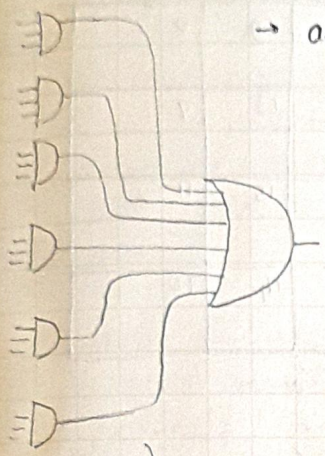
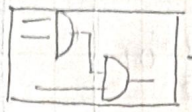
Minimisation of Switching Function 「コハビ」

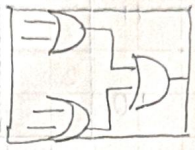
Say $f(x,y,z) = xyz' + xy'z + x'yz + xyz + x'yz' + xy'z'$



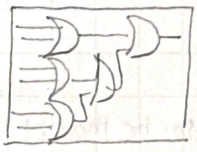
→ also as AND and OR gates are only 2 input devices:



→ 2 input AND



→ 4 input OR



→ 6 input OR

We used 17 gates here.

→ So # gates = O(literals) (l-1)

→ Having these many gates increases space and delay.

→ Complexity needn't always depend on # literals

Eg: $xyz + x + y' + z'$ & $xyz + xyz'$

Our objective:

Minimise no. of pdt terms in SOP & no other expression with same no. of pdt term with fewer literals. → (In $xyz \to x$ is a literal)

• Principle of minimising:

1. $Aa' + Aa = A$ (or) $xyz' + xyz = xy$ } Our obj is to find such combinations.

Now take $f(x,y,z)$ from above $x'yz' + x'y'z' + xy'z' + x'yz + xyz + xy'z$

① $\Rightarrow x'z'(y+y') + xy'z' + x'yz + xyz + xy'z$

$\Rightarrow z'(xy' + x') + yz + y'(xz)$ /* $x + x'Y = x + Y$

$\Rightarrow x'z' + y'z' + z(y+x)$

$\Rightarrow xz + x'z' + yz + y'z'$ (7 gates)

(OR)

② $x'z'(y+y') + y'(xz' + xz) + yz(x+x')$

$\Rightarrow x'z' + xy' + yz$ (5 gates)

③ $\Rightarrow x'y + yz' + xz$ (5 gates)

/* All of these are irredundant & among these ② & ③ are minimal */

• Irredundant (or) Irreducable expression:

→ That which can't be further reduced.

1. → More than 1 irredundant expr. are possible.
2. → Not all irreducible expr. are minimal.
3. → Minimal expr is not unique.

→ min{ irred. size() }

/*: We need repr. of the form xy' & xyz should be placed side by side.

No. 02
Date 08月18日 (木)
Contents

~ Karnaugh Map ~

• 3 variable: (x, y, z)

$z \backslash xy$	00	01	11	10
0	0	2	6	4
1	1	3	7	5

• Note: (4,3) side-by-side as they differ only by 1 literal

Now $f(x,y,z) = xyz' + x'y'z' + xy'z' + x'yz + xyz + xy'z$

$z \backslash xy$	00	01	11	10
0	1	1	0	1
1	0	1	1	1

$\therefore f(x,y,z) = \Sigma(0,2,3,7,4,5)$

• 4 variable: $(wxyz)$

/* 1st come 1st served */

$yz \backslash wx$	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

/* ✓ 9 is 1001 and 1 is 0001 so they can also be thought to be adj.

- So 1st & last column/row are also adjacent.
- So 9 can combine them.

The above combⁿ produces: $xz' + yz + xy'$ /* pencil one */

: $y'z' + x'y + xz$ /* <3 ^° ソは */

- As we are searching for those combns where Aa' & Aa so we need clubs of 2^n.

Eg:

$z \backslash xy$	00	01	11	10
0	0	2	6	4
1	1	3	7	5

Say $f(x,y,z) = \Sigma(0,1,2,3,5)$

$\Rightarrow f = x' + y'z$ //

- Selections made in Karnaugh maps are called as 'CUBES'.

eg:

$yz \backslash wx$	00	01	11	10
00	0	4	12	8
01	1	5	13	9
11	3	7	15	11
10	2	6	14	10

$f(w,x,y,z) = \Sigma(0,4,5,7,8,9,13,15)$

$= xz + w'x'y' + w'y'z'$

ॐ श्री कृष्ण

Date 08/11/19 (金)

Contents

농심라면

Objectives:

1. Cover all 1's with min. cubes. → signifies no. of pdt terms
2. Cubes are as large as possible → size of each pdt term

Steps:

1. Try to cover all 1's cell that cannot be covered by any other 1's cell. i.e. can only be covered by cube-size 2.
2. Select next cube-size = 4, 8 ....
3. Select min no. of large cubes that cover all 1's cell.

* IMPLICANT

$f(x, y, z) \sim g(x, y, z)$

g is an implicant of f if

1.) $g \subseteq f$

2. g is a pdt term

→ Prime Implicant:

An implicant that can't be reduced further.

So in a Karnaugh map, we are basically identifying the prime implicants.

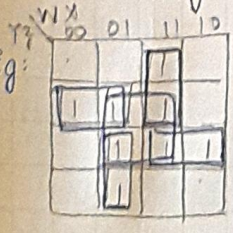
- Any subset of prime implicants that cover all 1's cell is a irredundant expression.

→ Essential Prime Implicant: A prime implicant which will be part of any irredundant expression.

- In the above example xz is an essential P.I.
- Rsn :- If a prime implicant covers an implicant that isn't covered by any other implicant. (→ prime)

- Any non-essential P.I is covering someone i.e. is covered by other P.I's.

Eg:



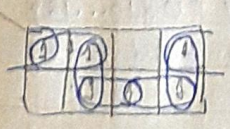
→ There are 5 P.I's

→ for minimal expression: essential P.I's are a must

→ So the smaller & mem ones are a must.

→ The bigger 4-mem one is redundant.

→ Here all P.I's (4) are essential. → unique irredundant expression which is minimal.



→ None of the P.I's are essential as every P.I is covered by atleast 2.

No. 04
Date 08月19日
(수)

Minimal P.O.S. & Don't Cares.

Steps: 0. Determine all PI's.

1. Determine all essential prime implicants and include them in the minimal expression.
2. Remove prime implicants that are covered by EPI's.
3. If step-2 covers all 1's cells then ⇒ gives a unique minimal expression
4. Otherwise select additional prime implicants ∋ OBJ. are met.

Quine Mccunskey

* Minimal POS *

Q: Find minimal POS for $f(w,x,y,z) = \Sigma\{5,6,9,10\}$.

/* It is not advisable to calculate min. POS by ~~E~~ mm. SOP & it sometimes does not give the minimal exp */

yz \ wx	00	01	11	10
00				
01		1		1
11				
10		1		1

yz \ wx	00	01	10	10
00	0	0	0	0
01	0		0	
11	0	0	0	0
10	0		0	

/* Applying K-map on 0's */

/* If we had taken 1's and then taken comp → 16 literals, so i... */

$\Rightarrow y'z' + w'x' + yz + wx$ → f'

$\therefore f = (f')'$

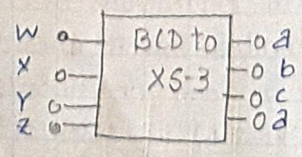
$= (y'z' + w'x' + yz + wx)'$ /* Demorgan's */

$= (y+z)(w+x)(y'+z')(w'+x')$

* Minimization with don't cares *

- In some cases we ~~are~~ only require some of the encodings (valid).
- Ex. BCD.: 0-9 ✓ (valid inputs)
  10-15 (don't cares)

	BCD				XS-3			
	w	x	y	z	a	b	c	d
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



→ Don't cares

$a = f_1(w,x,y,z) = \Sigma\{5,6,7,8,9\} + \emptyset\{10 \ldots 15\}$

$b = f_2(w,x,y,z) = \Sigma\{1,2,3,4,9\} + \emptyset\{10 \ldots 15\}$

$c = f_3(w,x,y,z) = \Sigma\{0,3,4,7,8\} + \quad "$

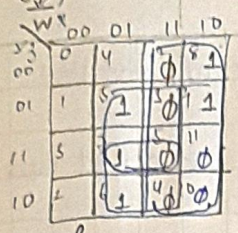
$d = f_4(w,x,y,z) = \Sigma\{0,2,4,6,8\} + \quad "$

Date: 05月19日 (金)

Contents

Basic Arithmetic: (1) $x+x=x=x\cdot x$
(2) $x+x'=1$; $x\cdot x'=0$
(3) $x+y+z=(x+y)+(x+z)$

Simplificatn Rules:
1. Absorptn: $x+xy=x(1+y)=x$
2. Simplificatn: $x+x'y=x+y$
3. Consensus: $xy+x'z+yz=xy+x'z$



*fa

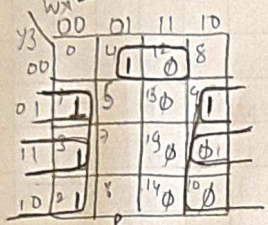
1. Without don't cares:

Our minimisation is : $w'xy + wx'y + w'xy'z$

2. with don't cares:

/* Our logic is that we treat ϕ's as 1's AT OUR CONVENIENCE.

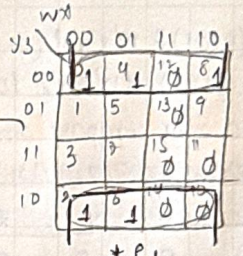
$\Rightarrow$ $xz + xy + w$



*fb

2. $b = xyz' + x'y'z + x'y$

$x'z$



*fd

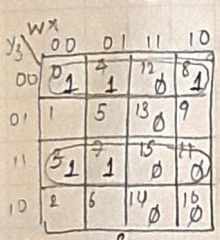
4. $d = y'z' + yz' = z'$

3. $c = y'z' + yz$.

Hence Now

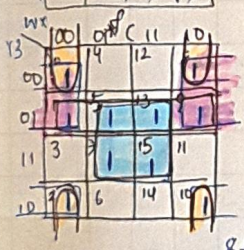
$f_a = xz + xy + w$ $\quad$ $f_c = y'z' + yz$

$f_b = x'y + x'z + xy'z'$ $\quad$ $f_d = z'$



Ex-1:

Solve $f(w,x,y,z) = \Sigma(0,1,2,5,7,8,9,10,13,15)$

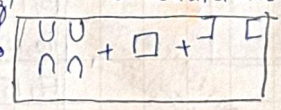


*fc

$\Rightarrow$ Ans $= x'y' + xz + x'y'z'$ -(1)

(2) $\because$ $y'z + xz + x'z'$ $\rightsquigarrow$ Irredundant.

(OR) $x'y' + xz + x'z'$



Stepwise: Find all prime Imp

$\rightarrow$ $\{x'y', xz, x'z', y'z'\}$

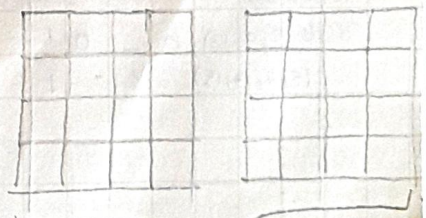
(ii) Find Essential PI $\{xz, x'z'\}$

(iii) $x'y'$ (or) $y'z'$ (selection of these) and

$\rightarrow$ So we have 2 min expressions.

// : To visualise 5 variable map take two blocks of 4 blocks. for 6 variable (4)x4 blocks.

Note: It becomes very difficult to solve/minimise using Karnaugh Maps for higher vables so we will use QUINE MCUNSKEY.



A 5 vable map.

No. 01

Date 05月22日 (月)

Contents

~ Quine Mcunskey - Tabulation Method. ~

→ Instead of directly finding cubes of size 4, 8 ... etc, I'll rather indentify and find cubes of size - 2. Most of these are implicants.

→ In ~~so~~ the 1st iteration if some implicants are prime; I'll keep them and extend those which ain't.

→ Continue.

→ 하오: $w'xy'z$ (0101) & $w'xyz'$ (0110) = 0111 ($wxyz$)

$w'xyz$ (0111) & $w'xy'z'$ (0100) = 0111 ✓

• I' should only compare and combine terms that differ by single bit

Take $f = \Sigma(0,1,2,5,7,8,9,10,13,15)$:

	w	x	y	z
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
8	1	0	0	0
5	0	1	0	1
9	1	0	0	1
10	1	0	1	0
7	0	1	1	1
13	1	1	0	1
15	1	1	1	1

Combe Compared in one gp only

** Adj gps are compared & we don't have to compare n² pdt terms

Groupings based on no. of 1

	x	y	z	w
(0,1)	0	0	0	–
(0,2)	0	0	–	0
(0,8)	–	0	0	0
(1,5)	0	–	0	1
(1,9)	–	0	0	1
(2,10)	–	0	1	0
(8,9)	1	0	0	–
(8,10)	1	0	–	0
(5,7)	0	1	–	1
(5,13)	–	1	0	1
(9,13)	1	–	0	1
(7,15)	–	1	1	1
(13,15)	1	1	–	1

// If (0,8) wasn't clubbed, it wd have been 8 F.

// X(1,10) as 2 bit diff

ONLY COMPARE THOSE WHERE – IS IN SAME PLACE

	w	x	y	z
(0,1,8,9)	–	0	0	–
(0,2,8,10)	–	0	–	0
~~(8,1,9~~ ~~(1,8,9)~~	~~–~~	~~0~~	~~0~~	~~–~~
(1,5,9,13)	0	–	0	1
(5,7,13,15)	–	1	–	1

$x'y'$

$x'z'$

(already cons.)

$y'z'$

xz

* All implicants of size 2 I* In the Karnaugh map*

→ Ir: If all pdt term is carried fwd;

• when all iterations are over if a pdt term doesn't have a tick becomes a PF

* Prime Implicant Chart:

	0	1	2	5	7	8	9	10	13	15
A	X	X				X	X			
B	X	X	X			X		X		
C		X		X			X		X	
D				X	X				X	X

* Here ABCD represents the PF

* Put crosses if a p.I has those.

* Select those P.I. which have 'X' that occur only once in a corr column

* Hence B, D [$x'z'$ & yz] are the P.I's.

08月22日 (月) ~ 08月25日 (木)

~ Selecting Prime Implicants ~.

- Because we already selected B & D ⇒ 0, 2, 8, 10, 5, 7, 13, 15 are already selected.
- I need to select a PI that contains 1 & 9.

Q:

$f(v,w,x,y,z) = \Sigma(13,15,17,18,19,20,21,23,25,27,29,31) + \phi(\Sigma(1,2,12,24))$. Find E.P.I & P.I's.

|*|r*|: For don't cares,

1. Consider them only in 1st iteration
2. If some of the don't cares can't be combined in 1st iteration, ignore them.

|*| For the above $f(v,w,x,y)$; the PI's are:

A: vx ; B: wxz ; C: $vwx'z'$; D: $vw'xz'$; E: $vw'xy$; F: $v'wxy'$; G: $w'x'yz'$; H: $wx'y'z$.

A PI	13	15	17	18	19	20	21	23	25	27	29	31
A: vx			X		X		X	$\boxed{X}$	X	$\boxed{X}$	X	X
B: wxz	X	$\boxed{X}$									X	X
C: $w'x'z'$									X			
D: $vw'xz'$						$\boxed{X}$	X					
E: $vw'xy$				X	X							
F: $v'wxy'$	X											
G: $w'x'yz'$				X								
H: $wx'y'z$			X									

C is cov. with D

- To cover 18 either of E | G needs to be selected.

Hence $f = A+B+D+E \mid A+B+D+G$.

Q: If ~~there~~ it not obvious to us as to which PI needs to be selected

★ Reduced Chart:

1. Remove all rows corresponding to essential prime implicants (EPI)
2. Remove a column that gets covered by EPI.

	18
E	X
F	
G	X
H	

Reduced chart.

Exe: $f(v,w,x,y,z) = \Sigma(0,1,3,4,7,13,15,19,20,22,23,29,31)$:

HW (19, 21, 27, 1, 2, 4) → find the PI's

08月25日
(木)
08月26日:

今何だ

P.I's

A = wxz
B = xyz
C = $w'yz$
D : $vwxy$
E : $vw'xz'$
F : $w'xyz'$
G : $v'wxy$
H : $v'w'y'z'$
I : $v'w'x'y'$

EPI

RARE CASE:

→ Ignore this as all terms in it are already covered by EPI.

Reduced Chart:

	0	1	4	20	22
D					X
E				X	X
F			X	X	
G		X			
H	X		X		
I	X	X			

→ Now it is not obvious as to which PI should be considered (2 per column)

→ Take a boolean expression $P = (H+I)(G+I)(F+H)(E+F)(D+I)$

$= EHI + EFI + DFI + EGH + DFGH$ → contains 4 terms

as each of D, E, F, G, H contain 4 terms so

→ So, finally there are 4 expressions:

1. $A+C+E+H+I$
2. $A+C+E+F+I$
3. $A+C+D+F+I$
4. $A+C+E+G+H$.

Steps:

1. Identify all the P.I's using tabular method.
2. Draw the PI chart
3. Select essential PI's
4. Select Non EPI
   - Reduced P.I chart
   - Boolean technique
   - Branching mtd.

Eg:
Say in a reduced chart:

A	X	X
D	X	
I	X	X

Selecting either A/E suffices.

~* Reduction of rows and/or columns in P.I. chart:

1. ROWS

	10	11	18	19	26
C			X	X	X
D			X	X	
E	X	X			
F	X	X			
G		X			X
H					X
I		X			

(i)

→ D || C
→ E || F
→ G || H + I → so consider C, E, G instead.

	10	11	18	19	26
C			X	X	X
E	X	X			
G		X			X

(ii)

- Now C & E are the new 'pseudo' EPI's
- But this couldn't be concluded from (i).

Date 08/12/26 日(月)

Contents

? Boolean

I: Remove redundant columns:

Taking the same eg.

	10	11	18	19	26
C			X	X	X
D			X	X	
E	X	X			X
F	X	X			
G		X		X	X
H				X	
I		X			

~ Logic is opposite to that in removal of rows

→ Take the columns 10 and 11

- 10 requires selection of 10 and E & F; whereas 11 requires E, F, G, I. Weisheme would I take G and I in addn
- So remove 11
- Similarly remove 19; as we already have a better 18
- And 26 is also of no necessity as it's ele have already been covered.

- Eg: $f(w,x,y,z) = \Sigma(0,1,5,7,8,10,14,15)$
  0 1 3 2 3 4

	w	x	y	z	
0	0	0	0	0	✓
1	0	0	0	1	✓
8	1	0	0	0	✓
5	0	1	0	1	✓
10	1	0	1	0	✓
14	1	1	1	0	✓
7	0	1	1	1	✓
15	1	1	1	1	✓

yz \ wx	00	01	11	10
00	X			X
01	X	X		
11			X	
10			X	X

PI: $w'y'z'$ ①
$wx'y$ ②
$wx'z'$ ③
$x'y'z'$ ④

It is thus difficult to obtain ALL PI's using K-map.

These allow P.I's.

(0,1)	0	0	0	–	[8,10]	1	0	–	0
[0,8]	–	0	0	0	[10,14]	1	–	1	0
(1,5)	0	–	0	1	[5,7]	0	1	–	1
[10,14]	–	1	1	0	[7,15]	–	1	1	1
[14,15]	1	1	1	–					

All these 8 are P.I's

	0	1	5	7	8	10	14	15
A	X	X						
B	X				X			
C		X	X					
D						X	X	
E							X	X
F				X	X	X		
G			X	X				
H				X				X

- CYCLIC PATHS & BRANCHING

Now there aren't any exclusively visible EPI

Using Boolean mtd:

→ $(A+B)(A+C)(C+G)(B+F)(D+F)(D+G)(E+H)$ ..

→ 7 terms (pdt)

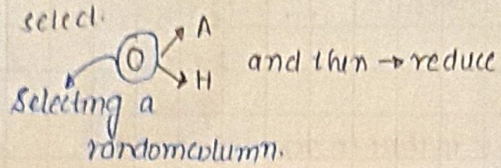
↳ But only 4 terms suffice.

Such a chart is called CYCLIC PI CHART

08/12/26 (9)
08/12/26 (1)

~ Heuristic Based Switching for Minimisation ~

- Arbitrarily select one term: (Since there are no ...)
- for 0 select:



and then → reduce

→ Apply the same rules

	1	5	7	10	14	15
A	✓					
B	✓	✓				
C		✓	✓			
D			✓		✓	
E					✓	✓
F				✓	✓	
G			✓			

B + D + F + H //

A + C + E + G

	5	7	8	10	14	15
B	✓	✓				
C	✓	✓				
D		✓				✓
E					✓	✓
F			✓		✓	
G			✓	✓		
H			✓			

* Heuristic Based Switching for Minimisation:

- Time complexity of Quine McCluskey:-
  - → we divide the inputs into groups based on no. of 1's.

$$\left.\begin{array}{l} 0\ 1's \rightarrow {}^nC_0 \\ 1\ 1 \rightarrow {}^nC_1 \\ \vdots \\ i\ 1's \rightarrow {}^nC_i \end{array}\right\} O(n) \text{ to write}$$

  - → Now in one iteration, comparing b/w two grps → $\Theta(n^3)$
  - → For n iterations → $O(n \times n \times n^3) = O(n^5)$
  - → Just to findout the prime implicants → $O(n^5)$
  - → Now for n vables we've atmost 2^n minterms; and no. of PI's in the worst case $O(3^n/n)$
  - → Hence Quine McCluskey method has very high time-complexity.

- Basic idea of Heuristic soln is:
  - ~ It gives near to (close) optimal soln that takes much less time.
  - → Say minimal Exp has 20 lit by QM → 5 min
    by heuristic say 21 lit (by H.I) → 3 min ✓

* ESPRESSO

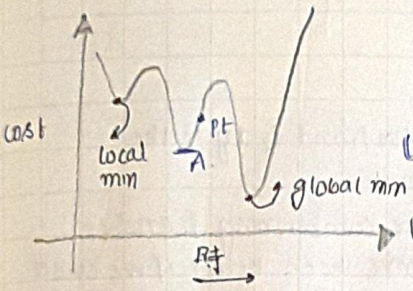
- → Fast time algorithm.
- → The pblm with QM is that they try to find out all the PI's which is exponential
- → The 1° intution here is that they never find out all the PI's.

Date 08月27日(土)

Contents

~ ESPRESSO ~

- It always keeps a set of PI's that covers all the minterms.
- It adds / removes minterms based on 時 available.
- It can stop anytime.
- Simulated Annealing Approach.



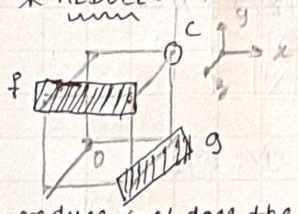
Fig(i) Heuristic Algo

(i) In Simulate An. Approach; start from a random st (P)
(ii) In our case; it is the given fn. (pdt term / minterms)
(iii) It minimises and reaches the nearest stable pt (reaches some local minima) (say A)
(iv) Then it tries to generate a random starting pt (by removing some PI (adding some PI) and keeps on going till it reaches the global minima.

```
do{
    do{
        reduce();
        expand();
        irredundant();
    } until cost stable
    reduce_gap()
    expand_gap()
    irredundant()
} (until time up)
```

```
Espresso{
    Expand();
    Irredundant()
```

* REDUCE:

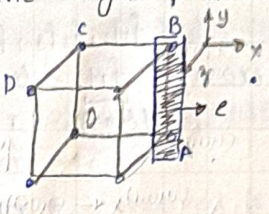


- reduce just does the opposite of expand.

→ It converts f, c, g back to {f1, f2, c, o, g} in one step.

* Expand:

- The original fn is the onset / impli-cants.

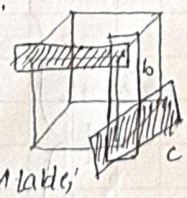


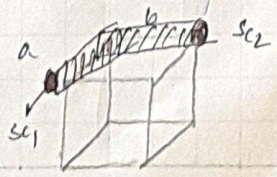
- vertices are the minterms

- Say initially minterms {0, b, c, d} = S
- It takes one / more terms in the set and tries to make a larger minterm
- S' = {e, c, d}
- Similar to QM table but we aren't trying to find out all the PI's here.

* Irredundant:

- Throw out the redundant expressions.
- In the example shown; b is a redundant expression.
- It removes b.
- It is similar to the PI chart in QM table, but since no. of PI's are less; it is faster.





* USES:

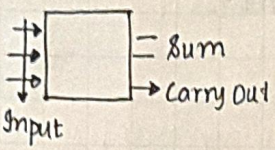
- Reduce: We got a soln; but not good; so we reduce the expression and again expands to find a better soln

* REDUCE-GAP:

- Rather than reducing randomly,
- For each minterm s; add those subcubes that are not covered by other subcubes.
  - ((They can be new starting pts.

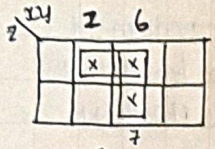
08月28日 (月)

~ Multi Function / Output Switching for Min ~

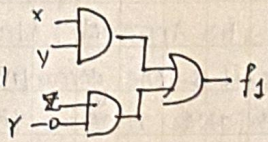


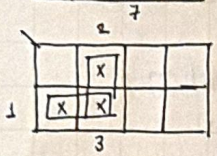
In logic minimalisation, we might lose the opportunity to

- Adder Circuit

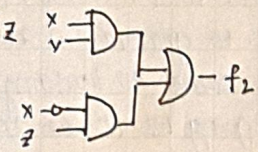


: $xy + yz'$





: $x'y + x'z$



- Now here b/w f_1 and f_2 xy is the common part.
- If we were combining f_1 and f_2 it would have been better if we could have used xy only once (commonly).

2 functions
f_1
f_2
f_1f_2

3 fns:
f_1, f_2, f_3
$f_1f_2f_3, f_1f_2; f_2f_3; f_3f_1,$

For n functions → $2^n - 1$ total possible combinations.

In our example $\underbrace{f_1f_2}_{(sim)} = (xy+yz')(x'y+x'z)$

$= xy + xy(x'z + yz') + x'y\,zz'$ (0)

$= xy + xyz' + x'yz$

$= xy(1) + x'yz$.

but actually:

$f_1 = x'yz + xyz' + xyz$; $f_2 = x'yz' + xy'z + x'yz$

f_1f_2: $x'yz$,

Steps:

1) Identify all prime implicants for $f_1; f_2 \cdots f_n$
2) Now take the Augmented Prime Implicants Chart.

/* Note say $x'y$ was common in f_2 and f_1f_2; select that in f_1f_2 to avoid renundancy */

1. A PI Chart

		f_1			f_2		
		[2]	6	7	[1]	2	3
f_1	xy		X	X			
	yz'	X	X				
f_2	$x'y$				X		X
	$x'z$					X	X
$f_1f_2f_3$	$x'yz$						✓

Explanat^n.

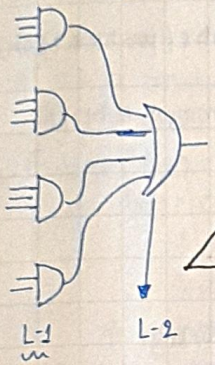
2. Reduced Chart.

Date Contents

05月29日(月)

Multi Level Logic Optimisation (コハビ-chap6)

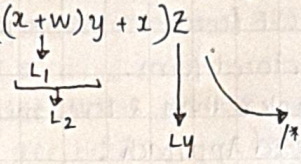
- Two level is SOP (or) POS [“and” and “or” constitute the two levels]
- Eg: $xyz + wxz + wyz + xz$.

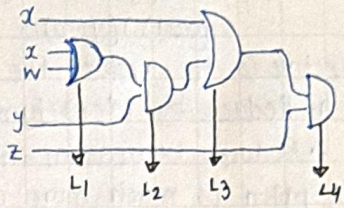


↳ It is also equivalent to

$(x+w)yz + w(xz)$

$((x+w)y + x)z$





/* like a recursive SOP definition */

- Many a time, 2 level representation might not be apt
- Has a larger “delay”. 「Delay (or) computational delay is the total time required to compute the output, by the circuit」 (In combinational circuit)
- 「Delay: In general sense is the longest combinational path between two registers」.
- 2 level circuits have fixed delay and is less than the multilevel } PROS of 2L

- In 2-level; our objective was to minimise the no. of minterms. In multilevel circuit this defn / objective is vague. So our objective is to minimise the no. of literals in a boolean circuit. } PROS of multilevel
- In the later part of this course: a circuit with n-variable → with 2n transistors.

Eg:

In the parity check code // that checks whether the no. of 1s received is even; we need to design a circuit that does so:

$\bar{A}$	A	B	C	D
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0

BC \ AA				
		1		1
	1		1	
		1		1
	1	1		

↳ Karnaugh map for $P(\bar{A}, A, B, C)$

↳ 8×4 ~ 32 literals (/minterm)

* But using multi level optimization; $P = (A \oplus B) \oplus (C \oplus D)$ → 4 literals.

// Extend to all 16 poss. inputs.

~Algebraic Methods for Multilevel Opt~

Eg: Eg: $xyz + xzw + vy + vw$
↳ Cmn subexp: $xz(y+w) + v(y+w)$
$= (xz+v)(y+w)$

- Our objective is to reduce the literal count.

How? : 1. Reduce by identifying CSE (common subexpression) — Cube (product term), Common subexpr.
2. Reduce expression in factored form.

↳ This pblm is much more complex than 2-level minimisation. We shall use "Heuristic Based Approach".

* Algebraic Method *

- I'll consider the boolean fn as a (represented as) Algebraic expression
- As we always start with SOP; the boolean prop. are anyway not utilisable.
- The only thing to be done is to replace the x' by some variable. I'll use the division oprn.

Differences:

Boolean	Algebra
$x \to x'$	×
$x + x' = 1$	×
$x \cdot x' = 0$	×
demorgan	×
⋮	

// Division Algo:

$$Divi = Q \cdot (d) + r.$$

Eg: $\underbrace{(a+b)}_{Quo} \,|\, \underbrace{a(c+d) + bc + bd + e}_{dividend} \,|\, \underbrace{(c+d)}_{Divisor} \to$

e (remainder) ↓ Q

$a(c+d) + b(c+d) + e$

$(a+b)\underbrace{(c+d)}_{Q} + \underbrace{e}_{R}$

- Obj: Look for CSE:

Q: where to look in?

A: Look in the kernel of the f^N. // Kernel → SOP with atleast two product terms. (cube). → Cube-free.

Cube | D | d | Q → Kernel 'Q'
co-kernel ↓ R

- atleast two cube | pdt term.
- + cube free.

// Cube free: doesn't have any common factor. (sub-product term)

// Eg: $ab + ac = a(b+c)$ Not a kernel
$ab + ac + d$ → a kernel.

- ↳ when I divide the fN by a cube | pdt term; when the quotient has ≥ 2 cube | pdt terms and is cube-free
- then that 'Q' is a kernel and the divisor is co-kernel.

Eg: Identify the kernels of:

$$f_D = ace + bce + de + g.$$

- Note: If D = 1 then Q is d is a kernel
- So $a f^N$ can be a kernel as well.
- So this fN has 3 kernels

D	Quotient of D/d: $ace+bce+de+g$	Q is kernel
a	ce	×
c	ae+be	×
e	a+b+d	✓
ce	a+b	✓

(ST) 09月 01日 (木) & 09月 05日 (月)

How to find out all kernels?

- 1. Identify all product terms and divide 'f' with each of them.
- 2. (Commutativity) : ac & ca are the same.
- 3. If a product term f occurs only in one pdt term of f; ignore it
  Exp: In the above 'f'; d; g occurs only once (in 1 pdt term) → ignore it.
- 4. Cubes (f, x) : Set of product terms in which x ~~co~~ occurs
  Say cubes(f, e) = {ace, bce}
  & cubes(f, c) = {ace, bce, d}.
  ∴ Cubes(f, c) ⊆ cubes(f, e)
  ⇒ ignore both 'c' & 'e' (individual) rather consider xy.
- 5. Define the levels of the kernels.
- 6. Apply recursively to obtain the factored form.

Eg: $xyz + wxy + yzw$:

- Levels of a kernel:
  - If a kernel has no kernel except itself it is of level 0:
  - If a kernel has atleast one kernel of level n-1 (at most) then it is a n-level kernel
  - Cokernels of a kernel has the same level.

Eg: $f = uwz + uxz + vwz + vxz + yz + uv$.

L-0	Co-kernel	Kernel
0	wz, xz	$w + v$
0	uz, vz	$w + x$
1	u	$wz + xz + v$
1	v	$wz + xz + u$
1	z	$uw + ux + vw + vx + y$
2	1	f

- **Intersection of Kernels:**

Algebraic M

1.

- Largest subset of common cubes.

$K(f_x) = \{ \underset{K_1}{(ac+bc+d)}, \underset{K_2}{(a+b)}, \underset{K_3}{(ace+bce+de+g)} \}$

$K(f_y) = \{ \underset{K_4}{(a+b+ce)}, \underset{K_5}{(cd+g)}, \underset{K_6}{(ad+bd+cde+ge)} \}$.

→ $K_1 \cap K_2 = \phi$; $K_2 \cap K_4 = a+b$; $K_3 \cap K_4 = \phi$

$K_1 \cap K_5$:

- 1st find corkernels in 1° rectangle & kernel in co-rectangle
- then $f = f_d f_q + r$ //algebraic where f_d f_q are algebraic
- Now simplify f_d & f_q; for a recursively following step 1 ~~till~~ as ~~much~~ far as poss
- Now combine common exp in $(f_d \cdot f_q)$ & f_r".
- Now literal count has decreased
- This mtd is : brute force

*Cube Literal Incidence Matrix : // All cubes are along the columns
// All the literals along header row

$f = UWZ + UXY + YZ + UV$

Cubes	U	V	W	X	Y	Z
UWZ	1	0	1	0	0	1
UXY	1	0	0	1	0	1
YZ	0	0	0	0	1	1
UV	1	1	0	0	0	0

Literals

* The elements $M[\,][\,] = 1$ if that literal is used in the cube else 0

- **Rectangle**
  → Subset of rows and columns in which all entries are 1.

	U	Z
UWZ	1	1
UXZ	1	1

These are CSE

Date 03 09月 05日 (月)

Contents

Co Rectangle: If (r, c) denote the rows subset and columns subset of the K-I Matrix then $(r, \bar{c})$ corresponds to the $\bar{c}$-rectangle of in that above Matrix

- Note: If (r, c) is such that NO OTHER LARGER rectangle contains it then it is called as PRIME RECTANGLE

  Eg: Co-rectangle $(\{uvz, uwz\}; \{v, x, y, w\})$ of the rectangle:

  $\{(uvz, uwz), \{u, z\}\}$ (r, c, $\bar{c}$)

- In the above rectangle;

Prime rect:
- $[\{uwz, uxz\}]; [\{u, z\}]$ for
- $[\{uwz, uxz, uv\}; \{u\}]$
- $[\{uwz, uxz, vz\}; \{z\}]$

are the prime rectangles

WITH

Corr. Co-rect:
- $[\{uwz, uxz\}; v, w, x, y]$
- $[\{v, w, z, uxz, ux\}; \{v, w, x, y, z\}]$
- $[\{uwz, uxz, z\}; \{v, w, x, y\}]$

being the corresponding co-rectangles

- Search for the co-kernels in the r:- $\{uwz, uxz\}$
- For us to search for co-kernels in a prime rectangle, no. of rows in it must be atleast 2,, wx, zx uz → $w+x$ ✓ (co-kernel, kernel)
- Then the corresponding kernel will be available in the co-rectangle; limited to the literals in the $\bar{c}$.

  Eg: in $\{uwz, uxz\}; \{v, w, x, y\} \Rightarrow w+x$ is a co-kernel

*After this recursively apply the above method on f_d, f_q & f_r and report the simplest ans

*EXTRACTION
- Cube extraction
- Kernel extraction

- Cube Extraction

  Consider that our network has $f_1; f_2$ and we need $f_3 = f_1 + f_2$ and $f_1 = uwz + uxz + yz + uv$ and $f_2 = vz + wyz$.

  - While finding f_3 we would like to have common subexp to simplify our tasks.

  (1) Make an aux. fn $f_a = f_1 + f_2 = uwz + uxz + yz + uv + vz + wyz$

  (2) Then make a cube literal incidence matrix but CONSIDER ONLY THOSE 1° rectangles which are spread in f_1 & f_2

- Algeb exp: If an exp doesn't look as if it can be simp further using boolean exp.

  else Boolean exp.

  Alg. Boolean factors, quotients, divisions

(3) For our example $[\{yz, wyz\}, \{y, z\}]$ is one such 1° rect

// Note don't club $yz + wyz$ or think that it could have been basis

// As both belong to separate fns initially

Cube ↓	Id	u	v	w	x	y	z
uwz	f_1	1	0	1	0	0	1
uxz	f_1	1	0	0	1	0	1
yz	f_1	0	0	0	0	1	1
uv	f_1	1	1	0	0	0	0
vz	f_2	0	1	0	0	0	1
wyz	f_2	0	0	1	0	1	1

Now $f_1 = uwz + uxz + f' + uv$

$f_2 = vz + wf'$

where $f' = yz$.

* Kernel Extraction:

Now f_1's kernels are $K(f_1) = \{(w+x), (uw + ux + y)\}$

f_2 " " $K(f_2)$ $\{w + x + yz\}$

Now rem these as cubes $f_1 = (aw, ax); (avw, aux, ay)$

$f_2 = \{aw, ax, ayz\}$

Kernel ↓	ID	aw	ax	avw	aux	ay	ayz
w+x	f_1	1	1	0	0	0	0
uw+ux+y	f_1	0	0	1	1	1	0
w+x+yz	f_2	1	1	0	0	0	1

Now,

consider the 1° rectangle spread through both f_1 & f_2.

* Decomposition:

09/7
08/7 (+)

Combinational Logic Design [J/K'-6]

I. XS-3 to grey code (or) some other cyclic code

Sl	w x y z	a' b' c' d'	a b c d
0	0 0 1 1	0 0 0 0	0 0 0 0
1	0 1 0 0	0 0 0 1	0 0 0 1
2	0 1 0 1	0 0 1 1	0 0 1 1
3	0 1 1 0	0 0 1 0	0 0 1 0
4	0 1 1 1	0 1 1 0	0 1 1 0
5	1 0 0 0	0 1 1 1	1 1 1 0
6	1 0 0 1	0 1 0 1	1 0 1 0
7	1 0 1 0	0 1 0 0	1 0 0 0
8	1 0 1 1	1 1 0 0	1 1 0 0
9	1 1 0 0	1 1 0 1	0 1 0 0

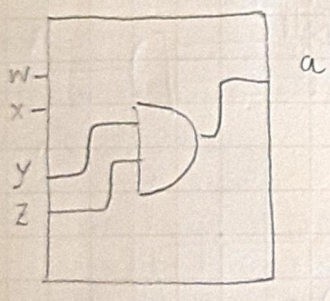
Only these circled elements are intersecting so consider these.

yz \ wx	00	01	11	10
00				1
01				1
10				1
10				1

fig: Not to scale,

// $f_3 = a = \Sigma(5,6,7,8)$

Similarly, find outputs for the others.



II. Parity Bit Converter:

→ Even parity:

w	x	y	z	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0

→ Sometimes K-map doesn't give the optimal design.

→ We must use multilevel optimisation

$f_1 = x'y'z + xy'z' + xyz' + \cdots$

$= x \oplus y \oplus z$.

III. If A, B are n bit no.s let's define:

if A > B : $f_g = 1$

elif A = B : $f_e = 1$

else : $f_f = 1$

// Consider A; B to be 2 bit no.s → so there are actualy $2^2 \cdot 2^2 = 16/2$ → outcomes, ½ max.

A_1 A_0	B_1 B_0	f_g	f_e	f_l
0 0	0 0	1	0	0
0 0	0 1	0	0	1
0 0	1 0	0	0	1
0 0	1 1	0	0	1
1				
0				

Out of those 16 are now ✓ ut.

B_1B_0 \ A_1A_0				
	2	1	1	1
	3	2	1	1
	1	3	2	3
	3	1	1	2

1: f_g
2: f_e
3: f_l //

fig: 4 bit comparator.

~MuX and Encoder~

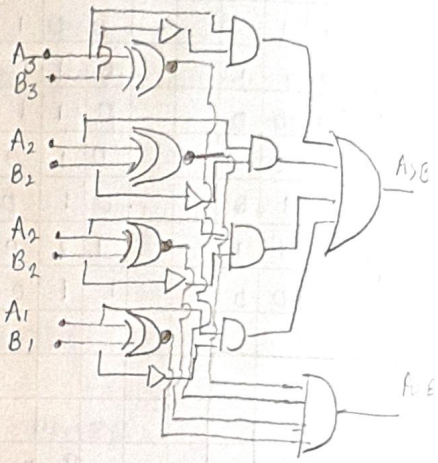
- 4 bit comparator:

$A = A_3 A_2 A_1 A_0$ $\quad A > B \Rightarrow A_3 B_3' + x_3 A_2 B_2' + x_3 x_2 A_1 B_1' + x_3 x_2 x_1 A_0 B_0'$

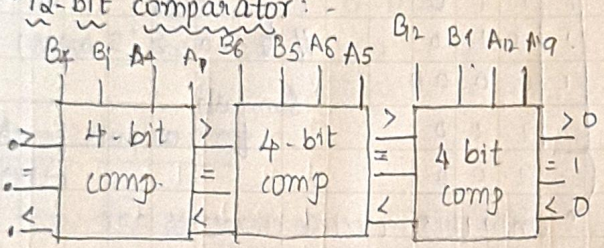
$B = B_3 B_2 B_1 B_0$ $\quad A = B \Rightarrow x_3 x_2 x_1 x_0$

$A < B \Rightarrow A_3' B_3 + x_3 A_2' B_2 + x_3 x_2 A_1' B_1 + x_3 x_2 x_1 A_0' B_0$.

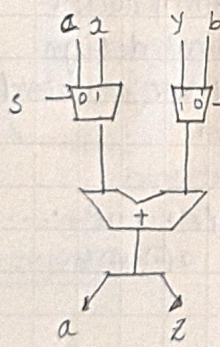
Let $x_i = A_i B_i + A_i' B_i' \; \forall \; i \in [0, 3]$
↳ XNOR



- 12-bit comparator:



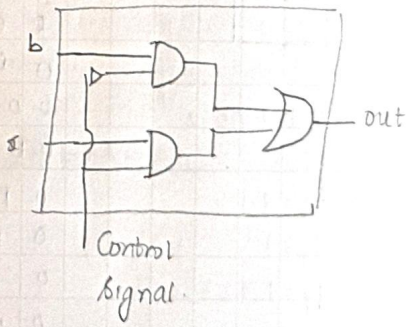
* Multiplexer /* multiplexes inputs */



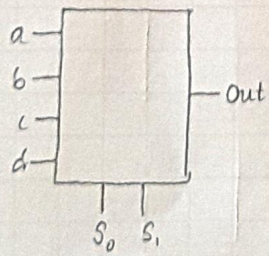
In clock-1: $a = b + c$.
clock-2: $z = x + y$.

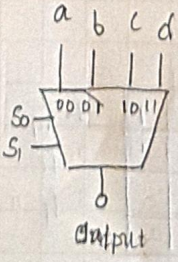
(2:1 MUX with a "cut")

// Having a cut signifies multiple input

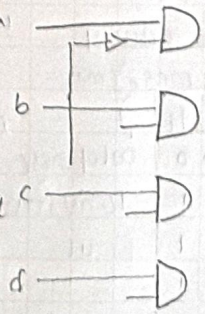


* 4:1 MUX:





// Notice that there aren't any cuts.

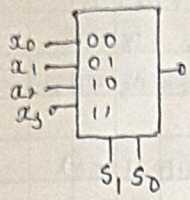


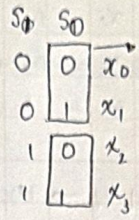
Date 09/1
04 09/1
(ॐ)

~ MuX and Encoder ~ [1/2-6]

(1) Q: Design a 4:1 MUX with a 2:1 MUX(s).

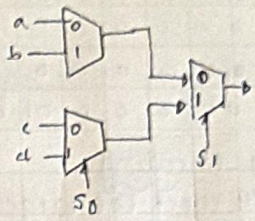
Sol^n





→ 1 MuX selects S_0

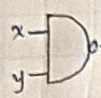
The other MuX enables choice of MuX



Note the equation for a 2:1 MuX is $S'a + Sb$

(2) Realise a NaNd gate using MuX.

Sol^n



→ eqN for a NaND gate:

$x' + y'$

→ putting $x = S$; $a = 1$

$S' = 0$; $b = y'$.

$\therefore f = x' + xy' = x' + y'$

(3) Implementing a Boolean f^N using MuX.:

Eg: $f = \Sigma(1,3,5)$

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

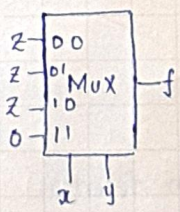
$\therefore \Rightarrow f = x'y'z + x'yz + xy'z$ (~ $x'z + xy'z$
$= z[x' + xy')]$
$= z[x' + y']$.)

→ Multi-level simplific^n.

We shall use a 4:1 MuX

→ and we would require 2 Select lines for this.

∴ Design^N:



Encoder

2^n inputs into n-outputs.

* We take an assumption that D_8 comes out to be 0.

D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	x_2	x_1	x_0
1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	1	1

1. Atleast one of the inputs is 1
2. Cannot distng b/w all 0's case

If SOP = $\Sigma(0,1,2,4,6,8,9) + \Sigma_{\phi}(5,7,11)$

$(f')' = \Pi(15,14,12,11,9,3,1) \cdot \Pi_{\phi}(?)$ 15, 14, 10, 13 (on 5, 1, 11 (on all)

0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

$x = D_4 + D_5 + D_6 + D_7$

$y = D_2 + D_3 + D_6 + D_7$

$z = D_1 + D_3 + D_5 + D_7$

* Priority encoding:

- I'll check the largest bit that contains 1. (say D_6 → then output is 6)
- And then ignore the remaining bits.
- And ignore all 0 case [so I'll include a valid checker that sets V to 1 for all 0 cases]

* Priority encoding for n=2:

V	D_3	D_2	D_1	D_0	x_1	x_0	V
0	0	0	0	0	x	x	0
1	0	0	0	1	0	0	1
1	0	0	1	x	0	1	1
1	0	1	x	x	1	0	1
1	1	x	x	x	1	1	1

- So there are 2, 4, 8 combns for 01, 10 & 11 respectively.
- The Kmap for this is:

D_1D_0 \ D_3D_2	00	01	11	10
00	0 X	4	12	8
01	1 1	5 1	13 1	9 1
11	3 1	7 1	15 1	11 1
10	2 1	6 1	14 1	10 1

for $x = D_2 + D_3$

Similarly

$y = D_2 + D_1 D_2$

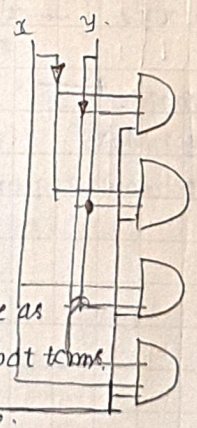
$V = D_0 + D_1 + D_2 + D_3$.

* Decoder:

n inputs - 2^n outputs.

x_1	x_0	D_0	D_1	D_2	D_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

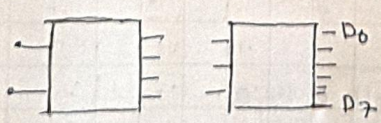
- Minimization isn't possible as we've to create all the pdt terms.

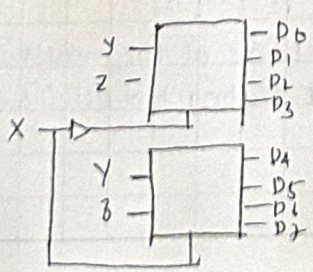


[In priority encoder
① all 0's ⇒ V=0
② else → V=1 & if D_i is the MSB 1 then out = i.]

[In decoders: n → 2^n
rev graph of encoder]

Q1 tpyer: Design a 3:8 Decoder using a 2:4 ~~deco~~ decoder.





x	y	z	
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3

for x = 1
we get $D_4 - D_7$
(from the 2nd part)

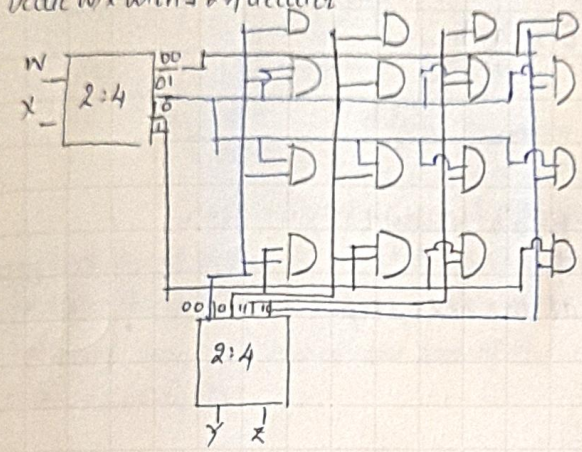
04 09A
4 012A
(A)

~ Decoder ~

- Design a 4:16 decoder using 2:4 decoders.

$f(w,x,y,z)$

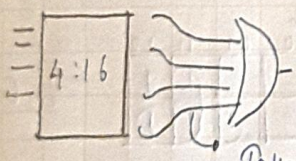
Decode w x with a 2:4 decoder → with another 2:4 decoder } And then combine them using AND gates.



// using 4→16 we would've required 16(+3) = 48 gates.

- Q: Can I prepare any de /realise any fN using a decoder.

Ans: Yes; as any fN is just a subset of the above map (4 variable fN).

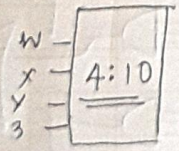


Add them.

Take the req minterms

* Decimal Decoder:

outputs are decimal no.s;



// 4 to 10 outp decoder. (0-9)

- Now since (10→15) are don't cares. They / This can be now simplified using K-map.

YZ \ WX	00	01	11	10
00	f0	f4	X	f8
01	f1	f5	X	f9
11	f3	f7	X	X
10	f2	f6	X	X

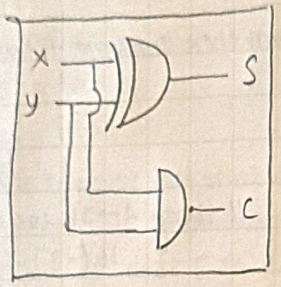
$f_8 = WZ'$

$f_9 = W'Z$

~ Adder ~

I) Half Adder: (It can add only single bit I's)

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



$S = a \oplus b$

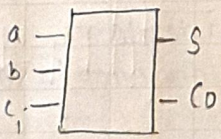
$C = a \cdot b$

- Note: Eventually I need to calc the "sum" for a 4-bit input.
  - So the carryout must be mitigated
  - Hence we also need a carry-in at the next step.

→ for the next step:

II) FULL ADDER

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



(i) for S:

C_i \ ab	00	01	10	10
0		1		1
1	1		1	

→ Can't be min further.

(ii) for C_o

C_i \ ab	00	01	11	10
0			1	1
1		1	1	1

$C_o = ab + (b+a)C_i$

- But since S and C_o are actually related we can optimise this further:

i.e. $S = a'b'c + abc + a'bc' + ab'c'$

$= (a'b' + ab)c + c'(a'b + ab')$

$= (\overline{a \oplus b})c + c'(a \oplus b)$

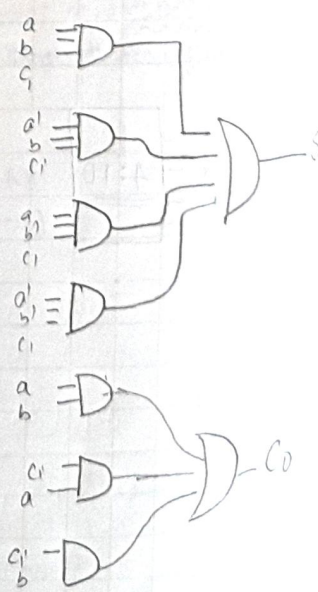
$= a \oplus b \oplus c_i$

$C_o = a'bc + ab'c + ab$

$= c(a'b + ab') + ab$

$= c\underbrace{(a \oplus b)}_{\text{cmn sub exp}} + ab$

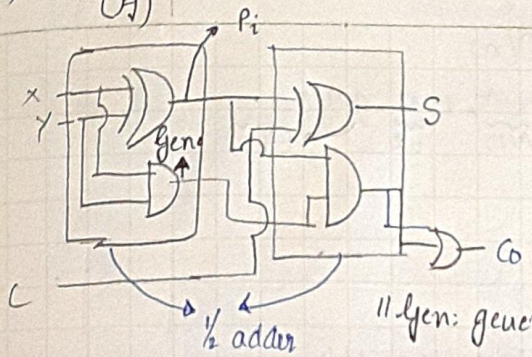
- But $a \oplus b$ is my ½ adder output

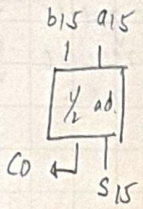


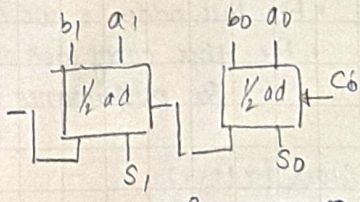
32 gates (8*16)

2-Level:

Date 04 (年) 09月 12日 (月) Contents







$S_0 = a_0 \oplus b_0 \oplus C_0$

$C_0 = C(a \oplus b) + ab$

// gen: generator.

- Delay → 3 unit. Carry & ~~2 for~~ sum (for each adder) (becomes cumullative)
  → (3n) overall

* Carry Look Ahead Adder
- → Used for fast calc
- → Const time calc; irr. of the size of the input.

~ 2-3 clock cycles

- whenever both a & b are 1 → generator shows 0 (gi) and carry is 1
- The XOR gate is called propogator (Pi) which is '1' when either of a/b = 1.
- P_i and g_i are generated (in 1 clock cycle) irrespective of C_{in}. for all the blocks.

Now S_0 & C_0:

$S_0 = P_0 \oplus C_0$ → Similarly $S_1 = P_1 \oplus C_1$

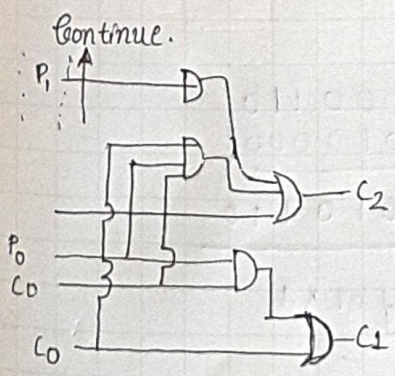
$C_0 = g_0 + P_0 C_0$ $\quad C_1 = g_1 + P_1 C_1$

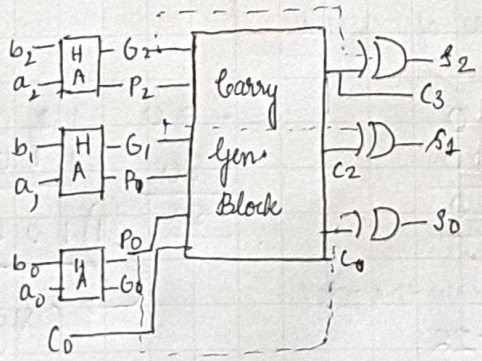
→ As C_1 is calculated rmm (in 1 cycle):

$C_2 = G_1 + P_1 (g_0 + P_0 C_0)$

→ Now C_2 depends only on g_1, P_0, g_0, C_0 which are (were available initially so all of C_i's can be calculated immediately.

* Carry Generator:





- Delay: for sum: 4 (as it depends on carry)

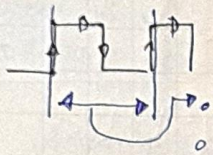
carry : 3.

一はな子

// Ripple Carry Adder //

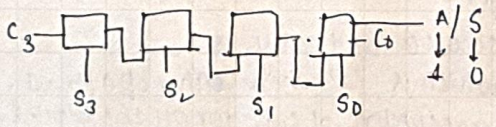
- For full adder : we require : $5n$ gates $\rightarrow \Theta(n)$
- For this carry look adder : we require $\underbrace{k \cdot nC_2}_{AND} + \underbrace{n}_{OR}$ gates $\rightarrow \Theta(n^2)$
  So performance comes with cost. i.e. area A.

* Clock (Y/N):



- Our register must get updated within one width
  - i.e. say S_0, C_0, C_4 are stored in $t = t_0$ within $t = t_0 + t_0$; our whole calc. must get over.
  - If we use full adder; our clock must be slow enough to accomodate the $5n$ calculations.

Note: We must equip our ripple carry adder to know, when our circuit generates overflow



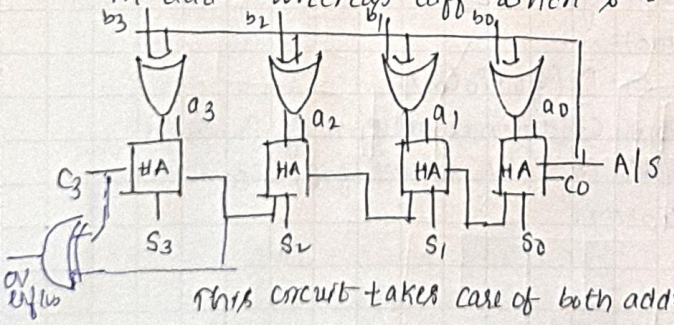
$(\pm A) \pm (\pm B)$

A−B: $A + 2^N - B$

$= 2^N + (A-B)$; $A \geq B$

$= 2^N - (B-A)$: $A < B$

- I've to design the circuit in such a way that the circuit accom. no additional changes in addn whereas diff. when S -



This circuit takes care of both addn & sub.

- Now we've to take care of overflow:

Take 8 bits

+70	0 1 0 0 0 1 1 0
+80	0 1 0 1 0 0 0 0
+150	0 (1) 0 0 1 0 1 1 0

overflow carry bit is 0 becomes sign bit & represented using 8

Conveying only this part → impl −22.

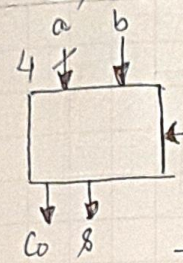
−70	1 0 0 0 1 1 0
−80	1 1 0 1 0 0 0 0
−150	(1) 1 0 0 1 0 1 1 0

carry bit is 1

Date 09月 16日 (金)

Contents

BCD Adder. (B.C.D. Adder)



• // a, b can be in [0...9]

// (inc. Cin)

→ S can be in [0...19]

→ The pblm is that we can't represent 19 in BCD. (in the reg sense...)

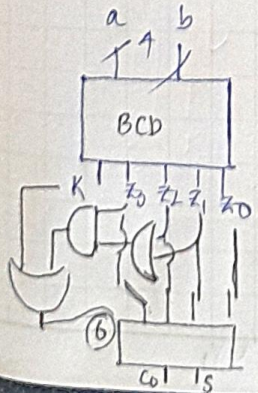
Cout	Z_3	Z_2	Z_1	Z_0	} (Binary) → (BCD)	Cout	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0		0	0	0	0	0
0	0	0	0	1		0	0	0	0	1
0	0	0	1	0		0	0	0	1	0
0	0	0	1	1		0	0	0	1	1
0	0	1	0	0		0	0	1	0	0
0	0	1	0	1		0	0	1	0	1
0	0	1	1	0		0	0	1	1	0
0	0	1	1	1		0	0	1	1	1
0	1	0	0	0		0	1	0	0	0
0	1	0	0	1		0	1	0	0	1
0	1	0	1	0		1	0	0	0	0
0	1	0	1	1		1	0	0	0	1
0	1	1	0	0		1	0	0	1	0
0	1	1	0	1		1	0	0	1	1
0	1	1	1	0		1	0	1	0	0
0	1	1	1	1		1	0	1	0	1
1	0	0	0	0		1	0	1	1	0
1	0	0	0	1		1	0	1	1	1
1	0	0	1	0		1	1	0	0	0
1	0	0	1	1		1	1	0	0	1

Same as the binary's.

Whenever Z_3 is 1; either of Z_2/Z_1 is 1 for 10-19

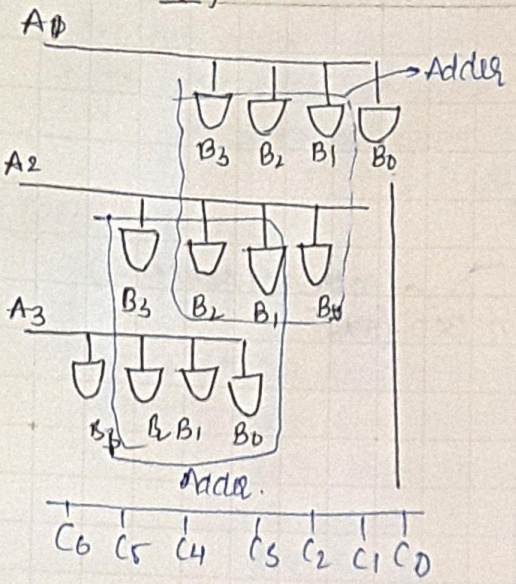
Designing this part is our challenge now.

for 10-19 :: $\underbrace{K}_{16-19} + \underbrace{Z_3(Z_2+Z_1)}_{\text{all } 10-19}$ } Observe that 6 has been added to these.



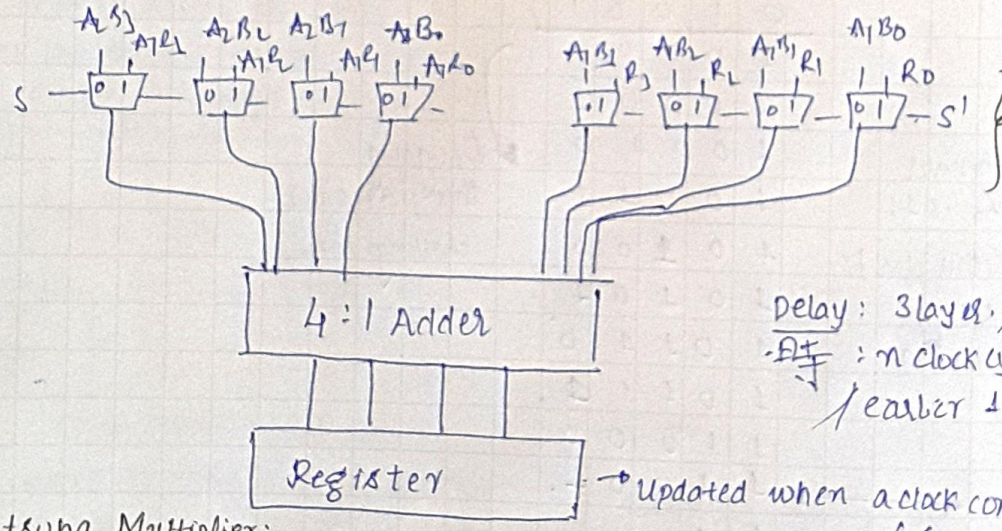
* Binary Multiplier:

$$
\begin{array}{cccccccc}
 & & & & B_3 & B_2 & B_1 & B_0 \\
 & & & & \times & A_3 & A_2 & A_1 \\
\hline
 & & & & A_1B_3 & A_1B_2 & A_1B_1 & A_1B_0 \\
 & & A_2B_3 & A_2B_2 & A_2B_1 & A_2B_0 & \times & \\
 & A_3B_3 & A_3B_2 & A_3B_1 & A_3B_0 & \times & \times & \\
\hline
C_6 & C_5 & C_4 & & C_3 & C_2 & C_1 & C_0
\end{array}
$$



- Say → $m \times n$ bit xplicatN.
- $m * n$ AND gates
- $n-1$ Adders are required
- n layers of AND gates + $n-1$ adders : $2n-1$ → delay
- But this is not optimal. (can't be done in 1 clock)
- This underlines the limitations of logical design using combinational circuits
- Rather than doing everything in 1-clock we can do the same in iterations
- So we need to store the intermediate results. Hence we need memory. (Seq. Circuits)

* Lets try to do in diff steps:



Delay: 3 layers.
: n clock cycles
/ earlier 1 clock cycle

→ Updated when a clock comes (S is also changed then)

* Karatsuba Multiplier:

- Our issue was with the width.
- Here we perform:

1 n bit xcatN ⇒ $3 \cdot n/2$ bit xplicatN

→ $n^{\log_2 3}$ 1 bit xpcatN

- x ; y

$x = x_1 B^m + x_0$ | $y = y_1 B^m + y_0$ | eg: $x = 12345$

$= 12 \times 10^3 \times 345$

x_1 x_0.

$xy = \underbrace{(x_1 y_0 + x_0 y_1)}_{z_1} B^m + \underbrace{(x_1 y_1)}_{z_2} B^{2m} + \underbrace{x_0 y_0}_{z_0}$

// Choose m correctly → $(1/2)\, xy$: $x_1 y_1$

- $xy = z_2 B^{2m} + z_1 B^m + z_0$

  $z_1 = (x_1 + x_0)(y_1 + y_0) - z_0 - z_2$.

  - Now again break the x_i; y_i s recursively
  - To obtain single bits.