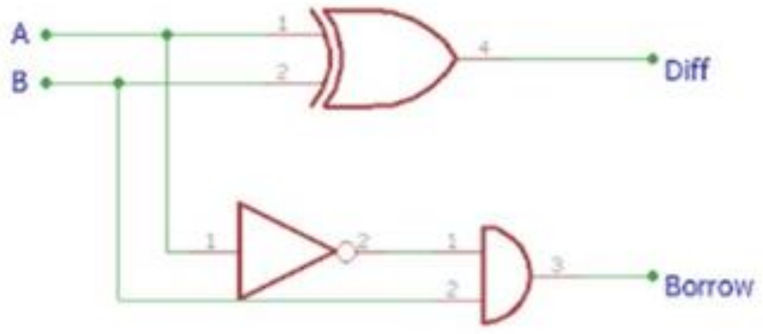


Half-subtractor



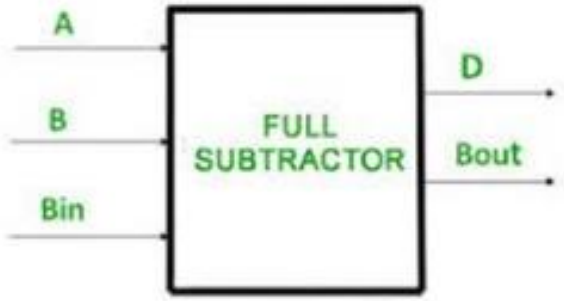


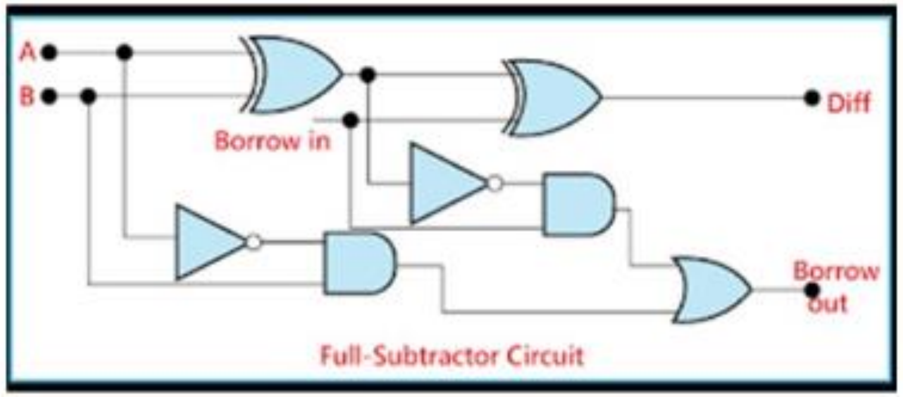
Inputs		Outputs	
X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

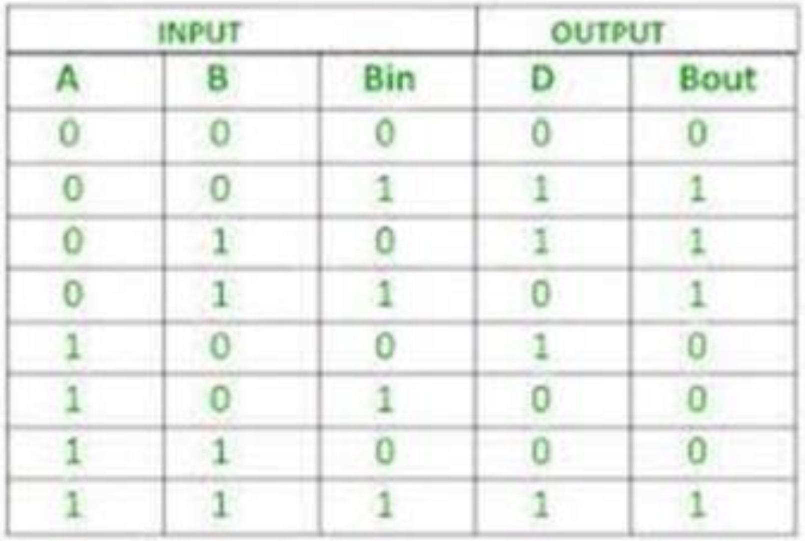


Half-Subtractor Circuit

Full-subtractor

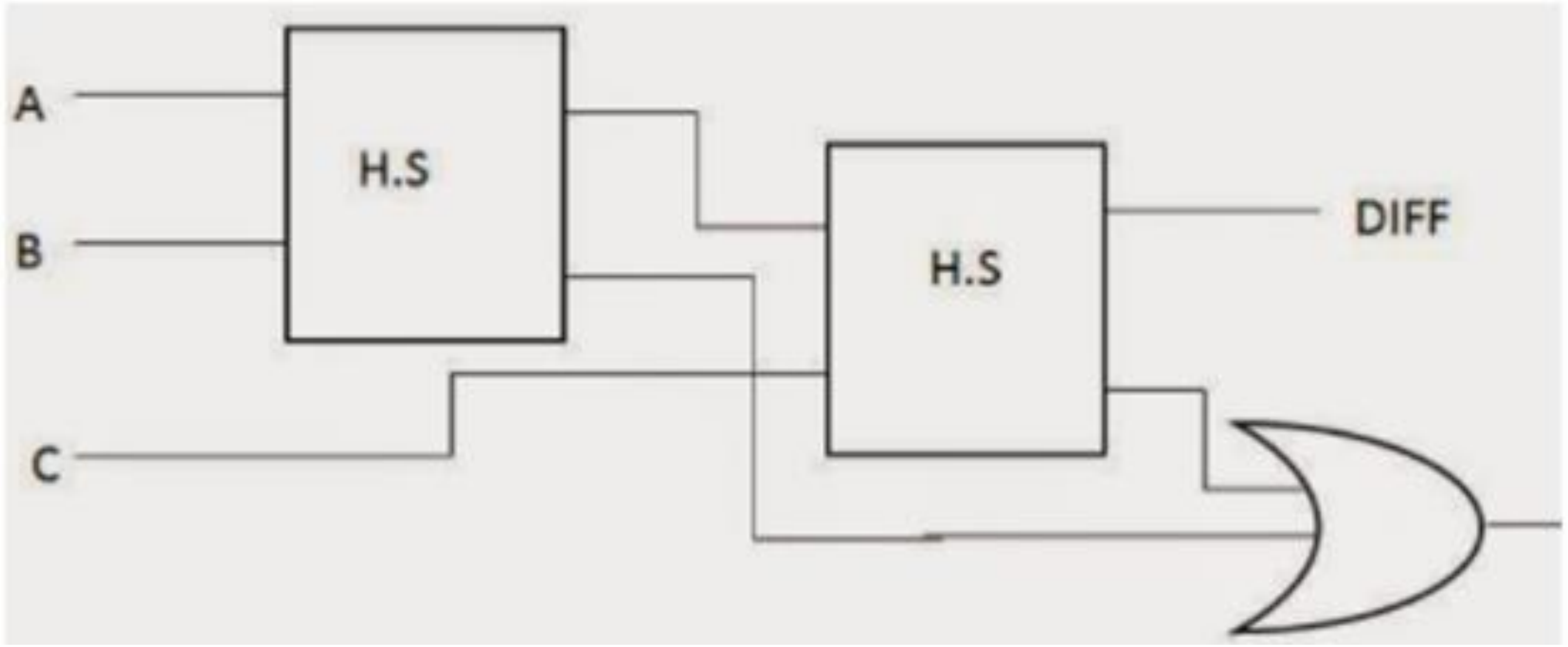




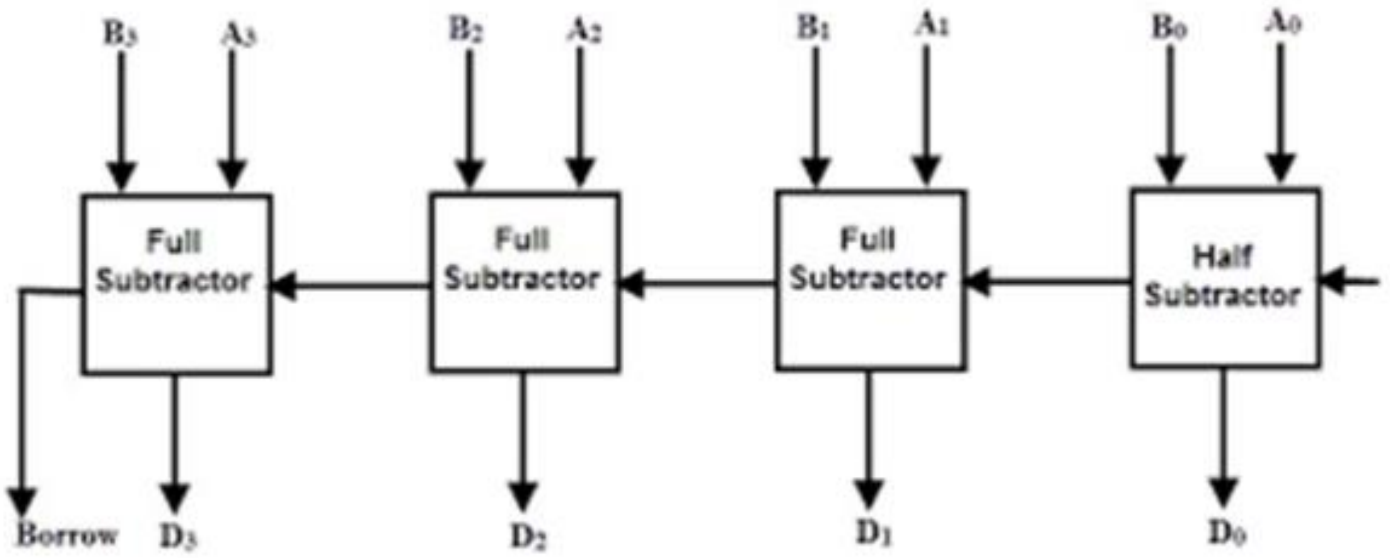


INPUT			OUTPUT	
A	B	Bin	D	Bout
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Full-subtractor using half-subtractor



4-bit full subtractor using 1-bit full subtractor

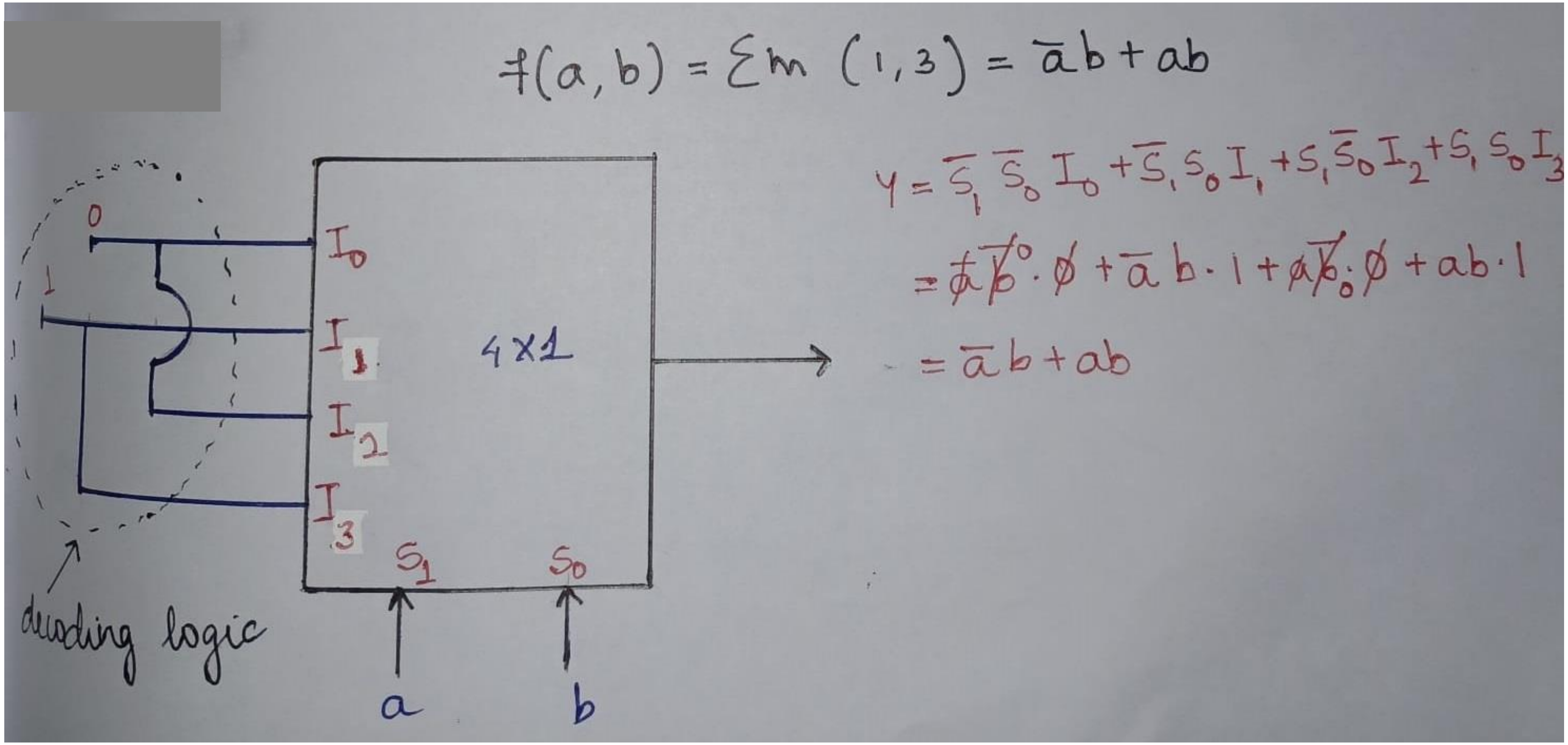


Agenda

- Full-adder implementation using half-adders
- Full-subtractor implementation using half-subtractor
- Full-adder realisation using NAND and NOR gates
- Full-subtractor realisation using NAND and NOR gates
- Implementing Boolean expression using NAND and NOR gates
- MUX and De-MUX

Boolean function implementation using 2x1 Mux

[redacted]



$$f(a,b) = \sum m(1,3) = \bar{a}b + ab$$

$$Y = \bar{S_1}\bar{S_0}I_0 + \bar{S_1}S_0I_1 + S_1\bar{S_0}I_2 + S_1S_0I_3$$

$$= \cancel{\bar{a}}\cancel{\bar{b}}^{0}\cdot\phi + \bar{a}b\cdot 1 + \cancel{a}\cancel{\bar{b}}_{0}\cdot\phi + ab\cdot 1$$

$$= \bar{a}b + ab$$

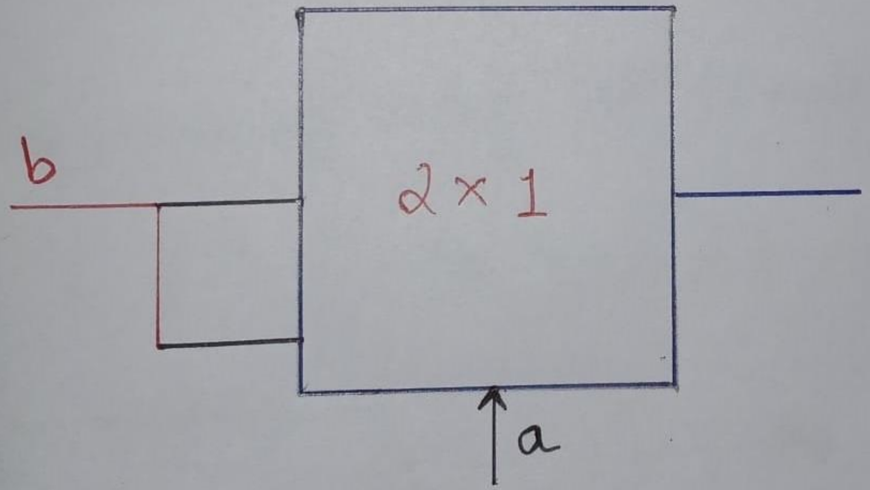
Boolean function implementation using 4x1 Mux

[redacted]

$$f(a,b) = \Sigma m(1,3) = \bar{a}b + ab$$

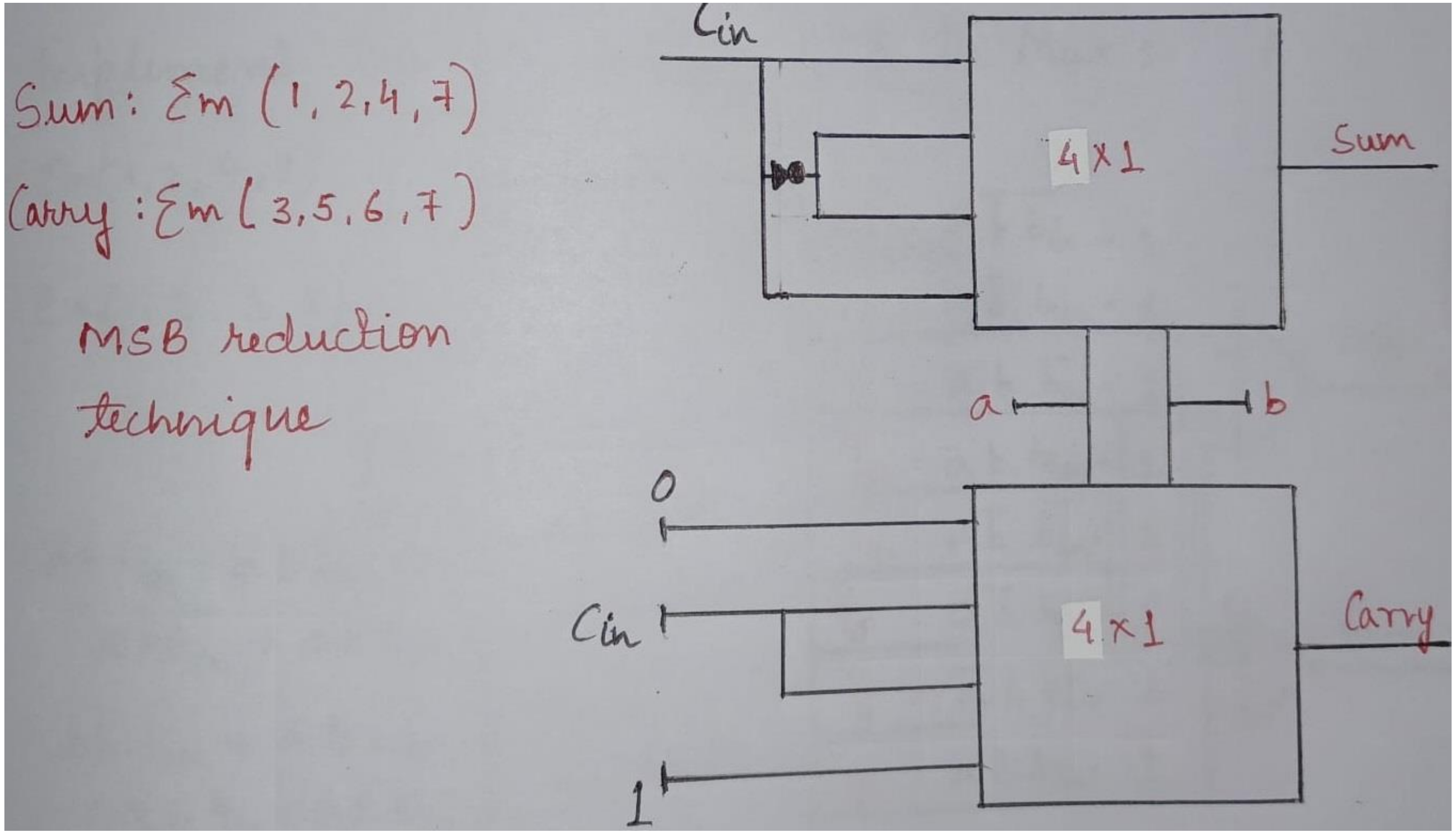
	a	b	y
0-	0	0	0
1-	0	1	1
2-	1	0	0
3-	1	1	1

	I_0	I_1
$\bar{b}$	0	2
b	$\textcircled{1}$	$\textcircled{3}$
	b	b



$$y = \bar{S_0} I_0 + S_0 I_1$$
$$= \bar{a} \cdot b + ab$$

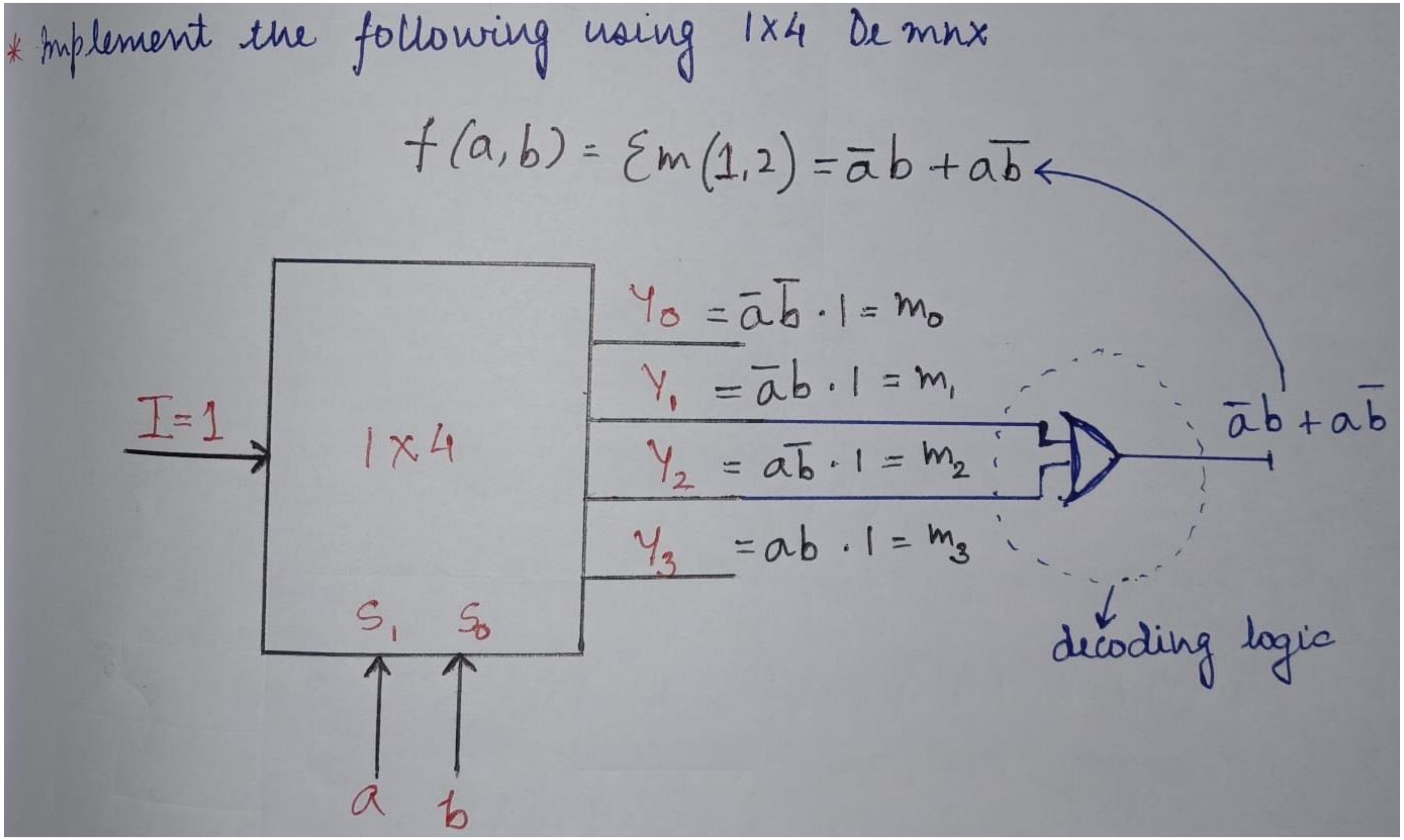
Full-adder using 4x1 Mux


Sum: $\Sigma m(1,2,4,7)$

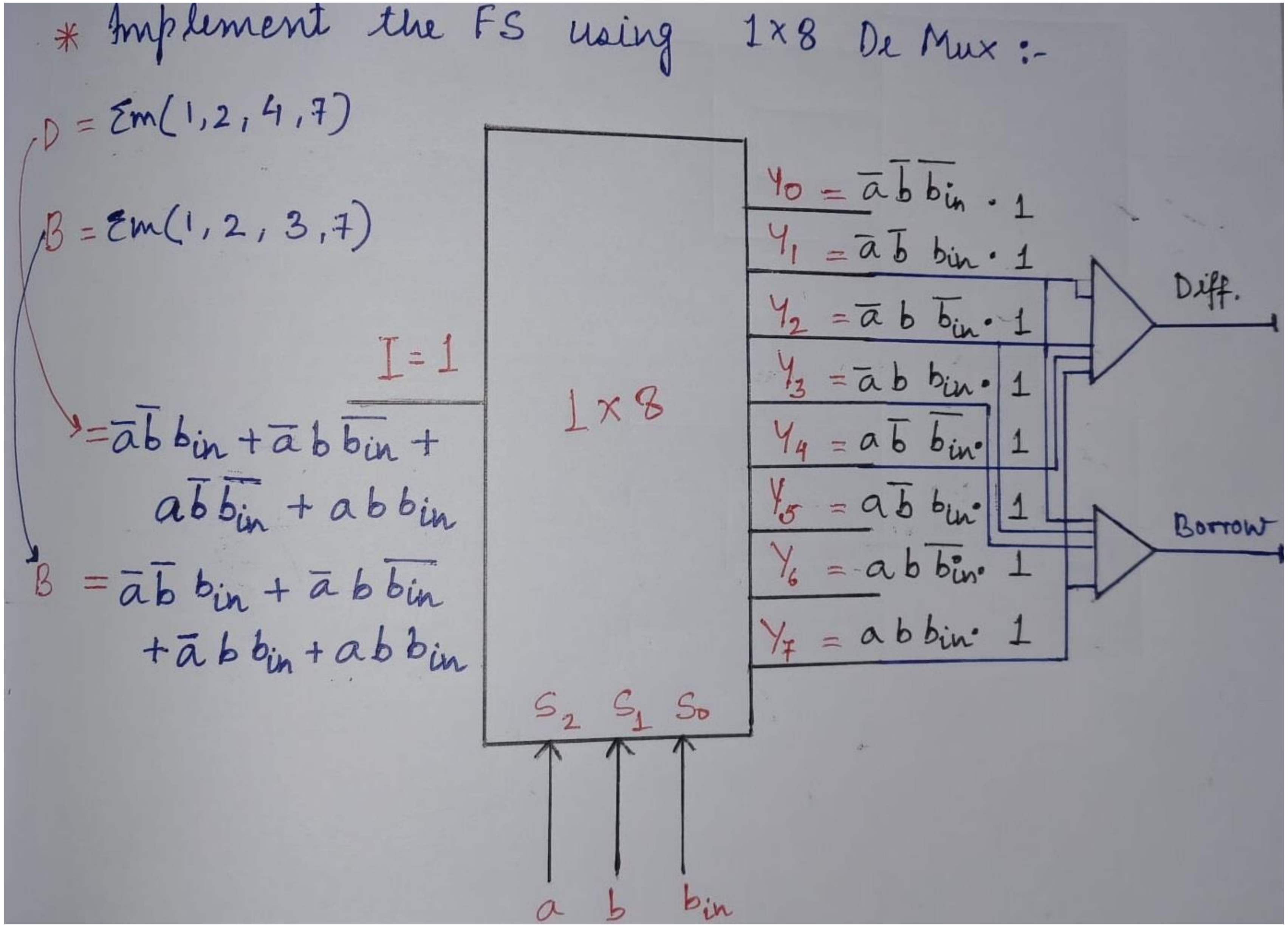
Carry: $\Sigma m(3,5,6,7)$

MSB reduction technique

Representing Boolean expression using a De-mux


* Implement the following using 1x4 De mux
$f(a,b) = \Sigma m(1,2) = \bar{a}b + a\bar{b}$
$Y_0 = \bar{a}\bar{b} \cdot 1 = m_0$
$Y_1 = \bar{a}b \cdot 1 = m_1$
$Y_2 = a\bar{b} \cdot 1 = m_2$
$Y_3 = ab \cdot 1 = m_3$
I=1
1x4
S_1
S_0
a
b
$\bar{a}b + a\bar{b}$
decoding logic

Representing a full-subtractor using a De-mux



* Implement the FS using 1×8 De Mux :-

$D = \Sigma m(1,2,4,7)$

$B = \Sigma m(1,2,3,7)$

$D = \bar{a}\bar{b}b_{in} + \bar{a}b\overline{b_{in}} + a\bar{b}\overline{b_{in}} + abb_{in}$

$B = \bar{a}\bar{b}b_{in} + \bar{a}b\overline{b_{in}} + \bar{a}bb_{in} + abb_{in}$

$Y_0 = \bar{a}\bar{b}\overline{b_{in}} \cdot 1$

$Y_1 = \bar{a}\bar{b}b_{in} \cdot 1$

$Y_2 = \bar{a}b\overline{b_{in}} \cdot 1$

$Y_3 = \bar{a}bb_{in} \cdot 1$

$Y_4 = a\bar{b}\overline{b_{in}} \cdot 1$

$Y_5 = a\bar{b}b_{in} \cdot 1$

$Y_6 = ab\overline{b_{in}} \cdot 1$

$Y_7 = abb_{in} \cdot 1$

Hazards in combinational logic

- Both inputs are given to a circuit at the same time, but they arrive at the final gate input at different times, due to the delays added by the gates in the path.
  - This difference in arrival times, create short duration of unwanted glitch at the gate output.
- Both A and B are changed at the same time
  - A = 1 to 0
  - B = 1 to 0
- How to resolve?
  - Use the paths in such a way that both paths have similar delay
  - This can be done using K-map
  - We won't be able to avoid 100% hazards, but they can be minimized.

mirafra TECHNOLOGIES

Thank you

Heading

- Input fields

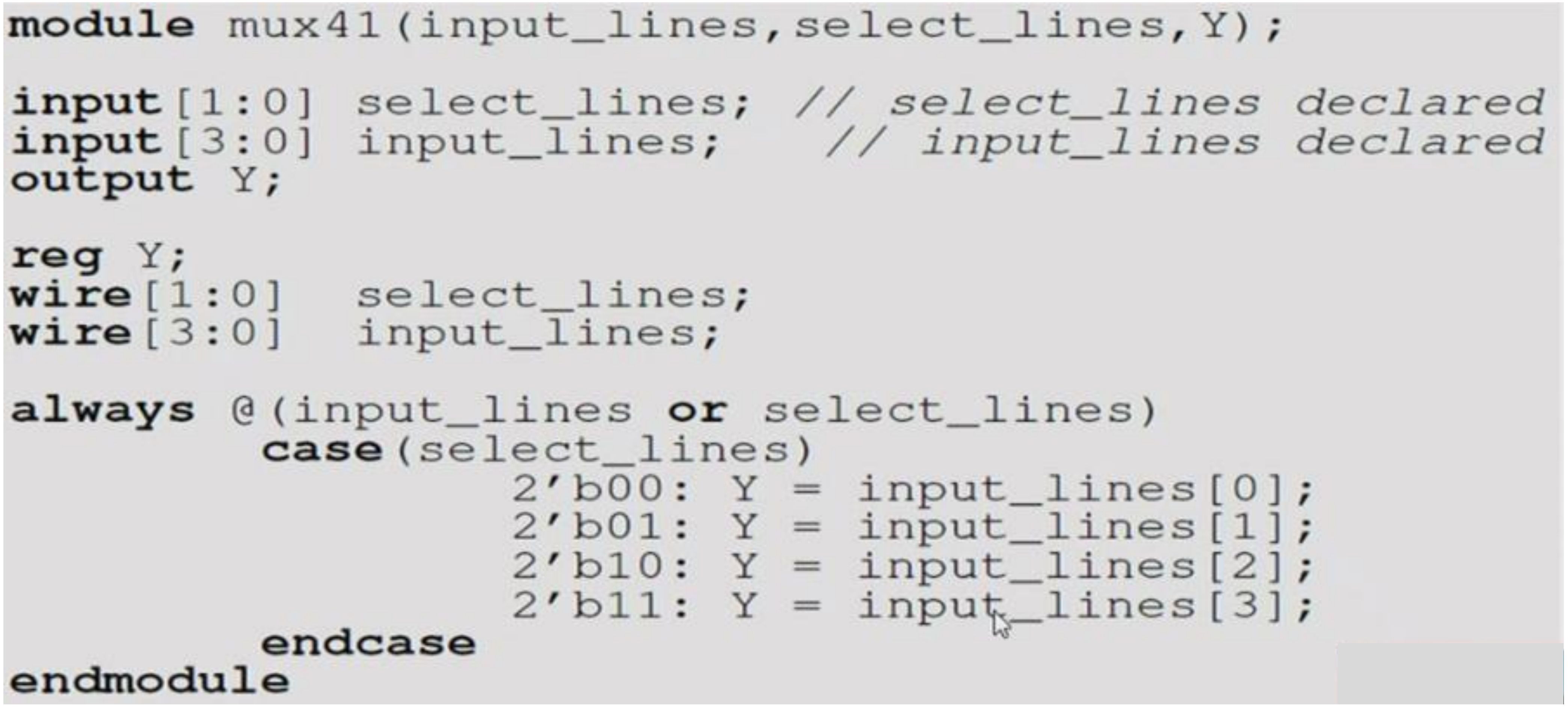
Problems on NAND and NOR

- Draw the CMOS representation of NAND and NOR gate?
- Implement y = AB + CD using NAND gates
- Implement (~X + ~Y) (Z + W)
- Implement (~A + B)C + ~F + DE using NAND and NOR gates
- Implement A~BC using 2-input NAND gates

Problems based on universal gates

- Implement y=AB+CD using NAND gates?
- Implement (~x+~y)(z+w)?
- Implement (~A+B)C+~F+DE using NAND and NOR gates?
- Implement A~BC using 2 input NAND gates?
- Realisation of full-subtractor using NAND and NOR gates.
- F(A,B,C) = (A.B.C)+(A'.B')+(A.B'.C)+(A.C')
  - How to convert this SOP to POS?
- AB + A~C + BC = AB + A~C , optimize this logic?

Multiplexer – Example



```
module mux41(input_lines,select_lines,Y);

input[1:0] select_lines; // select_lines declared
input[3:0] input_lines;   // input_lines declared
output Y;

reg Y;
wire[1:0]  select_lines;
wire[3:0]  input_lines;

always @(input_lines or select_lines)
       case(select_lines)
              2'b00: Y = input_lines[0];
              2'b01: Y = input_lines[1];
              2'b10: Y = input_lines[2];
              2'b11: Y = input_lines[3];
       endcase
endmodule
```

Combinational logic-based questions

- POS → SOP conversion and vice-versa
  - Minterms, maxterms => sigma, pie
- POS realisation using Mux
- Decoder realisation using Mux
- 1-bit and 2-bit comparator using Mux
- Full-adder and Half-adder using decoder
- 16x4 encoder realisation using smaller encoders
- Mux implementation using tri-state buffers
- Bigger decoder using smaller decoder
- Bigger Mux using smaller Mux
- Bigger demux using smaller de-mux
- Bigger comparator using smaller comparator
- Bigger full-adder using smaller full-adder

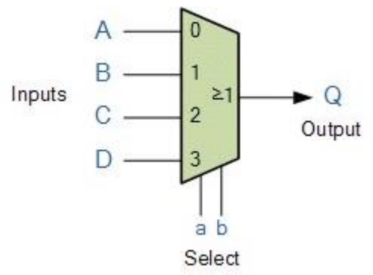
Problems based on Mux and De-Mux

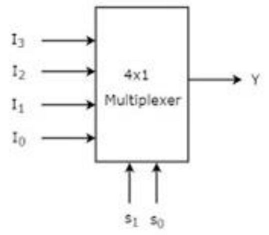
- Implement 4x1 Mux using 2x1 Mux
- Implement 8x1 Mux using 2x1 Mux
- Implement 8x1 Mux using 4x1 Mux
- Implement 5x1 Mux using 2x1 Mux
- Implement full-adder using 4x1 Mux
- Implement 16x1 Mux using 4x1 and 2x1 Mux
- Implement 32x1 Mux using 8x1 and 4x1 Mux
- Implement full-subtractor using 4x1 Mux
- Implement full-adder using 2x1 Mux only

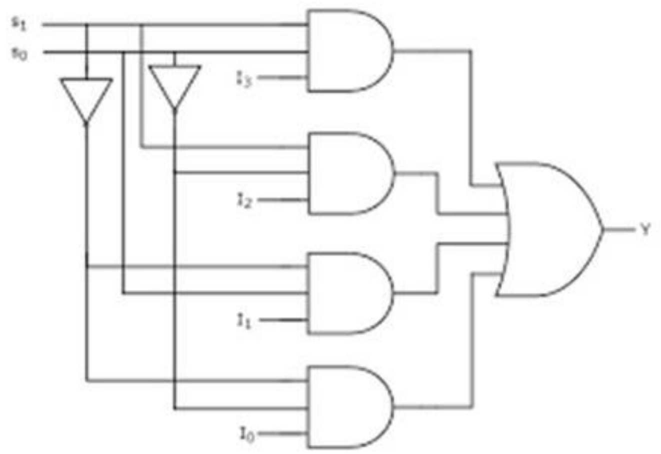
Combinational logic-based questions

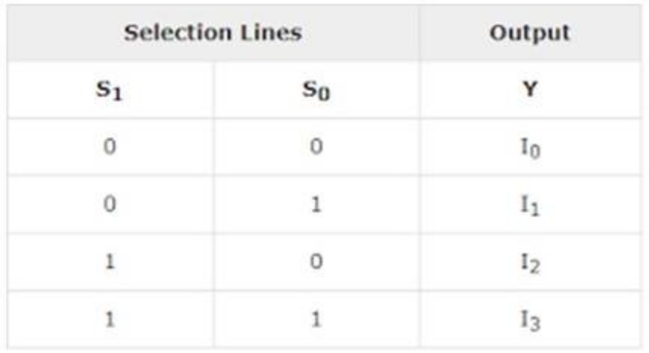
- Bigger full-subtractor using smaller full-subtractor
- Min terms realisation using 2:4 decoder
- Max-term realisation using 2:4 decoder
- Implement full-adder using de-mux
- Implement full-subtractor using de-mux
- Implement full-adder and half-subtractor using de-mux

Multiplexer









Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

- Used to select one among multiple inputs
- Generally used: 2x1, 2x1, 8x1
- Other types also possible: 3x1

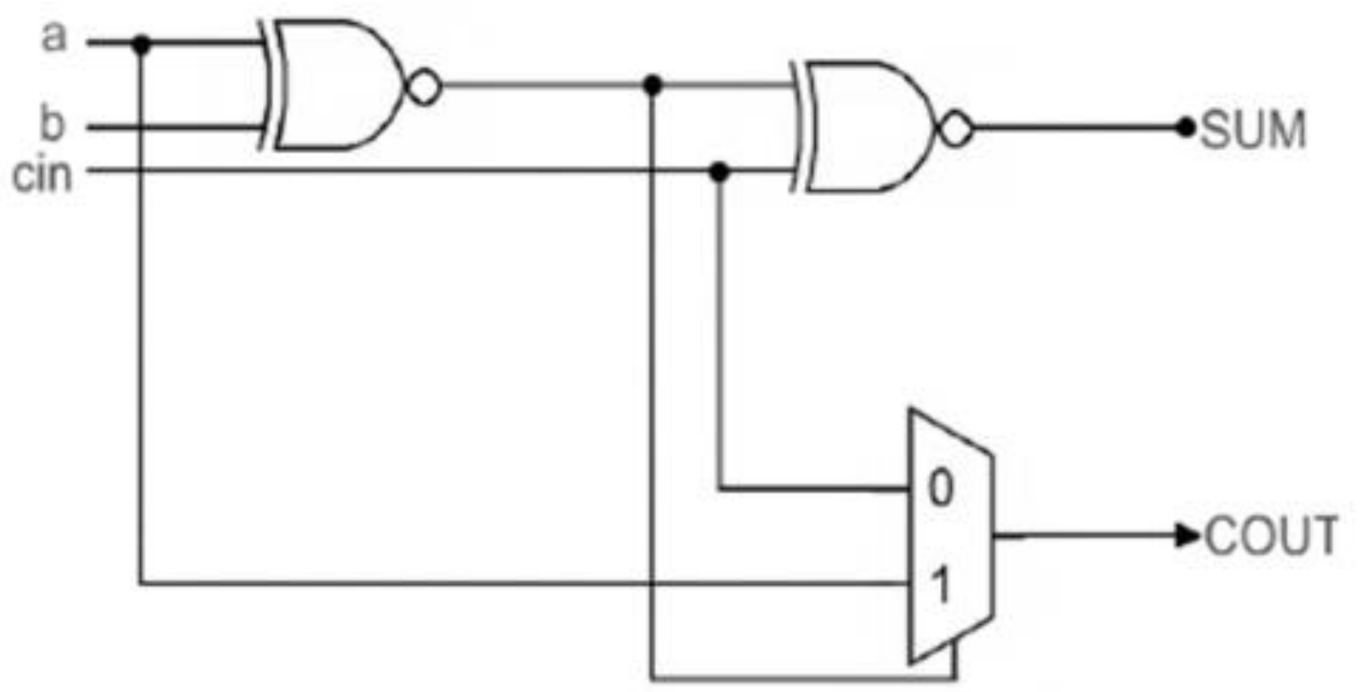
Multi-bit Mux Example

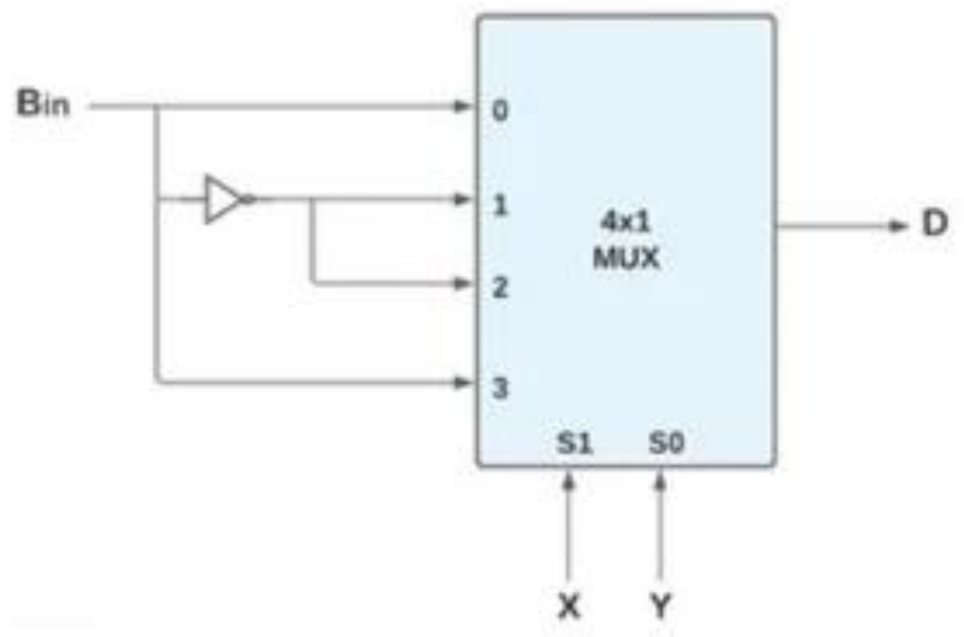
mirafra TECHNOLOGIES

```
module multiplexer_3bit_2_to_1 ( output [2:0] z,
                                 input  [2:0] a0, a1,
                                 input        sel );

  assign z = sel ? a1 : a0;

endmodule
```

mirafra
TECHNOLOGIES

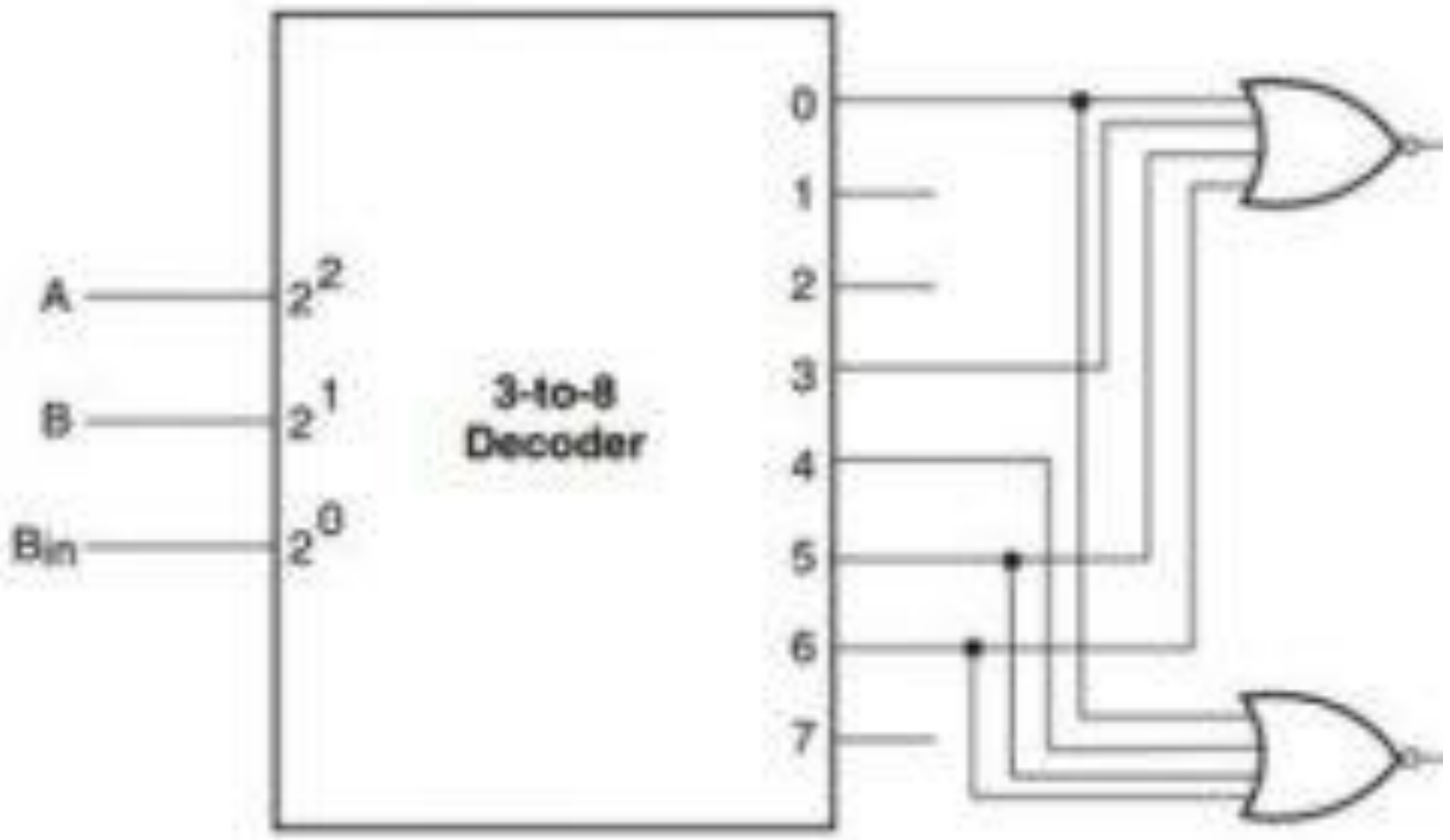
Full-adder using 2x1 Mux and XOR





Full-subtractor using multiplexers

mirafra
TECHNOLOGIES



Designing Encoders

mirafra TECHNOLOGIES

- We can derive the Boolean equation for each bit of the output by identifying those inputs for which the output bit is 1.
- The output bit is then the logical OR of those inputs.

Example:

- Design an encoder for use in a domestic burglar alarm that has sensors for each of eight zones. Each sensor signal is 1 when an intrusion is detected in that zone, and 0 otherwise. The encoder has three bits of output, encoding the zone as follows:

Zone 1: 000	Zone 2: 001	Zone 3: 010	Zone 4: 011
Zone 5: 100	Zone 6: 101	Zone 7: 110	Zone 8: 111

Designing Encoders – Example

Solution:

- The left-most bit of the output code is 1 when any of the zone 5 through zone 8 inputs is 1, so the equation for that output is the logical OR of those zone inputs. The equations for the other two output code bits are derived similarly.
- The valid output is the logical OR of all of the zone inputs.

Designing Encoders – Example

- Since all code words are used, we need a separate output to indicate when there is a valid code-word output.

```
module alarm_eqn ( output [2:0] intruder_zone,
                   output       valid,
                   input  [1:8] zone );

  assign intruder_zone[2] = zone[5] | zone[6] |
                            zone[7] | zone[8];
  assign intruder_zone[1] = zone[3] | zone[4] |
                            zone[7] | zone[8];
  assign intruder_zone[0] = zone[2] | zone[4] |
                            zone[6] | zone[8];

  assign valid = zone[1] | zone[2] | zone[3] | zone[4] |
                 zone[5] | zone[6] | zone[7] | zone[8];

endmodule
```

mirafra TECHNOLOGIES

Designing Priority Encoders – Example

- The port list is unchanged, since we need the same inputs and outputs for the encoder. The truth table for the priority encoder is,

zone								intruder_zone			valid
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(2)	(1)	(0)	
1	–	–	–	–	–	–	–	0	0	0	1
0	1	–	–	–	–	–	–	0	0	1	1
0	0	1	–	–	–	–	–	0	1	0	1
0	0	0	1	–	–	–	–	0	1	1	1
0	0	0	0	1	–	–	–	1	0	0	1
0	0	0	0	0	1	–	–	1	0	1	1
0	0	0	0	0	0	1	–	1	1	0	1
0	0	0	0	0	0	0	1	1	1	1	1
0	0	0	0	0	0	0	0	–	–	–	0

Designing Priority Encoders – Example

mirafra
TECHNOLOGIES

- Using conditional assignment:

```
module alarm_priority_1 ( output [2:0] intruder_zone,
                          output       valid,
                          input  [1:8] zone );

  assign intruder_zone = zone[1] ? 3'b000 :
                         zone[2] ? 3'b001 :
                         zone[3] ? 3'b010 :
                         zone[4] ? 3'b011 :
                         zone[5] ? 3'b100 :
                         zone[6] ? 3'b101 :
                         zone[7] ? 3'b110 :
                         zone[8] ? 3'b111 :
                         3'b000;

  assign valid = zone[1] | zone[2] | zone[3] | zone[4] |
                 zone[5] | zone[6] | zone[7] | zone[8];

endmodule
```

Designing Decoders

- Designing means deriving the Boolean equation for each output of a decoder by looking at the corresponding code word.
- Example: suppose we have an encoded 4-bit input signal (a_3 , a_2 , a_1 , a_0), and we need to determine the Boolean equation for the output corresponding to the code word 1011.
- The output is 1 only when,

a_3=1, a_2=0, a_1=1 and a_0=1

Thus, the output is the value of the expression = $a_3 \cdot \overline{a_2} \cdot a_1 \cdot a_0$

Designing Decoders – Example

mirafra TECHNOLOGIES

- Many ink-jet printers have six cartridges for different coloured ink: black, cyan, magenta, yellow, light cyan and light magenta. A multibit signal in such a printer indicates selection of one of the colours. Develop a Verilog model for a decoder for use in this inkjet printer.

- Soln: The decoder has three input bits representing the choice of colour cartridge and six output bits, one to select each cartridge.

Designing Decoders – Example

mirafra TECHNOLOGIES

- Many ink-jet printers have six cartridges for different coloured ink: black, cyan, magenta, yellow, light cyan and light magenta. A multibit signal in such a printer indicates selection of one of the colours. Develop a Verilog model for a decoder for use in this inkjet printer. The decoder has three input bits representing the choice of colour cartridge and six output bits, one to select each cartridge.

Solution:

Color	Codeword (c_2, c_1, c_0)
black	0, 0, 1
cyan	0, 1, 0
magenta	0, 1, 1
yellow	1, 0, 0
red	1, 0, 1
blue	1, 1, 0

Designing Decoders – Example

mirafra
TECHNOLOGIES

```
module ink_jet_decoder
  ( output black, cyan, magenta, yellow,
           light_cyan, light_magenta,
    input  color2, color1, color0 );

  assign black         = ~color2 & ~color1 &  color0;
  assign cyan          = ~color2 &  color1 & ~color0;
  assign magenta       = ~color2 &  color1 &  color0;
  assign yellow        =  color2 & ~color1 & ~color0;
  assign light_cyan    =  color2 & ~color1 &  color0;
  assign light_magenta =  color2 &  color1 & ~color0;

endmodule
```