INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION

INTERNATIONAL CENTRE FOR SCIENCE AND HIGH TECHNOLOGY

(INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS 34100 TRIESTE (ITALY) VIA GRIGNANO, 9 (ADRIATICO PALACE) P.O. BOX 586 TELEPHONE (040) 224522 TELEFAX (040) 224559 TELEX 460449 APH I

SMR/542 - 10

ICTP-INFN
SECOND COURSE ON BASIC VLSI DESIGN TECHNIQUES
18 February - 15 March 1991

Digital Design
"Logic Design"

Paolo FARABOSCHI
Dipartimento Ing. Biofisica ed Elettronica
Università di Genova
Via all'Opera Pia 11/A
Genova 16145
Italy

LOGIC

DESIGN

LOGIC DESIGN

Number Systems
- notations
- operations and properties
- implements

Boolean Algebra
- Laws
- Logic functions
- Canonical forms
- Minimization of logic functions
- Karnaugh Maps
- Hazards

ALGORITHMIC STATE MACHINE DESIGNS

- Sequential circuits
  - properties
  - ASM charts
  - memory elements
  - races
  - synthesis procedures

ASM with programmable devices
- Logic with PLA's
- Techniques for ASM design with ROM's

Number Systems

A number system is a language system consisting of an ordered set of digits with rules defined for addition, multiplication and other mathematical operators.

The radix or base of a number system specifies the actual number of digits included in its ordered set.

A number can have an integer and a fractional part set apart by a radix point:

$$(N)_r = [(\text{integer part}) \cdot (\text{fractional part})]$$

It is defined **JUSTAPOSITIONAL** notation the number representation where a number is represented by a sequence of symbols (digits) and the value of a given symbol is determined by the symbol itself and by its position in the sequence.

$$(N)_r = (a_{n-1}\, a_{n-2} \cdots a_i \cdots a_1 a_0 \bullet a_{-1} a_{-2} \cdots a_{-f} \cdots a_{-m})_r$$

Integer portion — radix point — fractional

The **POLINOMIAL** notation of a number $(N)_r$ can be written as follows:

$$(N)_r = \sum_{j=-m}^{n-1} a_j r^j$$

Examples:

$$(7)_{10} = (111)_2 = (21)_3 = (7)_8$$

$$111_2 = 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 4_{10} + 2_{10} + 1_{10} = 7_{10}$$

$$21_3 = 2\cdot 3^1 + 1\cdot 3^0 = 6_{10} + 1_{10} = 7_{10}$$

$$7_8 = 7\cdot 8^0 = 7_{10}$$

The justapositional notation allows to execute operations on numbers by using the symbols of the represented numbers, without taking into account the value of the symbols.

Thus, with the given numbers:

$$A = (a_{n-1} \dots a_0)_r \qquad B = (b_{m-1} \dots b_0)_r$$

we have:

$$A+B = \sum_{i=0}^{h} (a_i + b_i) \cdot r^i$$

where $h = \max(n-1, m-1)$

$$A \cdot B = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (a_i \cdot b_j) \cdot r^{i+j}$$

It is sufficient to build tables that specify the sum and the multiplication for every couple of symbols. The iterated application of those tables allows to compute sums and products for every couple of numbers.

For the binary system we have:

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

and then:

```
  111010 +
   10001 =
--------
 1001011
```

```
     110 ×
    1011 =
--------
     110
    110
   000
  110
--------
 1000010
```

Complements of numbers

Given a number in base r, the definition of the r's complement of the number is as follows:

$$(N)_{r,c} \triangleq r^n - N, \text{ if } N \neq \phi, \phi \text{ for } N = \phi$$

where n = number of digits in the integer portion of N

r = radix number

Examples:

$(147)_{10,c} = 10^3 - 147 = 853.00$

$(0.53)_{10,c} = 1 - 0.53 = 0.47$

$(1010)_{2,c} = 2^4 - 1010 = 00110$

With binary systems there are shortcut techniques that bypass the complementation operation.

One technique is as follows:

(1) Invert each digit of the number (logical complement)

(2) add 1 to the least significant digit of step (1)

For example, given 00010110.0110

step (1) 11101001.1001

step (2) + 1

11101001.1010

(r-1)'s complement.

The definition of the r minus one complement of a number N is as follows:

$$(N)_{r-1,c} \triangleq r^n - r^{-m} - N$$

where:

n = number of digits in the integer portion of N

m = number of digits in the fractional portion of N

r = radix number.

An (r-1)'s complement of a number in base 2 can be done simply by complementing each digit (0 or 1) of the number.

Subtraction

r's and (r-1)'s complements are very useful in the subtraction. They allows to use the same hardware for addition and subtraction

Subtraction with r's complements.

Given two positive base r numbers M and S, (M-S) goes as follows:

(1) Add M to the r's complement of S.

(2) Check the result for overflow carry.

(a) If an overflow carry results, discard it. The rest of the result is (M-S).

(b) If an overflow carry does not occur, the result of the first step is neg. Therefore, take the r's complement of the result and add a negative sign to the result.

Examples:

$(M-S) = (1010 - 0111)$

```
  1010
  1001  → 2's complement of 0111
overflow
carry discarded  1 ← 0011 → + 0011
                              answer
```

$(M-S) = (0111 - 1010)$

```
  0111
  0110  → 2's complements of 1010
no overflow carry  0 ← 1101 → - 0011
                              answer
```


Subtraction with (r-1)'s complements.

(1) Add M to the (r-1)'s complement of S.

(2) Check the result for overflow carry:

(a) If an overflow carry exists, add it to the least significant digit.

(b) If an overflow carry does not exist, the result is negative. Therefore complement the result, and add a minus sign in front.

$(M-S) = (1010 - 0111)$

```
  1010
  1000  → 1's complement of 0111
overflow carry  1 ← 0010
                  └──→ 1
                  ------
                  0011
                  answer
```

$(M-S) = (0111 - 1010)$

```
  0111
  0101  → 1's complement of 1010
no overflow carry  0 ← 1100 → - 0011
```

BOOLEAN ALGEBRA

Following E. V. Huntington ("Sets of indipendent postulates for the algebra of logic" Trans. Am. Math. Soc., vol. 5, 1904),

the deductive theory based on 4 axioms and 10 postulates is called Boolean Algebra

The axioms are the followings:

a) A set S

b) An equivalence relation, represented by the symbol "=", valid for the elements of S.

c) A binary operation, represented by the symbol "+", executable on the elements of S.

d) A binary operation, represented by the symbol "·", executable on the elements of S.

Huntington's Postulates

(1) (a),(b) A set of elements S is closed with respect to an operator if for every pair of elements in S the operator specifies a unique result which is also in the set S.

(2a) There exists an elements ϕ in S such that for every A in S, $A + \phi = A$.

(2b) There exists an element 1 in S such that for every A in S, $A \cdot 1 = A$.

(3a) $A + B = B + A$
(3b) $A \cdot B = B \cdot A$
} commutative laws

(4a) $A + (B \cdot C) = (A + B) \cdot (A + C)$
(4b) $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$
} distributive laws

(5) For every element A in S, there exists an element $\overline{A}$ such that $A \cdot \overline{A} = \phi$, $A + \overline{A} = 1$

(6) There exist at least two elements A and B in S such that A is not equivalent to B.

If we define the two-valued Boolean algebra system as follows:

A set $S = \{\phi, 1\}$,

"+" the operation defined by:

$1 + 1 = 1 \quad 1 + \phi = 1 \quad \phi + 1 = 1 \quad \phi + \phi = \phi$,

"·" the operation defined by:

$1 \cdot 1 = 1 \quad 1 \cdot \phi = \phi \quad \phi \cdot 1 = \phi \quad \phi \cdot \phi = \phi$,

"INVERTER" the operation defined by:

$\overline{1} = \phi \quad , \quad \overline{\phi} = 1$.

We find that the two-valued Boolean algebra system satisfies the postulates.

The following identities can be found:

$\phi \cdot A = \phi$	$1 + A = 1$
$1 \cdot A = A$	$\phi + A = A$
$A \cdot A = A$	$A + A = A$
$A \cdot \overline{A} = \phi$	$A + \overline{A} = 1$
$\overline{\overline{A}} = A$	

that can be used to develope and prove the following theorems:

$$A + A \cdot B = A$$

$$A + \overline{A} \cdot B = A + B$$

$$AB + A\overline{B} = A$$

$$AC + \overline{A}BC = AC + BC$$

$$AB + AC + \overline{B}C = AB + \overline{B}C$$

$$\overline{A \cdot B \cdot C \cdots} = \overline{A} + \overline{B} + \overline{C} + \cdots$$

$$\overline{A + B + C + \cdots} = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdots$$

(De Morgan Laws!

$$f(x, y, 1, 0, \cdot, +) = \overline{f}(\overline{x}, \overline{y}, 0, 1, +, \cdot)$$

Shannon's E.D. Theorem:

$$\sum_{n} f(x_1, x_2 \cdots x_n) = x_1 \cdot f(1, x_2 \cdots x_n)$$

)

It is possible to introduce other logical operators as follows:

NOR $a \downarrow b = \overline{a + b}$

NAND $a / b = \overline{a \cdot b}$

EXOR $a \oplus b = a\bar{b} + \bar{a}b$

EQUIVALENCE (EXNOR) $a \odot b = ab + \bar{a}\bar{b}$

The NOR and NAND operations are not associative so the notation: $a \downarrow b \downarrow c$ is interpreted as $\overline{a+b+c}$

and the notation $a / b / c$ as: $\overline{a \cdot b \cdot c}$

The NOR operation allows to build the set $\{+, \cdot, -\}$

The NAND " " " " " " "

LOGIC FUNCTIONS

A transformation from $\{0,1\}^n$ into $\{0,1\}$ is called n-variables logic function.

Since there are 2^n different combinations of n variables, the number of different n-variables functions is 2^{2^n}.

For a set of logical variables, we may define any desired function by giving the function value for each possible set of variables values. A tabular form with input variables on the left and the function on the right is very useful for this display. Such a display is called TRUTH TABLE.

Examples:

Let be x a logic function of three variables A, B, C, it can be shown as follows:

A	C	B	X
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Having chosen an ordering of the input variables (A, C, B in this case), the canonical truth table form lists all possible combinations of the variables' values, in binary numeric order.

EQUATIONS FROM TRUTH TABLE

1. SUM OF PRODUCTS FORM

Definition:

MINTERM or CANONICAL PRODUCT TERM:
The logical AND of exactly one occurrence of every variable of the function.

To derive a sum-of-products form for a function from a canonical truth table, write the OR of the minterms for which the function is true.

2. PRODUCT OF SUMS FORM

Definition:

MAXTERM OR CANONICAL SUM TERM:
The logical OR of exactly one occurrence of every variable of the function.

To derive a product-of-sums form of a function from a canonical truth table, write the AND of each maxterm for which the function is false.

3. NOR-NOR FORM

Definition:

FUNDAMENTAL NOR
The logical NOR of exactly one occurrence of every variable of the function.

To derive a NOR-NOR form for a function from a canonical truth table, write the NOR of every fundamental NOR that is true connected with the assignments for which the function is false.

4. NAND-NAND FORM

Definition:

FUNDAMENTAL NAND
The logical NAND of exactly one occurrence of every variable of the function

To derive a NAND-NAND form for a canonical truth table, write the NAND of every fundamental NAND that is false connected with the assignments for which the function is true

It is possible to derive other forms to express a truth-table. This is done by applying the the theorems of the BOOLEAN ALGEBRA. The following table reports 8 two levels forms and rules to follow to write a given expression

FORM	The assignments to consider are those for which the function is:	Rules for reading variable
AND-OR	True	asserted
OR-AND	False	not asserted
NAND-NAND	True	asserted
NOR-NOR	False	not asserted
NOR-OR	True	not asserted
NAND-AND	False	asserted
OR-NAND	True	not asserted
AND-NOR	False	asserted

MINIMIZATION OF LOGIC FUNCTIONS

The minimization process is used to reduce the number of logical operators used to express a logic function.

The Keys to Boolean minimization lie in the theorems of Boolean Algebra:

Thm 1: $A + AB = A$, $A(A+B) = A$

Thm 2: $A + \bar{A}B = A+B$, $A(\bar{A}+B) = AB$

Thm 3: $AB + A\bar{B} = A$, $(A+B)(A+\bar{B}) = A$

Examples:

(1) $F = CD + A\bar{B}C + AB\bar{C} + BCD = CD + A\bar{B}C + AB\bar{C}$

Thm 1

(2) $F = AB + BEF + \bar{A}CD + \bar{B}CD =$

$AB + BEF + CD(\bar{A} + \bar{B}) =$

$AB + BEF + CD(\overline{AB}) = AB + BEF + CD$

Thm 2

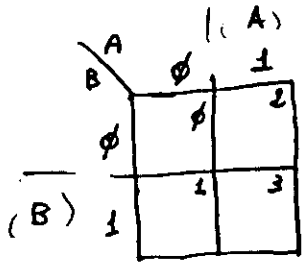
(3) $F = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + AB\bar{C} + A\bar{B}\bar{C} =$

Thm 3 Thm 3

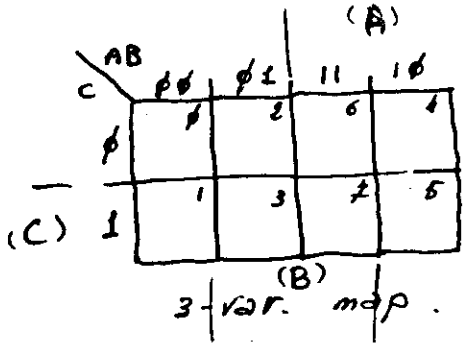
$\bar{A}\bar{B} + A\bar{C}$

KARNAUGH MAP (KM)

KM orders and displays the minterms in a geometrical pattern such that the application of the logic adjacency (Thm. 3) Theorem becomes obvious.



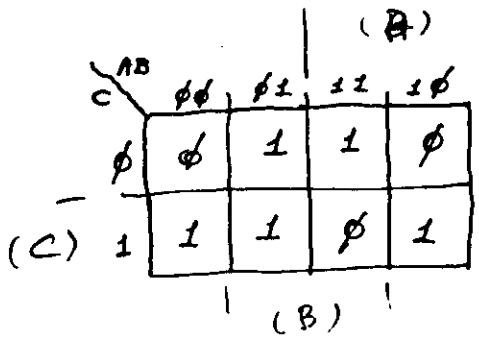
2-var. map



3-var. map.

Example:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



logic product P_1 <u>covers</u> a logic product P_2 if all the variables of P_1 are in P_2.

Example: $P_1 = x_1 x_0$

$P_2 = x_2 \bar{x}_1 x_0$

logic function f <u>covers</u> a logic function g if assignments do not exist for which f is false and g is true.

logic product is a PRIME IMPLICANT of the logic function f if:

1) f covers P
2) P is not covered by another logic product P^* that is covered by f and has fewer variables of P.

Theorem: A logic function f can be expressed by a minimal sum-of-products form, where all the logic products are prime implicants.

Example:

$$f(x_2, x_1, x_0) = x_2 \bar{x}_1 x_0 + x_2 x_1 x_0 + \bar{x}_2 x_1 x_0 + \bar{x}_2 \bar{x}_1 \bar{x}_0 =$$

$$\bar{x}_1 \bar{x}_2 + x_0$$

The prime implicants of f are:
$\bar{x}_1 \bar{x}_2$ and x_0.

Simplifying with K-Maps

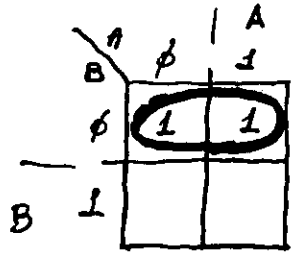
Consider the expression:

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

by using the identity $A \cdot B + \bar{A} \cdot B = B$ the expression can be simplified as follows:

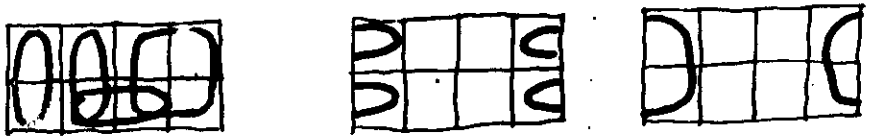
$$X = \bar{B}$$

The key point for simplification is that a term B is "anded" with both A and $\bar{A}$. On the KM this results in 1's both for $A=\phi$ and $A=1$ squares for $B=\phi$. The adjacent 1's are circled to remind that the variable A disappears from the simplified expression.



The drawing of circles among adjacent 1's is the basis for using KM in Boolean simplification.

Depending on the position of the 1's, we may have several circles, each spanning a grouping of one, two, four --- 1's.
Circling two 1's causes two canonical terms to collapse into one term; one variable drops out.
Four circled 1's bring four terms into one term, eliminating two variables.

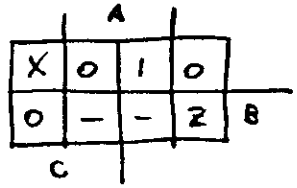


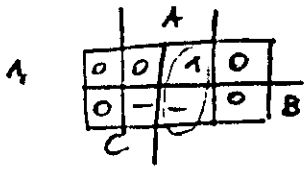
Typical circling of KM of 3-variables.

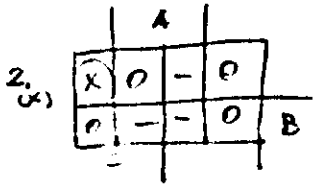
MAPS WITH VARIABLES

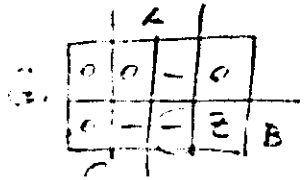
- It is possible to display and manipulate functions of many variables on a KM of smaller dimensions by entering the excess variables on the map
- To simplify functions with variables
  1. SET ALL VARIABLES TO 0 AND SIMPLIFY ON THE 1's REMAINING ON MAP
  2. RESTORE VARIABLES, SET 1's TO DON'T CARE AND MINIMIZE FOR EACH VARIABLE IN TURN
  3. COMBINE EXPRESSIONS GENERATED IN 1. AND 2.

EXAMPLE



1.  → $A \cdot \bar{C}$

2. (x)  → $X \cdot \bar{A}\bar{B}\bar{C}$

3.  → $\bar{B} \cdot C$

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

HAZARDS

- As the output of REAL gates has a finite delay (cannot change instantaneously), it is possible that for some transient

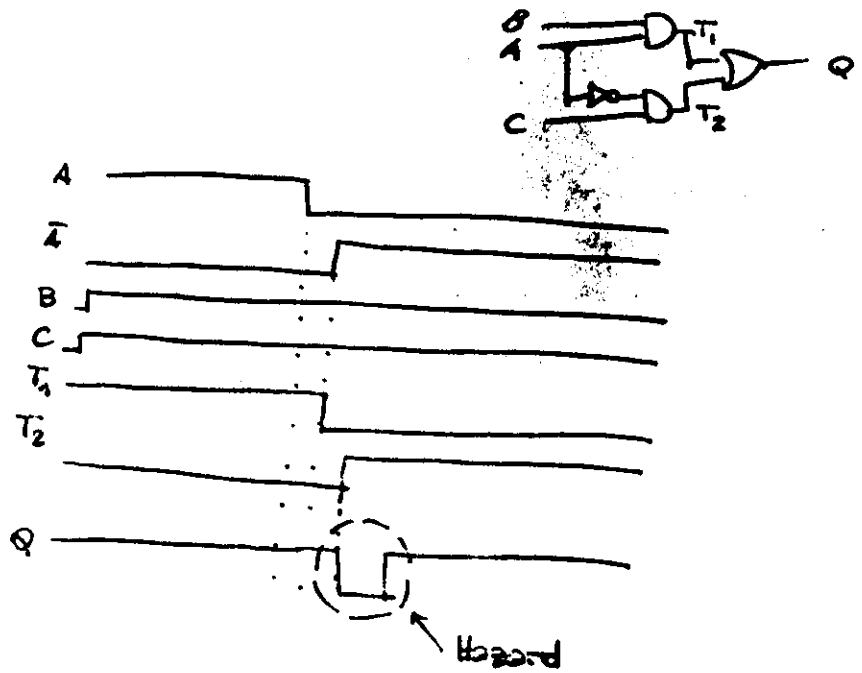
$$A = \overline{A}$$

which is a violation to boolean hypothesis

- Propagation delay (also caused by interconnections) can create spurious outputs (HAZARDS or GLITCHES)

- Example:

$$Q = A \cdot B + \overline{A} \cdot C$$



HAZARDS ELIMINATION

DEF.:

STATIC HAZARD when a change in an input variable causes a transitory change in an output variable that should have not changed

(⎍)

DYNAMIC HAZARD when a change in an input variable causes a transitory change in an output variable that should change

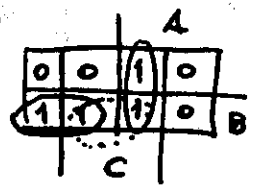
[more than 2 levels] (⎍⎍)

HAZARDS ELIMINATION

- SUM-OF-PRODUCTS

A function may have hazards if and only if its KM has adjacent 1's not enclosed in the same circle

Ex.:



To eliminate hazards: INTRODUCE REDUNDANT SQUARES

$$Q = AB + \overline{A}C + \underline{B \cdot C}$$

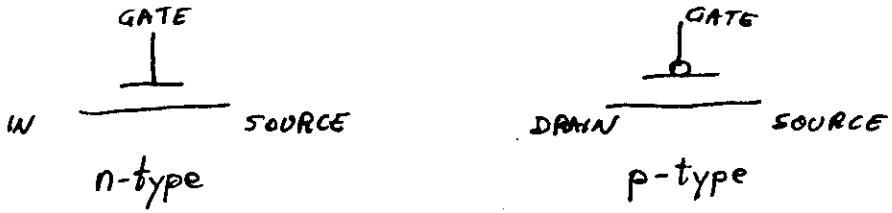
- GENERAL CIRCUITS

There is no technique
It's better to make output logic insensitive to hazards
(e.g.: through synchronization)

REALIZING LOGIC IN HARDWARE

CMOS technology provides two types of transistors:

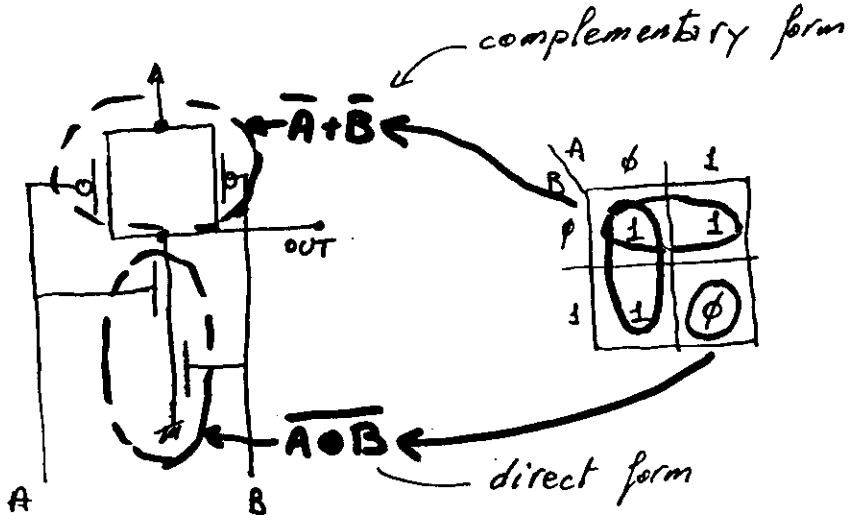
n-type transistor
p-type transistor



The gate controls the passage of current between the drain and source. Simplifying this to the extreme allows the MOS transistors to be viewed as simple on/off switches.

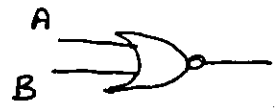
In the following we will assume that a "1" is a high voltage normally set to 5 volts, and called POWER or V_{DD}. The symbol "ϕ" will be assumed to be a low voltage, that is normally set to ϕ volts and called GROUND or V_{SS}.

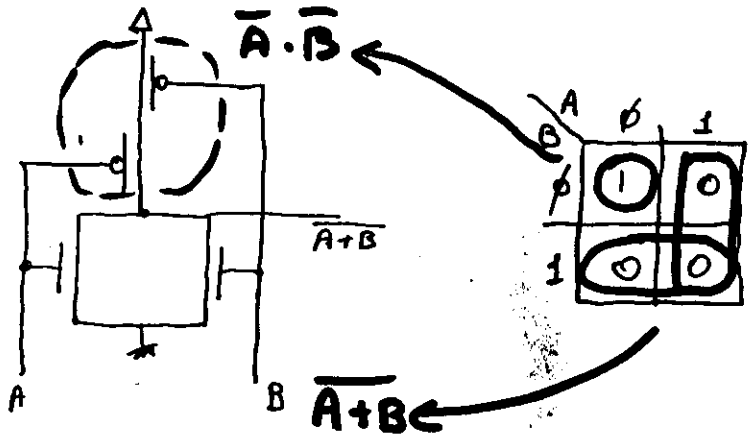
NAND GATE



To have perfect switching:
The p-structure is used for passing 1's
The n-structure is used for passing ϕ's

NOR GATE





CMOS LOGIC

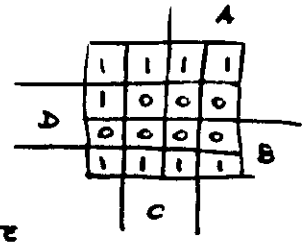
- CMOS properties :
  - If 2 n-mos are placed in series, the composite structure is on if both inputs are on → AND structure
  - If 2 n-mos are placed in parallel, the composite structure is on if one of two inputs is on → OR structure
  - A similar property (dual) is valid for p-mos structure

[IT IS POSSIBLE TO DERIVE CMOS combinatorial circuits DIRECTLY FROM EM]

- EXAMPLE

- Circling 0's :

$$\bar{F} = AD + BD + CD = (A+B+C)D$$



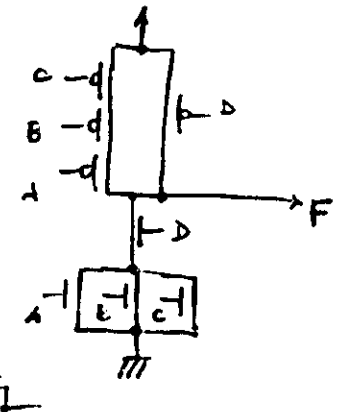
we derive the structure for the nmos part (function false, variables true)

- Circling 1's

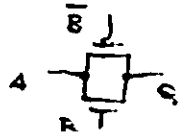
$$F = \bar{D} + \bar{A}\cdot\bar{B}\cdot\bar{C}$$

we derive the structure for the p-mos part (function true, variables false)

we can combine the 2 parts to form a CMOS gate



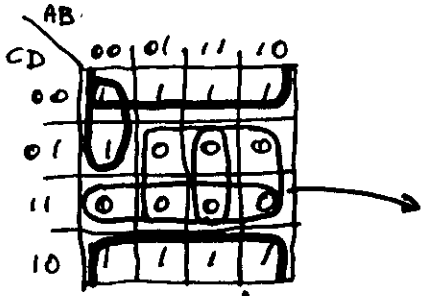
- NOTES :
  - number of variables is limited by capacitance factors (BODY EFFECT)
  - other approaches : TRANSMISSION GATES (for multiplexers)



COMPOUND GATES

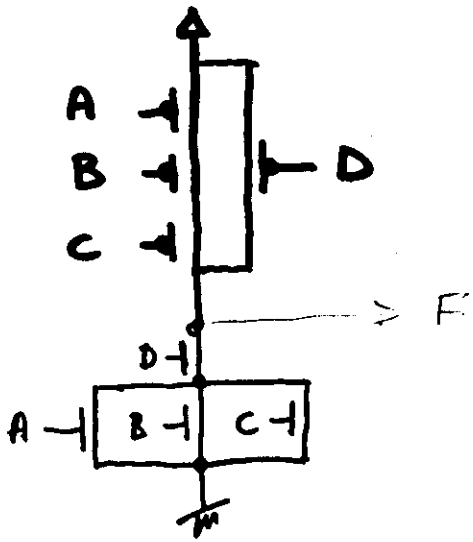
A compound gates is derived by using a combination of series and parallel switch structures.

Examples: $F = \overline{((A+B+C)\cdot D)}$

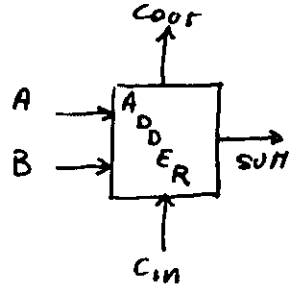


$C \cdot D + B \cdot D + A \cdot D = D \cdot (A+B+C)$

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



THE ADDER

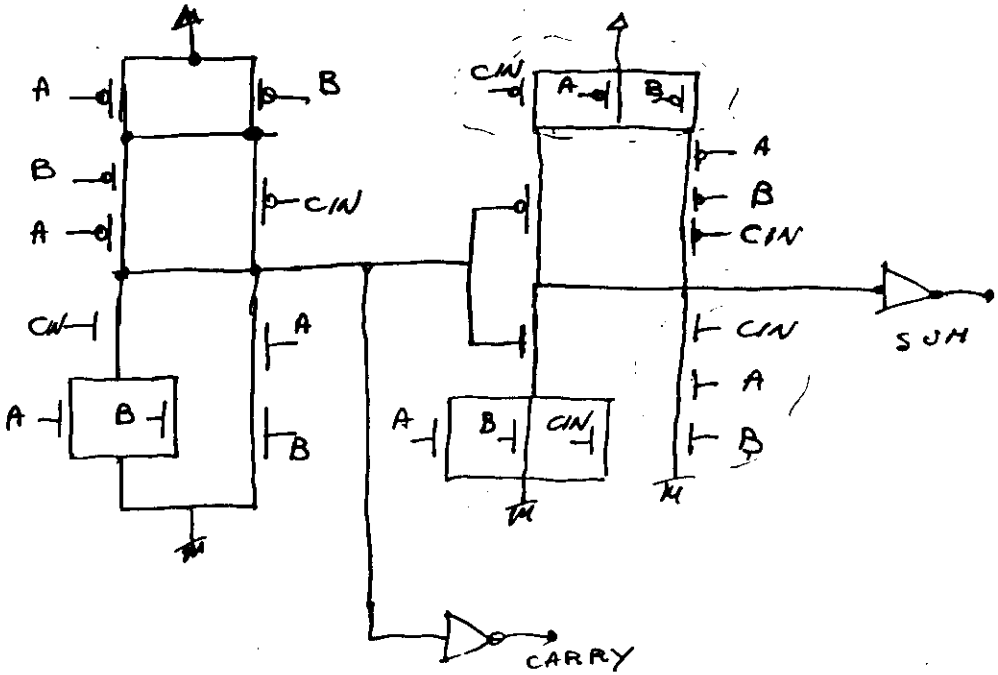


A	B	C_{in}	SUM	COUT
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

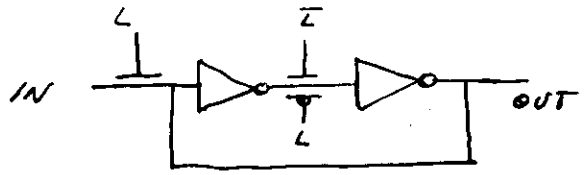
$COUT = AB + C(A+B)$

$SUM = ABC + A\overline{BC} + \overline{AB}C + \overline{A}B\overline{C} =$

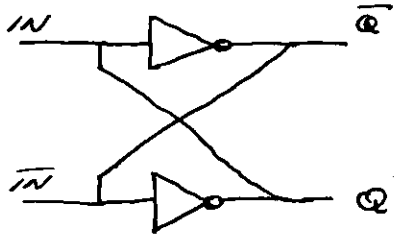
$\overline{COUT} \cdot (A+B+CIN) + A \cdot (B+CIN)$



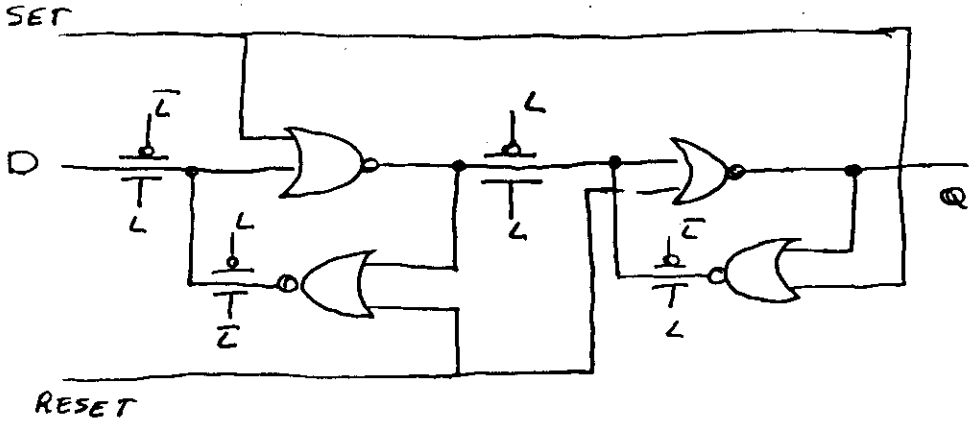
LATCH REGISTER



PSEUDO - STATIC LATCH



static Latch



STATIC D flip-flop with set and reset

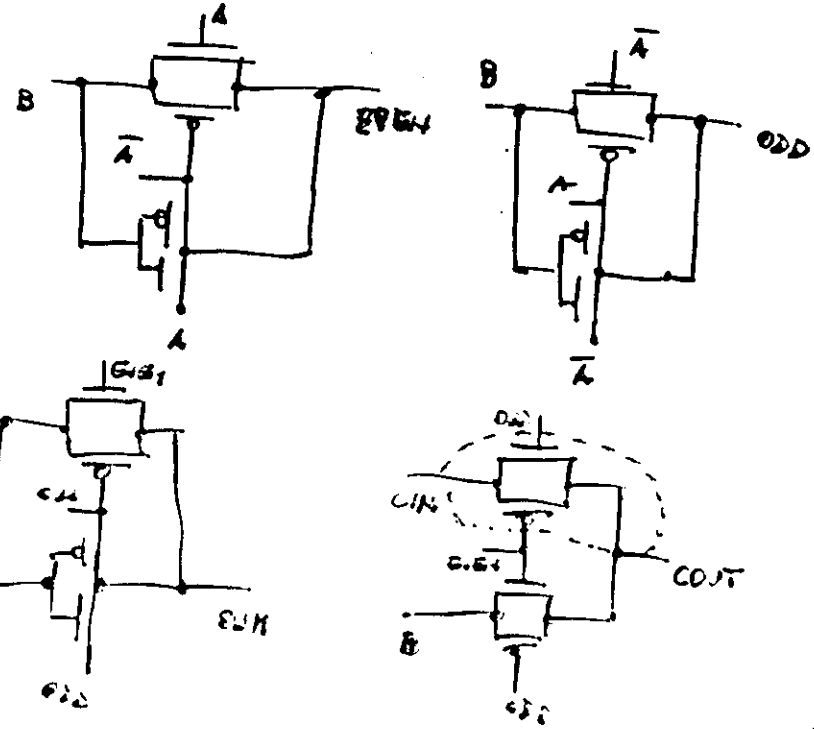
TRANSMISSION GATE ADDER

- cheaper way to implement an ADDER in CMOS logic.
- We can use TRANSMISSION GATES to synthesize

for instance : FULL ADDER :

$$ODD = A \oplus B = \bar{A}\cdot B + A\cdot\bar{B}$$
$$EVEN = \overline{A \oplus B} = \overline{ODD} = A\cdot B + \bar{A}\cdot\bar{B}$$

$$COUT = ODD \cdot C_{IN} + EVEN \cdot B$$
$$SUM = ODD \cdot \bar{C}_{IN} + EVEN \cdot C_{IN}$$



DIGITAL ARITHMETIC

FIXED POINT NUMBERS

SIGNED NUMBERS

A signed number is represented by its 2's complement

(Ex $+13 = 00001101$
$-13 = 11110101$)

FIXED POINT MACHINES

The algorithm remains the same regardless of how numbers are interpreted (as fractions or as integers)

(eg. : FRACTION: $0100.11 =$
$= 1 \cdot 2^2 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} =$
$= 4 + 0.5 + 0.25 = 4.75$)

FIXED POINT ADDITION

$S = A + B$

$A = a_{n-1}\ a_{n-2}\ \dots\ a_0$

$B = b_{n-1}\ b_{n-2}\ \dots\ b_0$

We discriminate among 3 cases

1) $A, B \geq 0 \Rightarrow S = |A| + |B|$

2) $B < 0,\ A \geq 0$

- $|A| > |B|$ $\Rightarrow S = |A| + 2^n - |B|$
  the carry is discarded (cout is positive)
- $|A| < |B|$
  there is no carry

3) $B < 0,\ A < 0 \Rightarrow S = 2^n - |A| + 2^n - |B| =$
$= 2^{n+1} - (|A| + |B|)$
the carry indicates a negative value

OVERFLOW

1. Whenever sign of the sum disagrees with sign of the operands, which are the same
$$V_1 = a_{n-1} \cdot b_{n-1} \cdot \overline{S_{n-1}} + \overline{a_{n-1}} \cdot \overline{b_{n-1}} \cdot S_{n-1}$$
2. Whenever the carry of the MSB of the sum and the carry of the sign disagree
$$V_2 = C_{n-1} \oplus C_{n-1}$$

1) RIPPLE CARRY

Based on full-adders

$$\begin{cases} S_i = a_i \oplus b_i \oplus C_i \\ C_{i+1} = a_i b_i + (a_i + b_i) C_i \end{cases}$$

worst case : n full-adder levels

2) CARRY LOOK-AHEAD

We define $\begin{cases} \text{Generate carry} & G_i = a_i b_i \\ \text{Propagate carry} & P_i = a_i + b_i \end{cases}$

$C_i = G_i + P_i \cdot C_{i-1}$

for more stages :

$C_1 = G_1 + P_0 C_0$

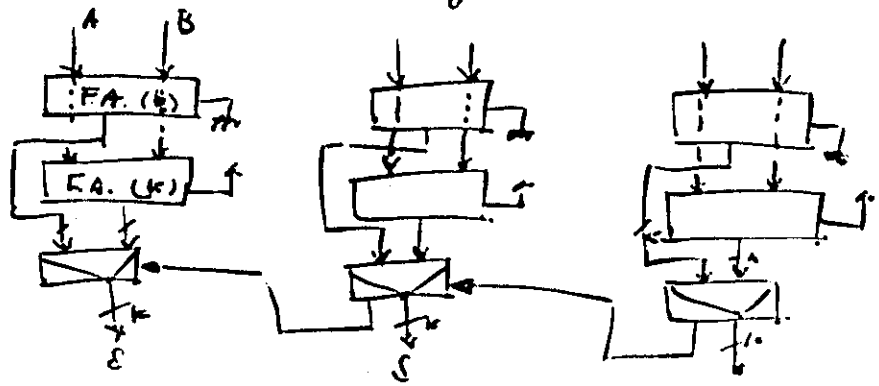
$C_2 = G_2 + P_2 G_1 + P_2 P_1 C_0$

$$C_i = G_i + \sum_{0}^{i-1} \left(\prod_{k+1}^{i} P_k \right) G_k + \prod_{0}^{i} P_k C_0$$

As number of high-order gate inputs are limited, we can organize addition in groups

3) CARRY SELECT ADDER

- 2 Addition are performed assuming $C_{in} = 0$ and $C_{in} = 1$
- Multiplexers then choose the right result



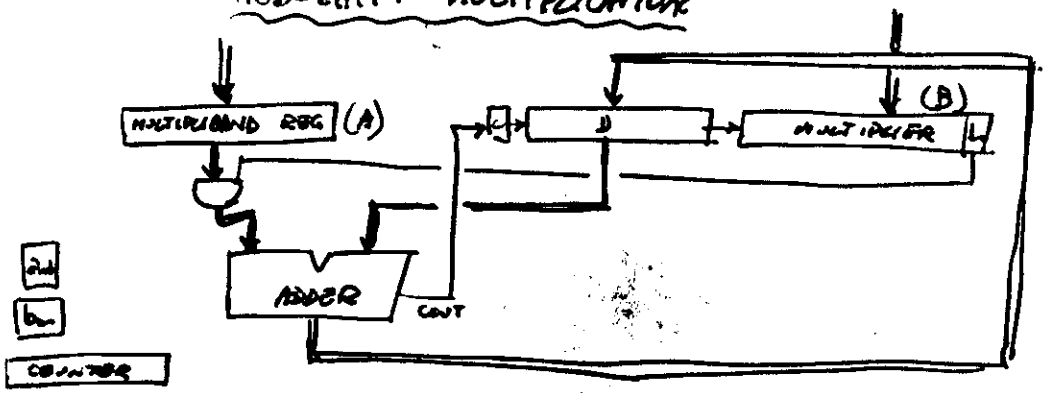
FIXED POINT MULTIPLICATION

$A = a_{n-1}\, a_{n-2} \ldots a_0$

$B = b_{n-1}\, b_{n-2} \ldots b_0$ $\qquad P = p_{2n-1} \ldots p_0$

Negative multipliers: in order to provide correct result we have to complement multiplier and then adjust the sign of the result.

ADD-SHIFT MULTIPLICATION



A: multiplicand (n)
B: multiplier (n)
D: high order bits of product (n)
C: carry (1)

1. $D \leftarrow 0$
   $C \leftarrow a_{n-1} \oplus b_{n-1}$
2. . bo determined IF A is to be added to D
   · Sum is loaded in D and carry in C
   · D and B are shifted (as a group) right 1 bit

BOOTH'S ALGORITHM

Idea: reduce the number of operations (additions) in add-shift technique with BIT SCANNING operations

1. SKIPPING OVER ZEROS:
   you can skip over 0s in the multiplier and then operate a shift of the right number of positions

2. SKIPPING OVER ONES:
   using the identity

$$0\,1\,1\ldots1\,1\,0\,0\,0 = 1\,0\,0\,0\,0 - 0\,0\,1\,0\,0$$

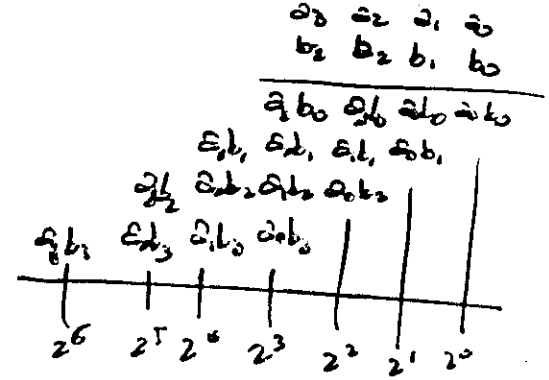
(k 1s $\Rightarrow$ $(k+1)$ 0s)

addition — subtraction

we can transform sequence of 1s in sequence of 0s

WALLACE TREES

Consider 4 bits operands



This leads to a parallel structure

(WHY: non additive multiply module)

NMM

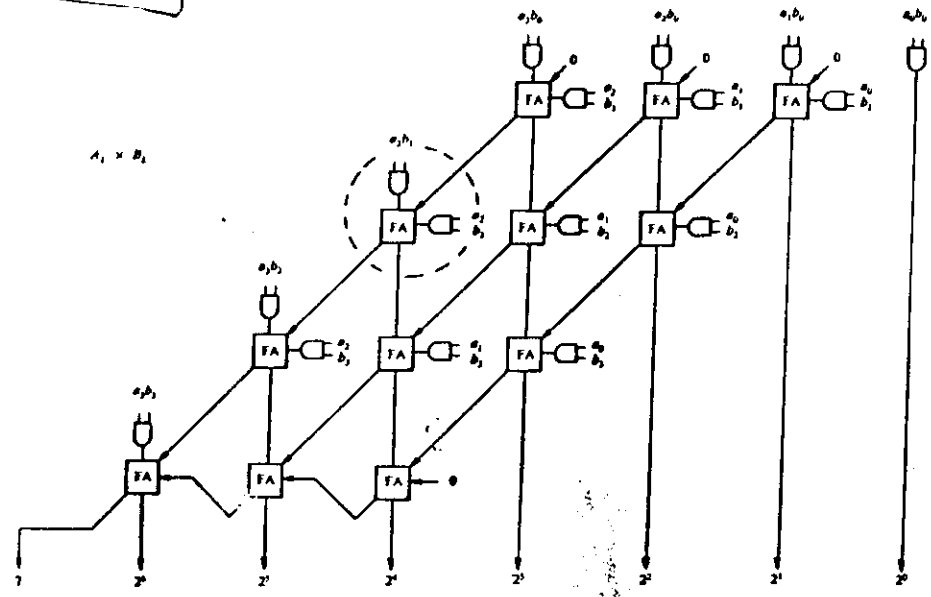
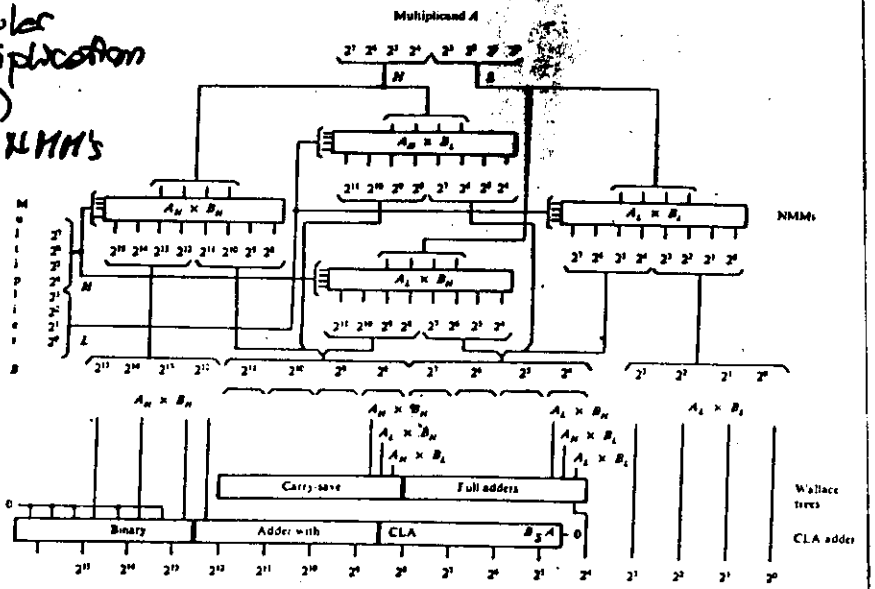
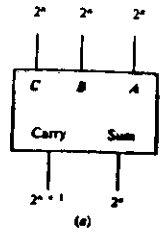


Figure 3.36 A 4 × 4 NMM.

Modular Multiplication (8×8) with NMM's

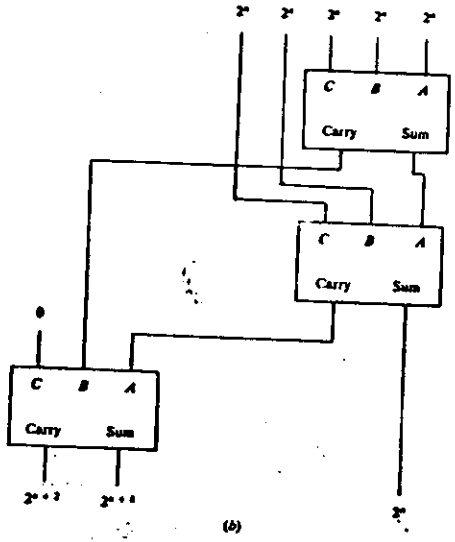


Carry save adder

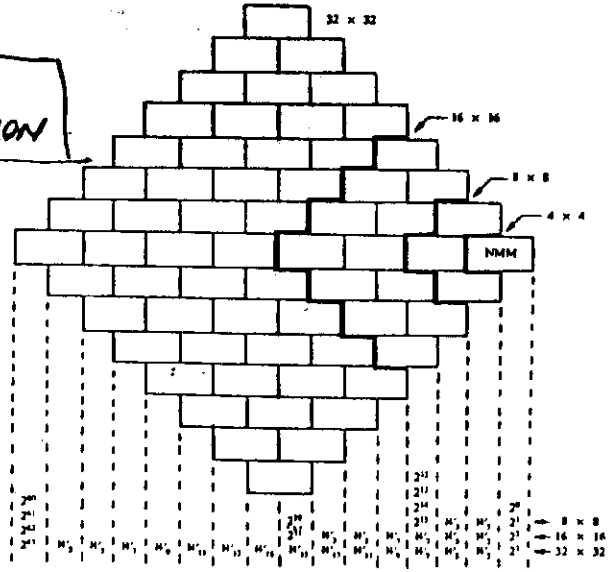


3-bit WALLACE TREE

5-bit WALLACE TREE



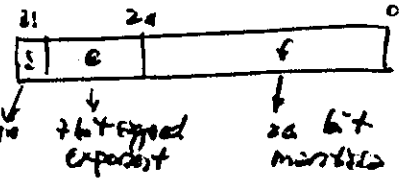
MODULAR MULTIPLICATION



FLOATING POINT NUMBERS

FORMAT $A = f \cdot 2^e$ { f = mantissa, e = exponent

(32 bits used:



)

- NORMALIZATION : there must be as many significant bits in f as possible → normalization

for a normalized fraction: $\frac{1}{2} \le f < 1$

- RANGE : → seven-bit exponent has → range $2^{-64} - 2^{63}$
at these limits are exceeded OVERFLOW or UNDERFLOW occur

BIAS EXPONENTS

- Necessity to compare frequently exponents
- To avoid involving the sign:
exponents are usually converted to positive by adding a positive BIAS to them at the FP number is formed.
- with n bits, Bias $= +2^{n-1}$

$$e_{biased} = e + 2^{n-1}$$

$$-2^{n-1} \le e_{unbiased} \le 2^{n-1} - 1$$

$$0 \le e_{biased} \le 2^n - 1$$

1.) simplification of the
2.) Representation of 0 with all 0s (lowest exponent

FLOATING POINT ADDITION

Using NORMALIZED operands

$A_1 = f_1 \cdot 2^{e_1}$

$A_2 = f_2 \cdot 2^{e_2}$ $\qquad \frac{1}{2} \le f_{1,2} \le 1 - 2^{-n}$

(n = number of mantissa bits)

$$A_1 + A_2 = f_1 2^{e_1} + f_2 2^{e_2}$$

$$= \left[f_1 + \left[f_2 \; 2^{-(e_1 - e_2)} \right] \right] 2^{e_1} \quad , \; e_1 > e_2$$

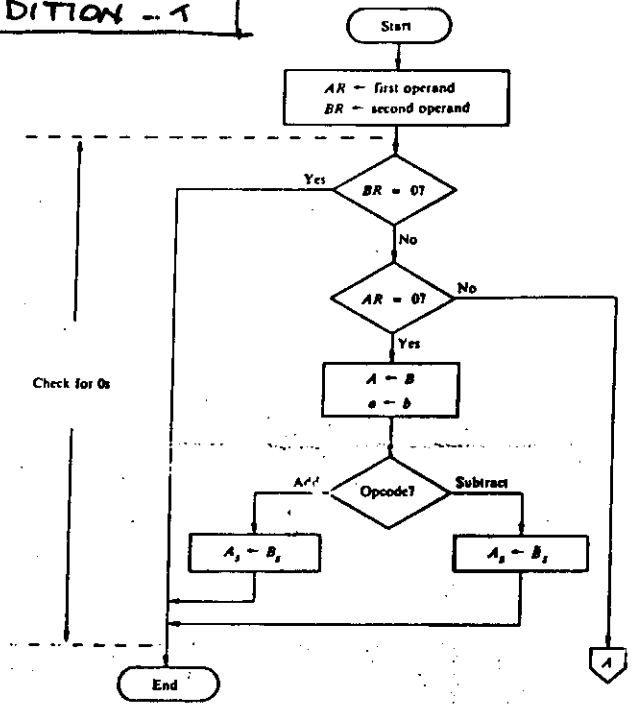
$$= \left[\left[f_1 \, 2^{-(e_2 - e_1)} \right] + f_2 \right] 2^{e_2} \quad , \; e_1 \le e_2$$

so, $e_1 > e_2$ → f_1 is added to f_2 aligned (right shifted)

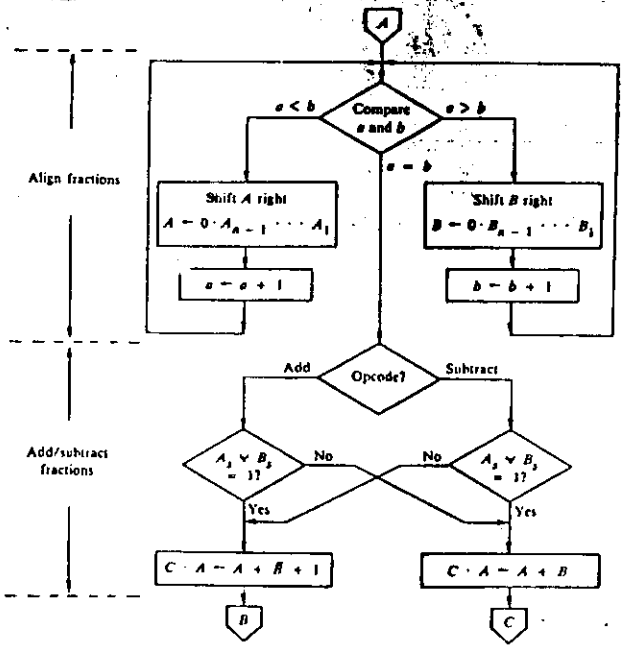
- In any case: THE TWO OPERANDS HAVE TO BE ALIGNED BEFORE ADDITION
  - compare magnitude of exponents
  - shift fraction with $|e_1 - e_2|$ position to the right
  - use the $\max(e_1, e_2)$ exponent for the result
- The result will have $0 \le |f| < 2$ (for a carry out) and needs to be normalized

- OVERFLOW, UNDERFLOW
  - when aligning operands and adjusting exponents significant bits may be lost from the right end → Fraction underflow
  - when the resulting fraction $\to 0$ → ? (usually the exponent is set to 0)

FLOATING POINT ADDITION - 1

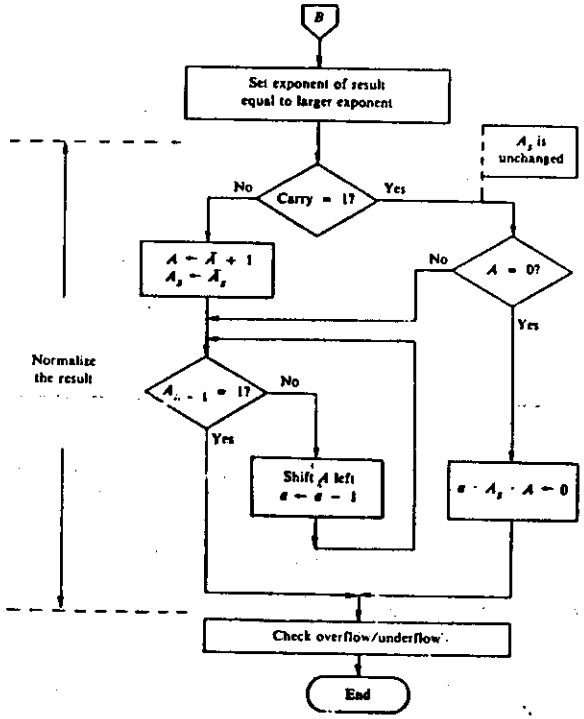


(a)

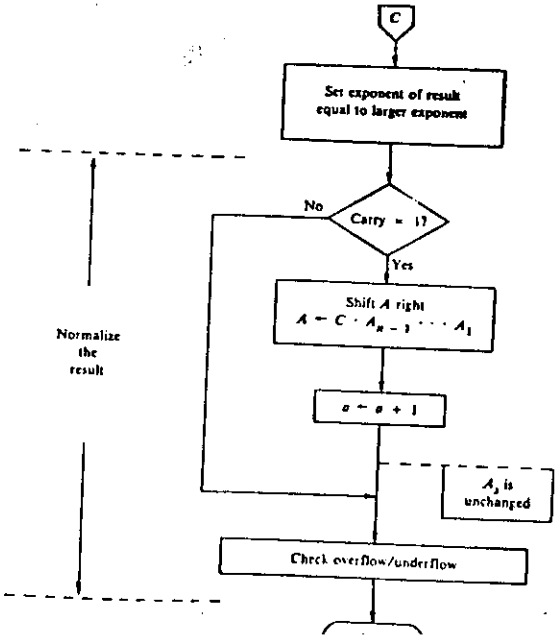


(b)

FLOATING POINT ADDITION - 2



(c)



FLOATING POINT ADDITION / SUBTRACTION UNIT

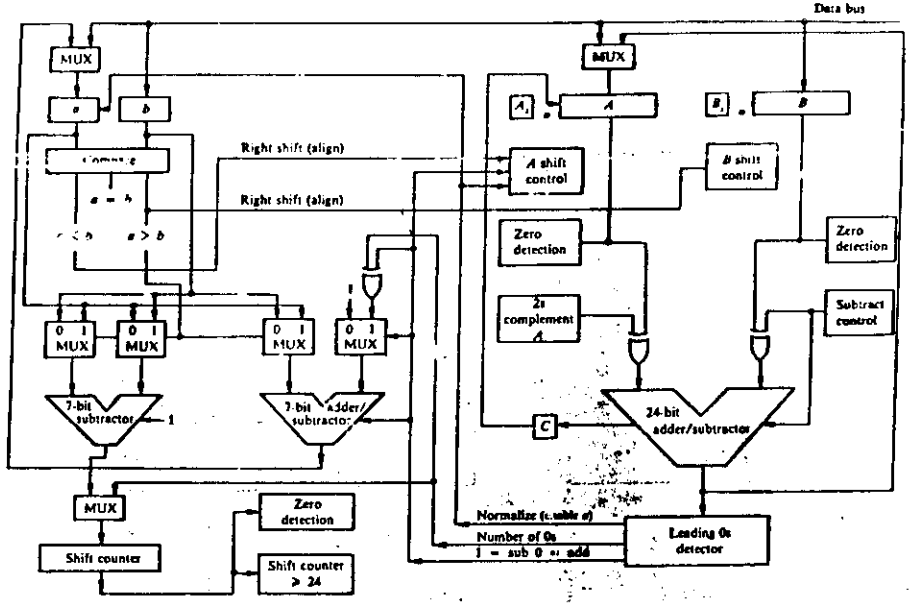


Figure 6.9 Floating-point addition/subtraction unit.

FLOATING POINT MULTIPLICATION

$$A_1 \times A_2 = (f_1 2^{e_1}) \cdot (f_2 2^{e_2}) =$$

$$= (f_1 f_2) 2^{(e_1+e_2)}$$

much simpler than addition

* $f_1 \cdot f_2$ is a fixed point multiplication
  no alignment is necessary
* $f_1 \cdot f_2$ and $e_1 + e_2$ can be operated in parallel
* Post normalization is necessary
  the resulting f:

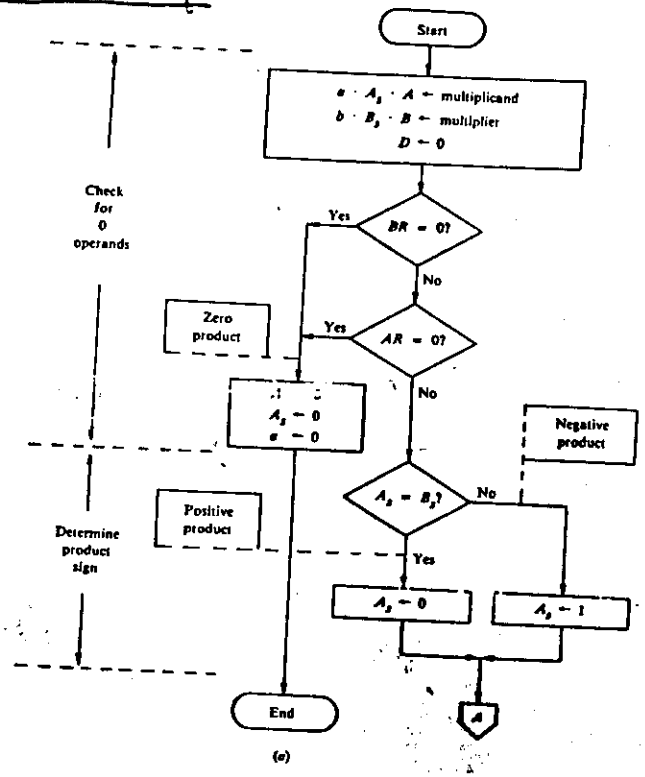
$$\frac{1}{4} \le |f_1 \cdot f_2| < 1$$

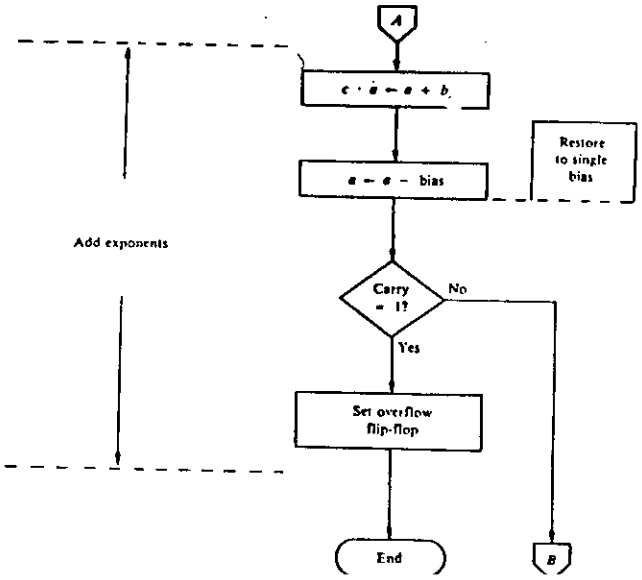
  if $\frac{1}{4} \le |f| < \frac{1}{2}$ we need post normalization (1 shift left)
  else if $\frac{1}{2} \le |f| < 1$ we do not need normalization

- OVERFLOW, UNDERFLOW
  - when exponent are added the number may be too large
  - " " are subtracted " " " too small (division)

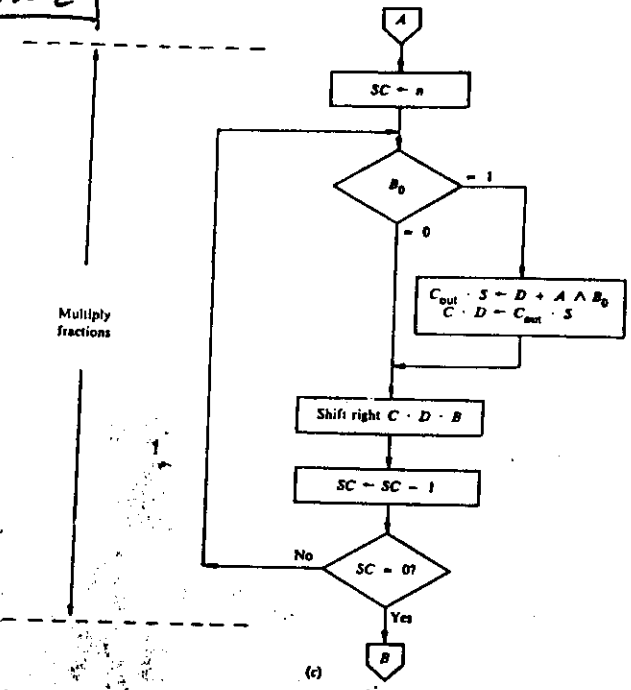
  (exponent over/underflow)

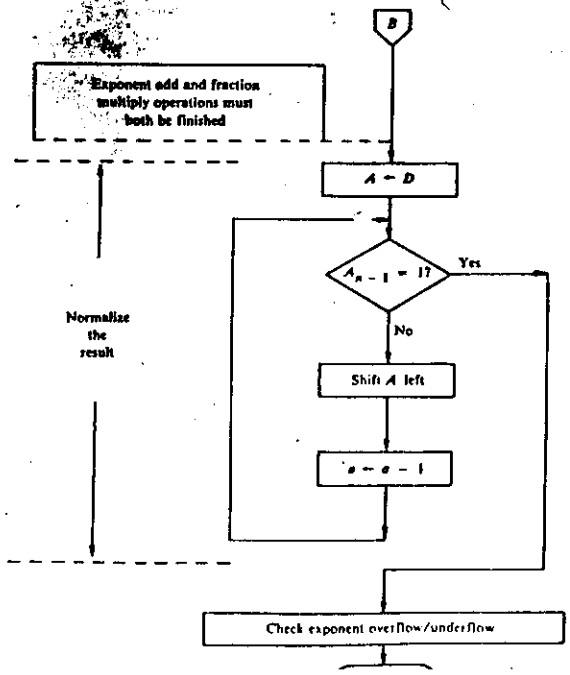
FLOATING POINT MULTIPLICATION - 1





FLOATING POINT MULTIPLICATION - 2





FLOATING-POINT MULTIPLICATION UNIT

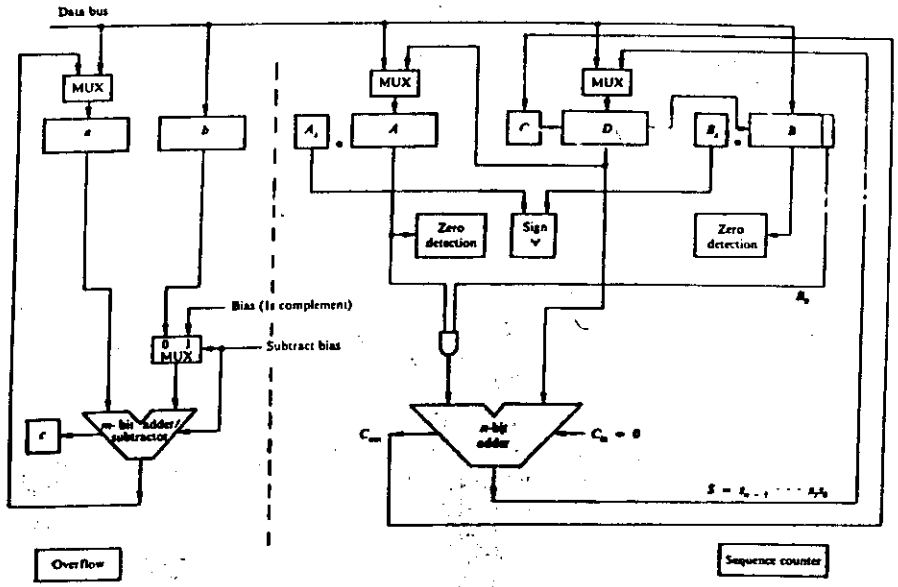


Figure 6.17 Floating-point multiplication unit.