CASPER

CS230: Digital Logic Design and Computer Architecture

Lecture-3: Combinational Circuits

https://www.cse.iitb.ac.in/~biswa/courses/CS230/main.html

Phones (smart/non-smart) on silence plz, Thanks

Contd.

1. **Minterm to Maxterm conversion:**
   rewrite minterm shorthand using maxterm shorthand
   replace minterm indices with the indices not already used
   E.g., $\mathbf{F}(A, B, C) = \sum m(3, 4, 5, 6, 7) = \prod M(0, 1, 2)$

2. **Maxterm to Minterm conversion:**
   rewrite maxterm shorthand using minterm shorthand
   replace maxterm indices with the indices not already used
   E.g., $\mathbf{F}(A, B, C) = \prod M(0, 1, 2) = \sum m(3, 4, 5, 6, 7)$

3. **Expansion of** $\mathbf{F}$ **to expansion of** $\bar{F}$ **:**

$$\text{E.g., } \mathbf{F}(A, B, C) = \sum m(3, 4, 5, 6, 7) \longrightarrow \bar{F}(A, B, C) = \sum m(0, 1, 2)$$

$$= \prod M(0, 1, 2) \longrightarrow = \prod M(3, 4, 5, 6, 7)$$

4. **Minterm expansion of** $\mathbf{F}$ **to Maxterm expansion of** $\bar{F}$ **:**
   rewrite in Maxterm form, using the same indices as $\mathbf{F}$

$$\text{E.g., } \mathbf{F}(A, B, C) = \sum m(3, 4, 5, 6, 7) \longrightarrow \bar{F}(A, B, C) = \prod M(3, 4, 5, 6, 7)$$

$$= \prod M(0, 1, 2) \longrightarrow = \sum m(0, 1, 2)$$

K-Maps

- Karnaugh Map (K-map) method
  - K-map is an alternative method of representing the truth table that helps visualize adjacencies in up to 6 dimensions
  - Physical adjacency ↔ Logical adjacency

2-variable K-map

A \ B	0	1
0	00	01
1	10	11

3-variable K-map

A \ BC	00	01	11	10
0	000	001	011	010
1	100	101	111	110

4-variable K-map

AB \ CD	00	01	11	10
00	0000	0001	0011	0010
01	0100	0101	0111	0110
11	1100	1101	1111	1110
10	1000	1001	1011	1010

Coffee points

Why 11 before 10 ?

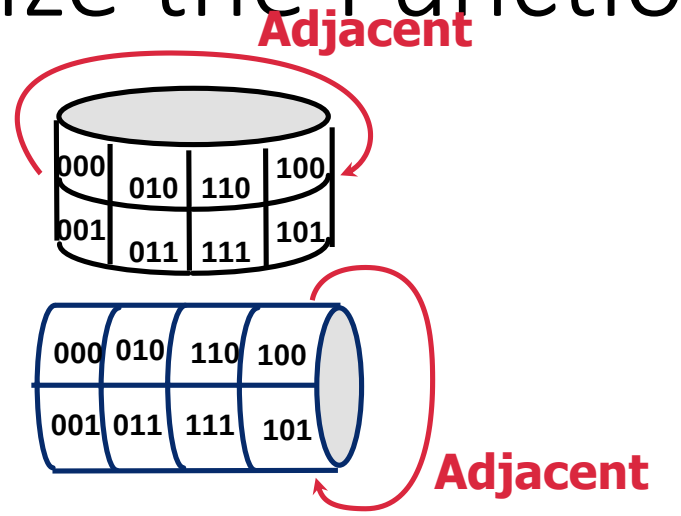
Coffee points

Why 11 before 10 ?

Numbering Scheme: **00, 01, 11, 10 is called a "Gray Code" — only a *single bit (variable) changes* from one code word and the next code word**

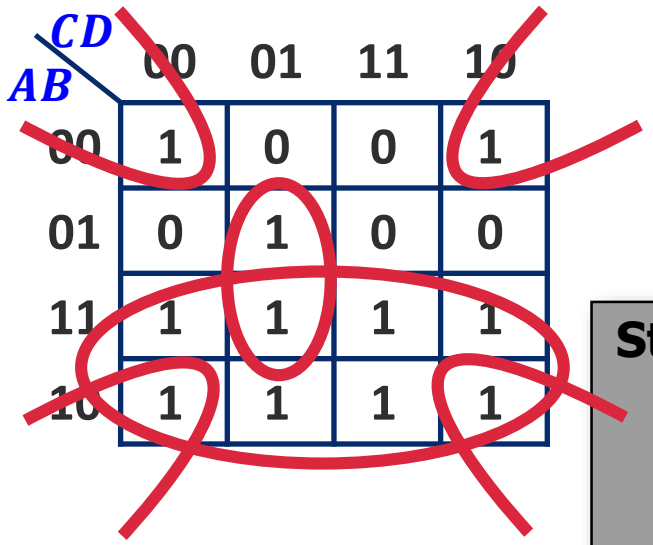
How? To minimize the Function

A \ BC	00	01	11	10
0	000	001	011	010
1	100	101	111	110



K-map adjacencies go "around the edges"
Wrap around from first to last column
Wrap around from top row to bottom row

How?



$$F(A,B,C,D) = \sum m(0,2,5,8,9,10,11,12,13,14,15)$$

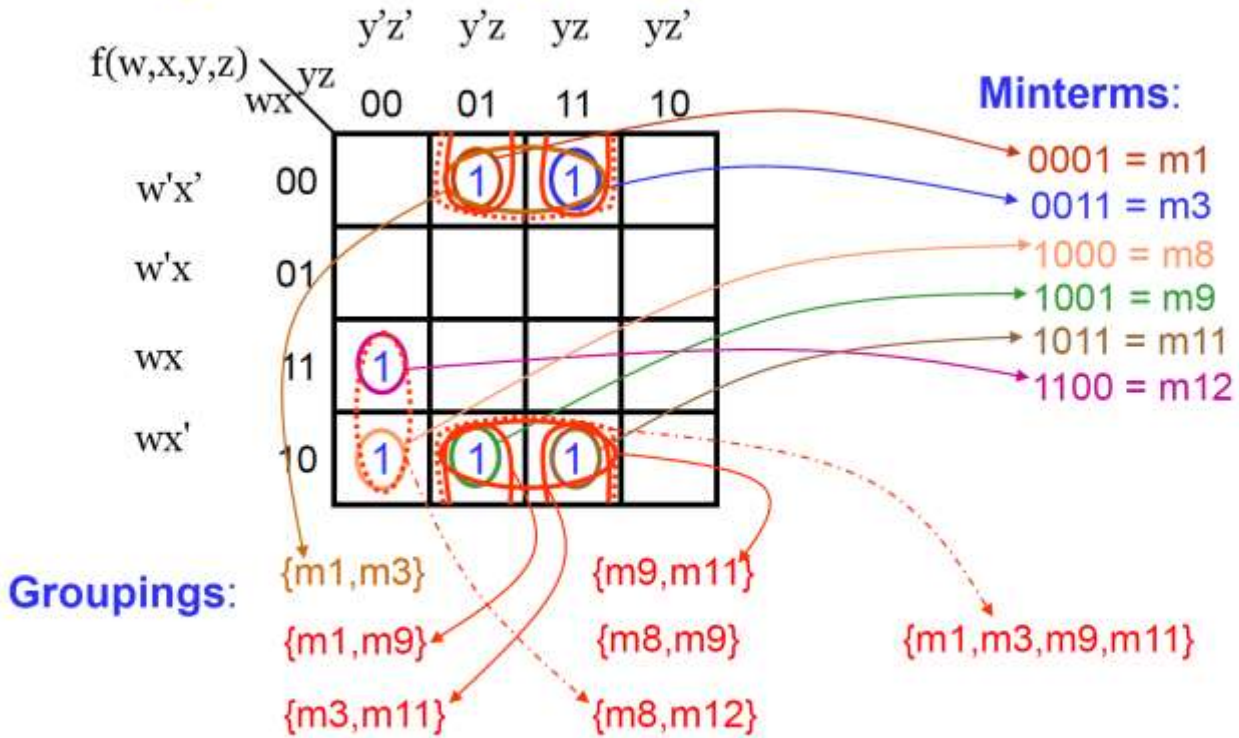
$$F = A + \bar{B}\bar{D} + B\bar{C}D$$

Strategy for "circling" rectangles on Kmap:

Biggest "oops!" that people forget:

Some more examples

Example: Grouping Minterms

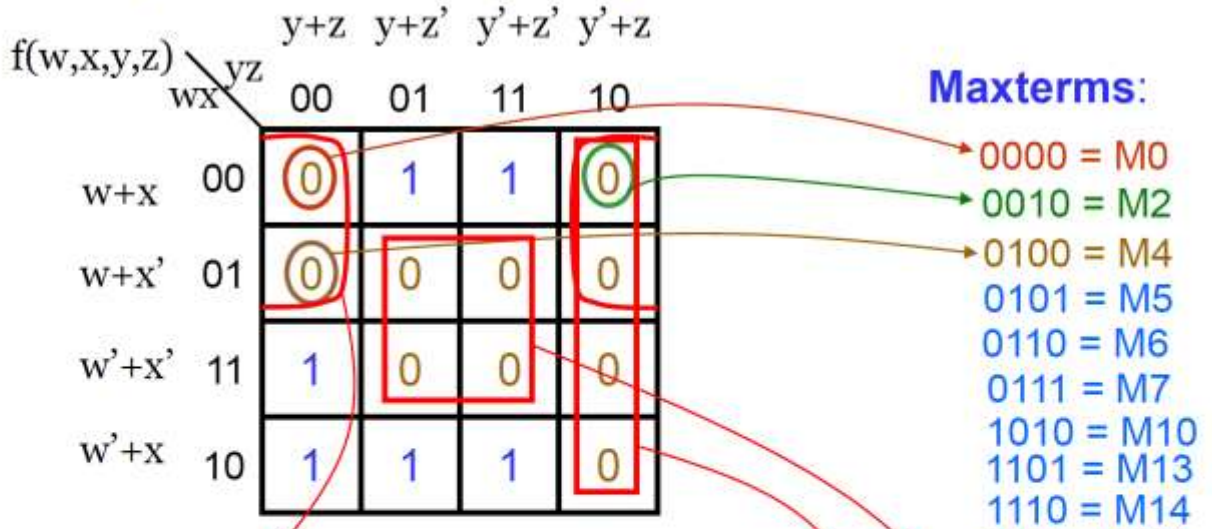


Simplified Expression: $f(w,x,y,z) = \{m1,m3,m9,m11\} + \{m8,m12\}$

$$= \bar{x}z + w\bar{y}\bar{z}$$

Some more examples

Example: Grouping Maxterms



Maxterms:

0000 = M0
0010 = M2
0100 = M4
0101 = M5
0110 = M6
0111 = M7
1010 = M10
1101 = M13
1110 = M14
1111 = M15

Groupings (note: all groupings are not listed):

{M0,M2,M4,M6}, {M4,M5,M6,M7}, {M5,M7,M13,M15},
{M2,M6,M10,M14}, {M6,M7,M14,M15}

Simplified Expression: f(w,x,y,z) = {M0,M2,M4,M6}{M5,M7,M13,M15}
{M2,M6,M10,M14}

$$= (w+z)(\bar{x}+\bar{z})(\bar{y}+z)$$

Why minimize?

Efficient resource usage

Resource scarcity

Summary

- Very simple guideline:
  - Circle all the rectangular blocks of 1’s in the map, using the fewest possible number of circles
    - Each circle should be as large as possible
  - Read off the implicants that were circled
  - Some of them may be “don’t care” (X) Try it yourself


PAUSE

Combinational Circuits

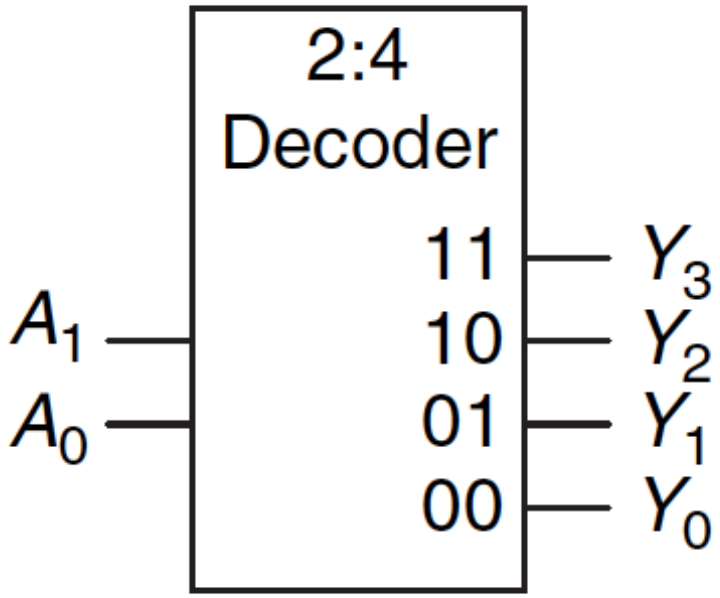
- Combinational logic is often grouped into larger building blocks to build more complex systems
- Hides the unnecessary gate-level details to emphasize the function of the building block
- Output is only dependent on the input
- We now examine:
  - Decoder
  - Multiplexer
  - Full adder

Decoder

- "Input pattern detector"
- n inputs and 2^n outputs
- Exactly one of the outputs is 1 and all the rest are 0s
- The output that is logically 1 is the output corresponding to the input pattern that the logic circuit is expected to detect
- Example: 2-to-4 decoder

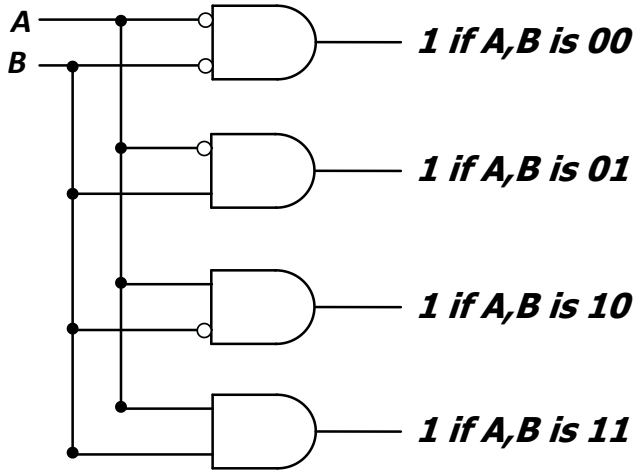
The complement of a decoder is encoder

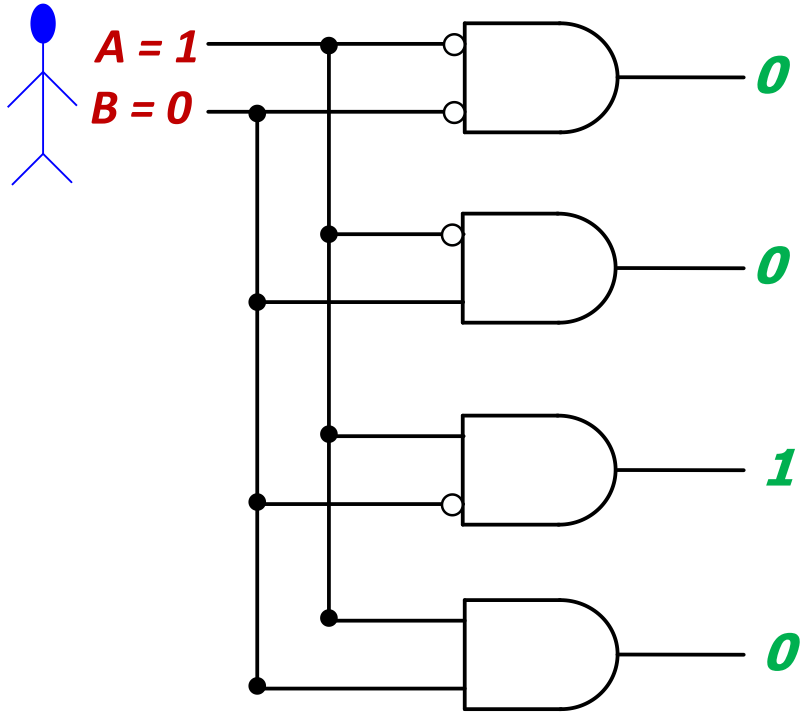
A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Contd.

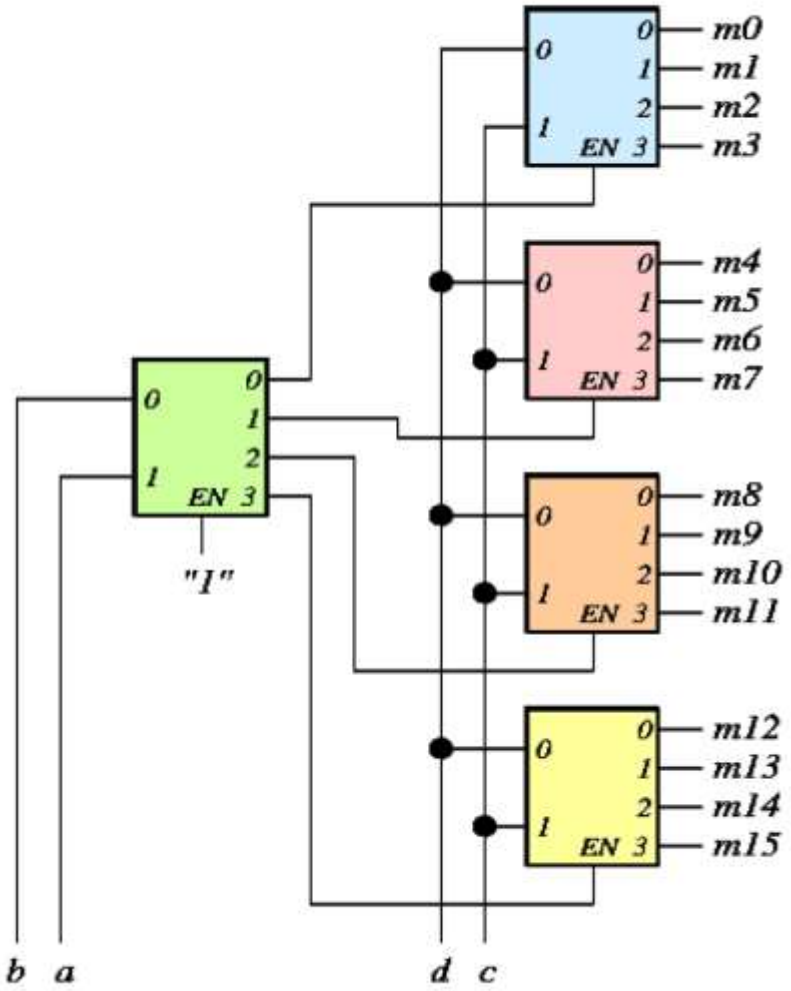
- n inputs and 2^n outputs
- Exactly one of the outputs is 1 and all the rest are 0s
- The output that is logically 1 is the output corresponding to the input pattern that the logic circuit is expected to detect
- Mostly decoders have an enable signal. Output is only generated if the enable is high.





- Build a 4x16 decoder using 2x4 decoders (decoder tree)..

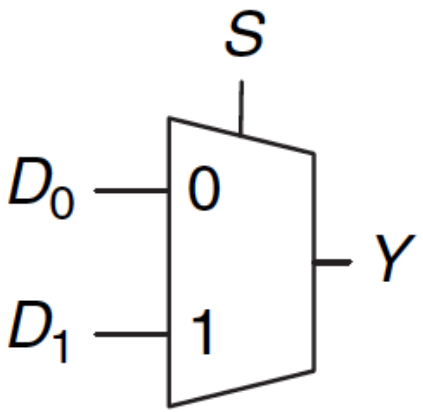
a	*b*	*c*	*d*	*EN*	*minterm*
0	0	0	0	1	m0
0	0	0	1	1	m1
0	0	1	0	1	m2
0	0	1	1	1	m3
0	1	0	0	1	m4
0	1	0	1	1	m5
0	1	1	0	1	m6
0	1	1	1	1	m7
1	0	0	0	1	m8
1	0	0	1	1	m9
1	0	1	0	1	m10
1	0	1	1	1	m11
1	1	0	0	1	m12
1	1	0	1	1	m13
1	1	1	0	1	m14
1	1	1	1	1	m15
X	X	X	X	0	0



Multiplexer

- Selects one of the *N* inputs to connect it to the output
  - based on the value of a $\log_2 N$-bit control input called select
- Example: 2-to-1 MUX

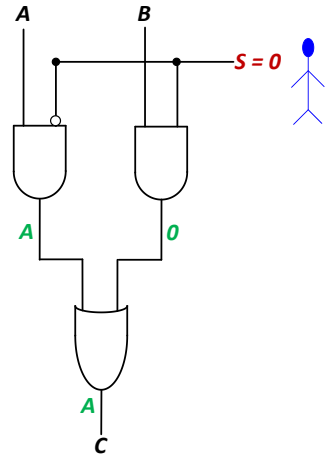
- When S is 0, Y is D_0 and when S is 1, Y is D_1

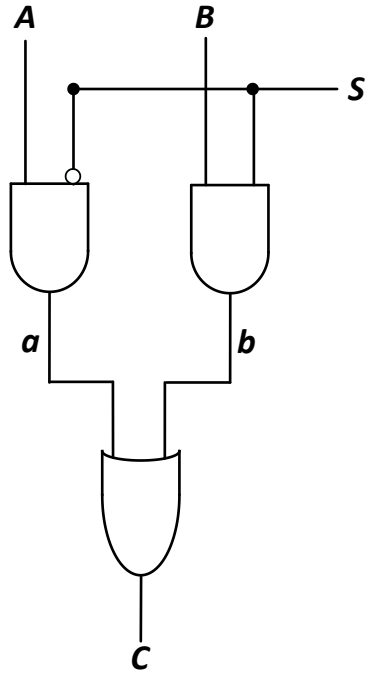


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

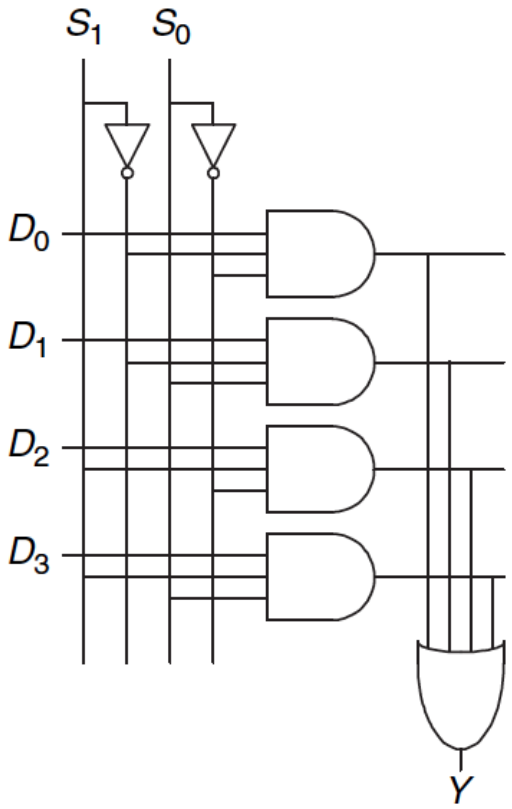
Contd.

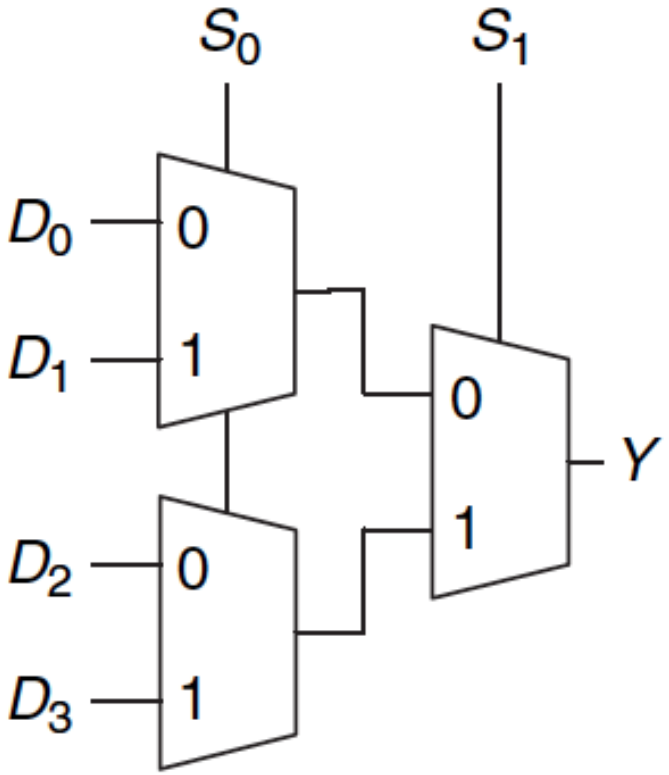
- Selects one of the *N* inputs to connect it to the output
  - based on the value of a $\log_2 N$-bit control input called select
- Example: 2-to-1 MUX





From 2 to 4:1 MUX

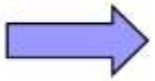




Adder

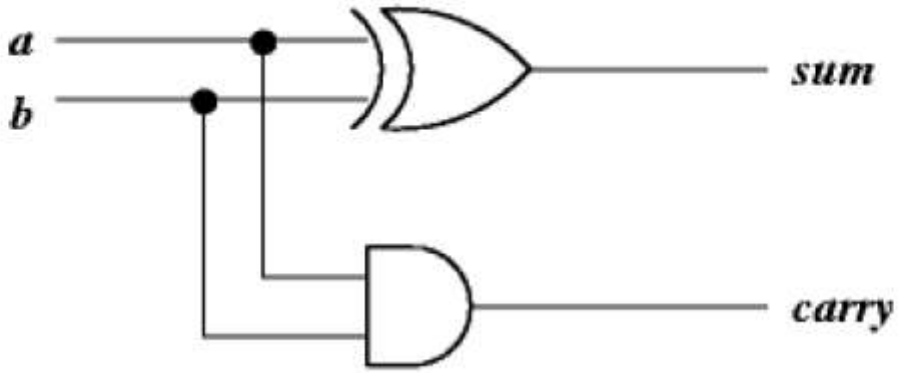
- The most basic arithmetic operation in a digital computer is addition.
- Half Adder is a combination circuit that performs addition of 2 bits.

Inputs		Outputs	
a	*b*	*Carry*	*Sum*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



$$Sum = \bar{a}b + a\bar{b} = a \oplus b$$
$$Carry = ab$$

Half adder



$$Sum = \bar{a}b + a\bar{b} = a \oplus b$$
$$Carry = ab$$

- Half adders cannot accept a carry input and hence it is not possible to cascade them to construct an n-bit binary adder.

Full adder

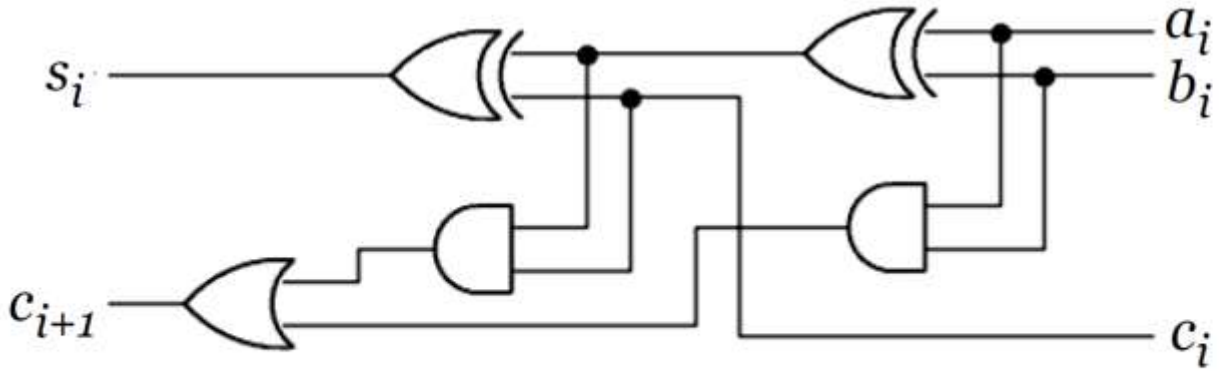
- Full Adder is a combinational circuit that forms the arithmetic sum of three input bits. It is described by the following truth table:

Inputs			Outputs	
c	b	a	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

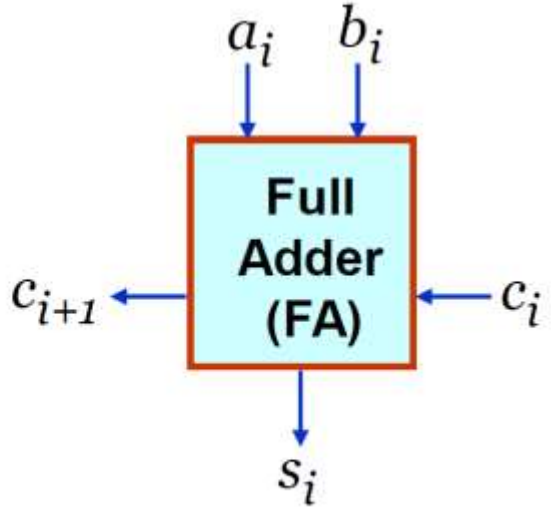
$$Sum = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc = a \oplus b \oplus c$$
$$C_{out} = ab + ac + bc = ab + c(a+b)$$

Implementation

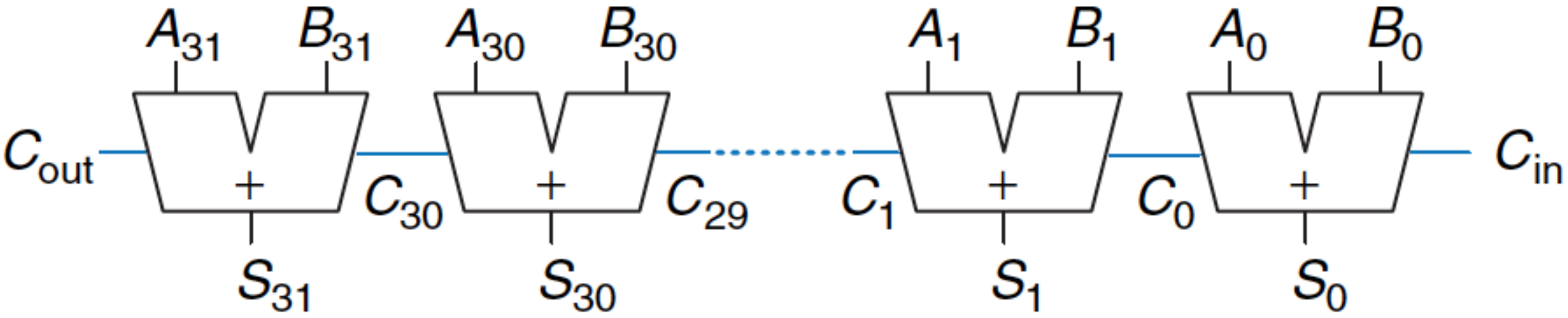
$$Sum = \bar{a}\bar{b}c + \bar{a}b\bar{c} + a\bar{b}\bar{c} + abc = a \oplus b \oplus c$$
$$C_{out} = ab + ac + bc = ab + c(a+b)$$



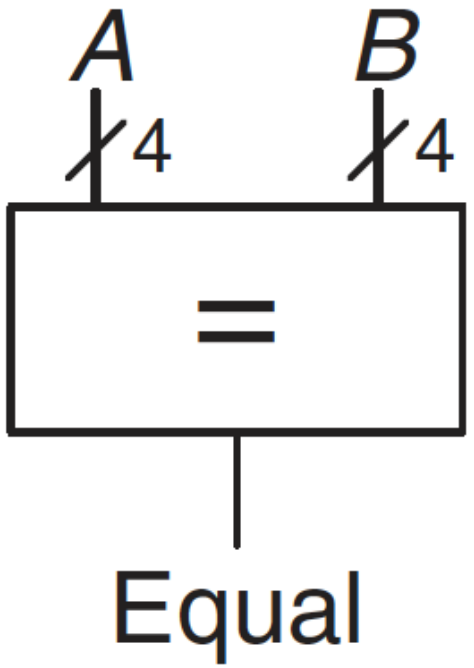
Full Adder at bit i

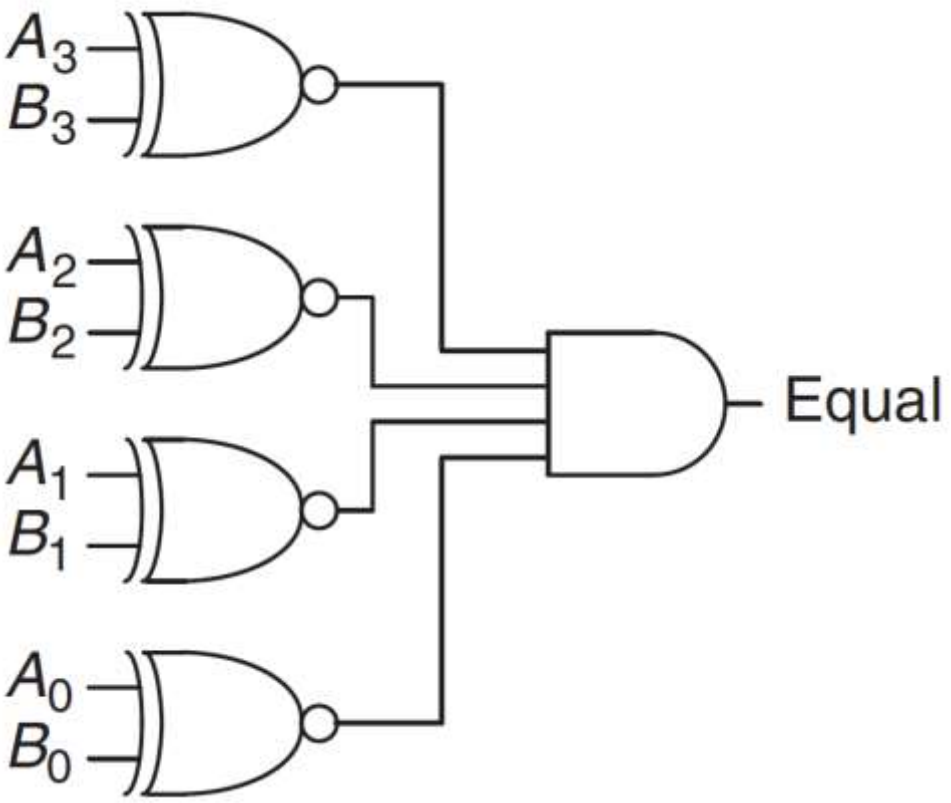


Ripple Carry Adder

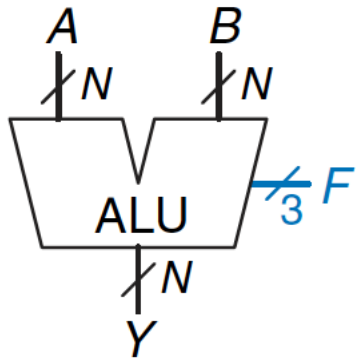


Comparator (Equality Checker)





ALU (Arithmetic Logic Unit)



ALU symbol

ALU operations

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT