UNIT – V

SUBSYSTEM DESIGN

Most chips are built from collection of subsystems like adders, regs, state machines etc.

Digital fns can be divided into:

- datapath operators
- memory elems
- ctrl structures
- I/O cells.

→ Area & delay costs can be reduced by optimizn of diff levels of abstraction:

layout, circuit, logic, Register transfer.

SHIFTERS

→ shifters are imp elems in many microprocr designs for arithmetic shifting, logical shifting & rotation functions.

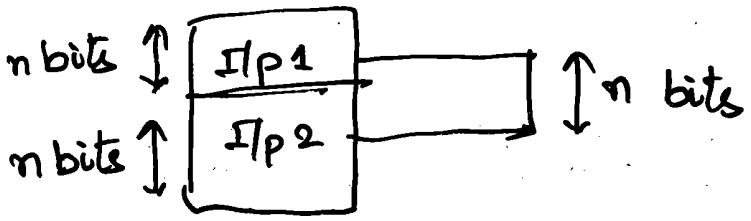
→ A barrel shifter can perform n-bit shifts in a single combinational fn.

→ Barrel shifter accepts '2n' data bits & 'n' ctrl signals and produces 'n' o/p bits.

→ It shifts by transmitting an n-bit slice of the '2n' data bits to the o/ps.

→ The position of the transmitted slice is determined by the ctrl bits; exact opern is determined by values placed at the data I/ps.

ex:- Right shift with zero fill.



data word 'd' into top i/p.
all 0's into bottom i/p.

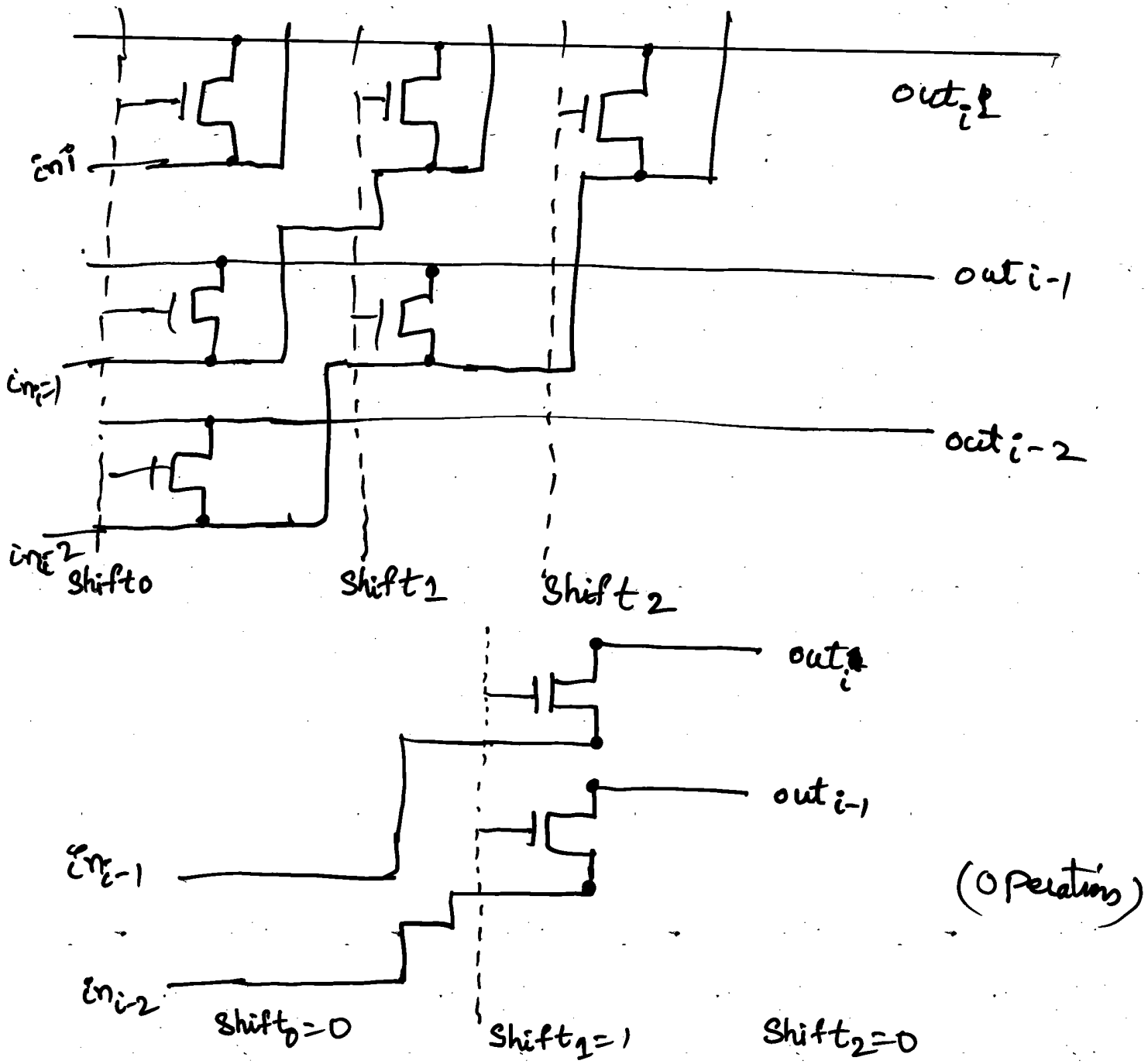
The o/p is right shift with zero fill.

→ by setting ctrl bits to select top-most 'n' bits is shift of zero.

Selecting bottom 'n' bits is an n-bit shift pushing entire word out of the shifter.

(2) Rotate Operation

When both top & bottom i/ps have same data. Rotate operation shifting out top bits of word causing those bits to reappear at the bottom of o/p



(Operation)

→ Barrel shifter with 'n' o/p bits is built from '2n' vertical by 'n' horz'l array of cells, each has single Tr & few wires.

→ Cell is Tx^m gate formed using single n-type Tr.

→ Ctrl lines run vertically; I/p run diagnally upward thru the system; o/p horizontally.

Ctrl lines set to '1', turns T_x^n gate in single column.

The T_x^n gate connect diagonal I/p wires to horz o/p wires; when col is turned on, all I/ps are shunted to the o/ps.

The length of the shift is determined by position of the selected column.

ADDERS:

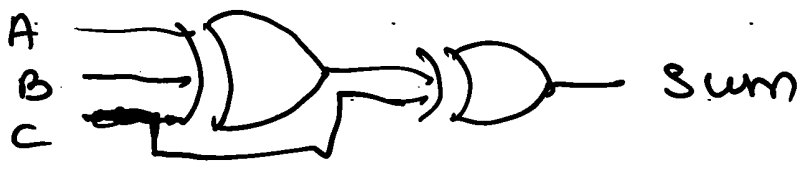
1) Full adder: 1-bit sum; 1-bit carry.

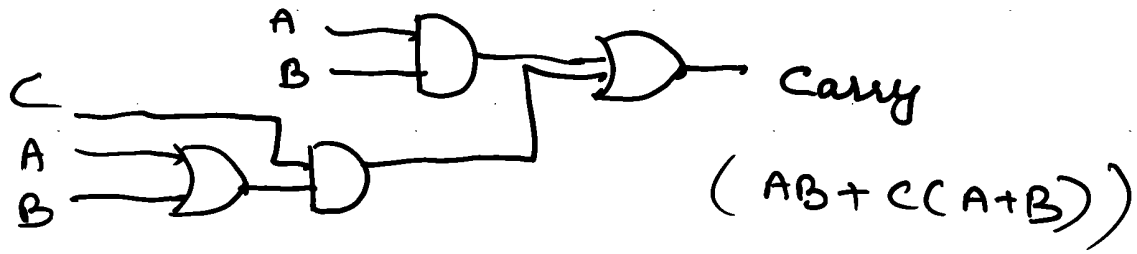
$$Sum = A\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + ABC$$

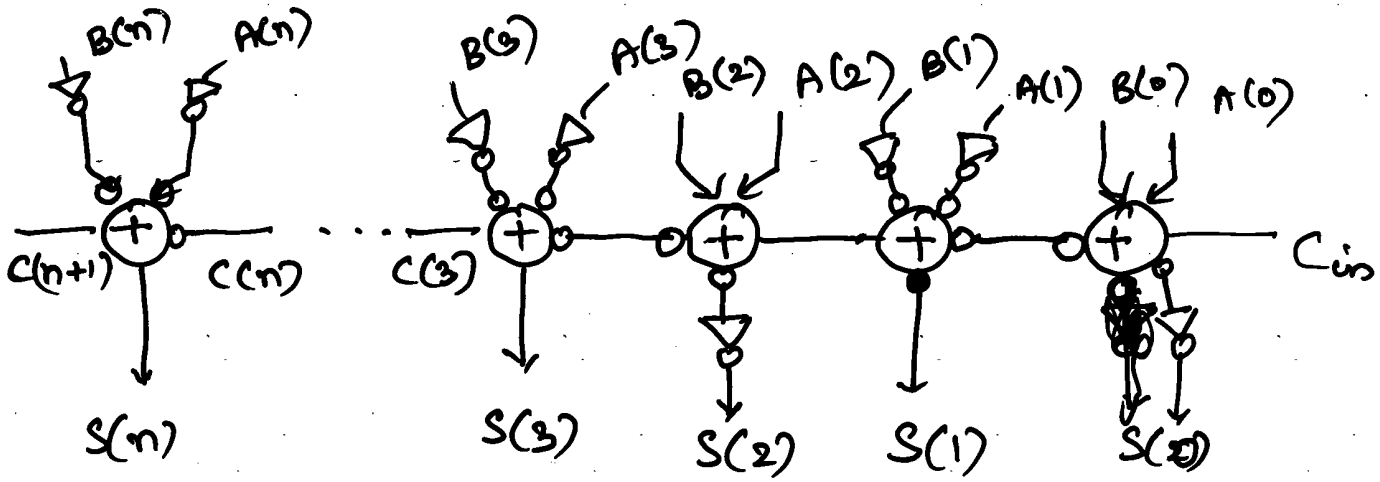
$$Sum_i = A_i \oplus B_i \oplus C_i$$

$$C_{i+1} = Carry_i = A_iB_i + B_iC_i + A_iC_i$$

Sum_i = Sum at i^{th} stage, C_{i+1} = Carry.



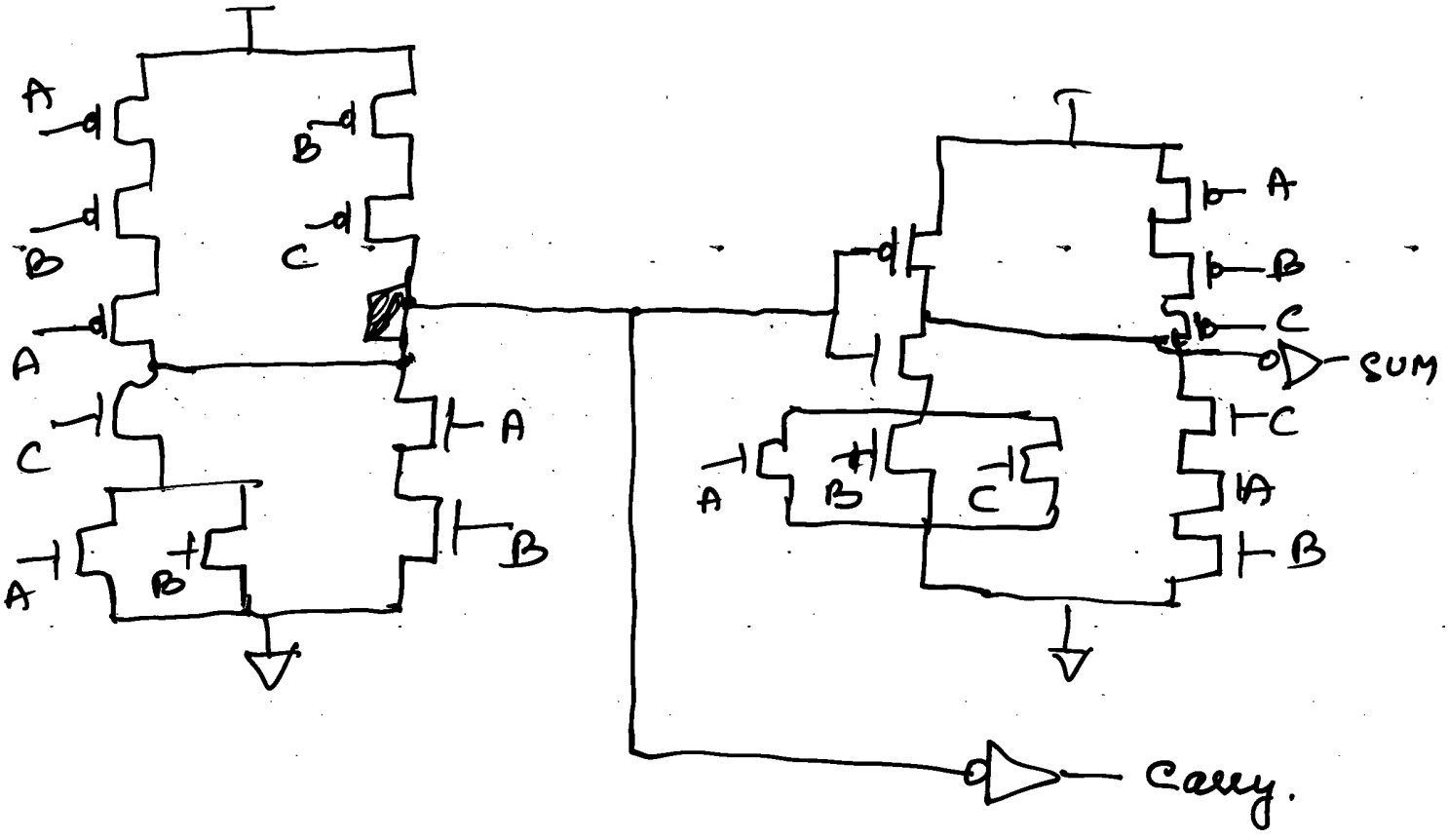




Single-bit adder (28 Trs)

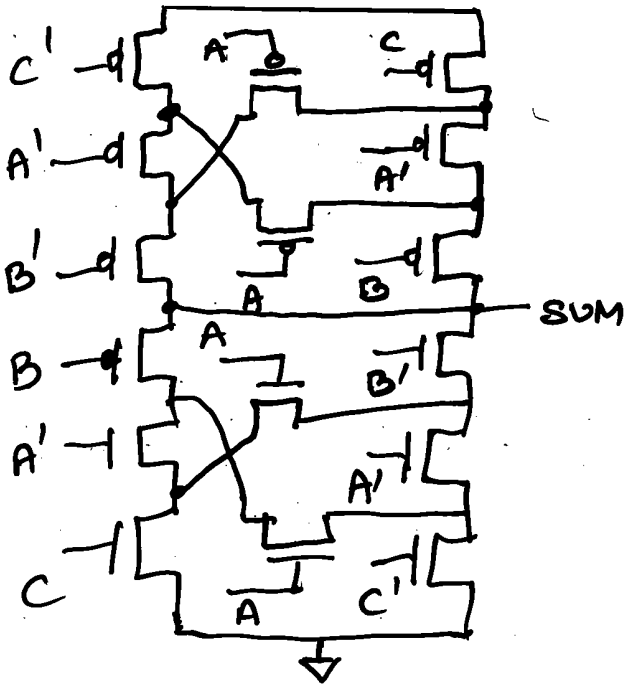
$$SUM = ABC + (A+B+C)\,\overline{CARRY}$$

$$C = ABC + (A+B+C)\left(\overline{AB + C(A+B)}\right)$$

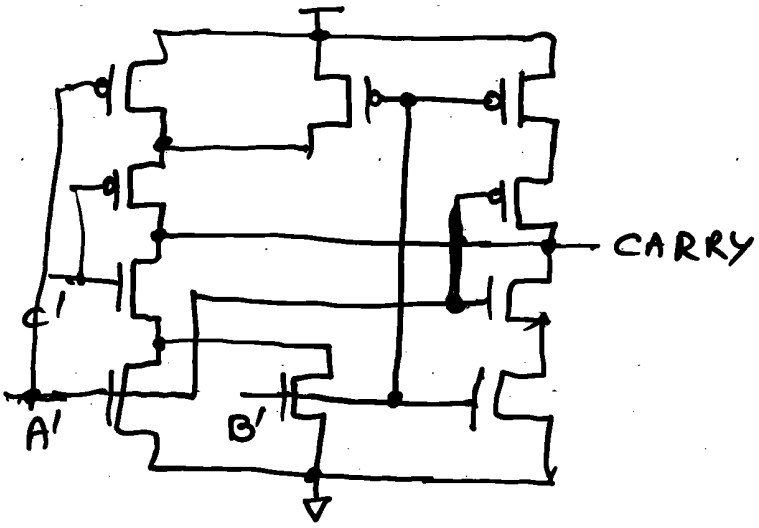


FA using XOR gates uses 32 transistors.

FA using cascaded logic gates uses 28 transistors.



$S = (A \oplus B) \oplus C = P \oplus C$



Carry = $AB + AC + BC$

CY = $\overline{A'B' + C'(A' + B')}$

(2) Bit-Parallel Adder (Ripple Carry Adder).

⇒ n-bit adder built by cascading 'n' 1-bit adders.

→ Ripple-carry adder is easy to design but is slow when 'n' is large.

→ The I/ps are n-bit A & B. The carry signal of stage 'i' is fed to the 'c' signal of stage 'i+1' & 'SUM' signal forms n bit o/p.

(3) n-bit subtractor.

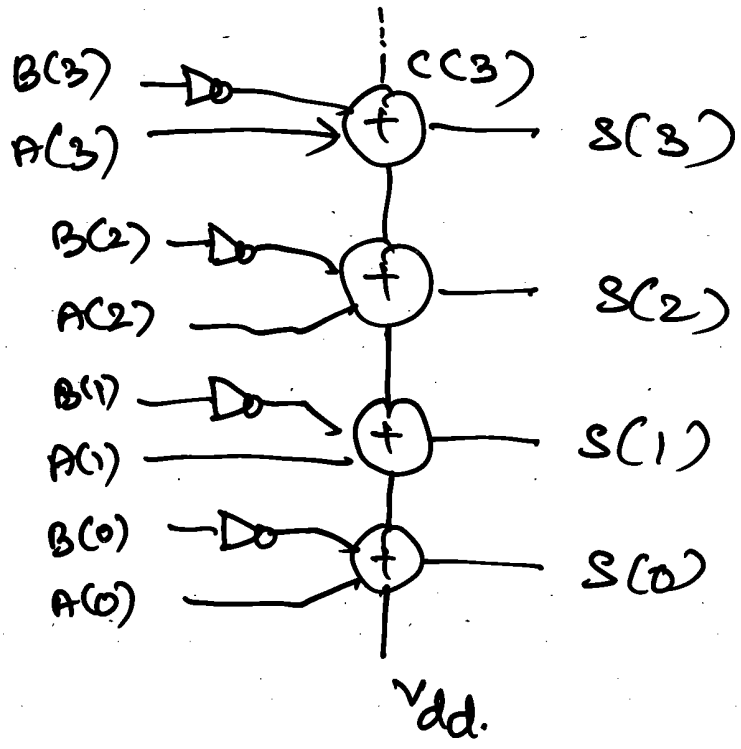
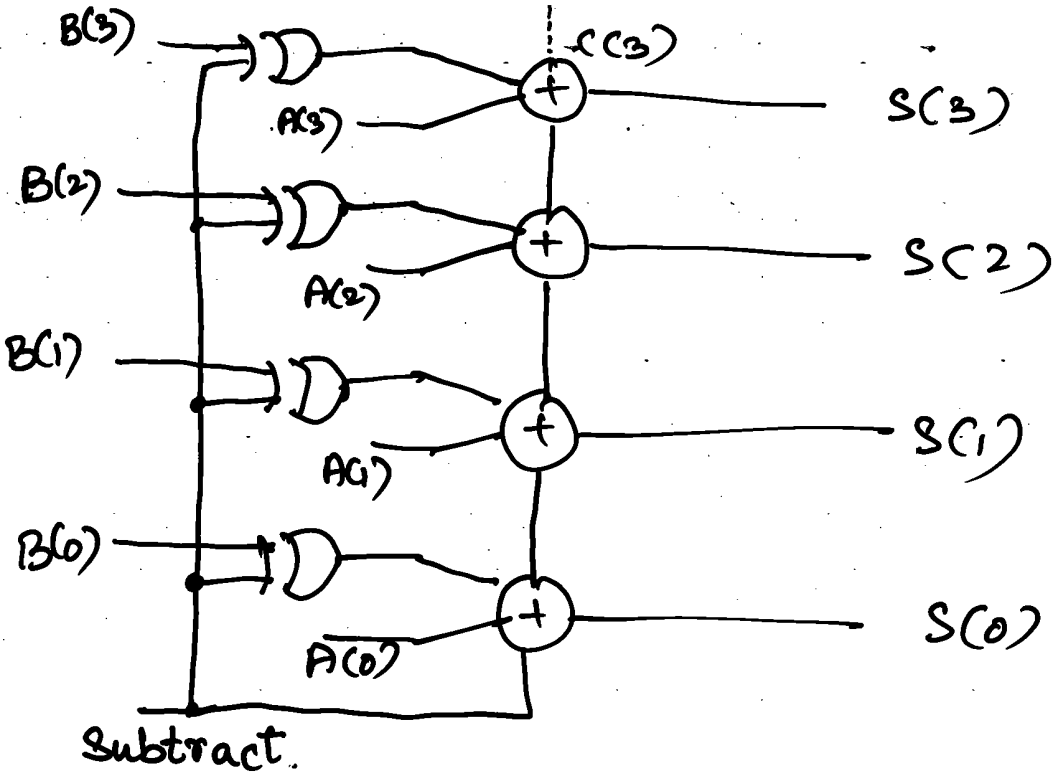


fig: A-B.

(4) Adder/Subtractor.

if subtract=0
S= A+B
else
S= A-B



(3) 4-bit subtractor.

fig: A-B.

(4) Adder/subtractor.

if subtract=0
S = A+B
else
S = A-B

(5) Carry-look ahead adder.

If a_i's, b_i's are I/Ps, P = Propagate
G = Generate

where

$$P_i = a_i + b_i$$
$$G_i = a_i b_i$$

Sum $S_i = c_i \oplus P_i \oplus G_i$

$$c_{i+1} = G_i + P_i c_i$$

$$\therefore c_{i+1} = G_i + P_i (G_{i-1} + P_{i-1} \cdot c_{i-1})$$

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + P_i P_{i-1} P_{i-2} c_{i-2}$$

Thus c_{i+1} depends on c_{i-2} & not on 'c_i' or 'c_{i-1}'.

→ The carry-look ahead units can be recursively connected to form a tree; each unit generates its own 'P' & 'G' values, which are used to feed carry-lookahead unit at the next level of the tree.

(7) Carry-Select Adder.

→ computes two versions of the addition with diff carry-ins, then selects the right one.

→ m-bit stages.

→ The 2nd stage computes 2 values: one assuming carry-in is '0' & another '1'.

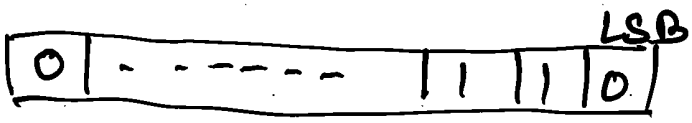
→ The carry out of prev stage is used to select which version is correct: Muxes ctrlled by prev stage's carry-out choose correct sum & carry-out.

→ This speeds up addn coz ith stage can be computing two versions of the sum in parallel with i-1th's computation of its carry.

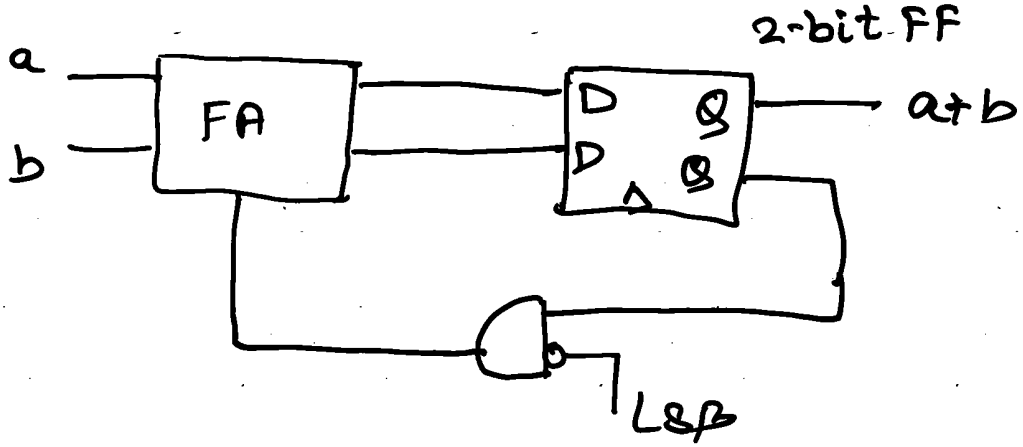
(8) Serial Adders:

→ Require many clk cycles to add two n-bit nos.

→ Small



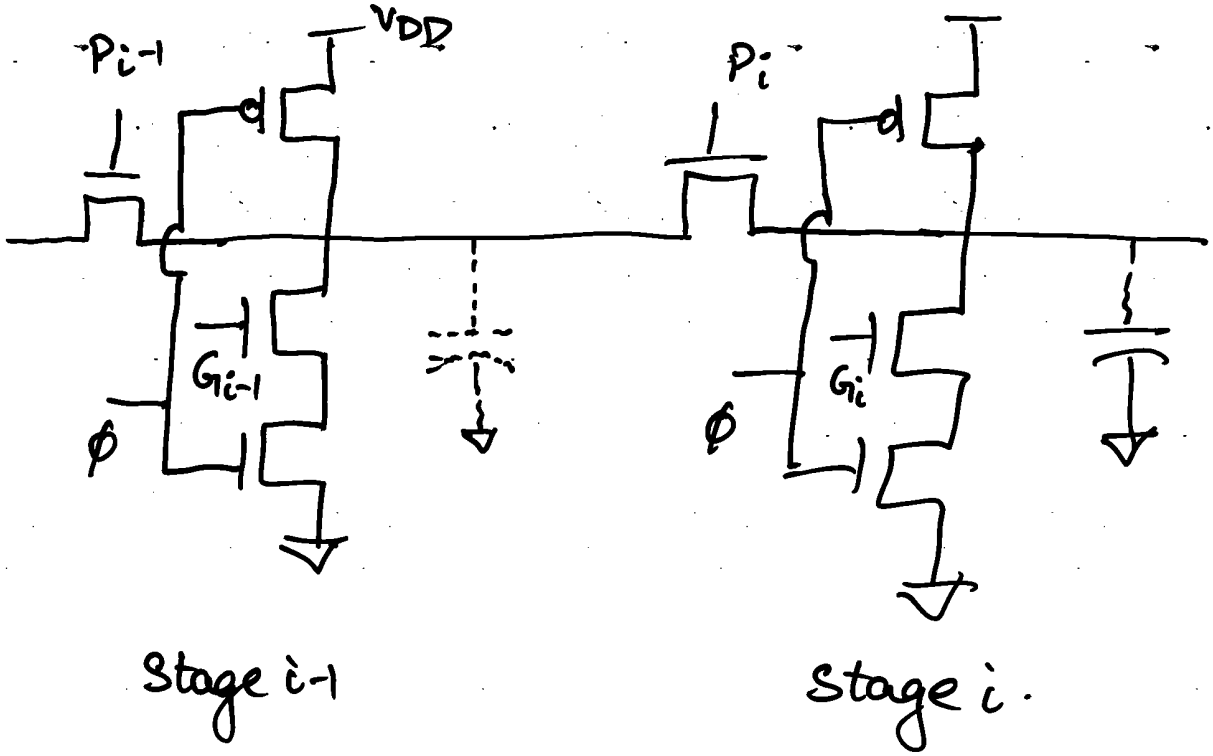
← data format



LSB = high when curr data bits are LSBs of the addends.

(9) Manchestar Carry Chain :

→ Precharged adder.



Stage i-1 Stage i.

→ Storage node holds complementary (C_i') charged to '1' during precharge phase.

→ If $G_i = 1$, during evaluate stage, storage node is discharged, producing carry into next stage.

If $P_i = 1$, then ith storage node is connected to $(i-1)^{th}$ storage node; ith storage node can be discharged by P_{i-1} P.d.

→ 16-bit adders. (Pwr Consumpn)

adder	Pwr (mv)
* Ripple-Carry	0.117
* Const width carry-skip	0.109
* Carry-lookahead	0.171
* Carry-select	0.216.

Carry-Save Adder :

→ Save carry & sum for each cycle.

→ n-bit adders: 'n' carries & 'n' sums.

→ n-bit CSA reqs '2n' registers.

→ reduces critical path. of reg, adder delay set up time into reg.

Tx - Gate Adder (TG)

(6 Trs)

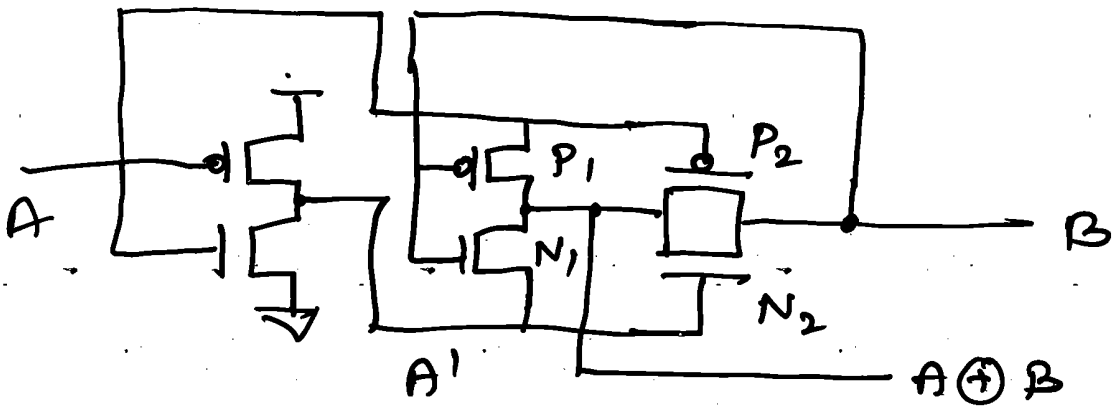


fig: Txn gate XOR. (Tiny XOR)

→ A = logic 1, A' = low, P_1 & N_1 : inverter with B' at o/p. Tx gate by pair P_2 & N_2 is open.

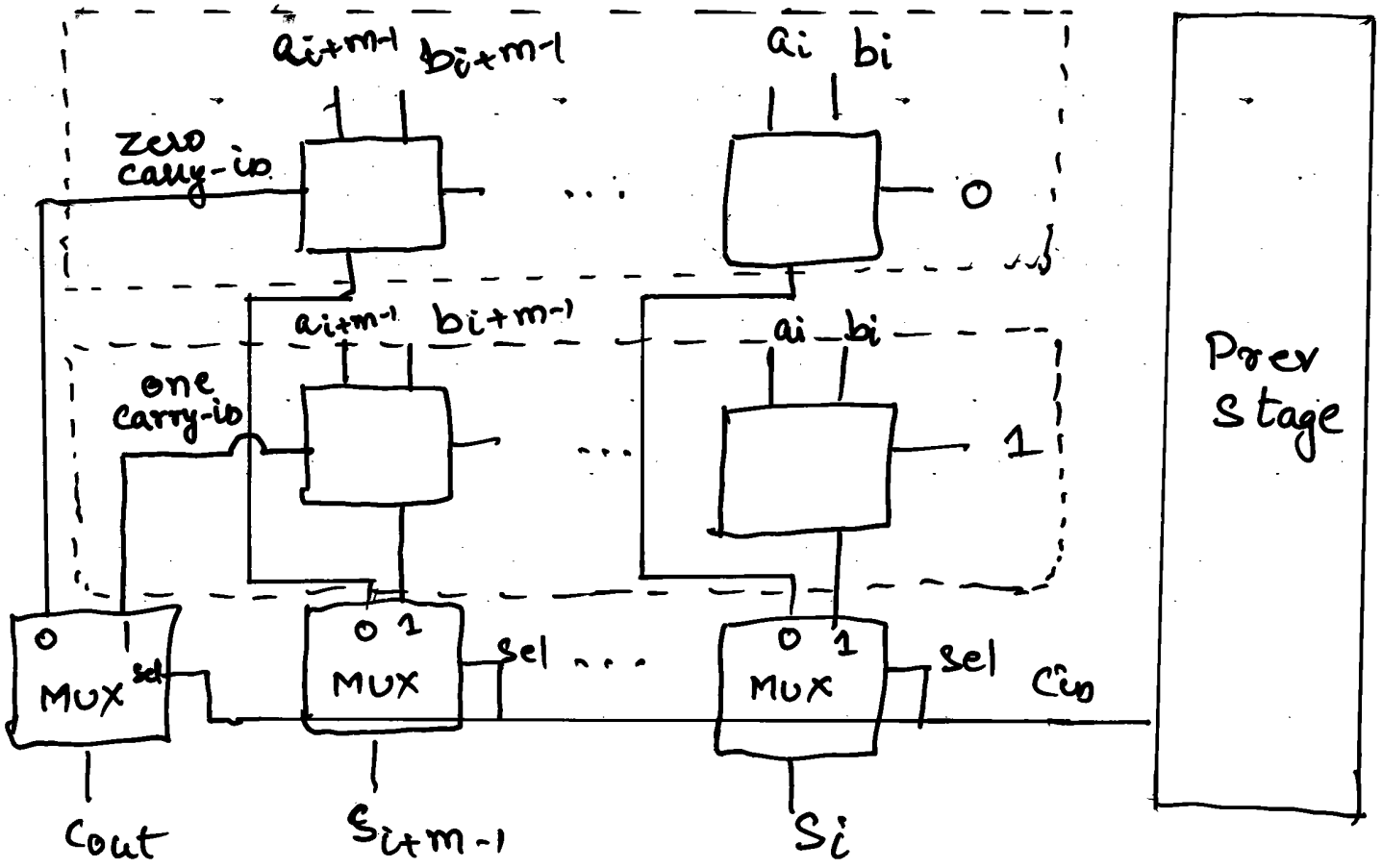
→ A = low, A' = high, (P_2, N_2) = closed, passing 'B' to o/p. The inv^r (P_1, N_1) is partially disabled

(6) Carry-Skip Adder:

→ This looks for cases where carry out of a set of bits is same as the carry in to those bits.

→ This adder is organized into m-bit groups; if the carry is propagated for every bit in the stage, then bypass gate sends the g stage's carry I/p directly to carry o/p.

→ divided into 2 groups of bits;
- * True Carry into group
- * True Propagate cond^n 'P' at every bit in the group is needed to cause carry to skip.



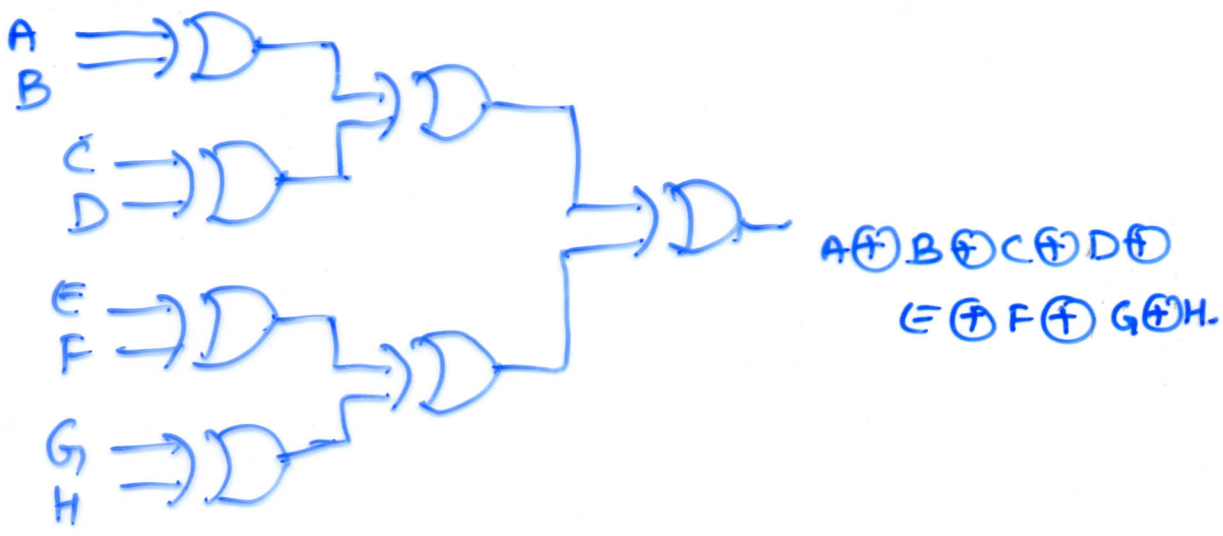
PARITY GENERATORS

Detects no. of 1s in i/p word as odd or even.

Function is

$$\text{Parity} = A_0 \oplus A_1 \oplus \cdots \oplus A_n.$$

(a) Static XOR tree.

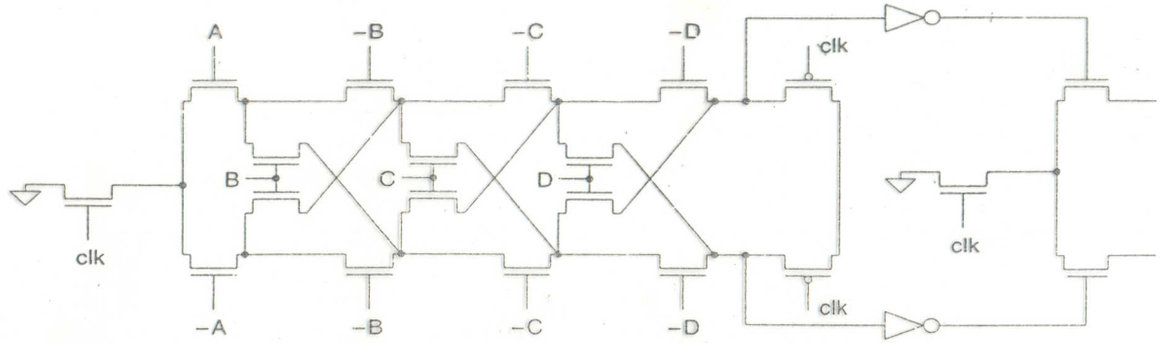


COMPARATOR:

→ Used to compare the magnitude of two binary nos.

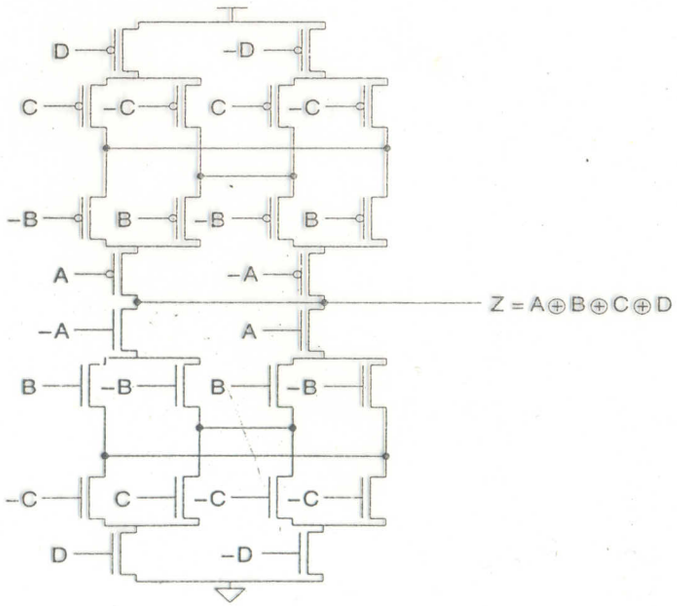
→ It can be built using an adder and complementer.

(a)



(b)

(dual-rail)



(c)

Figure 8.25 Parity generation: (a) static XOR tree; (b) dynamic version; (c) static 4-input XOR

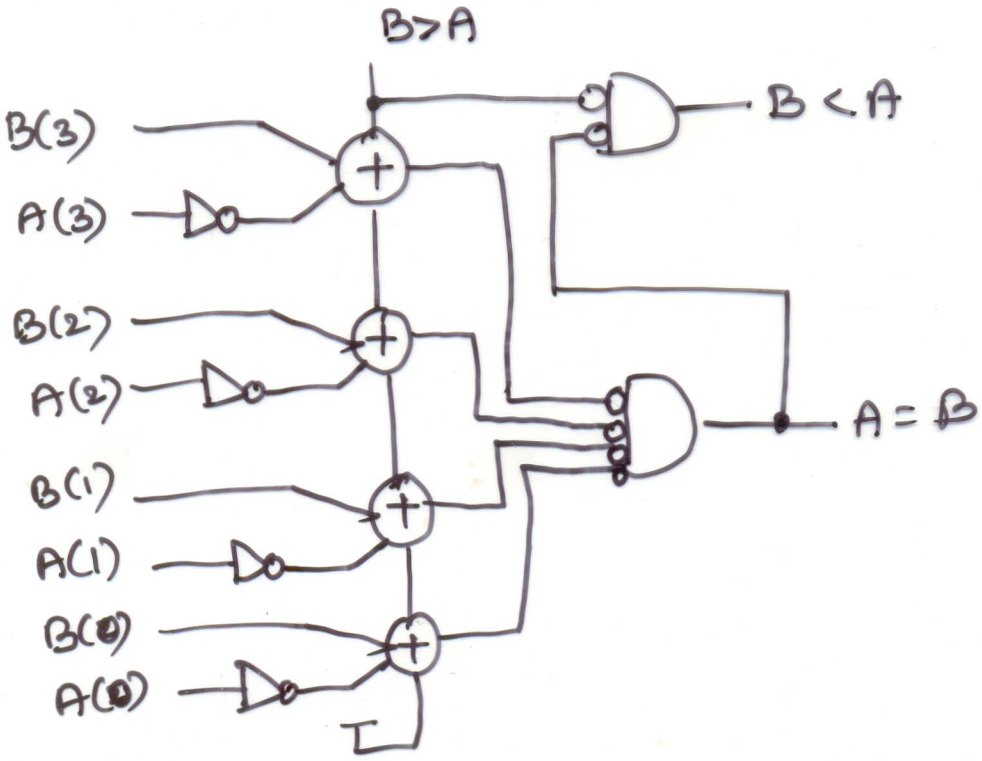


fig: Comparator using adder.

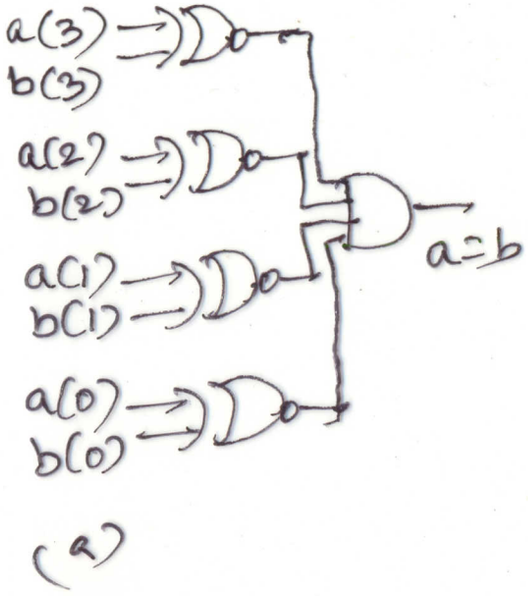
→ A zero detect (NOR gate) provides A=B signal while final carry o/p provides B>A signal.

→ other signals like A<B or A≤B, can be generated by logical combination of these signals.

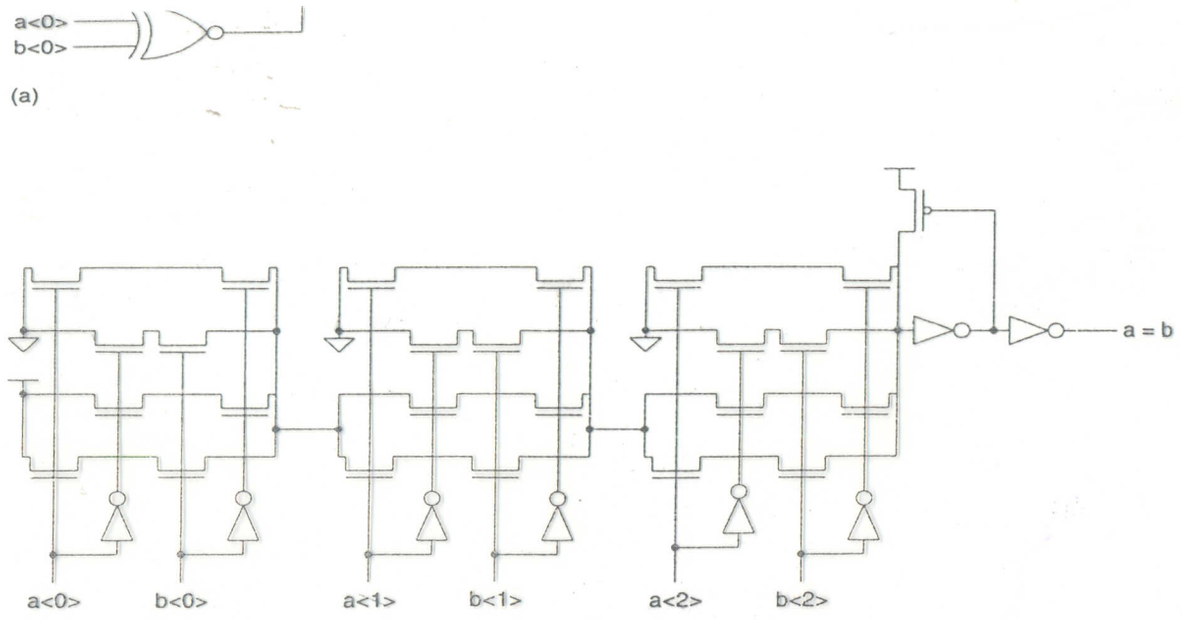
→ For equality, need only XNOR & AND gate.

→ fig(b): pass-gate logic. (Tx^n gate)
draws no DC curr, slow for long comps.

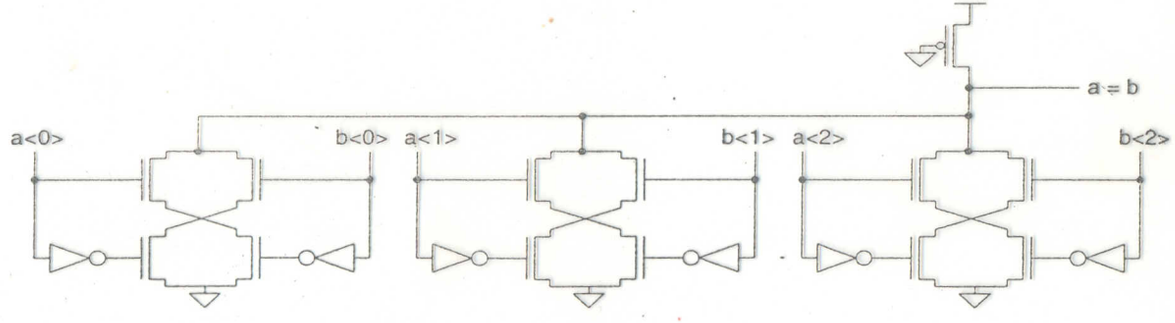
→ fig(c): merged XNOR/NOR gate.
draws DC curr, bt small & fast.



(a)

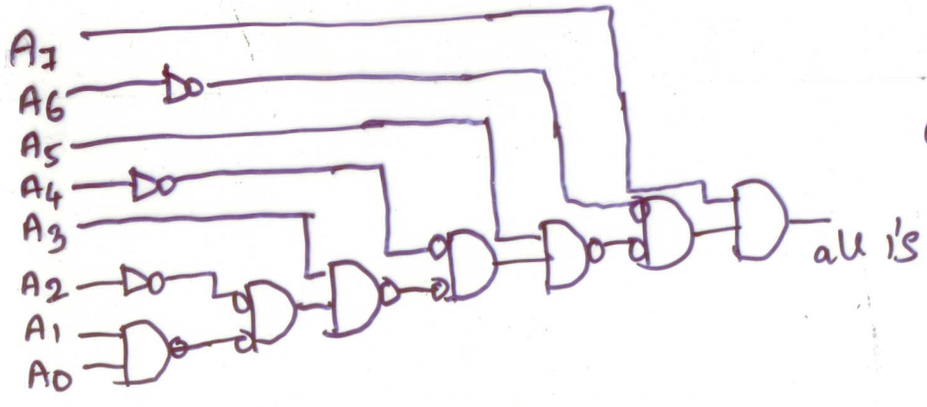


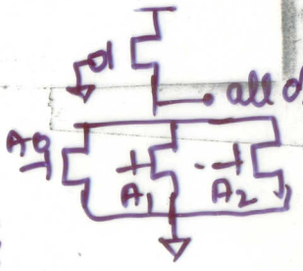
(b)



(c)

Figure 8.27 Comparator circuits: (a) XNOR based; (b) pass gate based; (c) pseudo-nMOS based





* $x_i = A_i B_i + A_i' B_i'$

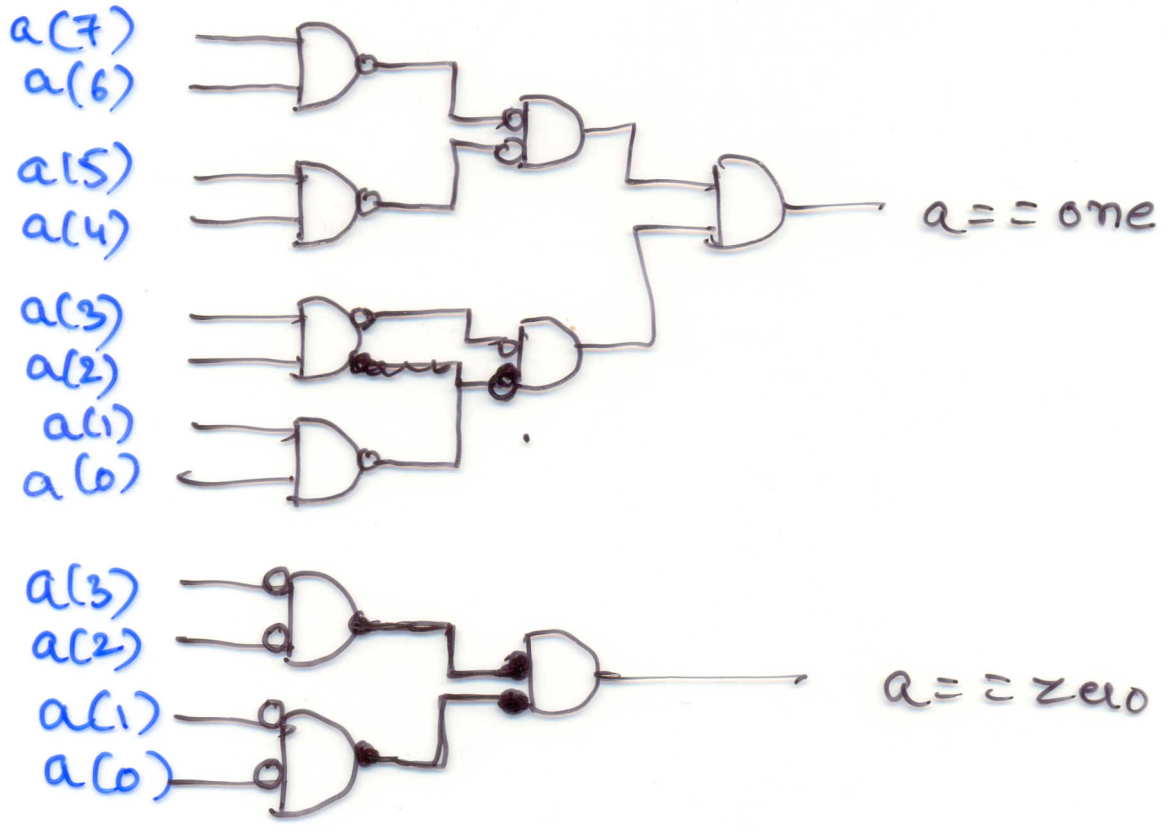
$(A = B) = x_3 x_2 x_1 x_0 = 1$

ZERO/ONE DETECTOR :

→ Detecting all ones or all zeros on wide words req large fan-in AND (or) OR gates.

→ Delay to o/p $\propto \log N$ (N = bit width of word)

→ Pseudo-nMOS NOR gate implementation of zero/one detector is very fast & small for word widths less than 32 bits.



(using alternate NAND & NOR)

COUNTERS:

→ Count goes thru a sequence of binary nos.

Asynchronous Counters

→ o/ps change at varying times w.r.t clk cycle or edge.

ex:- ripple counter.

→ The clking of each stage is carried out by the previous counter stage, & thus the time it takes the last counter stage to settle can be quite large for a long counter chain.

Synchronous Counters:

→ o/ps change at substantially same time.

ex:- up/ down Counter.

→ up/ down counter basically uses adder & a D-FF.

→ The speed is determined by the ripple-carry time from LSB to the MSB.

COUNTERS :

→ Count goes thru a sequence of binary nos.

Asynchronous Counters

→ o/ps change at varying times w.r.t clk cycle or edge.

ex:- ripple counter.

→ The clking of each stage is carried out by the previous counter stage, & thus the time it takes the last counter stage to settle can be quite large for a long counter chain.

Synchronous Counters :

→ o/ps change at substantially same time.

ex:- up/ down counter.

→ up/ down counter basically uses adder & a D-FF.

→ The speed is determined by the ripple-carry time from LSB to the MSB.

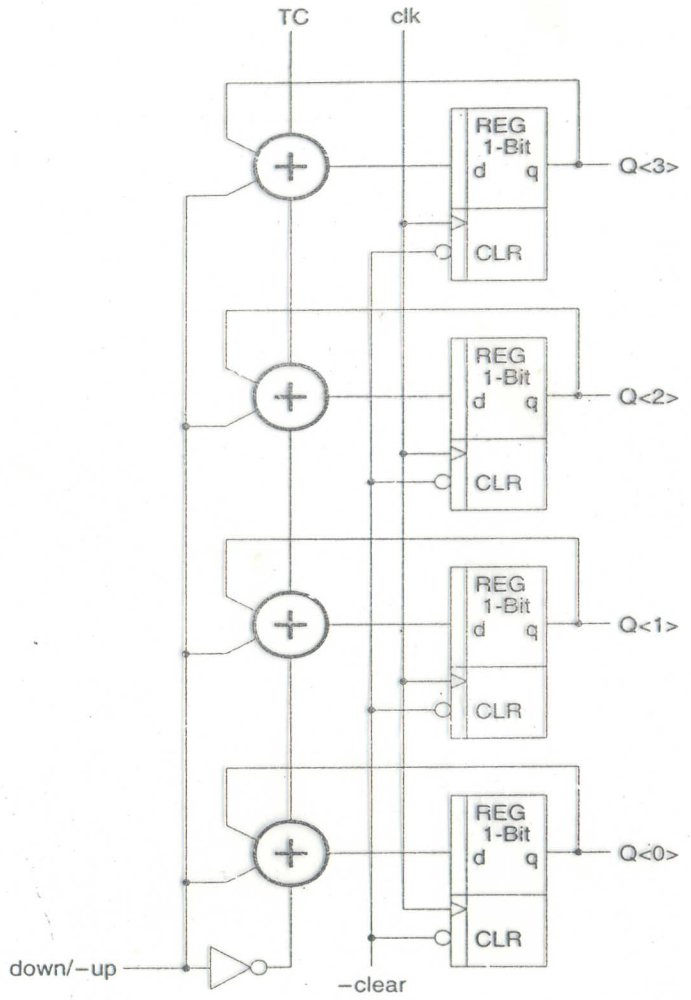


Figure 8.30 Synchronous up/down counter using adders and registers

ALUs :

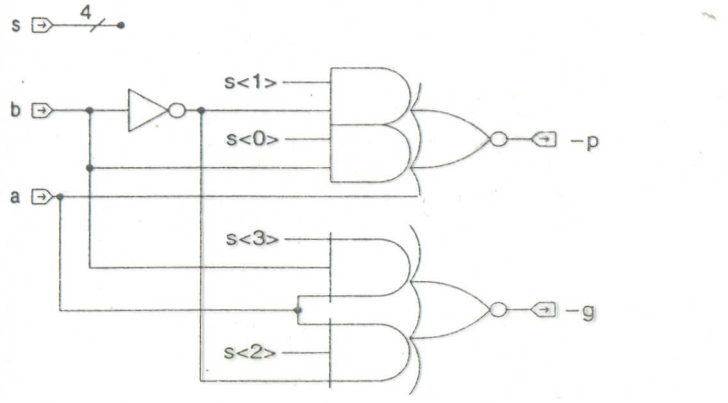
→ Arithmetic Logic Unit

→ Performs both arithmetic & logical oper^{n}s.

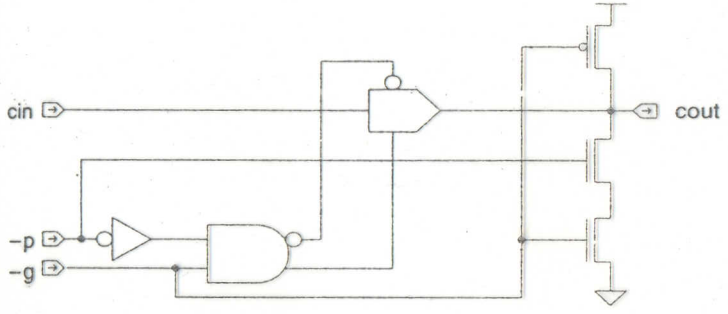
→ Basic ALU takes two data I/ps & a set of ctrl signals, called opcode.

→ The opcode along with ALU's carry-in, determine ALU's function.

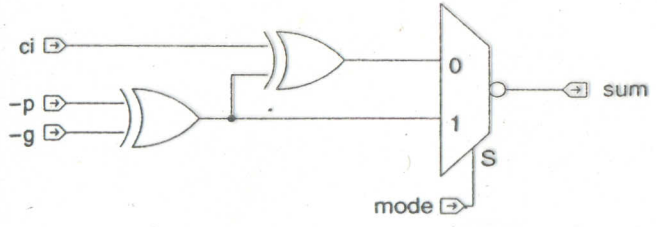
→ Logic to compute all possible ALU fns can be large unless carefully designed.



(a)



(b)



(c)

FIGURE 8.34 181 ALU

Signal mode = false (arithmetic)
= true (boolean).

ex:- S(0)=1, 'b' passed to o/p.

MULTIPLIERS

→ Used in Conv, Correlation, filtering, freq analysis.

→ Shift & add.

→ 2 steps: (i) Evaluation of partial products
(ii) Accumulation of shifted partial products.

→ partial products = ANDing of multiplicand & multiplier bit.

→ Classification of multipliers:
- Serial form
- Serial/parallel form
- Parallel form.

→ In Parallel multiplier, partial products computed in parallel.

ex:- $P = X \times Y = \sum_{i=0}^{m-1} x_i 2^i \cdot \sum_{j=0}^{n-1} y_j 2^j$

$$P = \sum_{k=0}^{m+n-1} P_k 2^k$$

{ P_k = Partial Prod = Summands }

For 4-bit Multiplier.

x_3 x_2 x_1 x_0 — Multiplicand

y_3 y_2 y_1 y_0 — Multiplier

x_3y_0 x_2y_0 x_1y_0 x_0y_0

x_3y_1 x_2y_1 x_1y_1 x_0y_1

x_3y_2 x_2y_2 x_1y_2 x_0y_2

x_3y_3 x_2y_3 x_1y_3 x_0y_3

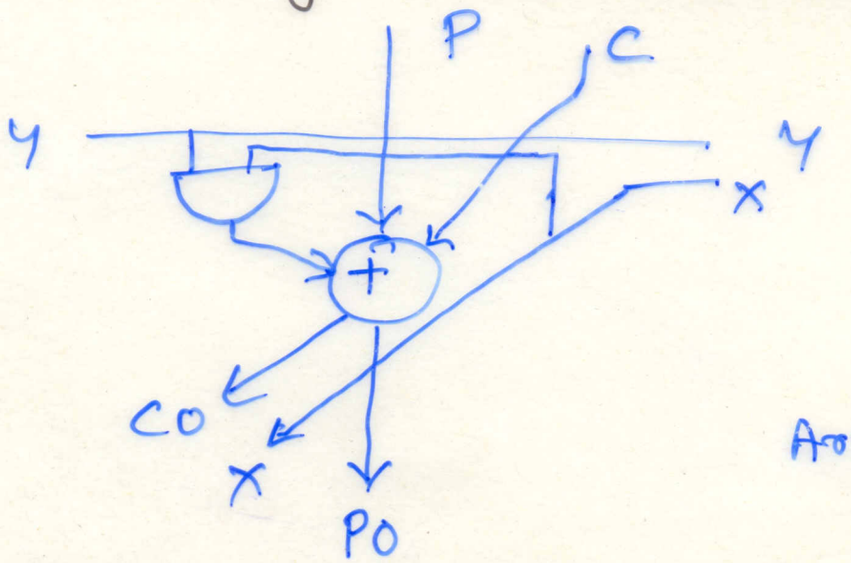
P_7 P_6 P_5 P_4 P_3 P_2 P_1 P_0 — Product

→ $n \times n$ Multiplier reqs:

$n(n-2)$ FAs, n HAs, n^2 AND gates.

→ Worst-case delay = $(2n+1)\tau_g$

τ_g = worst-case adder delay.



Array Multiplier Cell.

x_i : propagated diagonally from top right to bottom left.

Y_j : horizontally.

→ The bit-wise AND is performed in the cell, & SUM is passed to next cell below.

→ "Carry-out" is passed to the bottom left of the cell.

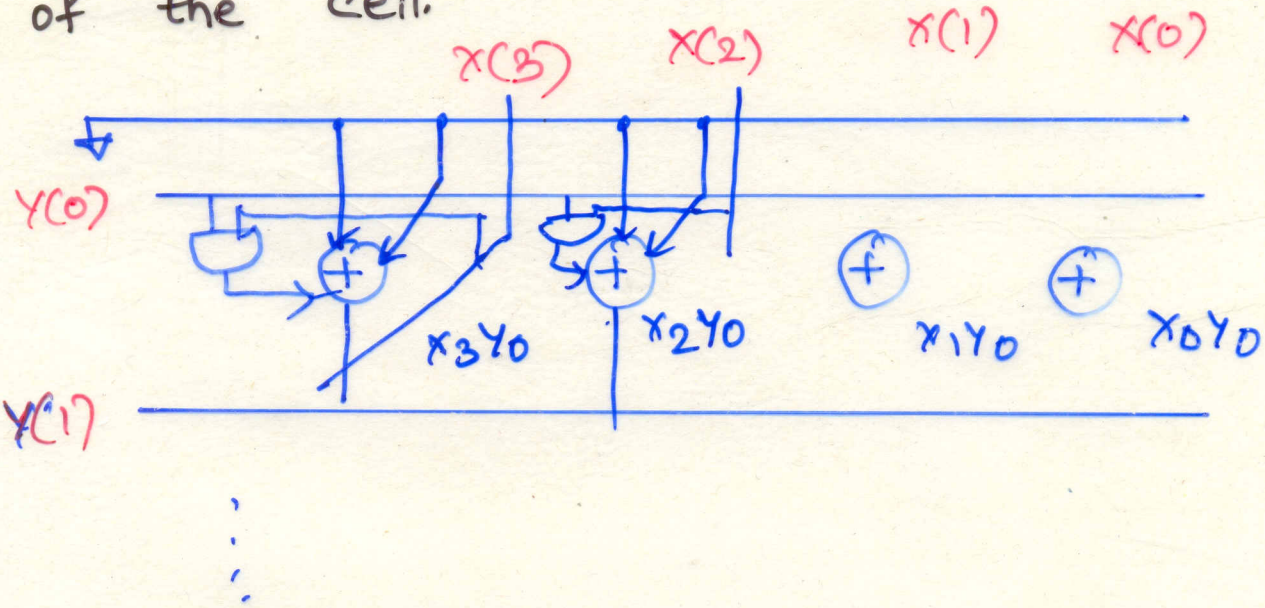


fig:- 4x4 array multiplier

Radix-n Multiplier :-

→ Above is radix-2 multiplier, coz computation done by one bit of multiplicand at a time.

→ High-radix multipliers reduce no. of adders & delay reqd to compute partial sums.

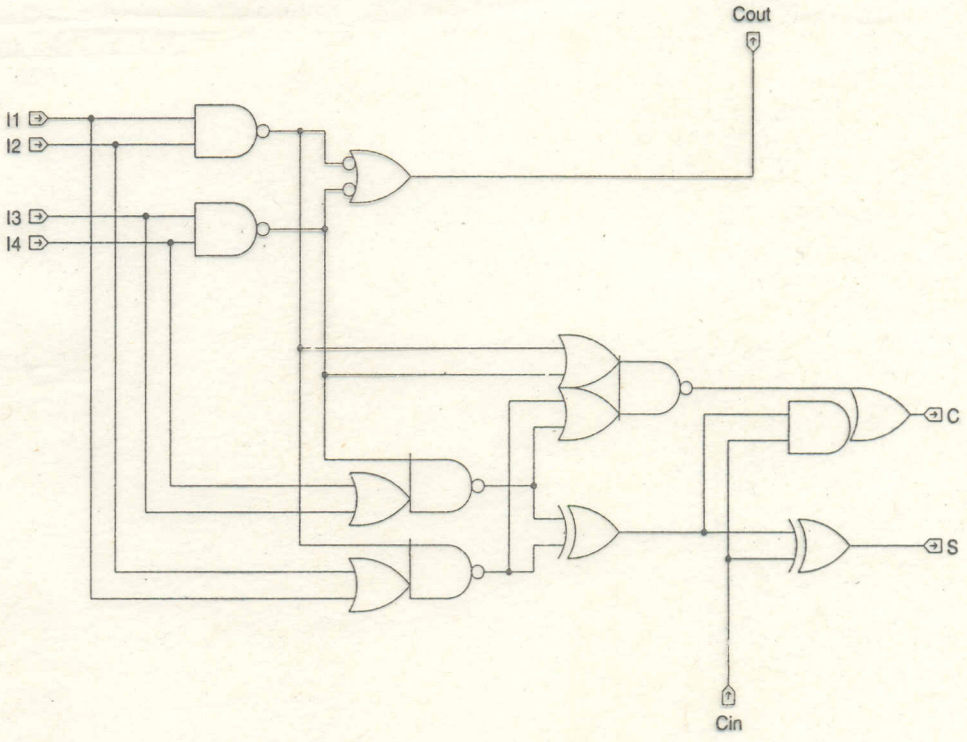


Figure 8.41 A 4:2 (5:3) compressor circuit

The two numbers X and Y are presented serially to the circuit (at different rates to account for multiplier and multiplicand word-lengths). The partial product is evaluated for every bit of the multiplier, and a serial addition is performed with the partial additions already stored in the register. The AND gate ($G2$) between the input to the adder and the output of the register is used to reset the partial sum at the beginning of the multiplication cycle. If the register is made of $N-1$ stages, then the 1-bit shift required for each partial product is obtained automatically. As far as the speed of operation is concerned, the complete product of $M+N$ bits can be obtained in MN intervals of the multiplicand clock.

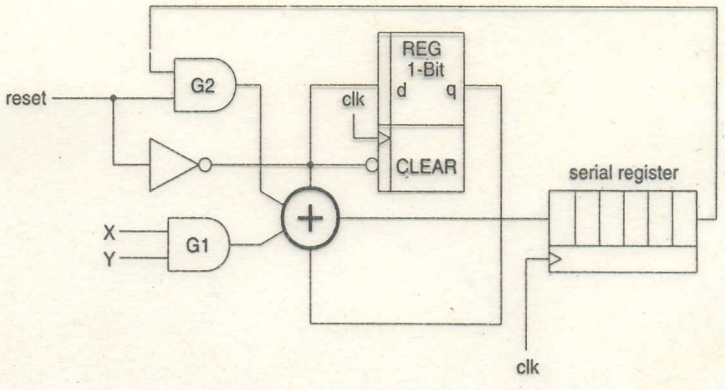


FIGURE 8.42 Serial multiplier

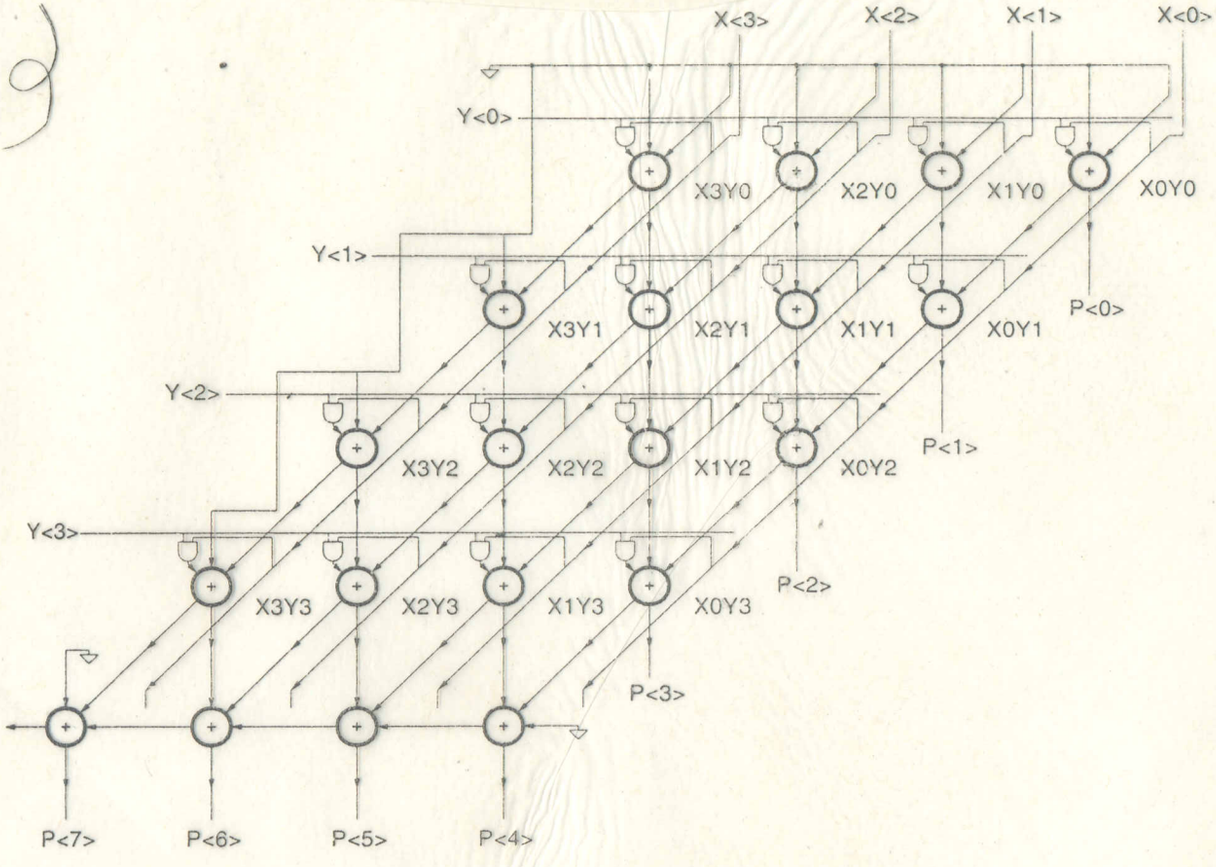


FIGURE 8.36 A 4 × 4 array multiplier

array cell). In this case the appropriate inputs to the first and second row would be connected to ground, as shown in Fig. 8.36.

The cell design for this multiplier is relatively straightforward, with the main attention paid to the adder. An adder with equal carry and sum propagation times is advantageous, because the worst-case multiply time depends on both paths.

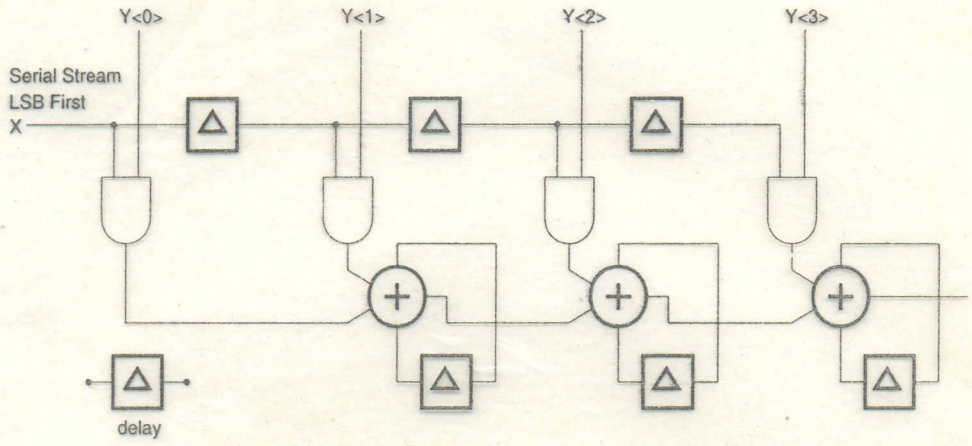


FIGURE 8.43 Serial/parallel multiplier

Using the general approach discussed previously, it is possible to realize a serial/parallel multiplier with a very modular structure that can easily be modified to obtain a pipelined system. The basic implementation is illustrated by Fig. 8.43. In this structure, the multiplication is performed by means of successive additions of columns of the shifted partial products matrix. As left-shifting by one bit in serial systems is obtained by a 1-bit delay element, the multiplier is successively shifted and gates the appropriate bit of the multiplicand. The delayed, gated instances of the multiplicand must all be in the same column of the shifted partial-product matrix. They are then added to form the required product bit for the particular column.

This structure requires $M + N$ clock cycles to produce a product. The main limitation is that the maximum frequency is limited by the propagation through the array of adders. The structure of Fig. 8.43 can be pipelined with the introduction of two delay elements in each cell, as shown in Fig. 8.44. If rounding or truncation of the product term to the same word length as the input is tolerated, then the time necessary to produce a product is 2M clock cycles. In this case the multiplier accumulates partial product sums, starting with the least significant partial product. After each addition, the result is an

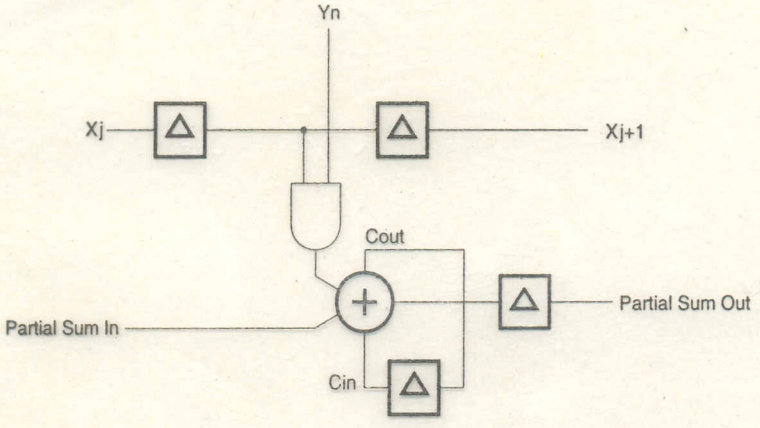


FIGURE 8.44 Pipelined serial/parallel multiplier

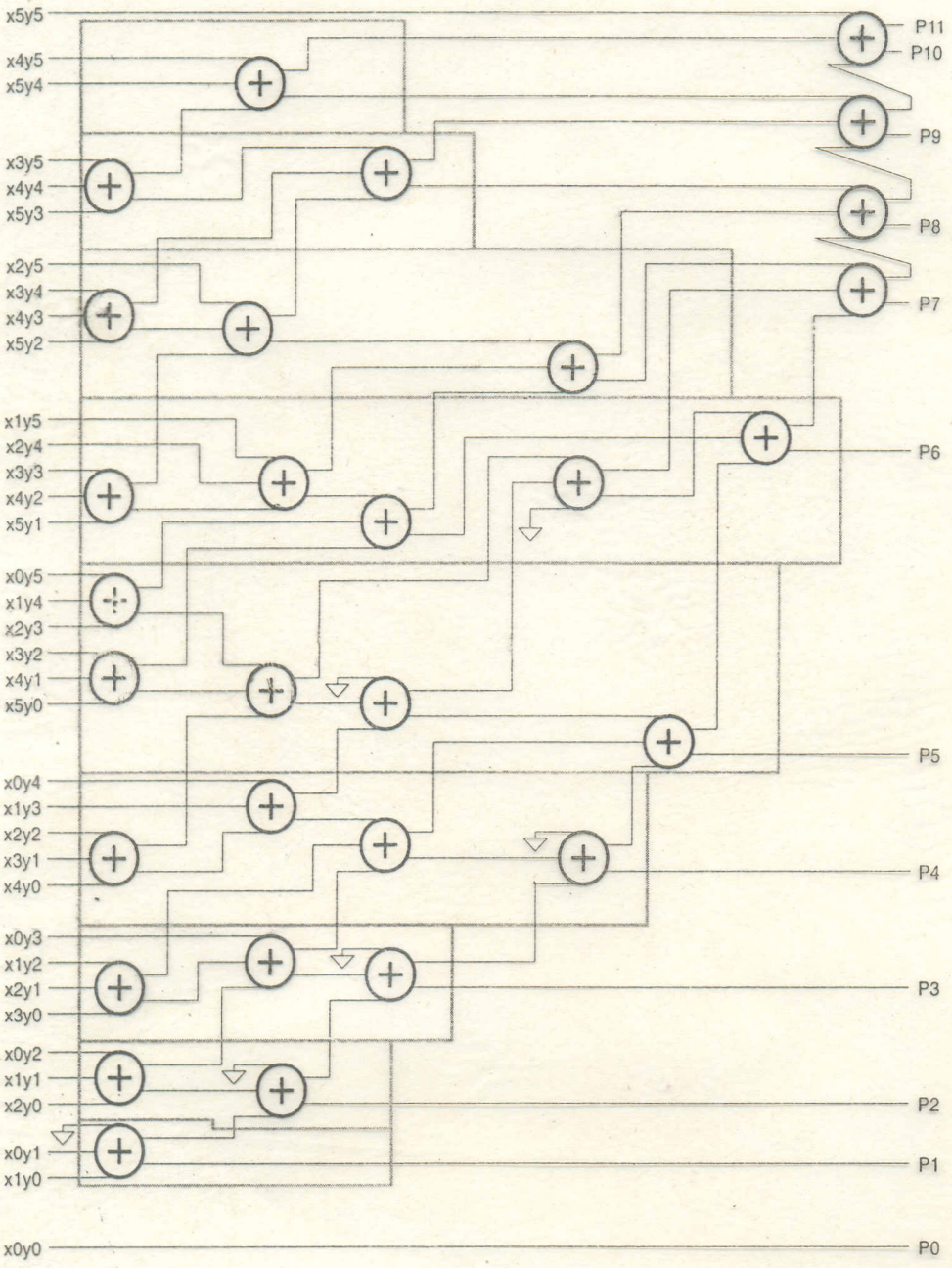


FIGURE 8.40 Wallace adder tree (for 6 × 6 multiplier)

layout for a 54-by-54 bit multiplier using the compressor shown in Fig. 8.41 may be found in Goto et al.[16]

8.2.7.4 *Serial Multiplication*

Multiplication may be performed serially. The simplest form of serial multiplier, shown in Fig. 8.42, uses the successive addition algorithm and is implemented using a full adder, a logical AND circuit, a delay element (i.e., either static or dynamic flip-flop), and a serial-to-parallel register.

Booth Multiplier examines 3 bits of multiplicand at a time to determine whether to add zero, 1*, -1*, 2*, -2* of that rank of multiplicand.

Booth-recoding table:

x_{i-1}	x_i	x_{i+1}	Operation	NEG	ZERO	TWO
0	0	0	add 0	1	1	0
0	0	1	add 2	0	0	1
0	1	0	sub 1	1	0	0
0	1	1	add 1	0	0	0
1	0	0	sub 1	1	0	0
1	0	1	add 1	0	0	0
1	1	0	sub 2	1	0	1
1	1	1	add 0	0	1	0

NOTE: ZERO : zeroes the operand.
NEG : inverts operand
TWO : multiplies value by 2 (left shift)

Booth Multiplier examines 3 bits of multiplicand at a time to determine whether to add zero, 1*, -1*, 2*, -2* of that rank of multiplicand.

Booth-recoding table:

x_{i-1}	x_i	$x_{(i+1)}$	Operation	NEG	ZERO	TWO
0	0	0	add 0	1	1	0
0	0	1	add 2	0	0	1
0	1	0	sub 1	1	0	0
0	1	1	add 1	0	0	0
1	0	0	sub 1	1	0	0
1	0	1	add 1	0	0	0
1	1	0	sub 2	1	0	1
1	1	1	add 0	0	1	0

NOTE: ZERO : zeroes the operand.
NEG : inverts operand
TWO : multiplies value by 2 (left shift)

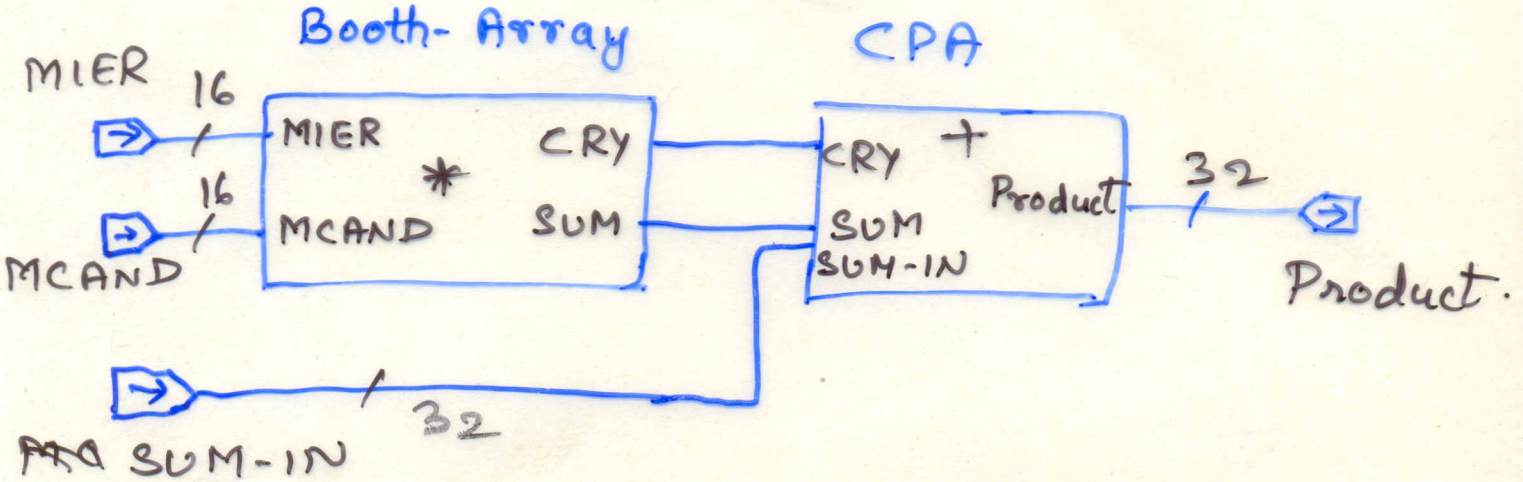


fig: 16x16 booth recoded multiplier (Schematic)

→ Booth-array accepts two 16-bit I/ps, & feeds CPA. (Carry Propagate Array).

→ The CPA also accepts 32-bit I/p to perform multiple accumulates.

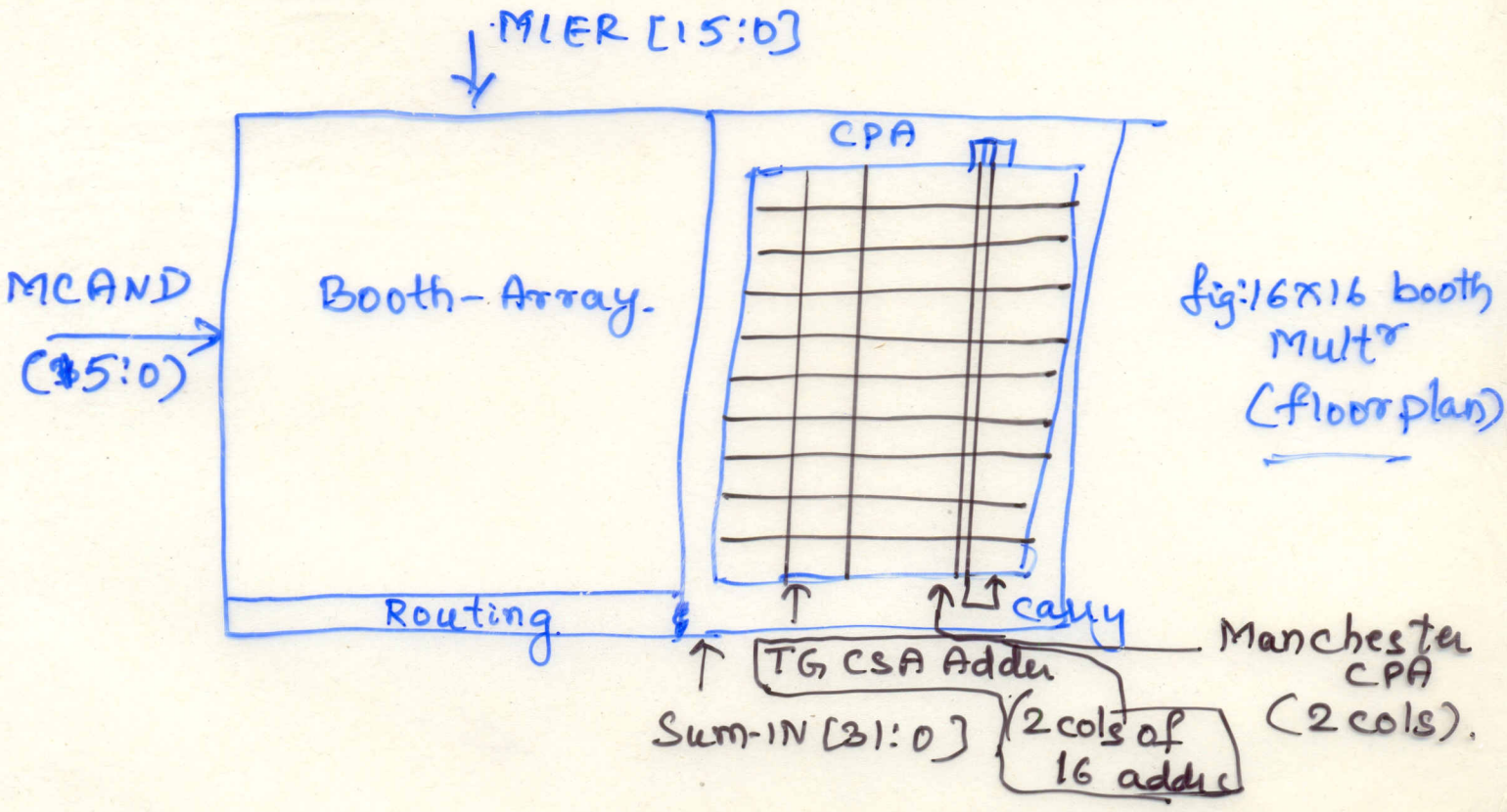


fig: 16x16 booth Multr (floorplan)