shiva sai

Unit – III

Mos layers

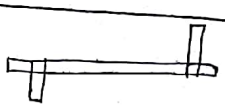
Mos circuits are formed on four basic layers

n – diffusion

P – diffusion

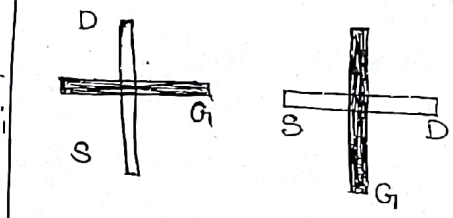
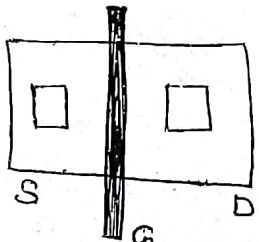
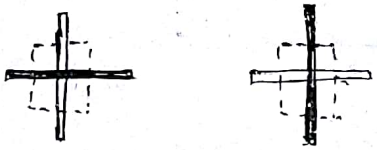
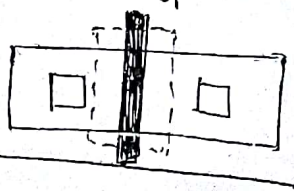
Polysilicon

metal.

Color	Stick Encoding	Layers
GREEN		n-diffusion
RED		Polysilicon
BLUE		Metal 1
BLACK		Contact cut

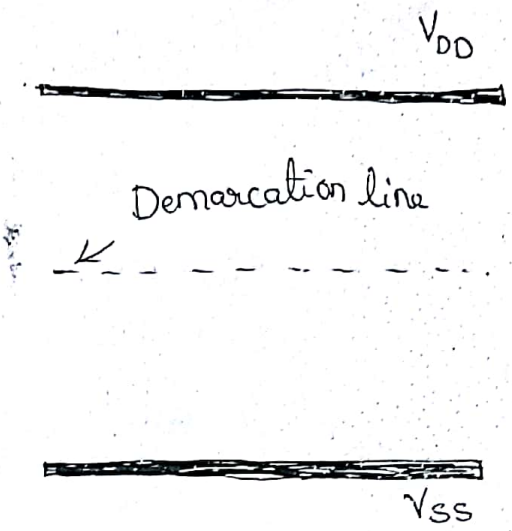
* Stick diagram:

The diagrams which are used to convey the information through the use of a color code is nothing but stick diagram.

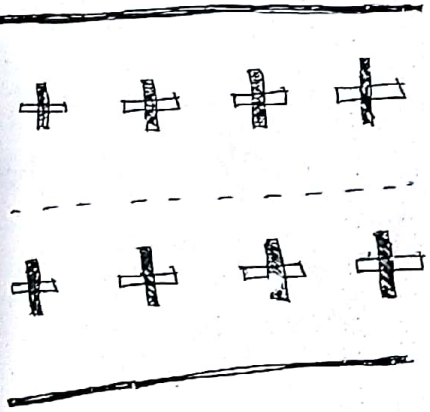
FEATURE	FEATURE (STICK) (MONOCHROME)	FEATURE (SYMBOL) (MONOCHROME)
n-type enhancement mode transistor		
n-type depletion mode transistor		

CMOS Design style

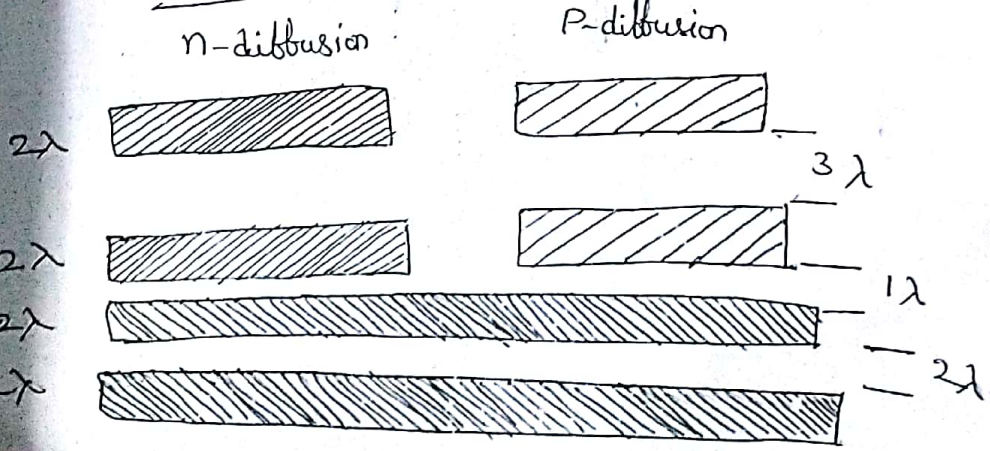
a) Rails and demarcation line



b) naud P transistors



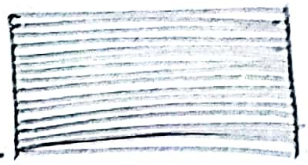
Lambda based Design Rules for Wires



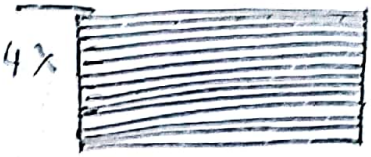
Metal 1

3λ

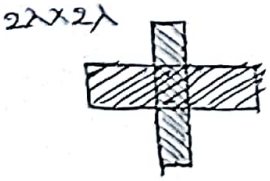
Metal-2



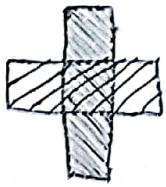
4λ.



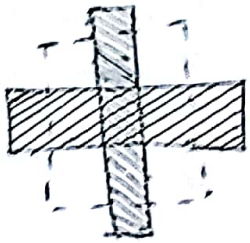
Minimum Size transistor design rules



nMOS (enhancement)



PMOS (enhancement)



nMOS (depletion

Polysilicon

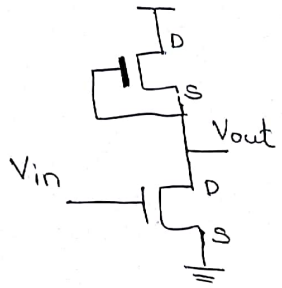
n diffusion

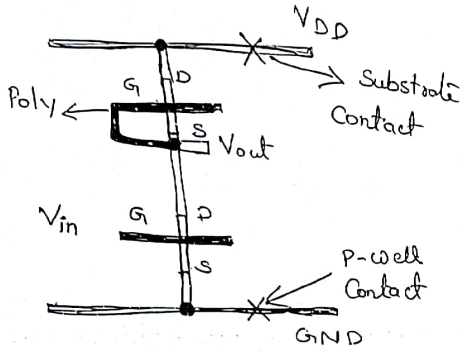
P diffusion

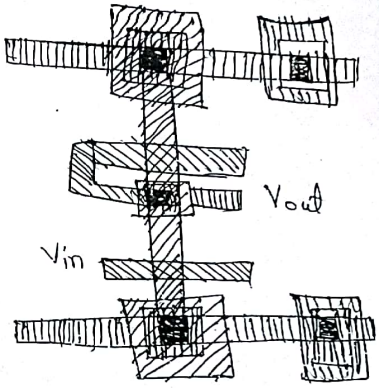
channel

Polysilicon overthing

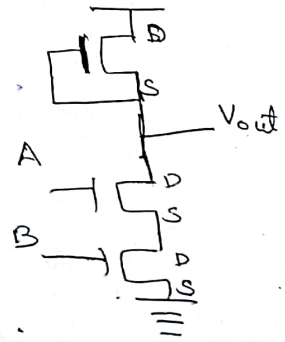
Pseudo NMOS Inverter

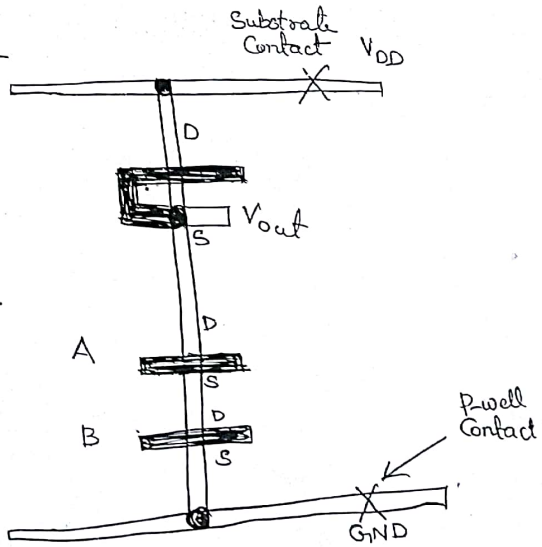


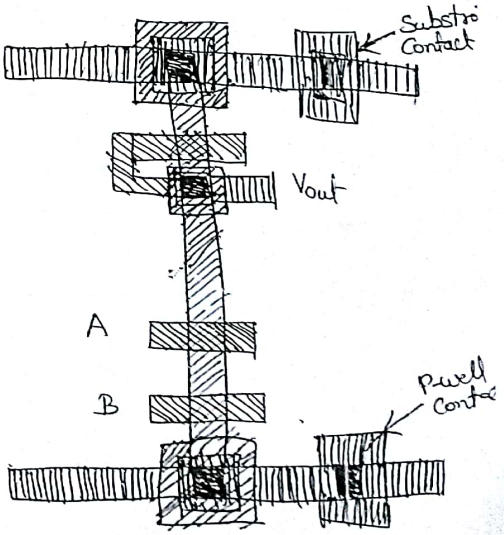




Pseudo NMOS NAND





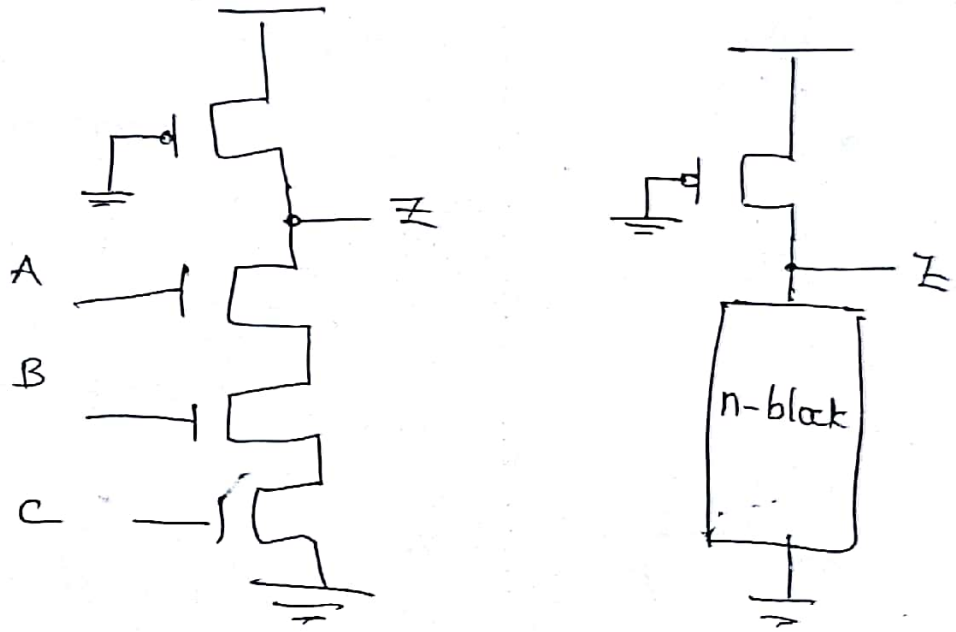


Other forms of CMOS

1) Pseudo - nMOS logic.

with three input nand

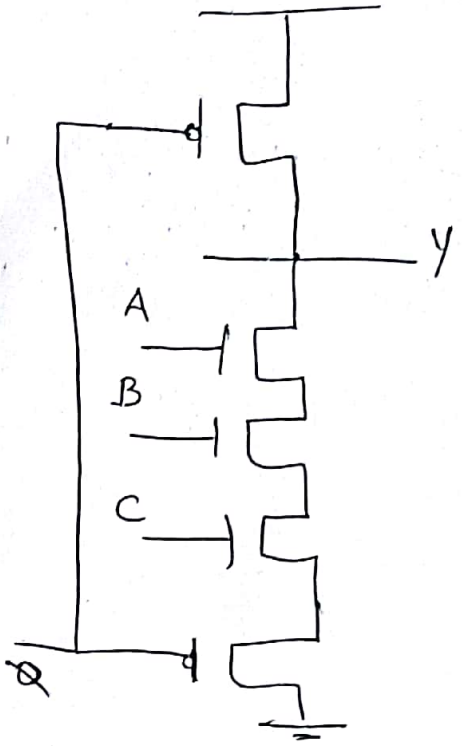
like depletion mode pull up transistor with a P-transistor with gate connected to Vss



For this ty of Pseudo - nMOS logic the Z P.u/Z P. d ratio is 3/1.

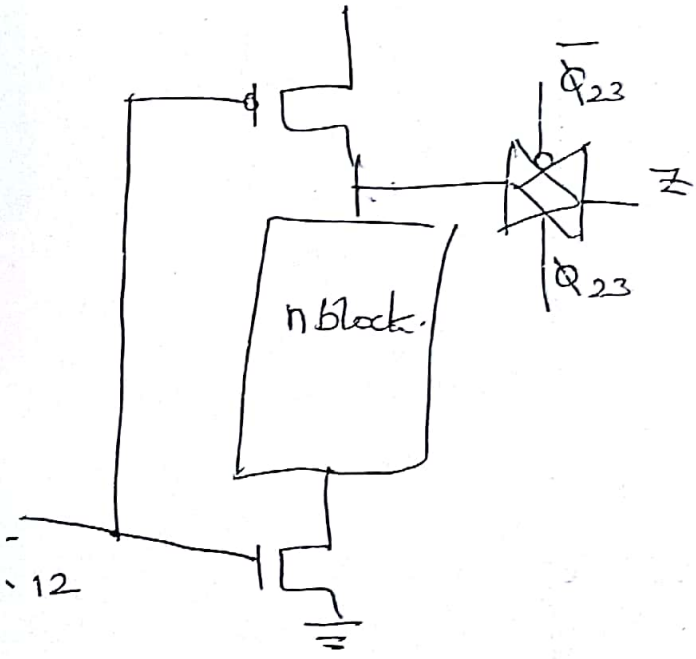
2) Dynamic

1) Charge sharing may be a problem
2) delays more.

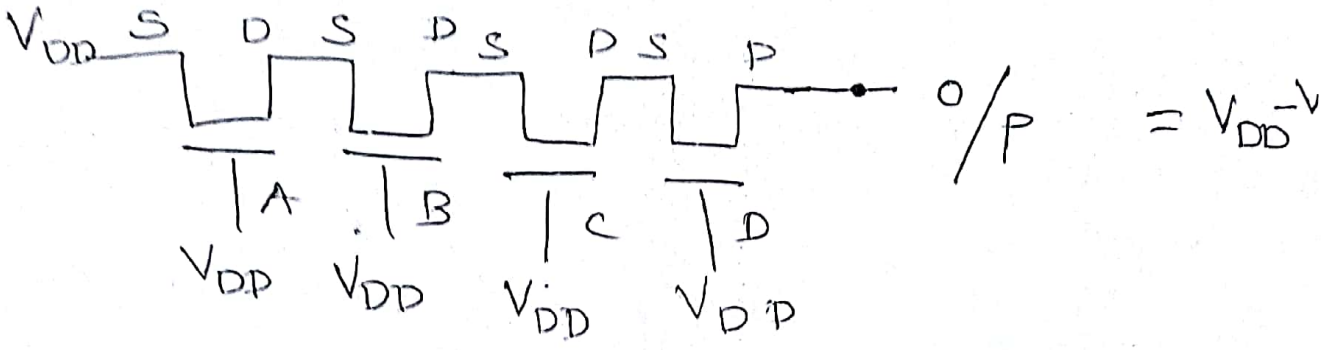


Single phase structures cannot be cascaded.

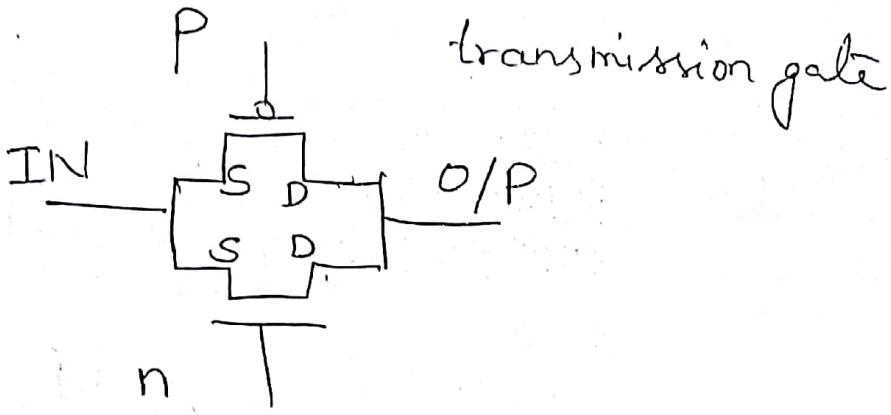
Better to use four phase clocking



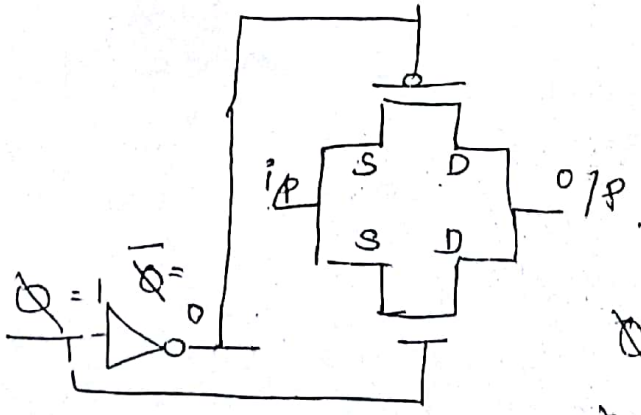
Pass transistor



transmission logic



logic



Q = 0,

Q = 1, $\bar{Q}$ = 0 O/P = i/p.

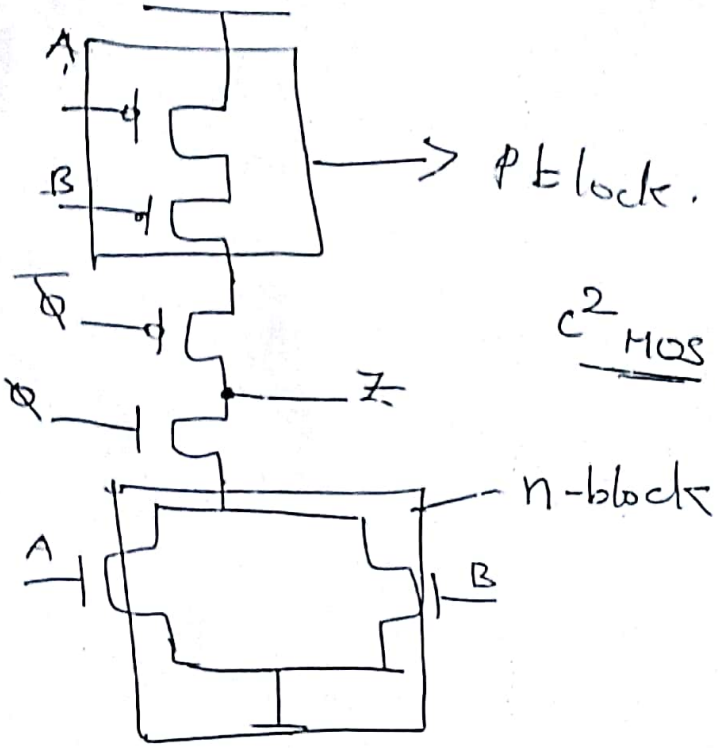
Q = 0, $\bar{Q}$ = 1 O/P = Z.

Both will be off.

CMOS domino

Use only single phase

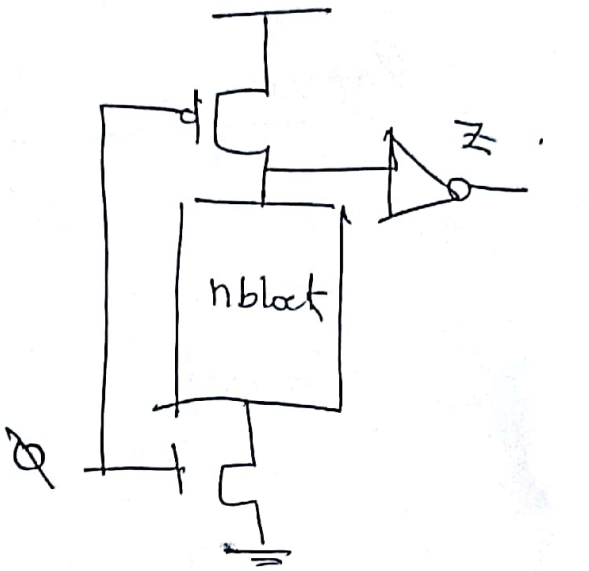
C^2MOS



→ P block.

C^2 MOS.

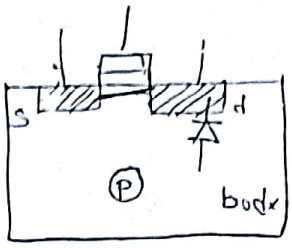
CMOS domino



1) Parasitics are small
2) Small area
3) Charge distribution may be prob
4) free of gliches.

Channel Length Modulation

I_{ds} is independent of V_{ds} for a transistor is in Saturation. The reverse biased p-n junction between the drain and body forms a depletion region with a width L_d that increases with V_{ds}.



Channel length because of depletion region $L_{eff} = L - L_d$

Assume Source Voltage is close to body Voltage so, $V_{db} \sim V_{ds}$.

Hence increasing V_{ds} decreases the effective channel length. Shorter channel length results in higher Current. Thus I_{ds} increases with V_{ds} in Saturation. This can be modeled by multiplying by a factor of $(1+\lambda V_{ds})$

$$I_{ds} = \beta \frac{(V_{gs}-V_t)^2}{2}(1+\lambda V_{ds})$$

λ is channel length modulation.

Body effect

The potential difference between the Source and body V_{sb} affects the threshold Voltage.

The threshold Voltage can be modeled as

$$V_t = V_{to} + \gamma(\sqrt{\phi_s + V_{sb}} - \sqrt{\phi_s})$$

where V_{to} is the threshold Voltage when the Source is at the body potential, ϕ_s is the Surface potential at threshold.

γ is the body effect coefficient, the range is 0.4 to 1 $V^{1/2}$

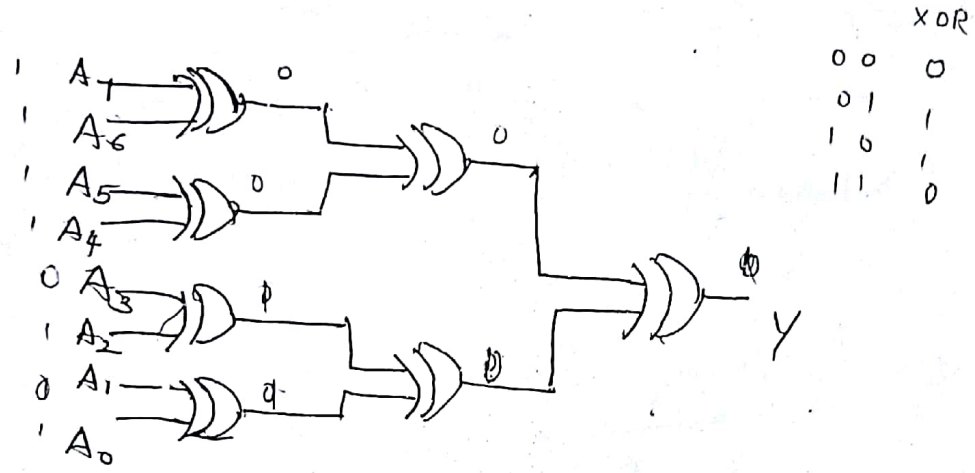
$$\phi_s = 2\phi_T \ln \frac{N_A}{n_i}$$

$$\gamma = \frac{t_{ox}}{\epsilon_{ox}} \sqrt{2q\epsilon_{Si} N_A} = \frac{\sqrt{2q\epsilon_{Si} N_A}}{C_{ox}}$$

Parity

A parity bit can be added to an N-bit word to indicate whether the number of 1's in the word is even or odd. In even parity, the extra bit is the XOR of the other N bits, which ensures the (N+1)-bit coded word has an even number of 1's.

$$A_n = \text{Parity} = A_n \oplus A_1 \oplus A_2 \oplus \cdots \oplus A_{-1}$$



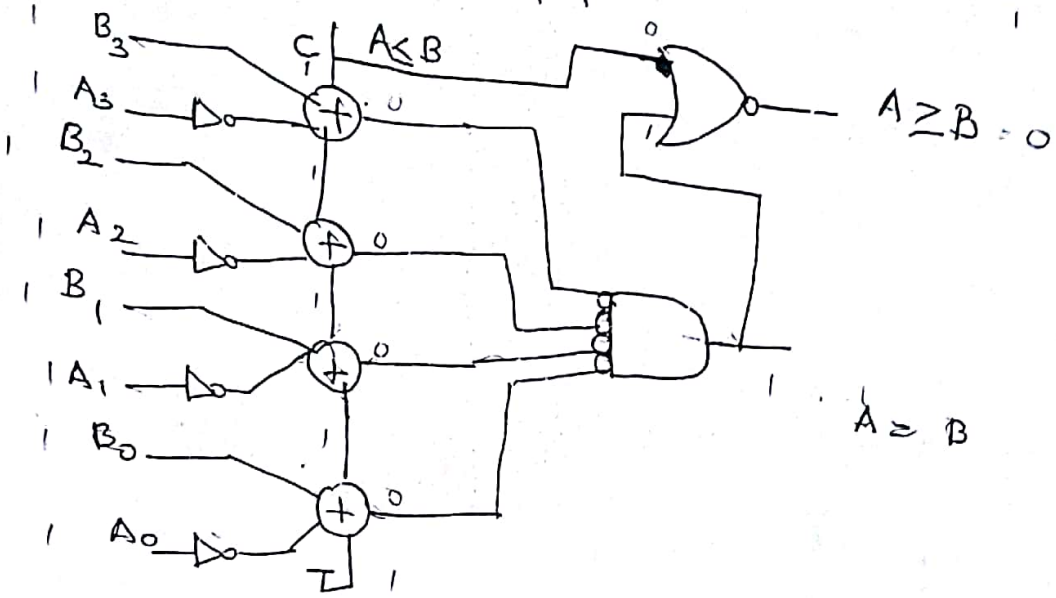
If output y is '0' then it says that number of 1's are even. If the output y is '1' then it says that number of 1's are odd.

Comparator

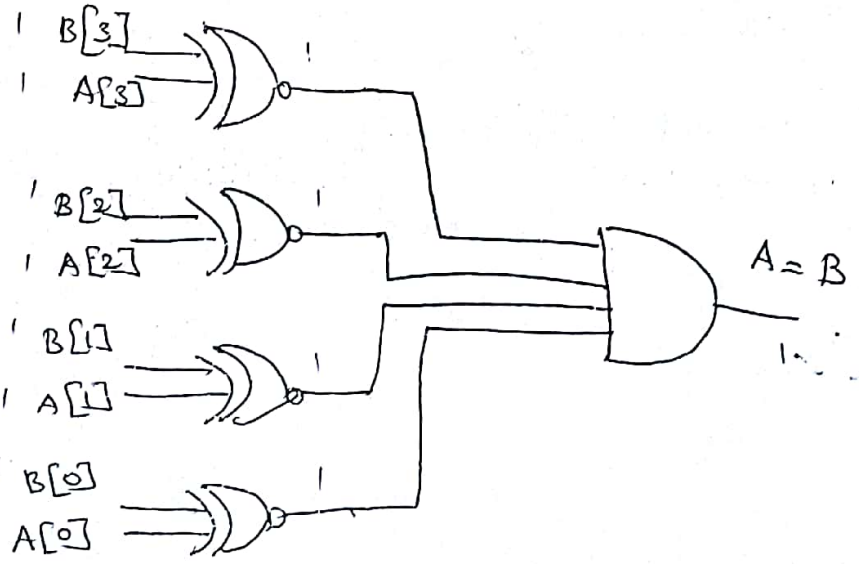
Magnitude Comparator

X NOR	XOR		OR	NOR
1	0	00	0	1
0	1	01	1	0
0	1	10	1	0
1	0	11	1	0

$$B - A = B + \bar{A} + 1$$



Equality Comparator



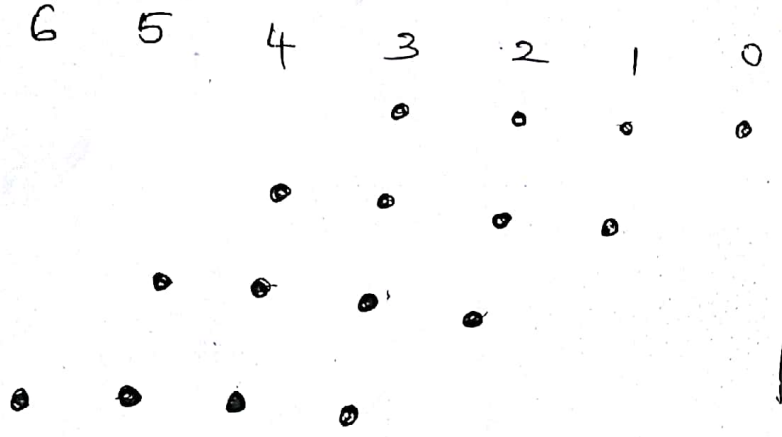
Tree mult

delay reduced.

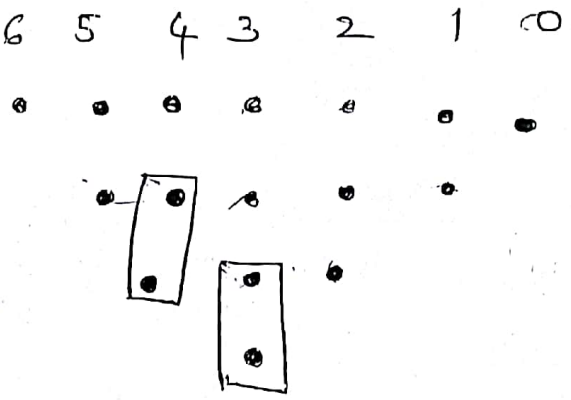
only six FA, six HA for carry save

here 3 FAs, 3 HA.

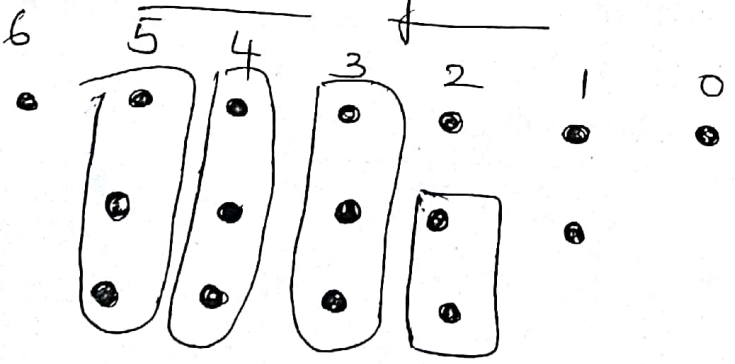
Partial products



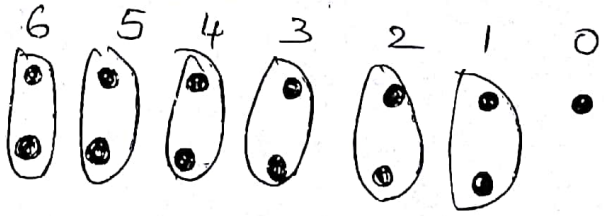
First Stage



Second stage



Final



$$t_p = O(\log_{3/2}(N))$$

Booth multiplier

→ to reduce partial products.

↳ Radix-2

```
5   0101 × Multiplicand
2   0010 × Multiplier
    ----
    0000
   0101
  0000
 0000
-------
0001010
```

I/P	O/P
00	0
01 (+ve)	−1
10	+1
11	0

```
00100
0+1 −1 0
01 −1 0
```

$3 \times 2 = 6$ 4−2   5−1 $3\times(4-2) = 6$

```
(0101) x = −x
0 1 −1 0
--------
000 0000
00 1011
0 0101
0000
--------
0 0 0 1 0 1 0
  8 4 2 1
  10.
```

```
0101
0−110
-------
 0000
0101
1011
0000
-------
0110100
```

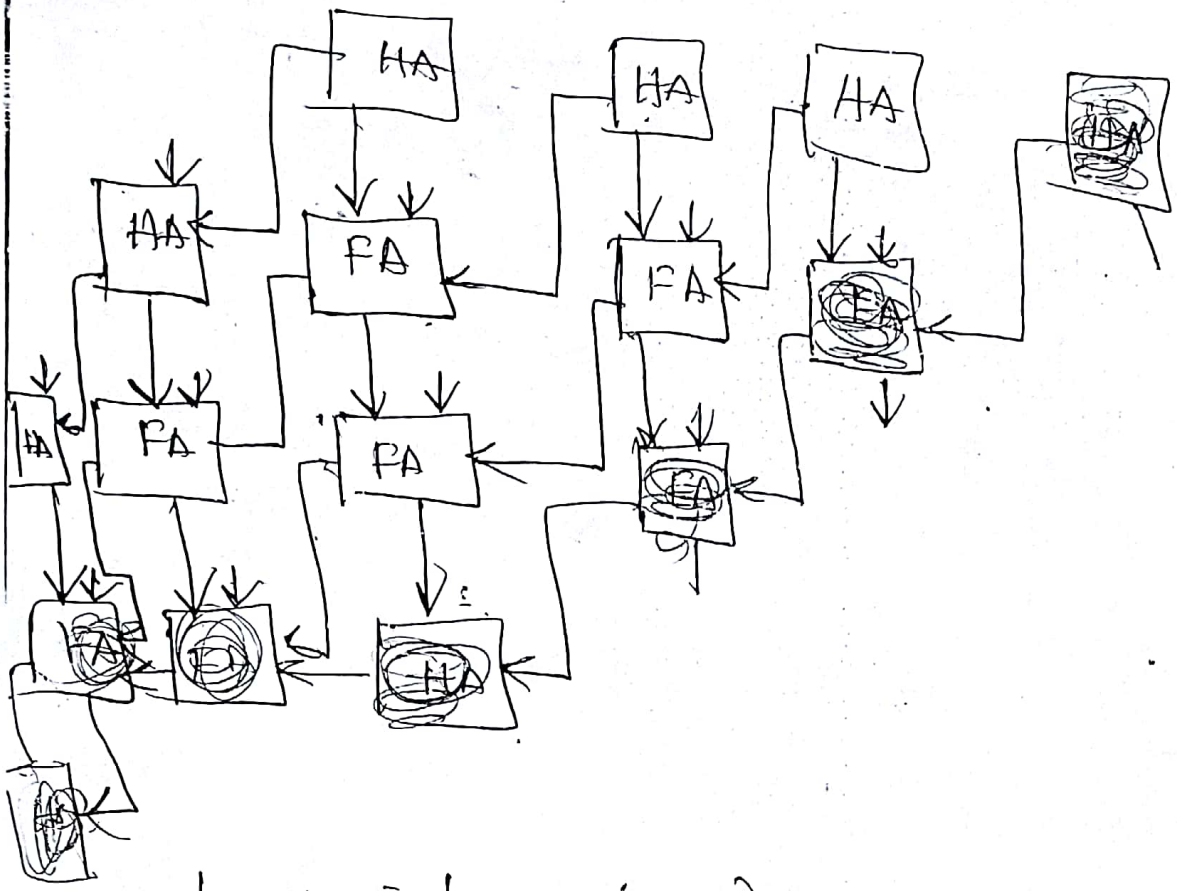
```
0
1010
   1
-----
0011

1010
   1
-----
1011
```

```
0000 0000
1 11011
0 0101
0000
--------
0001010
```

```
00010
11100
```

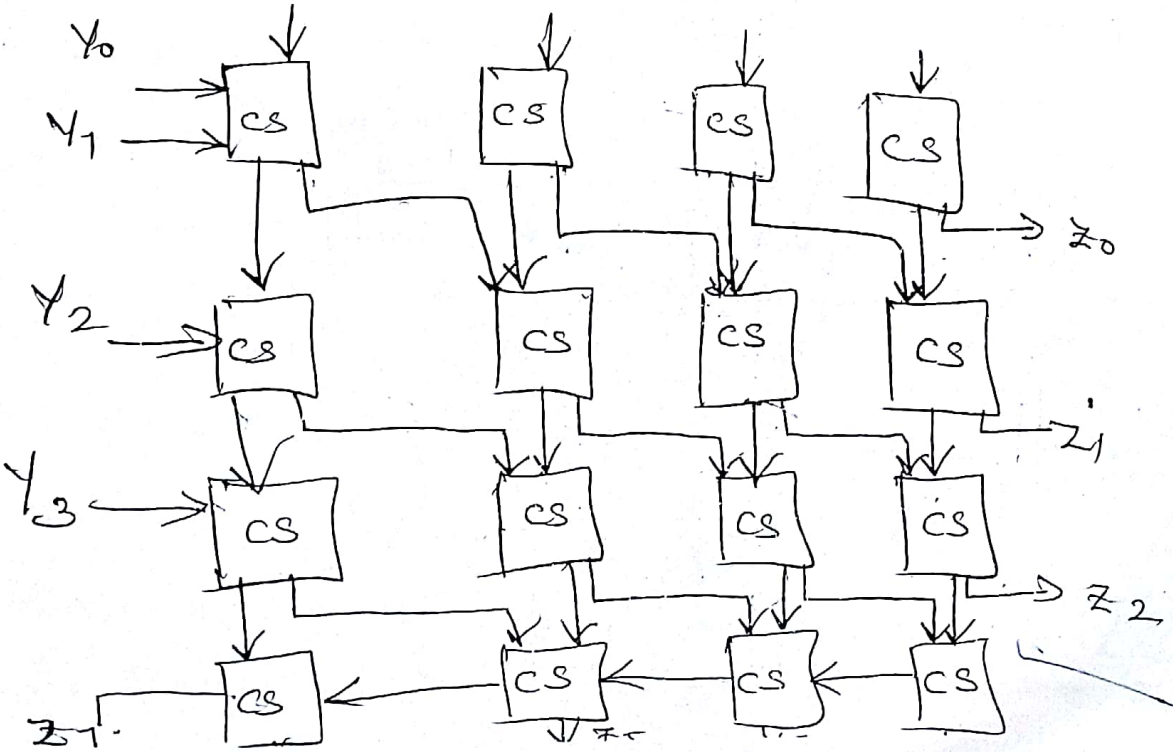
```
0000000
111011
00101
10000
--------
0001010
```

Carry Save multiplier

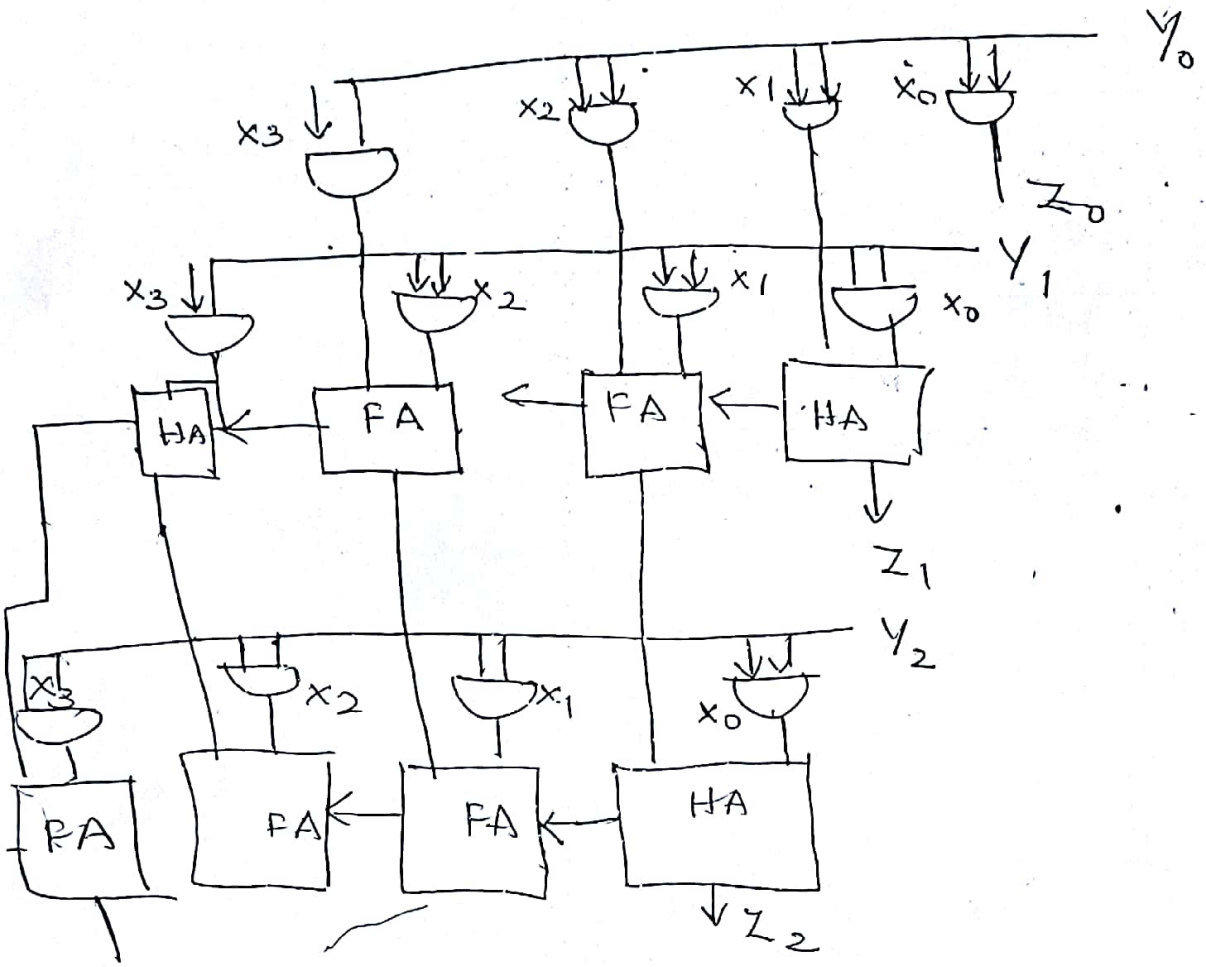


$$t_{mult} = t_{and} + (N-1)t_{carry} + t_{merge}.$$

Worst case critical path delay is shater.

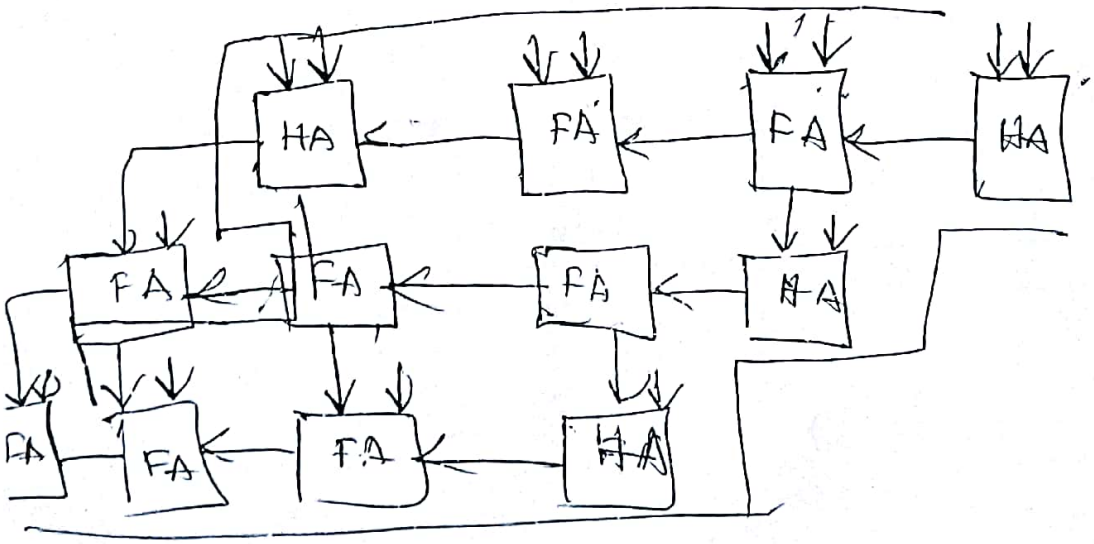


Array Multiplier



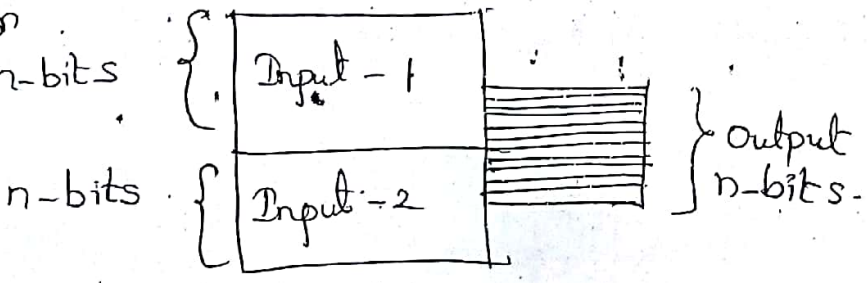
Z_6 Z_5 Z_4 Z_3

$t_{mul.} = [(M-1) + (N-1)]\, t_{carry} + (N-1)\, t_{sum} + t_{and}$



Shifters

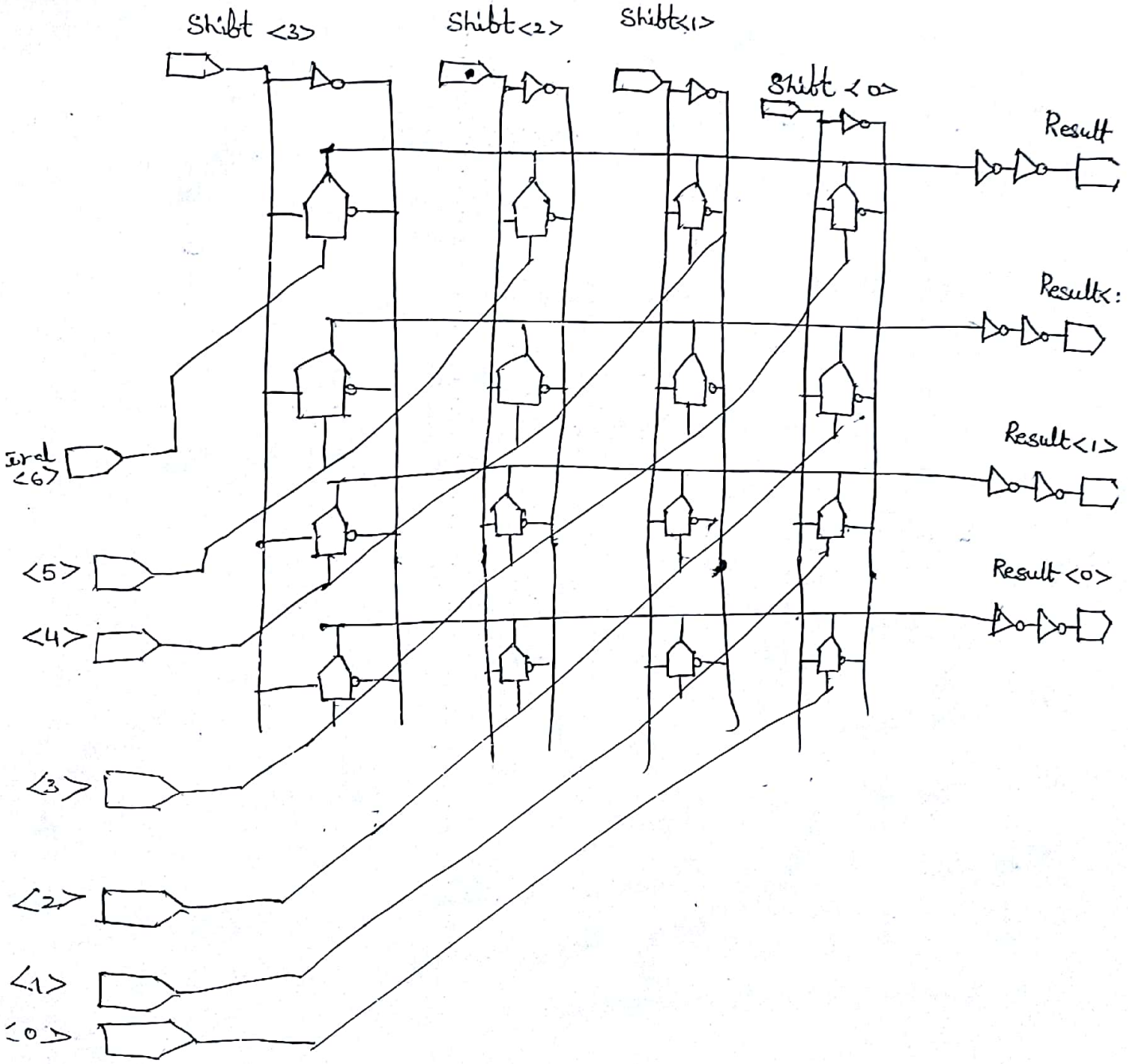
- A shifter is mainly used for arithmatic operation Shifting is equivalent to multiplication by powers of two. The simplest shifter is a shift register that shifts data by one position per clock cycle.
- Shifters are integral part of VLSI subsystems, and is used for arithmatic shifters, logical shifting rotation functions and floating point arithmatic
- A barrel shifter has n-data inputs, n-data output and a set of control inputs that specify how to shift the data between input and output
- Barrel shifter can perform n-bit shift in a single Combinational function and can rotate, extend sign as well. A basic architecture of barrel shifter is shown.



- A 4x4 barrel shifter using transmission gate is implemented as shown.
- The value to be shifted (literal <6:0>) and the shift amount (shift <3:0>) are the input to the shifter. The value of output (result <3:0>) for various shift

Values and literals are shown as

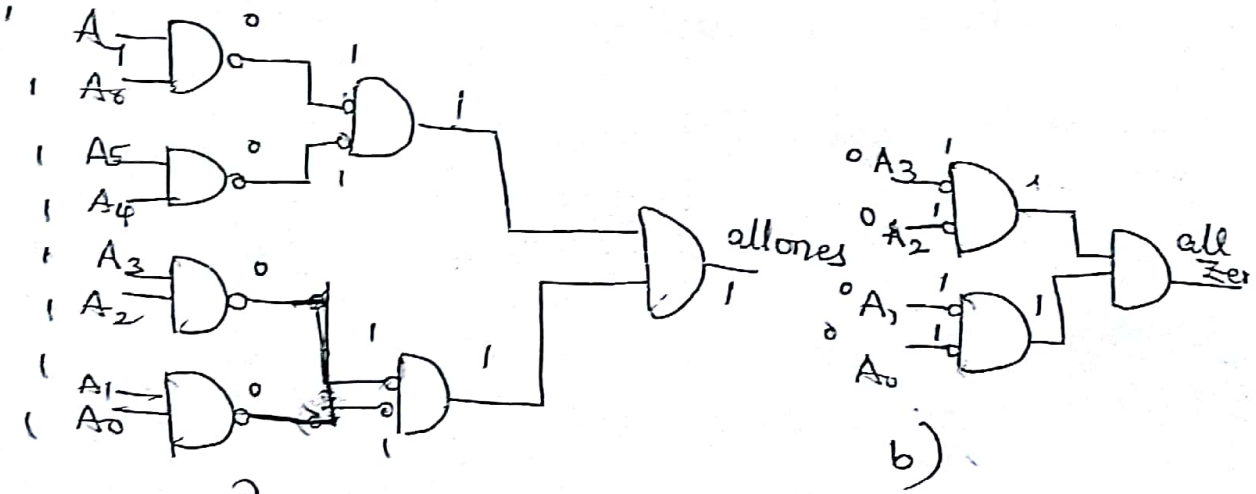
Shift	Result
0	literal <3:0>
1	literal <4:1>
2	literal <5:2>
3	literal <6:3>



Zero/One Detectors

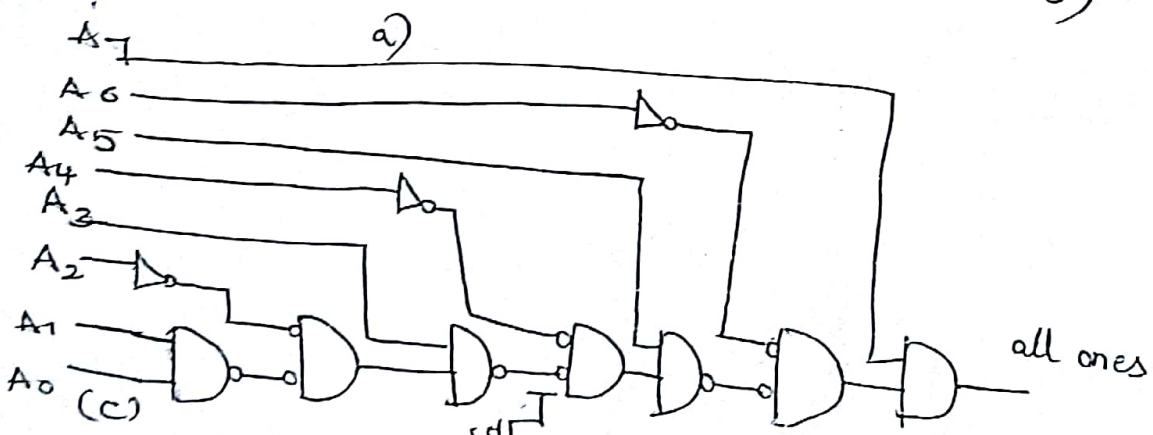
Detecting all 1's & 0's on wide N-bit words requires large fan-in AND & NOR gates. Here in this circuit alternate NAND and NOR gates have been used. The path has log N stages. The minimum logical effort is achieved with a tree alternating NAND gates and inverters and the pd logical effort is

$$G_{and}(N) = \left(\frac{4}{3}\right)^{\log_2 N} = N^{\log_2 4/3} = N^{0.4}$$

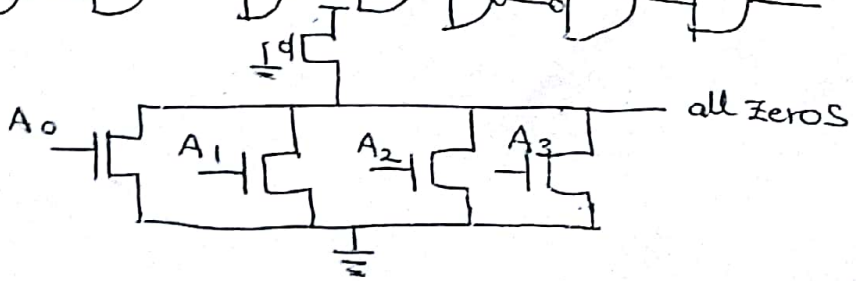


b)

a)



(c)



A rough estimate of the path delay driving a path electrical effort of H using static CMOS gates is

$$D = (\log_4 F)\, t_{FO4} = (\log_4 H + 0.415 \log_4 N)\, t_{FO4}.$$

where t_{FO4} is the fanout of 4 inverter delay.

If the word being checked has a natural skew in the arrival time of the outputs, the designer might consider mimicking the adder delay in the detector. It is the delay from the last changing output A_7 is a single delay.

Another fast detector uses a pseudo-nMOS or dynamic NOR structure to perform the "wired-OR" as shown. This works well for words up to about 16 bits. For larger words the gates can be split into 8–16 bit chunks to reduce the parasitic delay and avoid problems with subthreshold leakage.

The Carry-Lookahead Adder*

The Monolithic Lookahead Adder When designing even faster adders, it is essential to get around the rippling effect of the carry that is still present in one form or another in both the carry-bypass and carry-select adders. The *carry-lookahead* principle offers a possible way to do so [Weinberger56, MacSorley61]. As stated before, the following relation holds for each bit position in an N-bit adder:

$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1} \tag{11.15}$$

The dependency between $C_{o,k}$ and $C_{o,k-1}$ can be eliminated by expanding $C_{o,k-1}$:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1} C_{o,k-2}) \tag{11.16}$$

In a fully expanded form,

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\ldots + P_1(G_0 + P_0 C_{i,0}))) \tag{11.17}$$

with $C_{i,0}$ typically equal to 0.

This expanded relationship can be used to implement an N-bit adder. For every bit, the carry and sum outputs are independent of the previous bits. The ripple effect has thus been effectively eliminated, and the addition time should be independent of the number of bits. A block diagram of the overall composition of a carry-lookahead adder is shown in Figure 11-20.

Such a high-level model contains some hidden dependencies. When we study the detailed schematics of the adder, it becomes obvious that the constant addition time is wishful thinking and that the real delay is at least increasing linearly with the number of bits. This is illustrated in Figure 11-21, where a possible circuit implementation of Eq. (11.17) is shown for $N = 4$. Note that the circuit exploits the self-duality and the recursivity of the carry-lookahead equation to build a mirror structure, similar in style to the single-bit full adder of Figure 11-6.[4] The large fan-in of the circuit makes it prohibitively slow for larger values of N. Implementing it with simpler gates requires multiple logic levels. In both cases, the propagation delay increases. Further-

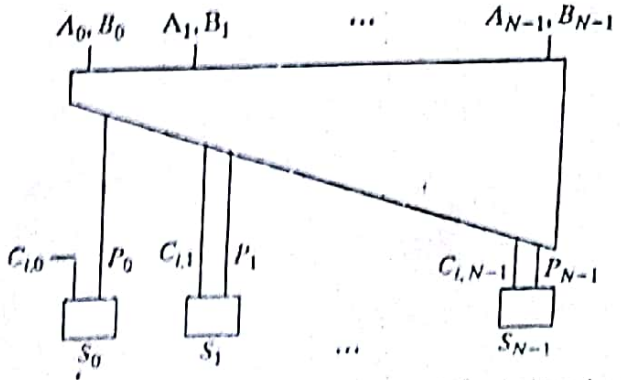


Figure 11-20 Conceptual diagram of a carry-lookahead adder.

[4] Similar to the mirror adder, this circuit requires that the *Propagate* signal be defined as $P = A + B$.

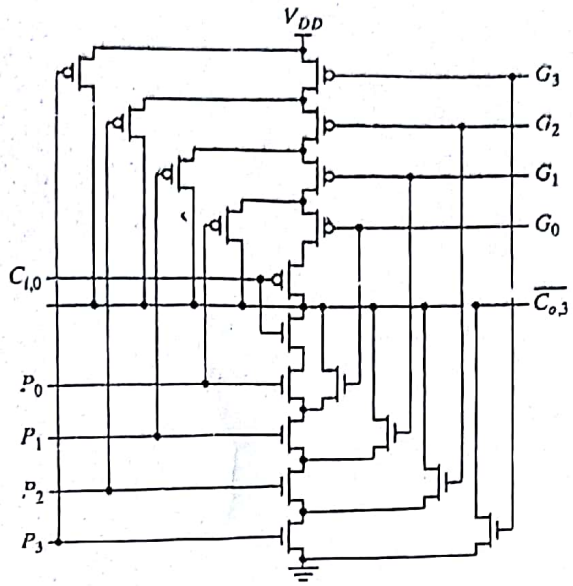


Figure 11-21 Schematic diagram of mirror implementation of four-bit lookahead adder (from [Weste85]).

more, the fan-out on some of the signals tends to grow excessively, slowing down the adder even more. For instance, the signals G_0 and P_0 appear in the expression for every one of the subsequent bits. Hence, the capacitance on these lines is substantial. Finally, the area of the implementation grows progressively with N. Therefore, the lookahead structure suggested by Eq. (11.16) is only useful for small values of N (≤ 4).

The Logarithmic Lookahead Adder—Concept For a carry-lookahead group of N bits, the transistor implementation has $N+1$ parallel branches with up to $N+1$ transistors in the stack. Since wide gates and large stacks display poor performance, the carry-lookahead computation has to be limited to up to two or four bits in practice. In order to build very fast adders, it is necessary to organize carry propagation and generation into recursive trees. A more effective implementation is obtained by hierarchically decomposing the carry propagation into subgroups of N bits:

$$
\begin{aligned}
C_{o,0} &= G_0 + P_0 C_{i,0} \\
C_{o,1} &= G_1 + P_1 G_0 + P_1 P_0 C_{i,0} = (G_1 + P_1 G_0) + (P_1 P_0) C_{i,0} = G_{1:0} + P_{1:0} C_{i,0} \\
C_{o,2} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0} = G_2 + P_2 C_{o,1} \\
C_{o,3} &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{i,0} \\
&= (G_3 + P_3 G_2) + (P_3 P_2) C_{o,1} = G_{3:2} + P_{3:2} C_{o,1}
\end{aligned}
\qquad (11.18)
$$

$C_{0,0} = G_0 + P_0 C_{i,0}$

$C_{0,1} = G_1 + P_1 (G_0 + P_0 C_{i,0}) = G_1 + P_1 G_0 + P_1 P_0 C_{i,0}$

$C_{0,2} = G_2 + P_2 (C_{0,1}) = G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_{i,0})$

$= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0}$

$C_{0,3} = G_3 + P_3 C_{0,2} = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0})$

$= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0$