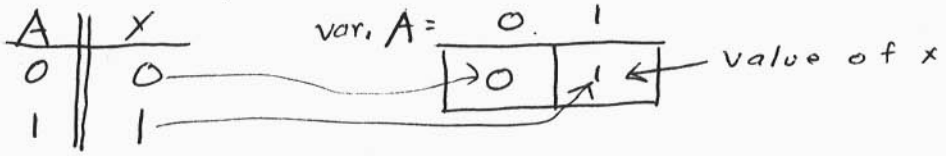
Either the canonical form or the first expression can be reduced algebraically – but gets a bit messy.

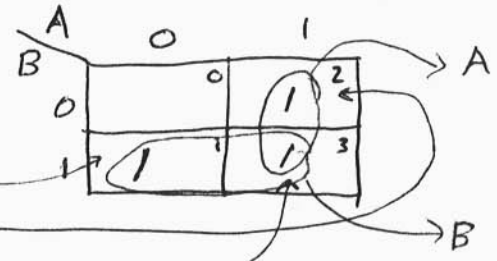
Motivates alternate way of determining the groupings:

One variable map: (trivial)



Two variable map

min term	A	B	Z
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

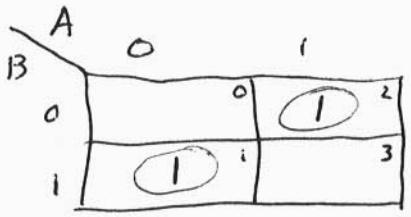


Adjacent min-terms

$Z = \underbrace{\bar{A}B + AB}_{B(\bar{A}+A) = B} + \underbrace{AB + A\bar{B}}_{A(B+\bar{B}) = A}$

$Z = B + A$

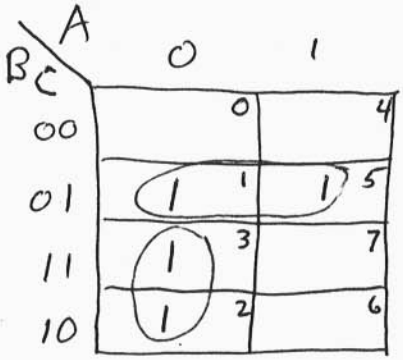
m	A	B	Z
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0



Non-adjacent terms $Z = \bar{A}B + A\bar{B}$

3 variable map

n	A	B	C	X
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	0



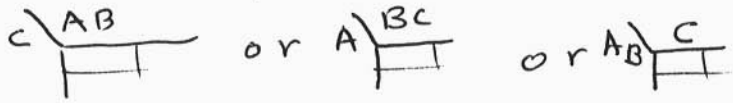
$$X = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}BC + \bar{A}B\bar{C}$$

(1) (5) (3) (2)

$\bar{B}C$ $\bar{A}B$

$$X = \bar{B}C + \bar{A}B$$

or can set up as



with same results

Note ordering of rows-columns: corresponds to counting 0, 1, 2, 3, ··· but in Gray Code.

Algorithm for Gray code:

append extra 0 if needed → $b_3\ b_2\ b_1\ b_0$ ← binary code bits

$g_2\ g_1\ g_0$

i.e. $g_i = b_i \oplus b_{i+1}$

ex: 1 1 0 0 (binary for decimal 12)

$g_0 = b_0 \oplus b_1 = 0 \oplus 0 = 0$

$g_1 = b_1 \oplus b_2 = 0 \oplus 1 = 1$

$g_2 = b_2 \oplus b_3 = 1 \oplus 1 = 0$

$g_3 = b_3 \oplus (b_4 = 0) = 1 \oplus 0 = 1$

hence $12_{10} = 1100_{\text{binary}} = 1010_{\text{Gray}}$

Characteristic of Gray counting is that only one bit at a time changes in count sequence:

e.g. in binary going 7 → 8 gives 0111 → 1000

Gray " " " 0100 → 1100

decimal:	0	1	2	3	4	5	6	7
Gray	0000	0001	0011	0010	0110	0111	0101	0100
	8	9	10	11	12	13	14	15
	1100	1101	1111	1110	1010	1011	1001	1000

By using Gray code ordering of row/columns we ensure that each cell is "adjacent" to cells to left and right + above and below.

"Adjacent" = min-terms in adjacent cells differ by only one bit.

Groupings: Basic idea in using K map is to find groups of adjacent cells

Group of 2 cells ⇒ one variable eliminated

" " 4 " two " "

8 " three " "

For each suitable group result is that retain only the product term for which the literals remain <u>unchanged</u> throughout the group.

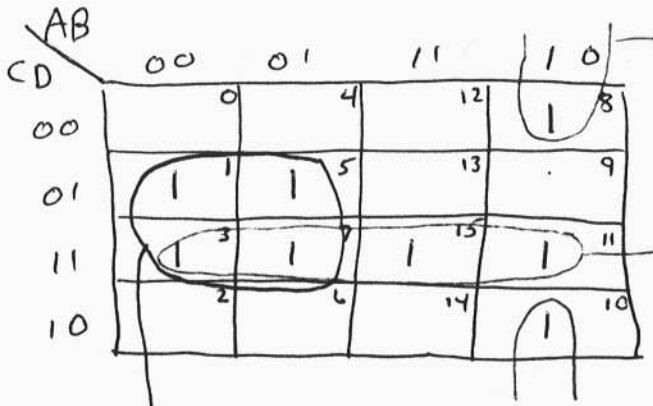
Note that cells on diagonals are not adjacent,

" " corner cells are adjacent.

" " all cells (with a 1 in them) must be accounted for, either as part of a group or by themselves.

4 variable map:

Assume from a TT A B C D | Z get following map



→ for this 2 loop, C changes so term is $A\bar{B}\bar{D}$

→ for this 4 loop, A, B change so term = CD

→ for this 4 loop, B, C change so term is $\bar{A}D$

so Z =

note: have:

$$\underbrace{\bar{A}\bar{B}\bar{C}D}_{①} + \underbrace{\bar{A}B\bar{C}D}_{③} + \underbrace{\bar{A}\bar{B}CD}_{③} + \underbrace{\bar{A}BCD}_{⑦}$$

$$= \bar{A}\bar{C}D(\bar{B}+B)' + \bar{A}CD(\bar{B}+B)'$$

$$= \bar{A}D(\bar{C}+C) = \bar{A}D$$

nomenclature:

A grouping = a prime implicant

An essential prime implicant is a grouping not contained in a larger grouping.

Grouping: AKA a loop, an enclosure

Home exercise: use K map to reduce the Swiss airline problem.

Don't care states often used in K maps to simplify expressions.

May be taken as 1's or 0's (or ignored) as convenient.

Example: code conversion between BCD and excess 3 codes.

	BCD				Excess three			
	A3	A2	A1	A0	B3	B2	B1	B0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	d	d	d	d	d			
11	d	d	d	d	d			
12	d	d	d	d	d			
13	d	d	d	d	d			
14	d	d	d	d	d			
15	d	d	d	d	d			

A_3A_2 / A_1A_0

	00	01	11	10
00	10	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

A_3A_2 / A_1A_0

B1

	00	01	11	10
00				
01				
11				
10				

A_3A_2 / A_1A_0

B3

(5, 6, 7, 8, 9)

	00	01	11	10
00			d	1
01		1	d	1
11		1	d	d
10		1	d	d

$B3 = A3 + A_2A_1 + A_2A_0$

B0

	00	01	11	10
00				
01				
11				
10				

A_3A_2 / A_1A_0

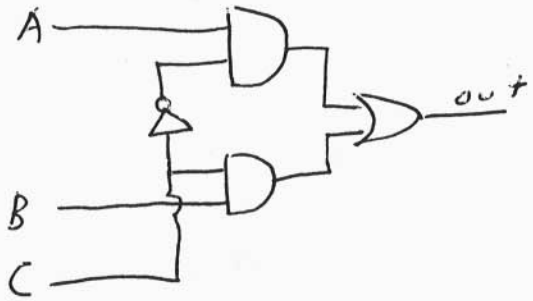
B2

	00	01	11	10
00		1	d	
01	1		d	1
11	1		d	d
10	1		d	d

$B_2 = A_2\bar{A}_1\bar{A}_0 + \bar{A}_2A_1 + \bar{A}_2A_0$

MUX

Consider two ways of looking at:



Can regard as plain vanilla circuit, write TT and fill in 8 row etc to get

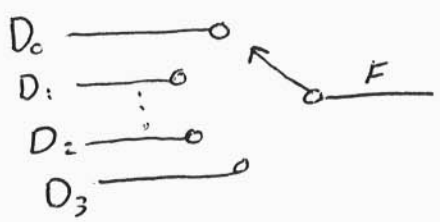
$out = B \cdot C + A \cdot \bar{C}$

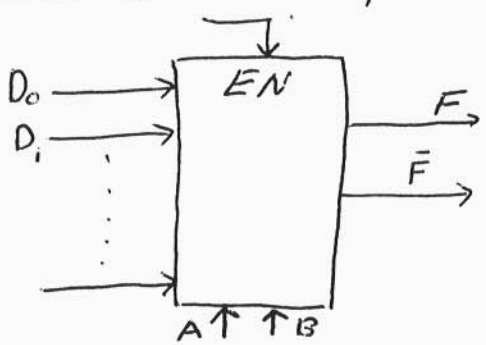
OR can note that only one of the two AND gates is ennabled at a given time and OUT then copies its input. Can describe as

C	A	B	out
0	0	x	0 } = A
0	1	x	1 } = A
1	x	0	0 } = B
1	x	1	1 } = B

Can regard the circuit as just a rotary switch controlled by C: it is the simplest form of a multiplexer (AKA MUX)

Bigger one chip MUX's are available equivalent to and with general pinout as:





D_0 --- are the input "data" lines, 4 or 8 or 16 etc

F is output line } one or other
$\bar{F}$ is $\overline{\text{output}}$ line } or both may be available

A, B are control, AKA address lines

4 inputs ⇒ A, B

8 " ⇒ A, B, C

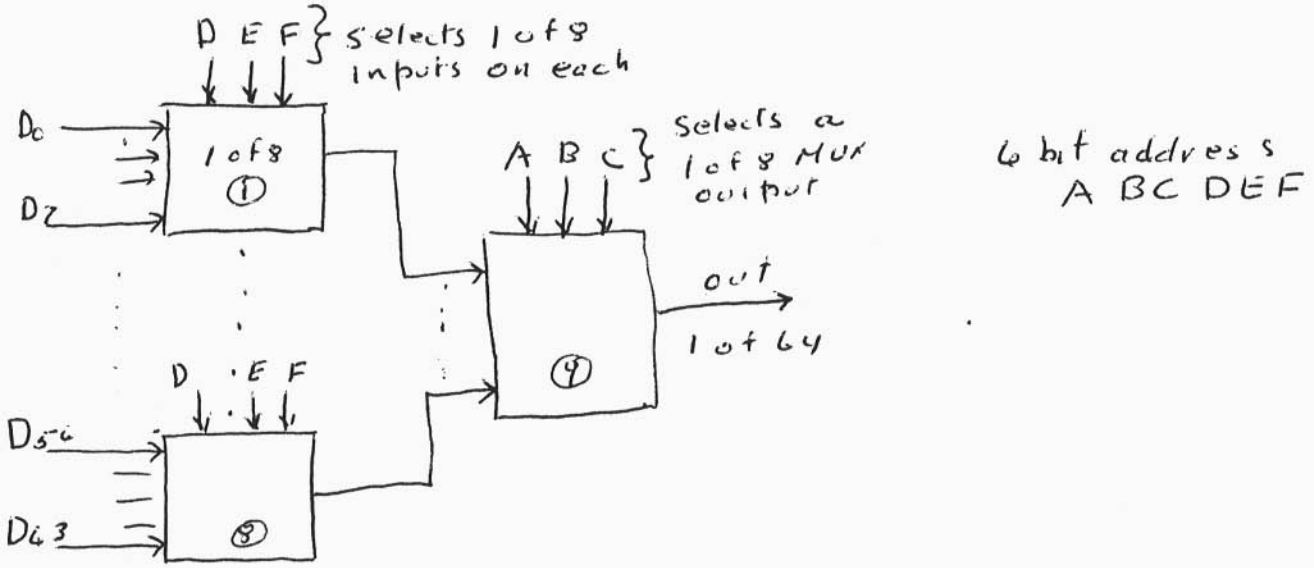
16 " ⇒ A, B, C, D

EN is an enable input; can force output high (or low) regardless of anything else going on. Often also called a "chip select" line.

74150 = 1 of 16 MUX 74151 = 1 of 8 MUX

74152 = 1 of 8 MUX 74153 = dual 1 of 4 MUX

MUX's can be cascaded in various ways to obtain larger ones: e.g. use 9 - 1of8 to get a 1 of 64 MUX



(EN's not shown)

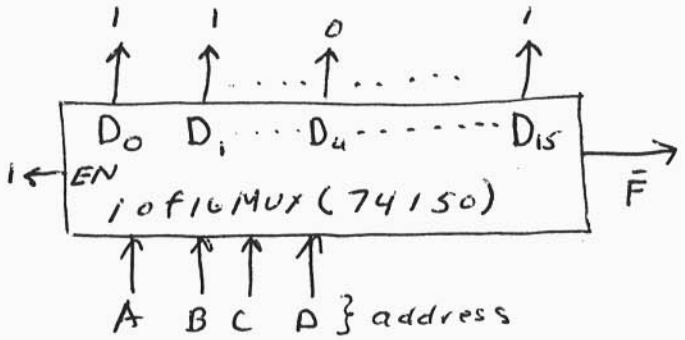
MUX applications

Ex 1:

Say said a TT something like:

	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
4	0	1	0	0	0
15	1	1	1	1	1

Can get direct implementation

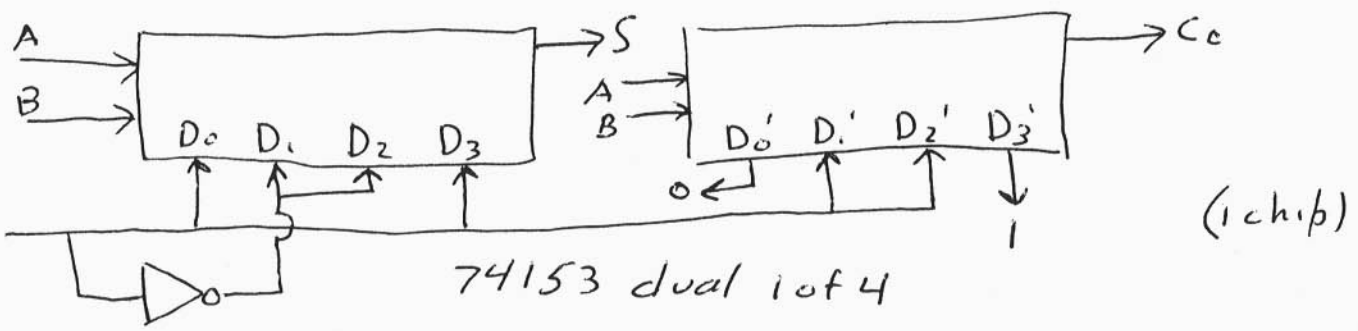


Ex 2: Full adder, inputs A, B, C_i; outputs S, C_o

↙ partition

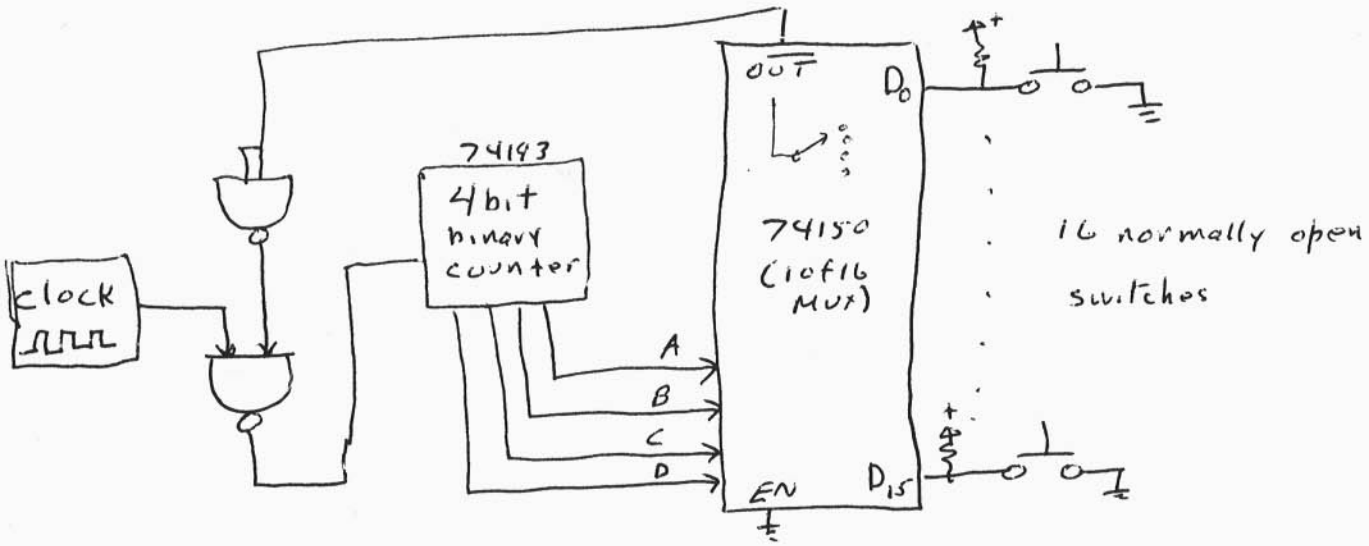
A	B	C_i	S	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

for S	for C_o
$D_0 = C_i$	$D_0' = 0$
$D_1 = \bar{C}_i$	$D_1' = C_i$
$D_2 = \bar{C}_i$	$D_2' = C_i$
$D_3 = C_i$	$D_3' = 1$



A MUX application:

Consider following circuit:



Assume a clock and straight binary 4bit counter (counts 0000 → 0001 → 0010 etc)

No switch closed, all Data inputs high so $\overline{OUT} = 0$ and so top NAND supplies a 1 to bottom NAND which in turn lets clock go thru to counter. MUX then scans each D input in turn.

If a switch is closed then output of MUX changes state (this MUX has $\overline{\text{F}}$ output) and stops clock to counter. At this point counter state specifies which switch was pressed (ie closed.)

Hence this is a "hex keyboard encoder". Obviously more must be added to make this a practical keyboard encoder. Like what?