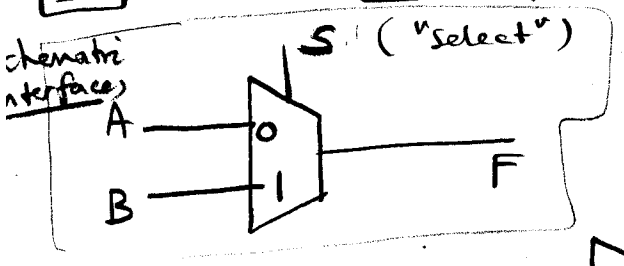
Lecture 2 Continued & Lecture #3 | LECTURE 3

Implementation of intermediate combinational logic building blocks

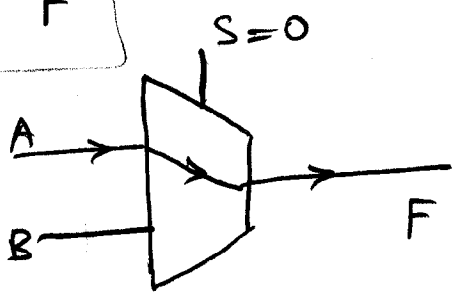
I Mux: (multiplexer)

2 inputs → 2:1 mux → 1 output

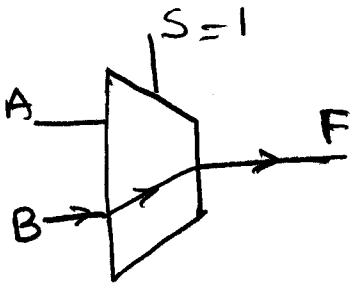
[mux symbol]



operation:



If S=0, F=A

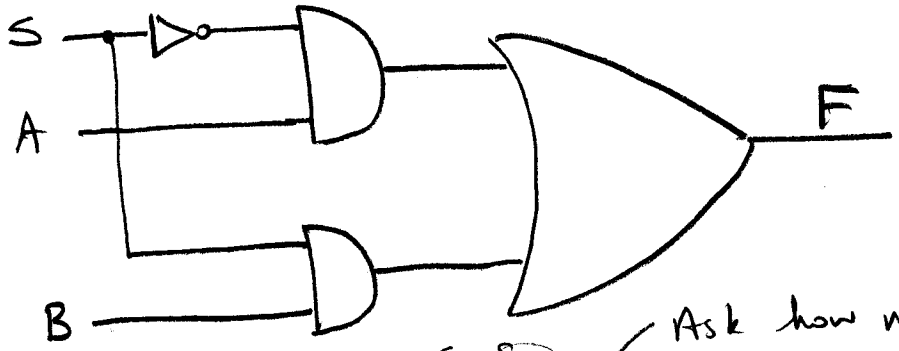


If S=1, F=B

- How can we implement a 2:1 mux using AND, OR and NOT gates? [Implementation]

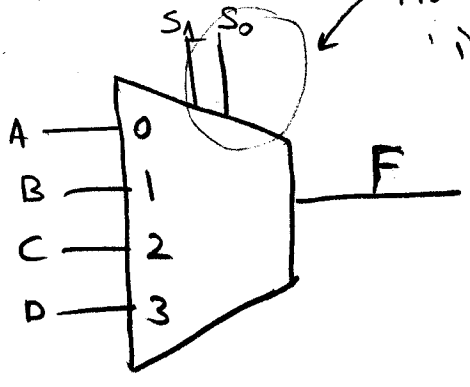
• First, write down the Boolean expression for F. (ask class)

$$F = S' \cdot A + S \cdot B$$



A 4:1 mux

Ask how many control inputs we need.

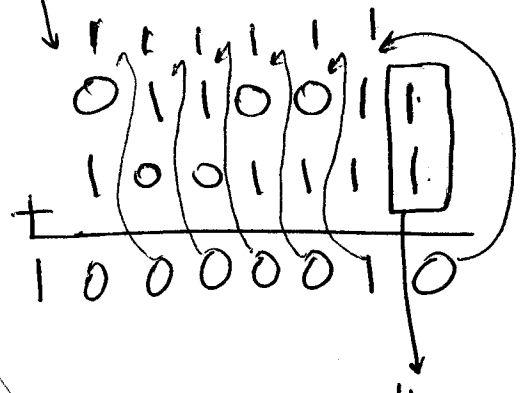


$$F = S_1' \cdot S_0' \cdot A + S_1' \cdot S_0 \cdot B + S_1 \cdot S_0' \cdot C + S_1 \cdot S_0 \cdot D$$

Emphasize interface & implementation

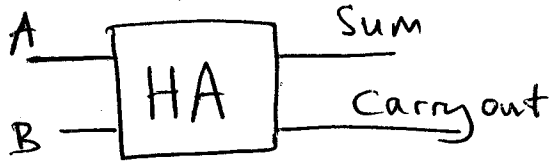
II Design of Adder:

give discussion of unsigned numbers.



Half-adder

- Half-adder create Interface: (schematic):



[Reading: Ch 4.1–4.4, 4.
Ch 6.1]

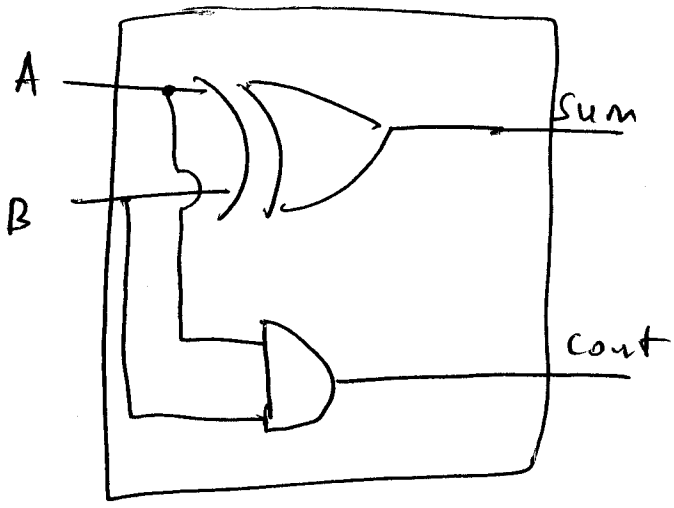
Behavior/operation:

A	B	Sum	cout
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

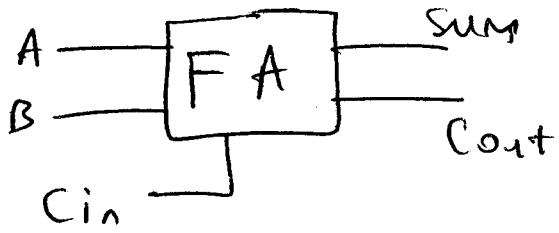
$$Sum = A'B + AB' \quad (= A \oplus B)$$

$$cout = A \cdot B$$

* Implementation



- Full-adder:



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

$$Sum = A'B'C + A'BC' + AB'C' + ABC$$

$$= A'(\underbrace{B'C + BC'}_{(B \oplus C)}) + A(\underbrace{B'C' + BC}_{(B \oplus C)'})$$

$$= A \oplus (B \oplus C) = A \oplus B \oplus C_{in}$$

(XOR is associative)
(you will show this in your hw assignment)

- XOR: basis operation for binary addition. (but cannot track carries)

Ex) $1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0$.

XOR of an odd number of 1's is 1.
even 1's is 0

$$\boxed{Sum = A \oplus B \oplus Cin}$$

$$Cout = A'BC + AB'C + ABC' + ABC.$$

$$= BC(A + A') + A(B'C + BC')$$

$$= BC + A \cdot (B \oplus C)$$

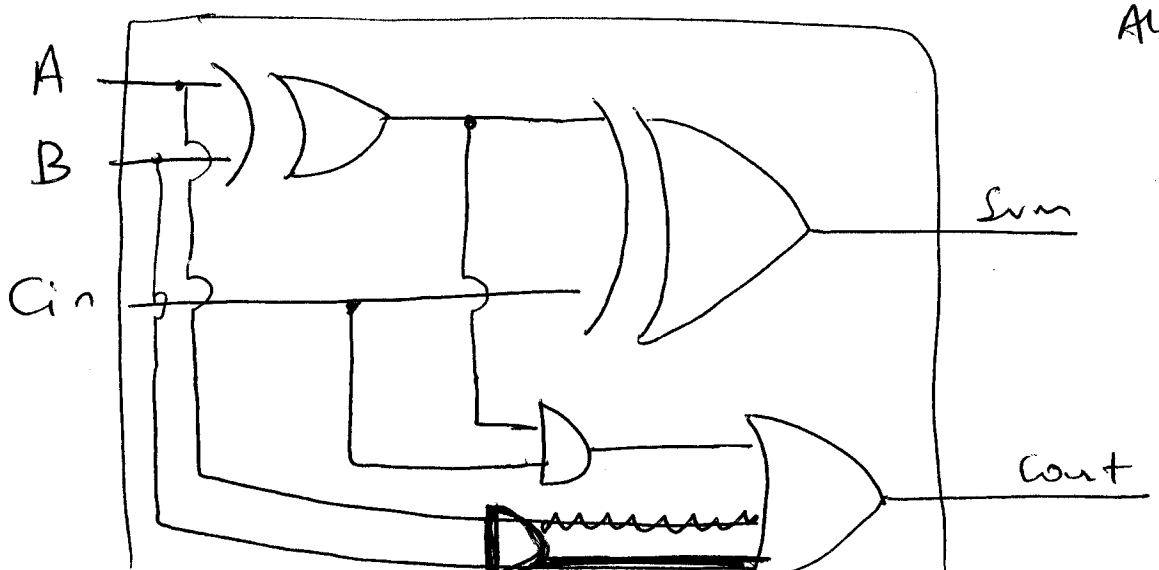
$$= C(A'B + AB') + AB(C + C')$$

($C + C' = 1$)

$$= C(A \oplus B) + AB.$$

$$\boxed{Cout = Cin \cdot (A \oplus B) + A \cdot B.}$$

[Note:
$Cin(A \oplus B) + A \cdot B$
Also possible implementation

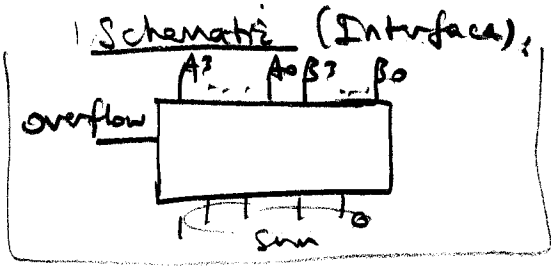


- How do we implement n-bit addition?

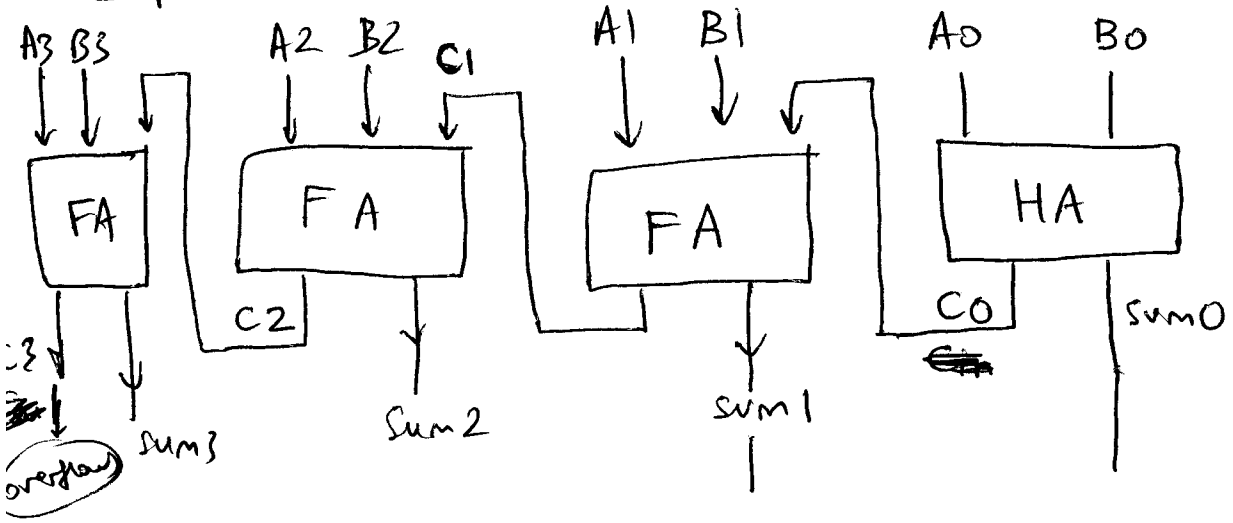
A3 A2 A1 A0

B3 B2 B1 B0

Schematic (Interface):



Implementation:



- Ripple-carry adder

(slow: as n ↑, the delay scales linearly with n.)

- does not handle overflow.

(Ask class: How can I detect the overflow in the addition of 2 unsigned numbers?

Answer: carryout from the last stage indicates overflow. for unsigned numbers.)

Overflow Detected = C3 above.)

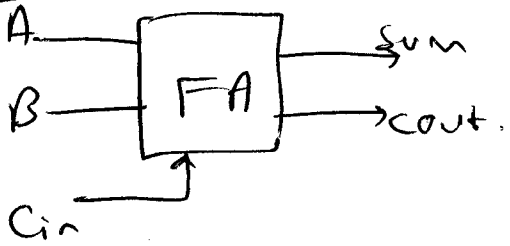
~~01/15/2008~~

ECE 152A Lecture #2

— Boolean Minimization (K-maps using)

~~(Reading for this week: Ch 5-8 Skip 5-6 variable K-maps.)~~

Recall our full-adder example for last lecture. Go back to:



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

→ to page 5

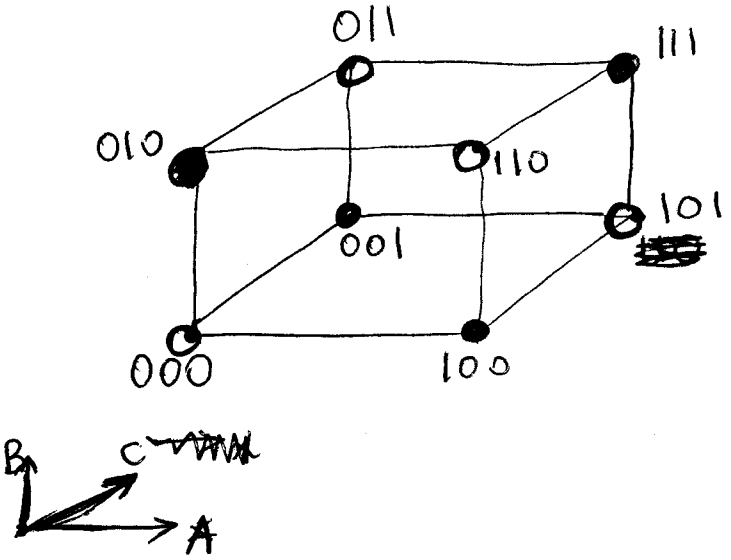
Last time: we did algebraic simplification.

This time: understand geometrically what we're doing when simplifying & design a better method for simplification

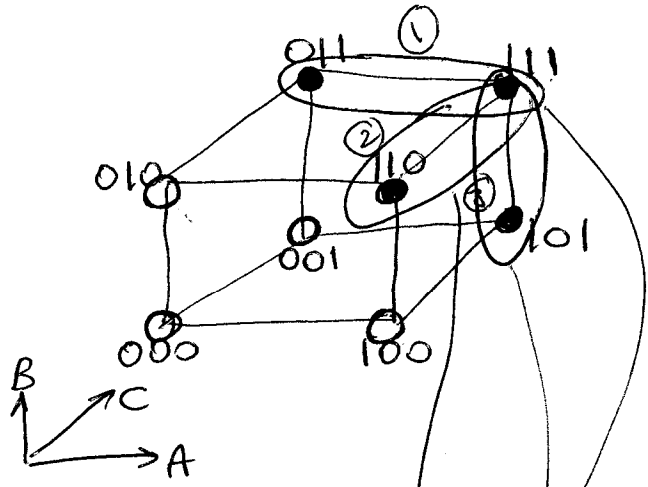
Boolean cube:

represent the input vectors in a vector space:

Sum



Cout



$$Cout = A'BC + AB'C + ABC' + ABC$$

$$\overset{①}{=} (A' + A)BC + AB'C + ABC'$$

OR do ②

$$= A'BC + AB'C + AB(C' + C)$$

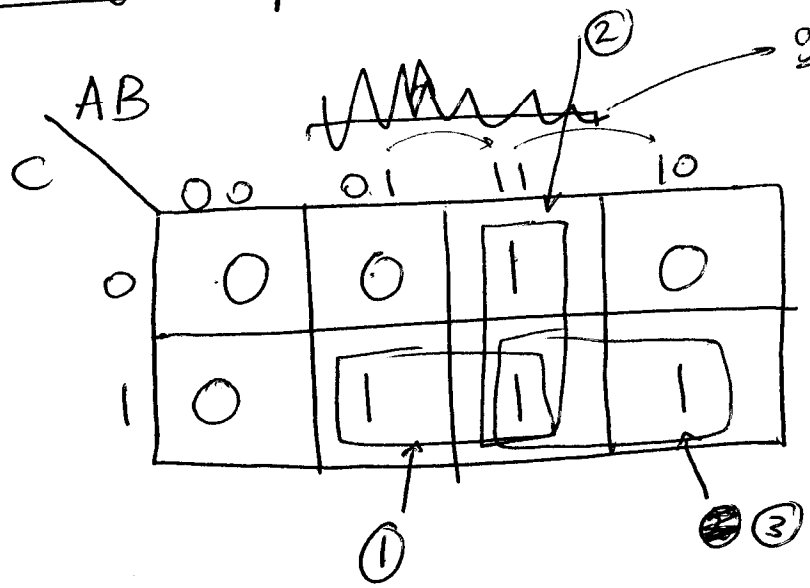
OR do ③

$$= A'BC + A(B' + B)C + ABC'$$

- Hence, main insight is: Algebraic simplification by grouping corresponds to grouping of "minterms" on Boolean (on-terms) cube.

- If the number of inputs is ~~not~~ not very high (~~well~~ well address ≤ 4), then grouping by visualization tools ~~may be faster~~ is faster/easier.

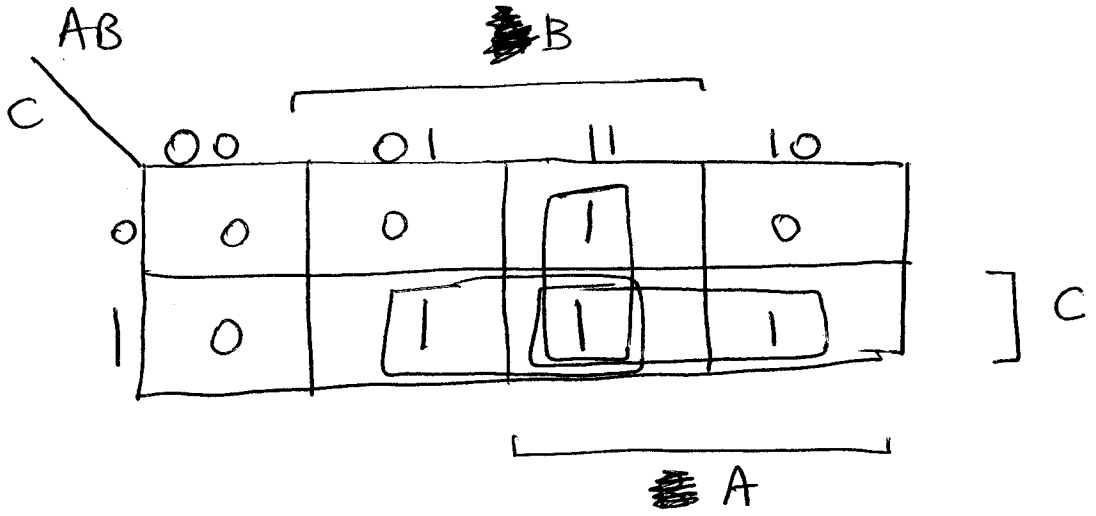
- Karnaugh map:



gray code: (define it)
- adjacency on Boolean cube
- wrap around

- In the future: we will go directly from the truth table to the K-map.
- An important skill is to be able write down the resulting algebraic expression after minimization (grouping) on the K-map.

~S, above:



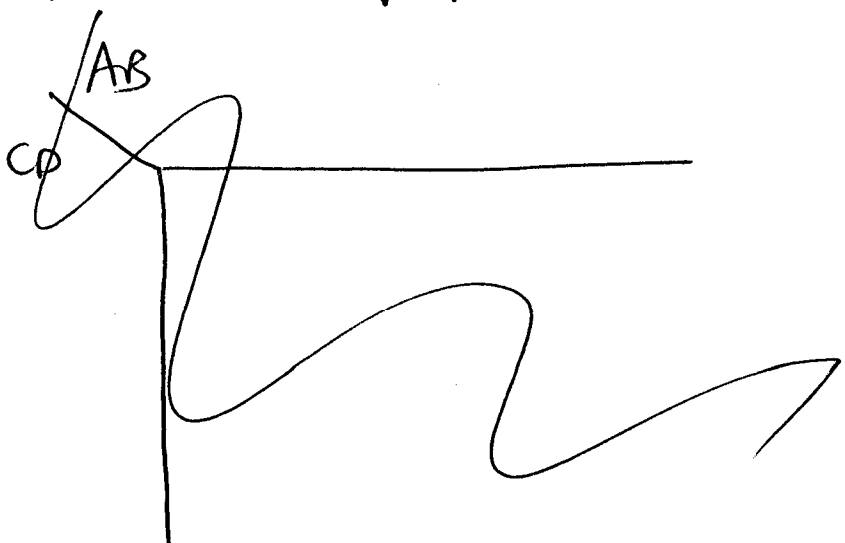
$$\boxed{C_{out} = AB + BC + AC.}$$ ← the "majority function" (3-inputs)

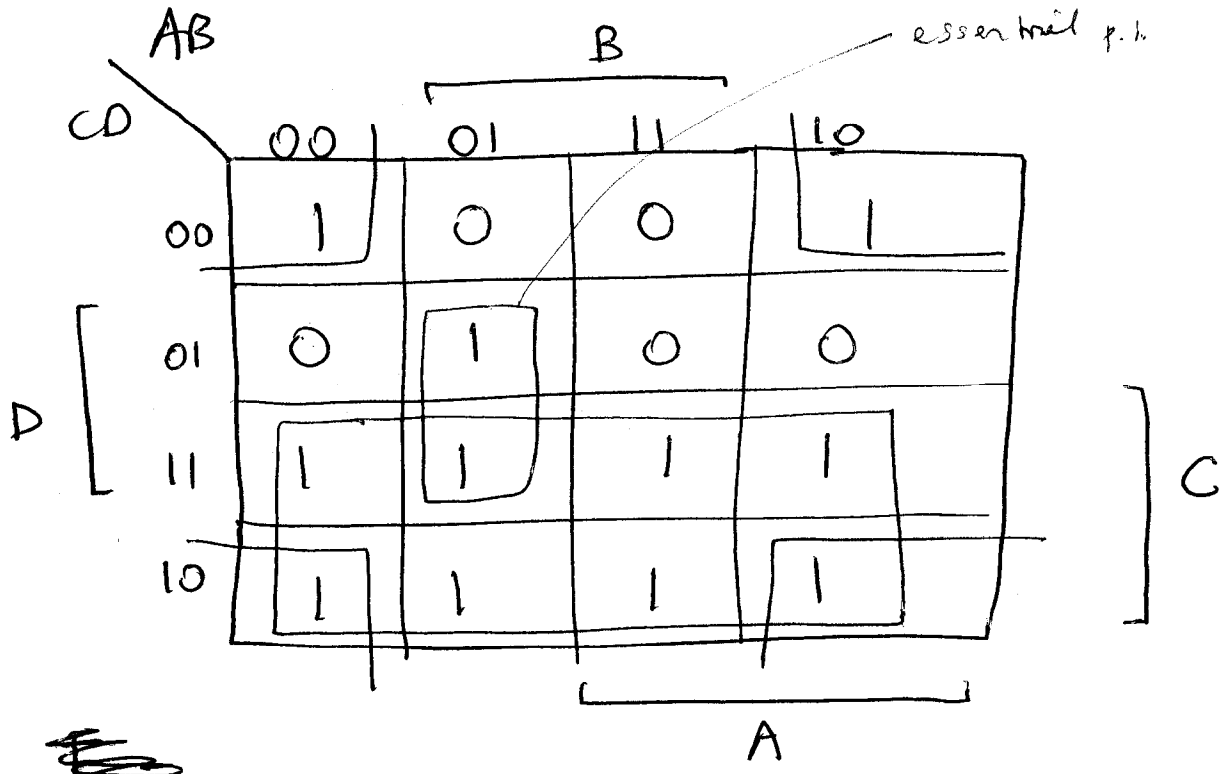
$$= (C_{in}(A+B) + AB)$$

(is ~~equal to one~~ 1 ~~Rule~~ iff the majority of the inputs is 1)

I have illustrated above how to do simplification using K-maps

Another Example: K-map of a 4-variable function:





(all p.i.'s here are essential)

$$F = C + A'BD + B'D'$$

Rigorous treatment of these concepts:

- Two-level, logic canonical forms:
  - Any Boolean function can be written in a unique canonical form:
    - 2 forms:
      - ~~Minimal~~ canonical Sum of products (minterm expansion)
      - canonical product of sums: (maxterm expansion)

Sum of products :

A	B	C	minterms	
0	0	0	$A'B'C'$	$= m_0$
0	0	1	$A'B'C$	$= m_1$
0	1	0	$A'BC'$	$= m_2$
1	1	1	ABC	$= m_7$

→ an ANDed product of literals in which each variable appears exactly once either in true or complemented form.

Ex) AB is not a minterm.

• $ABB'C$ is not a minterm.)

$$C_{out} = A'BC + AB'C + ABC' + ABC$$

canonical Sum of products form.

— minterm expansion

$$= m_3 + m_5 + m_6 + m_7$$

Product of Sums : (maxterm expansion)

maxterm: an ORed sum of literals in which each variable appears exactly once in true or complemented form

A	B	C	maxterms	
0	0	0	$A + B + C$	$= M_0$
0	0	1	$A + B + C'$	$= M_1$
0	1	0	$A + B' + C$	$= M_2$
0	1	1	$A' + B + C$	$= M_3$
1	0	0	$A' + B + C$	$= M_4$
1	0	1	$A' + B + C'$	$= M_5$
1	1	0	$A' + B' + C$	$= M_6$

$$C_{out} = m_3 + m_5 + m_6 + m_7.$$

$$(C_{out})' = m_0 + m_1 + m_2 + m_4 \qquad \text{(Ask class)}$$

$$= A'B'C' + A'B'C + A'BC' + AB'C'$$

$$C_{out} = \left((C_{out})'\right)'$$

$$= \left(A'B'C' + A'B'C + A'BC' + AB'C'\right)'$$

$$= \underbrace{(A+B+C)}_{M_0} \cdot \underbrace{(A+B+C')}_{M_1} \cdot \overbrace{(A+B'+C)}^{M_2} \cdot \underbrace{(A'+B+C)}_{M_4}$$

$$= M_0 \cdot M_1 \cdot M_2 \cdot M_4 \quad \text{(maxterm expansion)}$$

Hence: $C_{out} = m_3 + m_5 + m_6 + m_7 = M_0 \cdot M_1 \cdot M_2 \cdot M_3.$

$$C_{out}' = m_0 + m_1 + m_2 + m_3 = M_3 \cdot M_5 \cdot M_6 \cdot M_7$$

Two-level simplification: go to the Boolean cube example now

& come back

"Minimum sum of products" : Sum of products with the minimum number of product terms and out of all such ones, one with the minimum number of literals.

(not a unique expression)

"minimum product of sums" : — defined similarly

---

implicant : is an ANDed expression (product) that makes the function true. (So a single 1 or a group of 1's on a K-map).

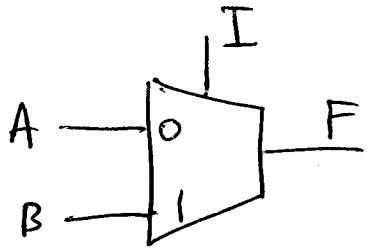
prime implicant : an implicant that cannot be combined with other terms to simplify further by theorem.

essential prime implicant : a prime implicant that is necessary in the expression in order to cover a "1".

(go to example on p. 5)

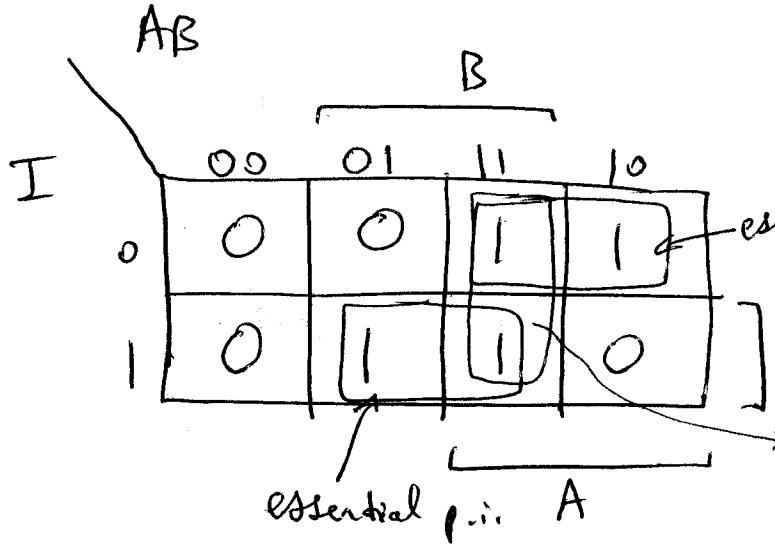
Example: 2-bit comparator:
(see sheet)

Example: 2-1 mux: example.
(Go back to)



$$F = I \cdot B + I' A .$$

A	B	I	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



$$F = I \cdot B + I' \cdot A .$$

$+ (AB)$ ← consensus term.

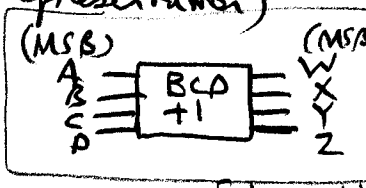
Incompletely Specified Functions

Ex) Design a combinational circuit that takes as input a Binary-Decimal-coded (BCD) digit X ($X \in$ [ie. 0 ... 9]) and outputs $X+1$ (mod 10). ie. $0 \to 1$, $1 \to 2$, .. $8 \to 9$, $9 \to 0$. The input is guaranteed (by external circuitry) to lie in range 0 ... 9.

Design: Need 4 bits to represent the input (unsigned binary representation) and 4 bits for output

$\overline{A}$ $\overline{B}$ $\overline{C}$ $\overline{D}$

↑ MSB ... ↑ LSB



Schematic (Interface)

So, we use the numbers 0000 ("0") $\to$ 1001 ("9"). Note that the input values "10" $\to$ "15" will never occur, ~~but we~~

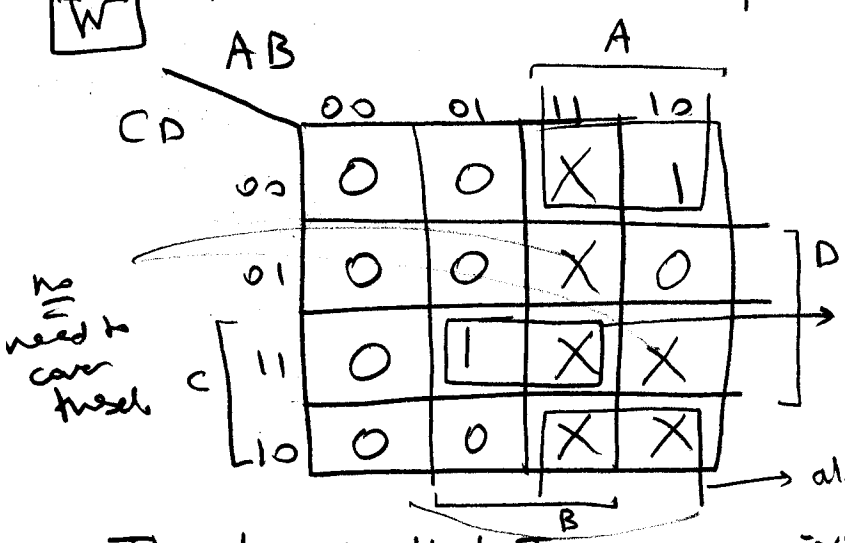
We can actually use this fact to our advantage when we do logic minimization.

- Let's write down the truth table for the "increment by 1" circuit.

A	B	C	D	W	X	Y	Z	
0	0	0	0	0	0	0	1	
0	0	0	1	0	0	1	0	
0	0	1	0	0	0	1	1	
0	0	1	1	0	1	0	0	
0	1	0	0	0	1	0	1	
0	1	0	1	0	1	1	0	
0	1	1	0	0	1	1	1	
0	1	1	1	1	0	0	0	
1	0	0	0	1	0	0	1	
"9" → 1	0	0	1	0	0	0	0	← note wrap-around
1	0	1	0	X	X	X	X	"don't cares"
1	0	1	1	X	X	X	X	
1	1	0	0	X	X	X	X	
1	1	0	1	X	X	X	X	
1	1	1	0	X	X	X	X	
1	1	1	1	X	X	X	X	

Logic minimization: — need to draw a K-map for each output variable.

W

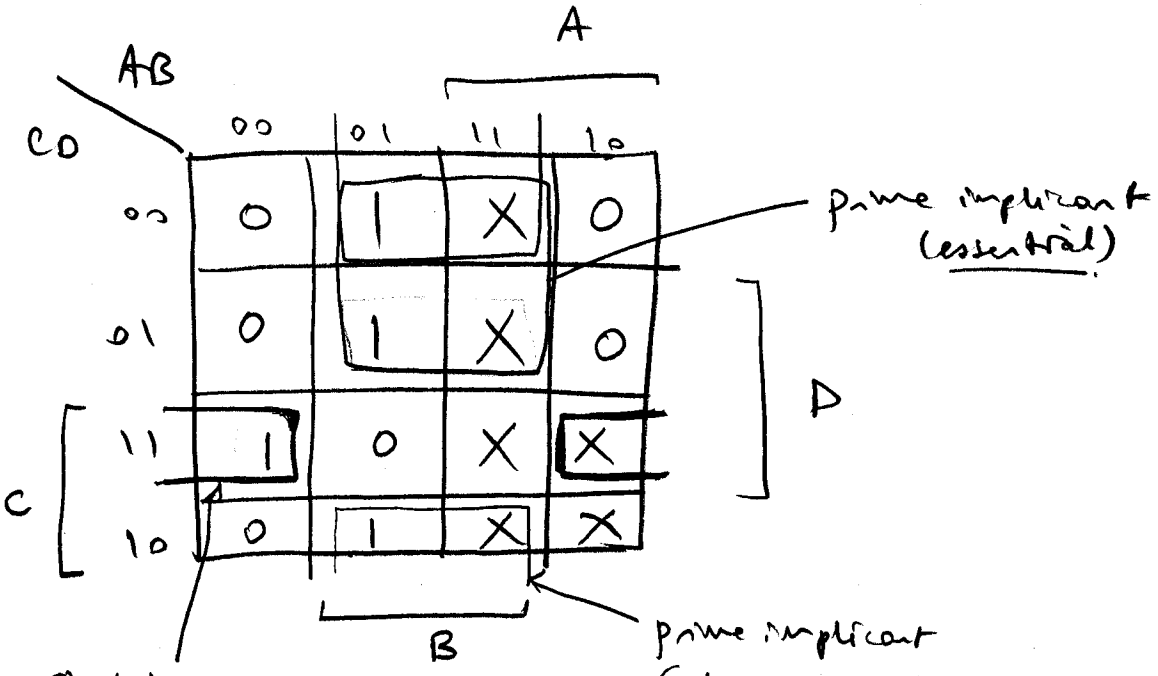


- The key is that I can use "X"'s as 0's or 1's as I wish because those input values will not occur anyway.
- But: I don't need to cover all X's!
- Hence, above, we have to draw two 1's... etc.

$$W = BCD + AD'.$$

(a minimum SOP expression for W.)

X



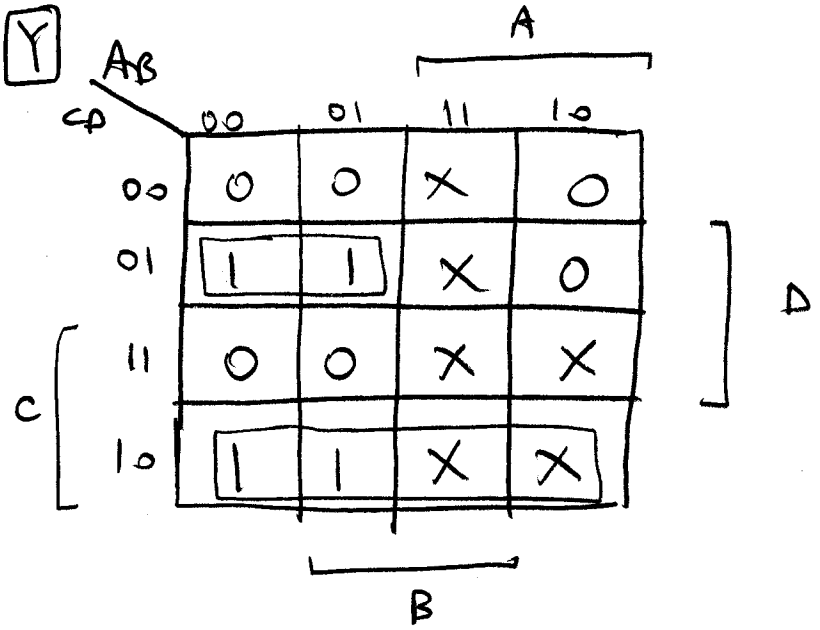
$$X = B'CD + BC' + BD'$$

~~Do Y and Z as homework.~~ Exercise

Similarly, we find the following for Y & Z.

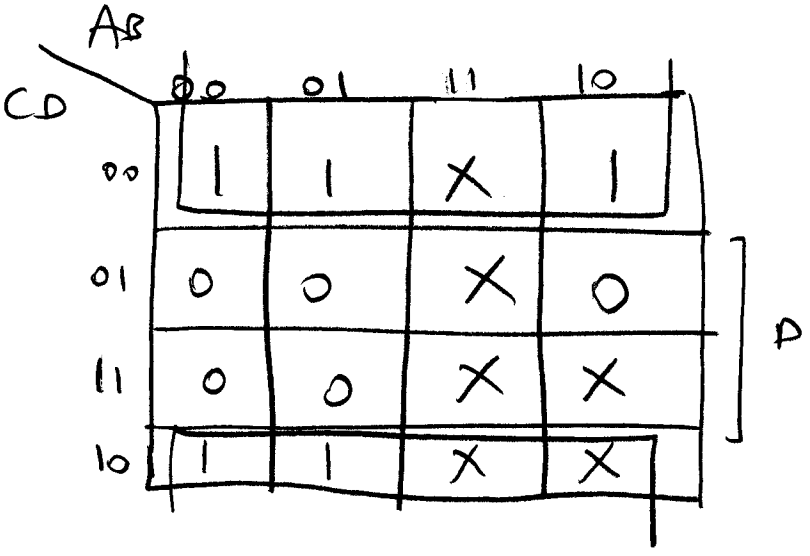
Answers:

Y



$$Y = A'C'D + CD'$$

Z



$$Z = D'$$