Computational Elements in CMOS Technology

In this section, we look at Barrel shifter, Adder and simple Multiplier design.

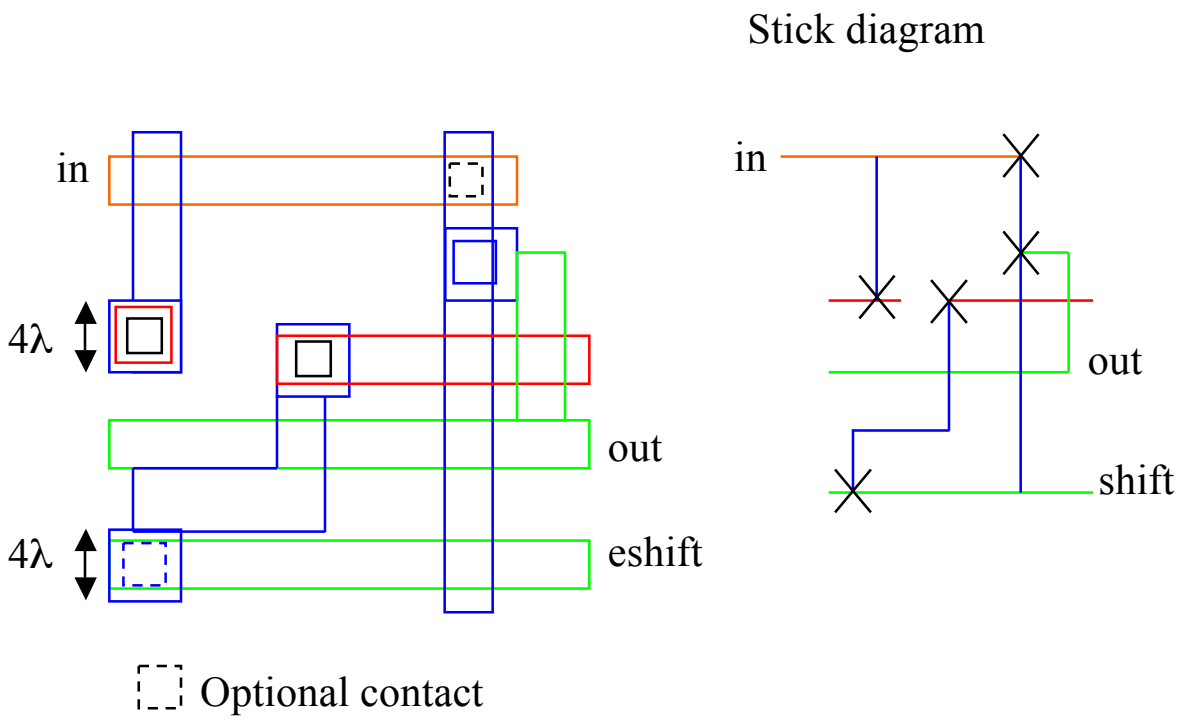
Barrel Shifter

- Uses transmission gates
- Barrel Shifter should perform any shift

For 4 bit case, (i) 4 input lines, (ii) 4 line for 4 output bits, (iii) Gate inputs for selecting any of 0, 1, 2, 3 bit shift operations. There are two arrangements as shown in the figures for this function implementation. The cross bar switch arrangement is not as efficient as needs 16 different control signals. On the other hand, in the barrel shifter arrangement, a single high poly line can achieve the shifting operation. For instance, 0 shift is achieved by Sϕ high so that inϕ → outϕ, in1 → out1, in2 → out2, in3 → out3. Setting S1 high does in1 → outϕ, in2 → out1, in3 → out2, inϕ → out3 which is a 1 bit right shift operation.

The regularity in the Barrel Shifter schematic is clear. The one transistor standard cell can now be configured easily

depending on constraint. One way to get the standard cell is as shown in the block.



Here shift and output signal are carried by diffusion. Input is carried by poly as shown. The area of this cell is quite small. The optional contact and metal on the left make the ladder pattern of shift control possible.

This implementation is not proper if the number of bits involved is large and drive needed is higher. Also, to provide drive, a buffer is needed even if there is no shifting.

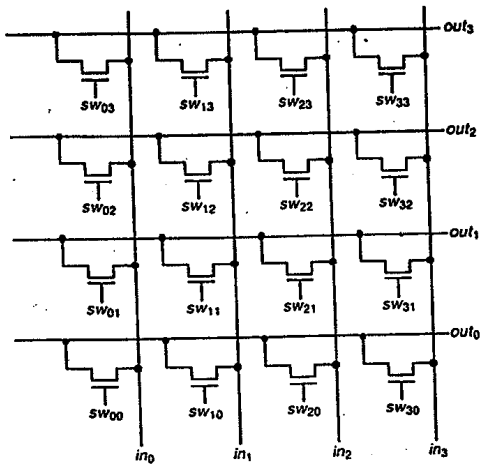


Figure 8–6 4×4 crossbar switch

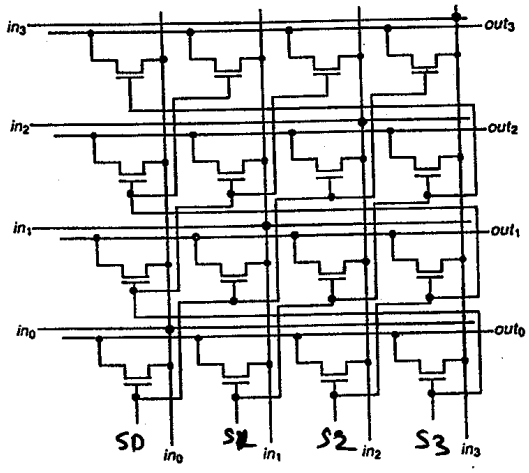


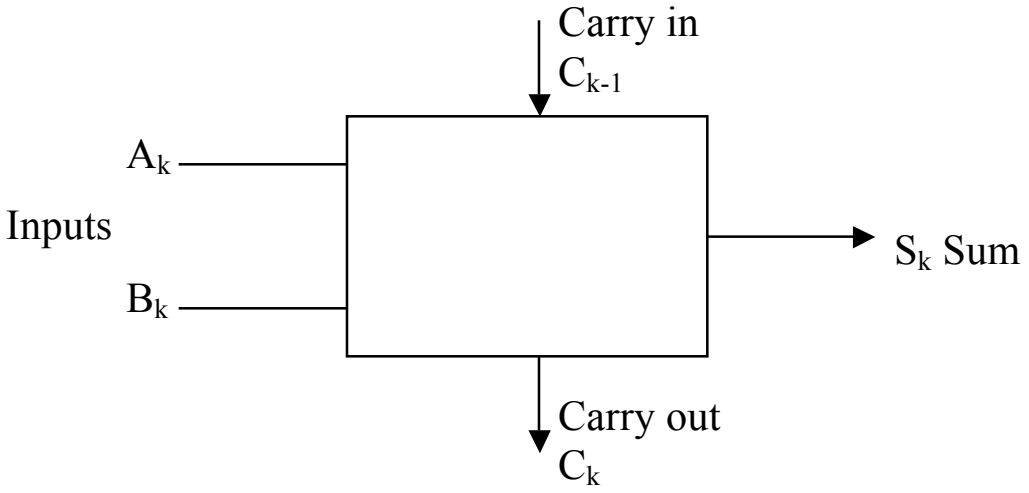
Figure 8–7 4×4 barrel shifter

Adder Design

An adder forms a vital part of ALU system. The truth table for a 1 bit full adder is as shown.

Inputs			Outputs	
A_K	B_K	Previous carry C_{k-1}	S_K	C_K
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

Schematically, the element is as shown.

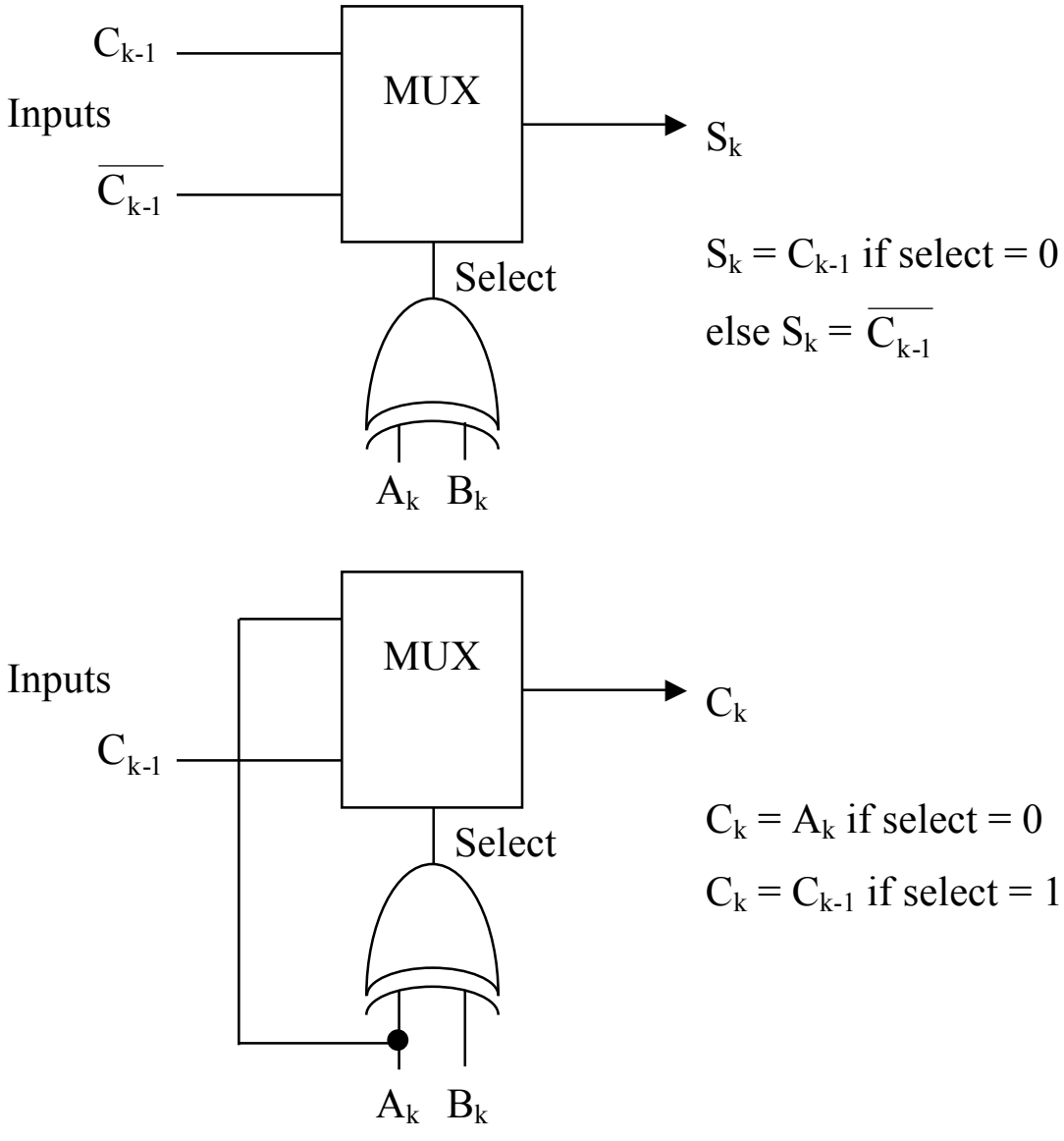


The functionality can be described as

If $A_k = B_k$ then $S_k = C_{k-1}$ Else $S_k = \overline{C_{k-1}}$.

If $A_k = B_k$ then $C_k = A_k = B_k$ Else $C_k = C_{k-1}$.

The simple implementation of first part is as shown.



Needs inverter XOR and Multiplexer cells. Can be cascaded for more bits

If each adder cell has a delay of τ, then this type of n-bit adder has a total summation time of $n\tau$ if τ is carry bit delay. For the adder cell implementation above, τ will be a sum of the XOR and 2 to 1 multiplexer delay. Hence more efficient cell implementations are sought.

A set of equivalent Boolean Expressions for S_k and C_k of one bit adder, which can be more efficiently implemented, are

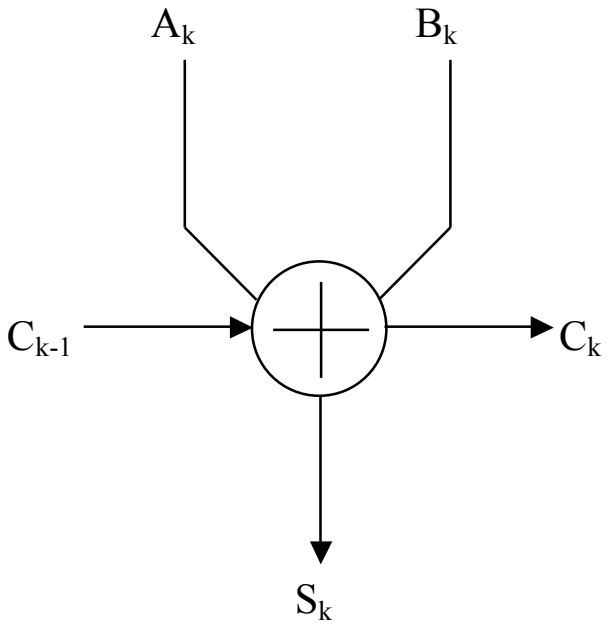
$$C_k = (A_k + B_k)C_{k-1} + A_kB_k$$

$$S_k = \overline{C_k}(A_k + B_k + C_{k-1}) + A_kB_kC_{k-1}$$

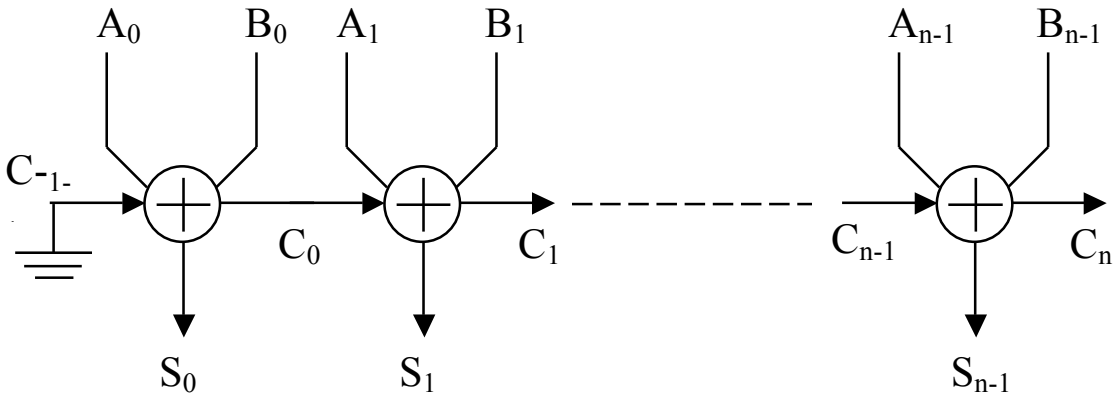
$$= A_k \oplus B_k \oplus C_{k-1}.$$

Obviously, it is easy to implement $\overline{C_k}$ and $\overline{S_k}$.

A more compact representation of one bit full adder cell is as shown.



Hence cascaded ripple carry implementation is as shown.
Carry bits propagate sideways. Sum bits are available once carry is known.



n bits

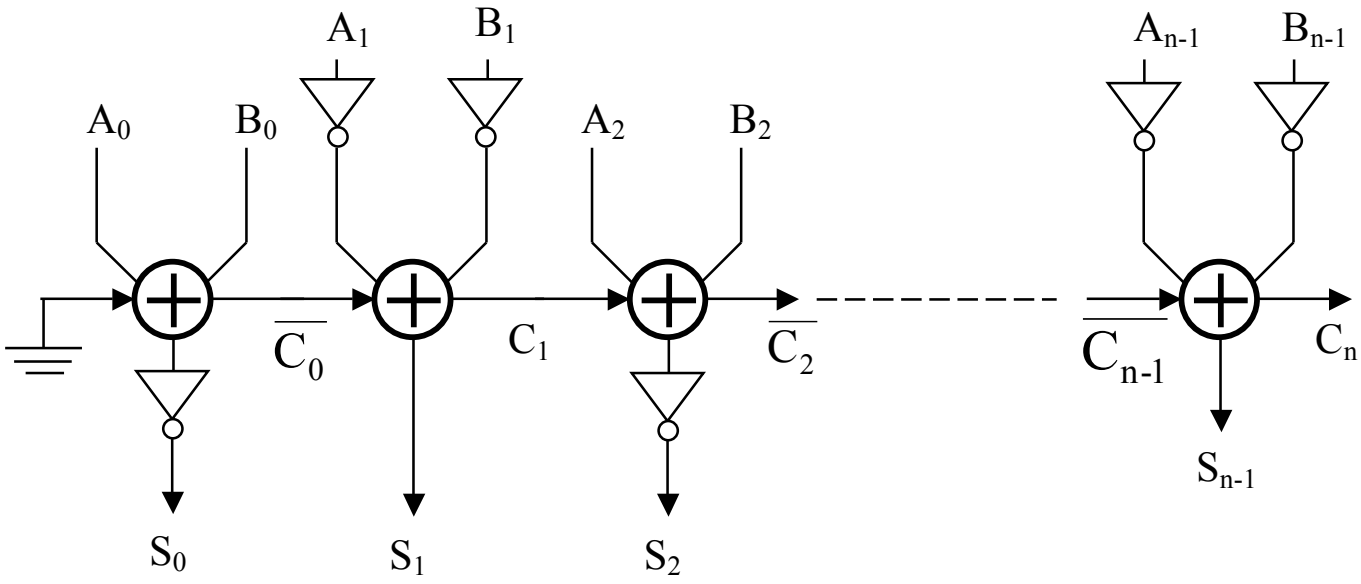
Since we generally do not need each C_k, an efficient cascading with either C_k or $\overline{C_k}$ is possible with a new cell with bold adder circle.

The bold adder cell below has $\overline{C_k}$ and $\overline{S_k}$ as its output when its input has actual bits to be added and carry as these are much faster to implement.

Since, for the first and every odd cell original bits and carry are the inputs, the output of these cell is actually $\overline{C_k}$ and $\overline{S_k}$.

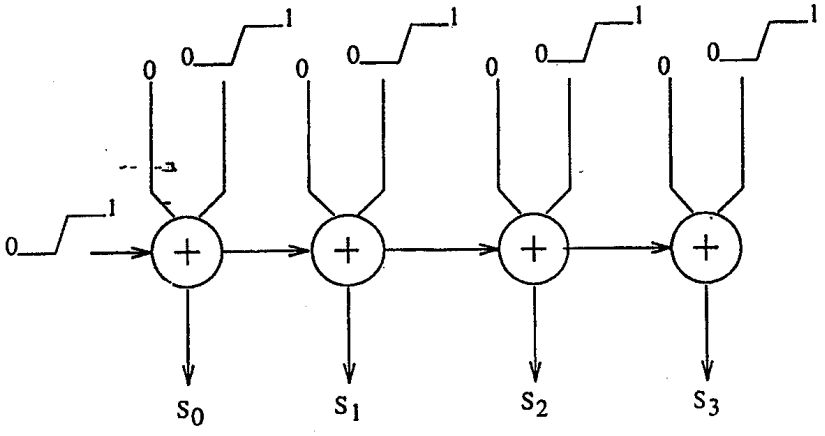
Hence the output of the cell needs to be inverted to get S_k. Thus this is done for every odd cell.

For an efficient implementation, carry propagation delay must be reduced. Hence every even cell must be able to directly use $\overline{C_k}$ as input and there must be no need to invert $\overline{C_k}$. As input bits to be added are available at the same time, they are inverted for all even cells. For second and other even cells, S_1, S_3, S_5, S_7, ….are directly available as all inputs are inverted. The output with such inverted bits as input will directly yield S_k and C_k (needed for odd cells) in place of $\overline{C_k}$ and $\overline{S_k}$. You can verify this Bullion equivalence.



This cascading balances delays and avoids using inverter for each carry. $\overline{C_0}$ delay allows time for inversion of A_1, B_1, … etc.

Such ripple carry adders may suffer from glitch phenomenon depicted in the following figure. Hence careful optimization is needed so that this does not pose a serious problem.



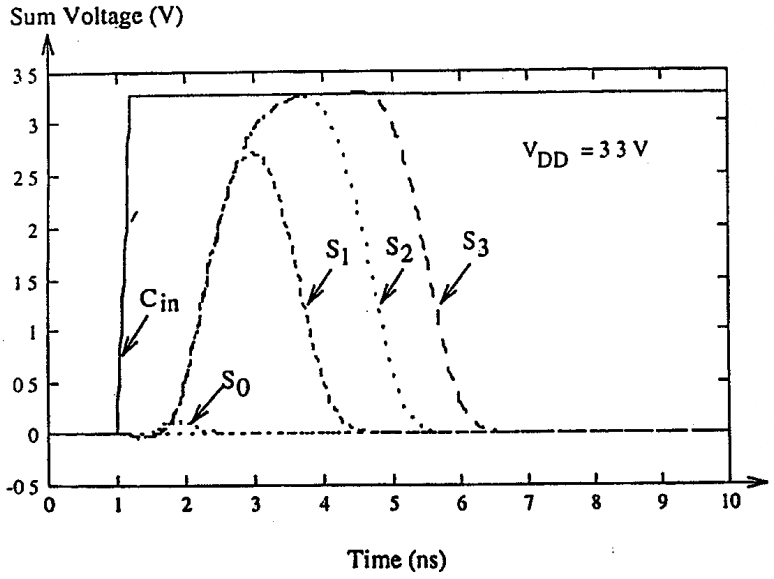


Figure 7.4 Sum voltage waveforms with glitching phenomenon.

Even more efficient implementations of adders are needed for the speed requirements today. Some of these implementations now will be discussed.

CLA Adder Implementation

In carry look ahead (CLA) adder, carry for all the bit positions is predicted in advance so that the sum of each bit position can be carried out independently. The adder actually uses intermediate or auxiliary inputs known as propagate (P) and generate (G) signals defined by

$$\overline{P_i} = \overline{A_i \oplus B_i} \qquad \oplus \text{ indicates XOR}$$

$$\overline{G_i} = \overline{A_i\, B_i}$$

the carry and sum is predicted for any bit position with the help of P_i's and G_i's. The carry expressions are:

$$C_1 = G_1 + P_1\, C_0$$

$$= \overline{\overline{G_1} \bullet (\overline{C_0} + \overline{P_1})}$$

$$C_2 = \overline{\overline{G_2} \bullet (\overline{P_2} + \overline{G_1} \bullet (\overline{C_0} + \overline{P_1}))} = \overline{\overline{G_2} \bullet (\overline{P_2} + \overline{C_1})}$$

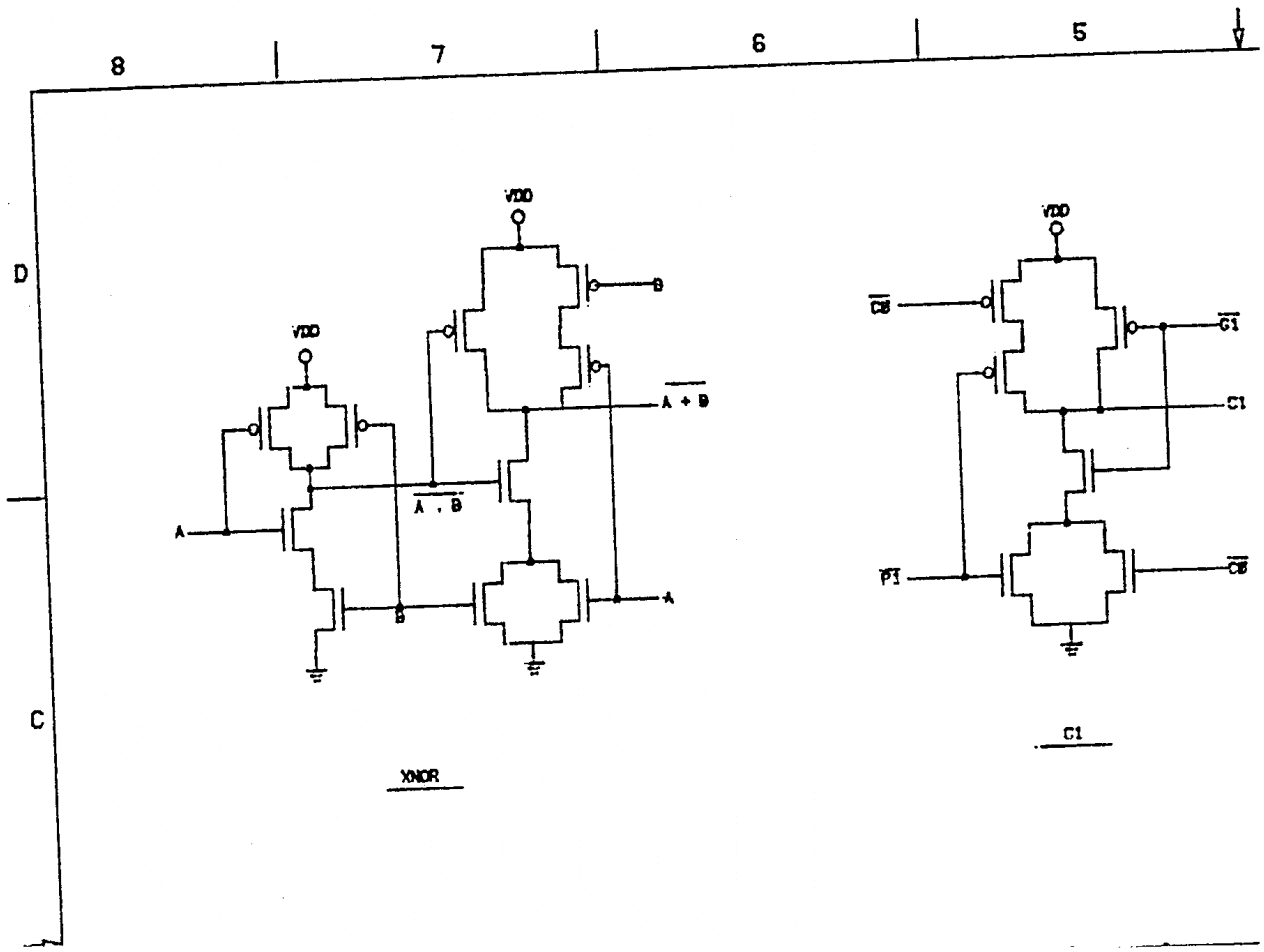
$$C_3 = \overline{\overline{G_3} \bullet (\overline{P_3} + \overline{G_2} \bullet (\overline{P_2} + \overline{G_1} \bullet (\overline{C_0} + \overline{P_1})))}$$

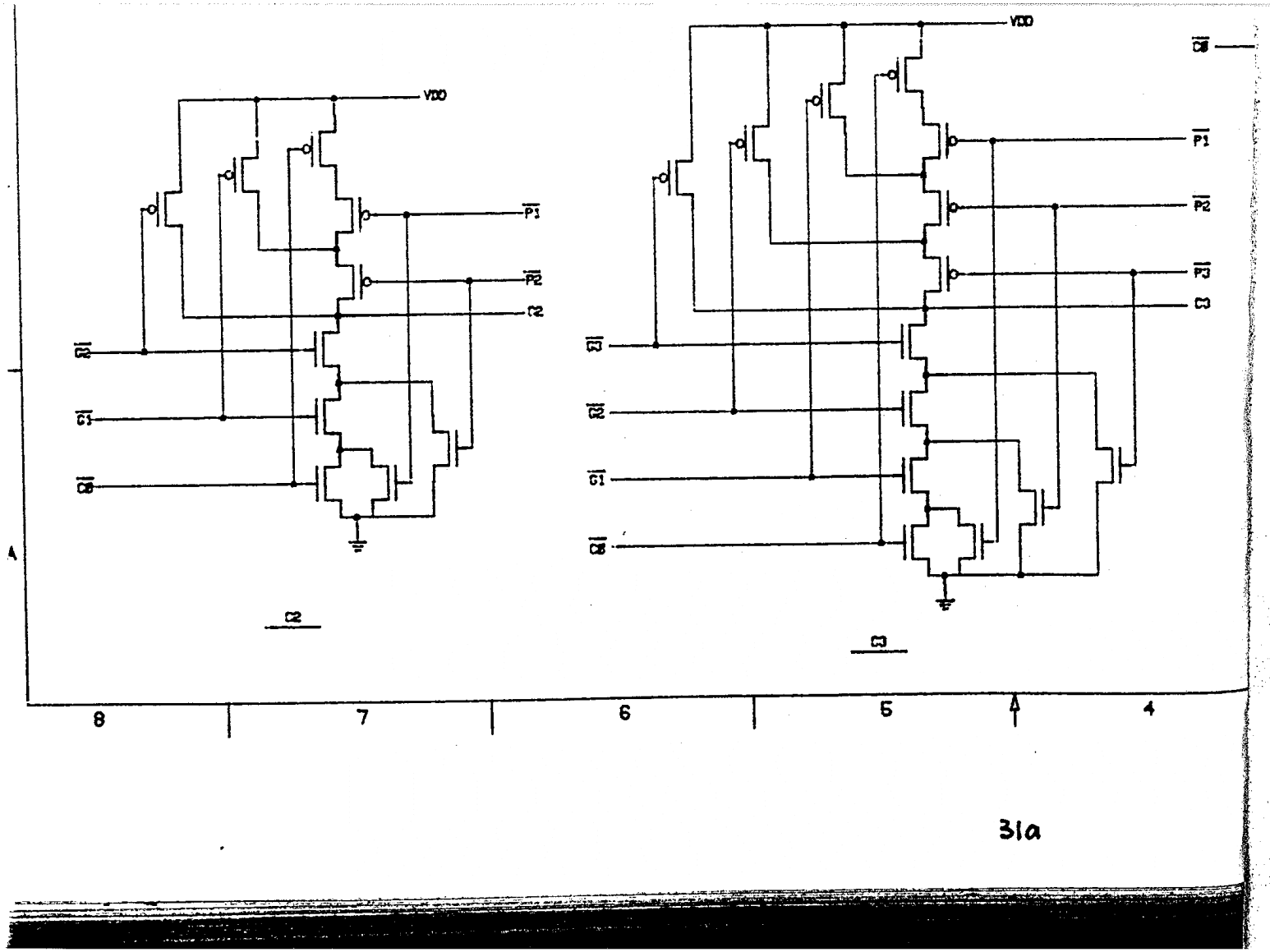
$$C_4 = \overline{\overline{G_4} \bullet (\overline{P_4} + \overline{G_3} \bullet (\overline{P_3} + \overline{G_2} \bullet (\overline{P_2} + \overline{G_1} \bullet (\overline{C_0} + \overline{P_1}))))}$$

and so on.

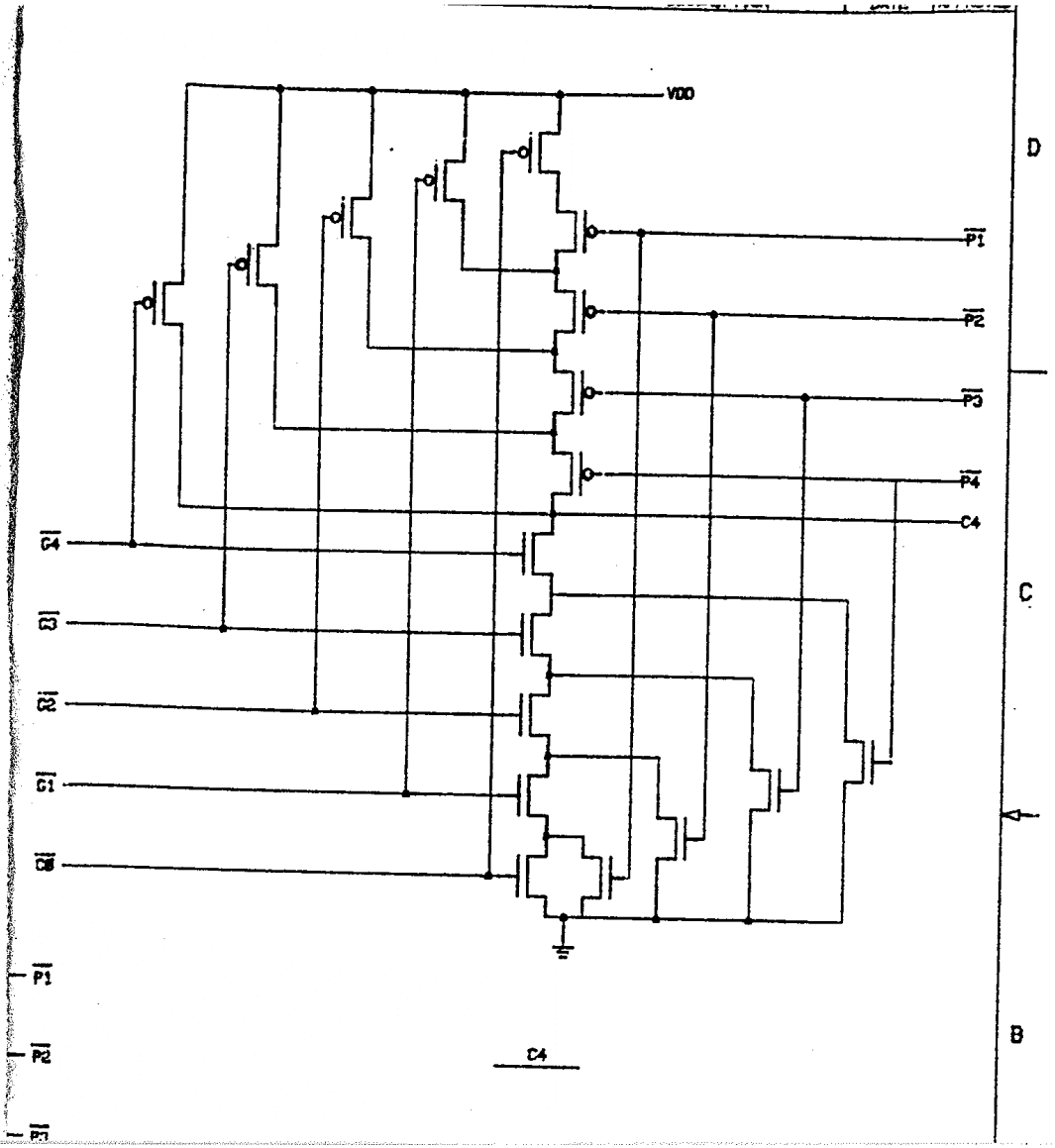
And S_i = Sum at bit position i = $\overline{\overline{P_i} \oplus C_{i-1}}$

As is clear to predict carry for more than 4 bit positions needs a very large overhead and 5 series connected devices. Hence, the CLA unit is limited to 4 bits and 2 or 3 of these are used to do the addition. Since C4 is predicted very fast, next carry bits are also available. The schematics for the carry positions and XNOR are shown next along with the layout of XNOR and CLA unit.


8
7
6
5
D
C
VDD
B
A + B
A . B
A
B
A
XNOR
VDD
CB
G1
C1
P1
CB
C1



31a


VDD
D
C
B
P1
P2
P3
P4
C4
C3
C2
C1
C0
C4

INTEGRATED CIRCUIT DESIGN

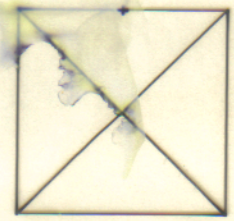
DATE : Sep 10, 1992
FILENAME : table_t.

SCALE : 0 microns/inch
USER : lws

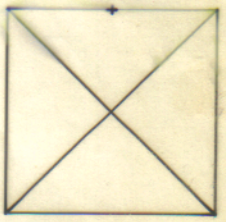
PSD_cover

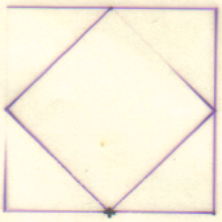
NSD_cover

PSD_contact

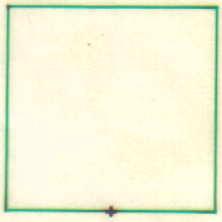


NSD_contact



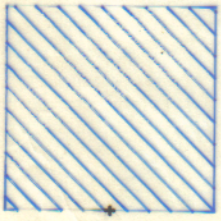


m1_m2_contact

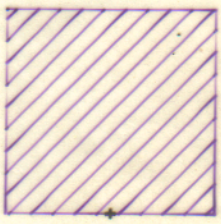


active_area

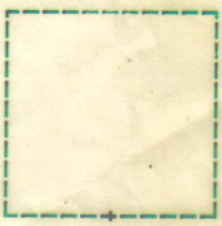
polysi



metal1



metal2

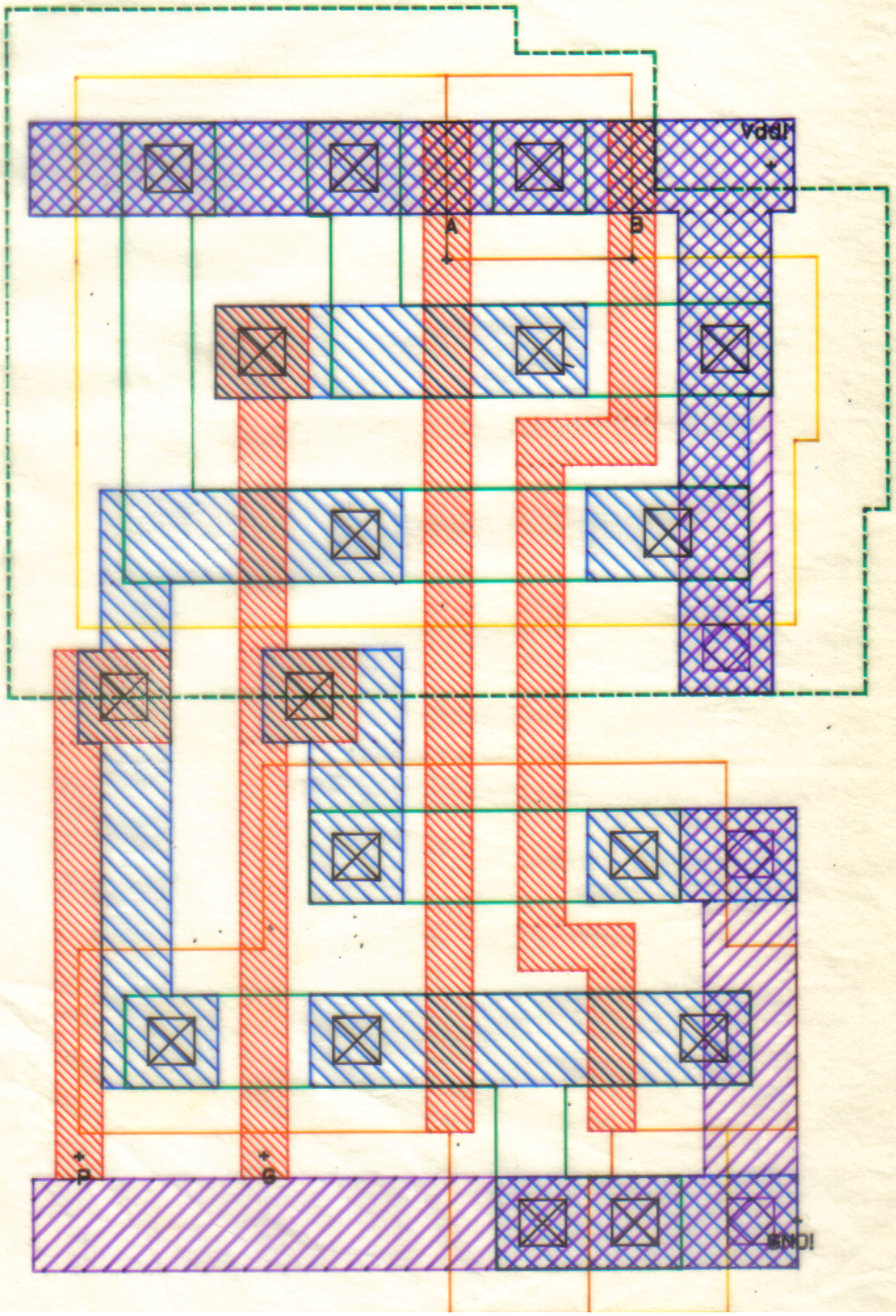


nwell

INTEGRATED CIRCUIT DESIGN

DATE : Sep 9, 1992
FILENAME : xnor1

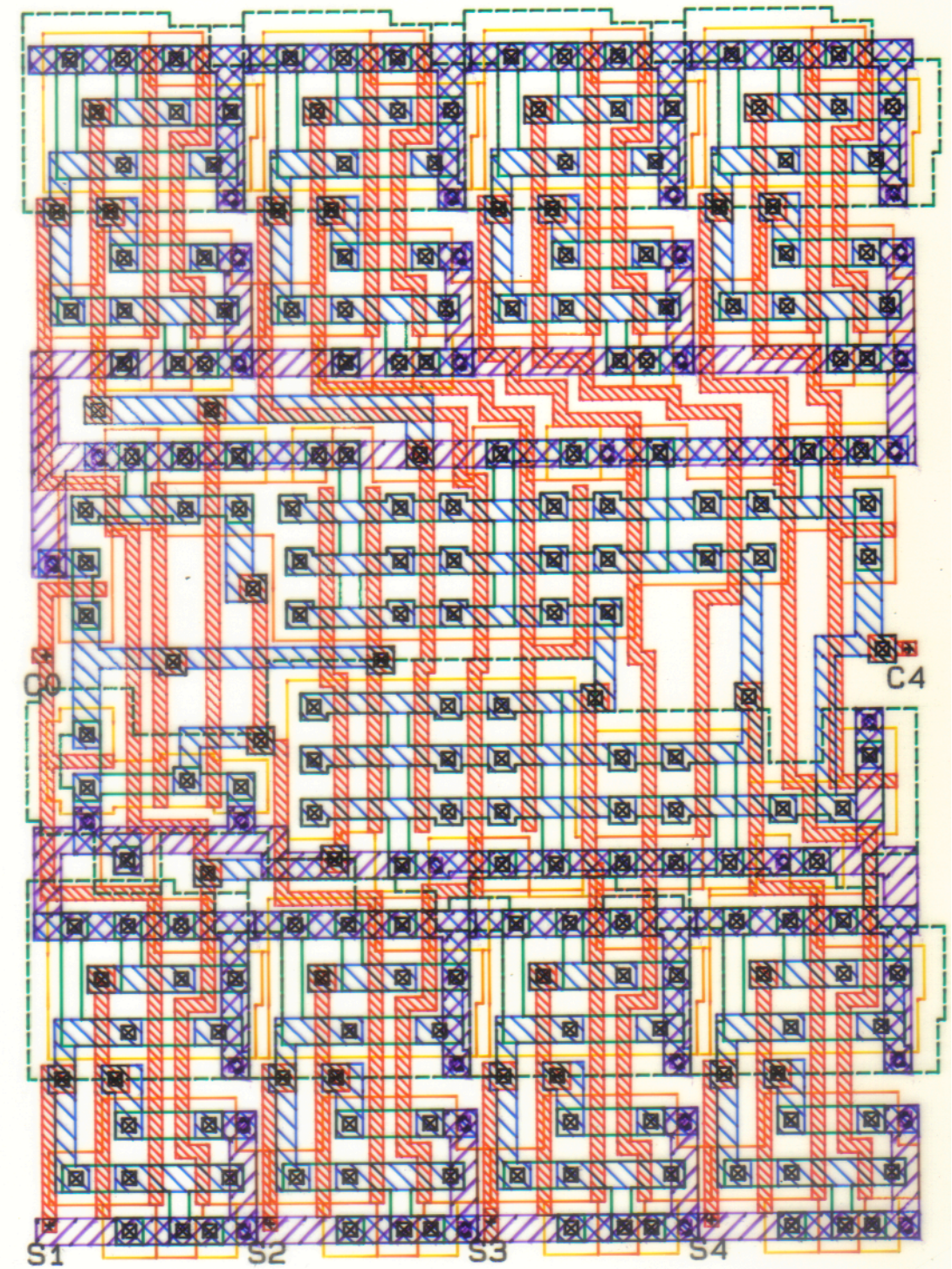
SCALE : 4 microns/inch
USER : lws



INTEGRATED CIRCUIT DESIGN

DATE : Sep 9, 1992
FILENAME : cla

SCALE : 13 microns/inch
USER : lws



Carry Select Adders (CSA)

- Uses small RCA implementation (3-4 bit).
- Sums are calculated assuming $C_{in} = 0$ and $C_{in} = 1$. The sum bits transferred to the output in one shot once carry is known using multiplexer.
- Architecture is as shown.
- If k_1 is one bit adder delay then for n bits, RCA delay $= k_1 n$. If m parallel paths (blocks) are used and k_2 is carry multiplexer delay

$$\text{CSA delay} = k_1 \frac{n}{m} + k_2 (m - 1)$$

- Optimization needed to get m.

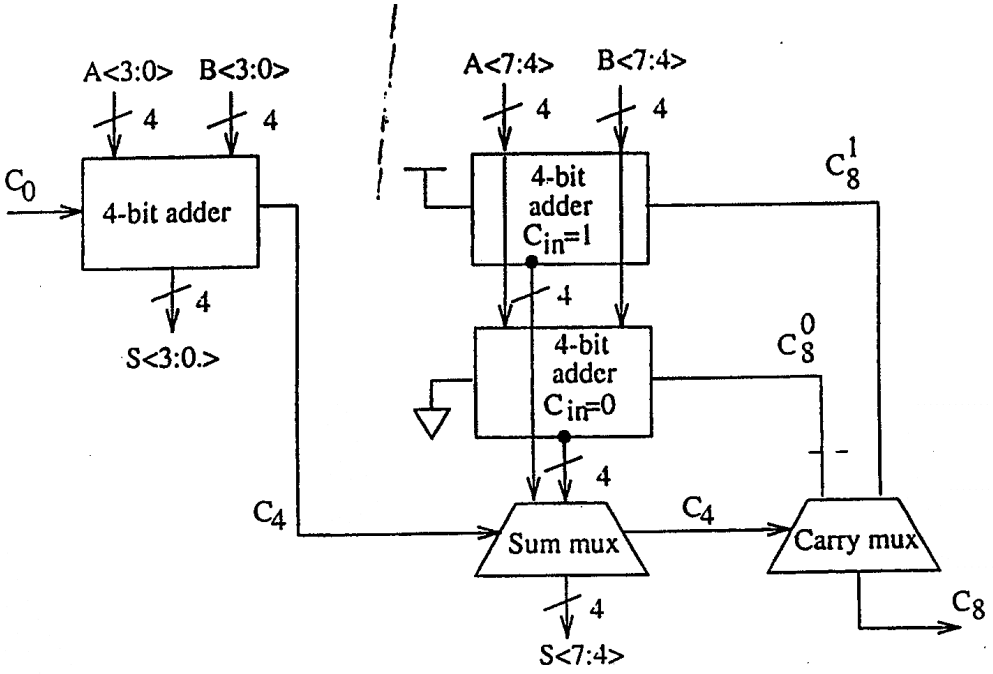


Figure 7.15 An 8 bit architecture of carry select adder.

hapter 3, simulations show that the optimal staging of a 32-bit CS adde
g TGs is 4-4-7-9-8 at 3.3 V power supply voltage. This implementatio
egular and easy to layout, however it has a higher occupied area than th
A.

4 Conditional Sum Adders

960 Sklansky considered the Conditional Sum Adder (CSA) as the fastes
from a theoretical point of view [2, 3]. The concept behind this architectur
xplained using the basic circuit of Fig. 7.16. This example is for a 4-bi
ditional sum adder. It uses two types of cells: i) the conditional cell, and ii
multiplexer. For each bit there is one conditional cell circuit. It compute
sums and two carries: S^0 and C^0 are calculated for a carry in zero, and S
C^1 are calculated for a carry in one. The selection of the true sum is don
the first carry in and the previous carries. The true final carry out (C_4 i
7.16) is also selected.

Comparison

- CLA fastest at high power.
- RCA slow but regular and low power.
- CSA gives compromise.

Design of Multipliers

- Braun Multiplier- Simple unsigned multiplier

  Consider two n-bit unsigned numbers (binary)

$$X = \sum_{i=0}^{n-1} X_i \, 2^i$$

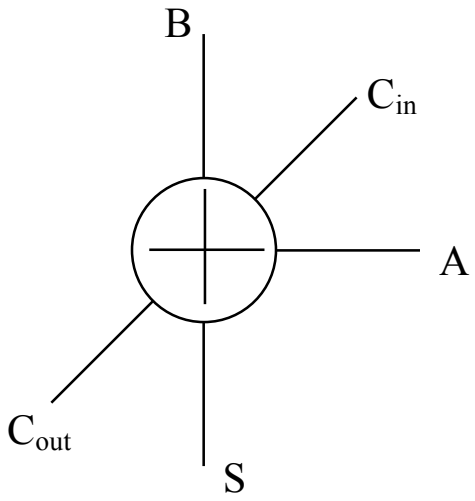
$$Y = \sum_{j=0}^{n-1} Y_j \, 2^j$$

$$\therefore XY = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} X_i Y_j \, 2^{i+j}$$

  Some issues are clear.

- Product binary number has 2n bits.
- $X_i Y_i$ terms can be calculated using AND gates.
- For a given value of (i + j), there are many i, j combinations, which have the same (i + j) value, that contribute to the $(i + j)^{th}$ bit. For example, for 2^4 bit position, X_0Y_4, X_1Y_3, X_2Y_2, X_3Y_1, X_4Y_0 contribute and have to be added.
- Adder cells are required to do these additions.
- General multiplication process is as shown. Carry propagates sideways.
- 4 bit × 4 bit multiplication with arbitrary bits.

- Adder cell block.



A – B inputs

S sum

4 × 4 configuration as indicated.

- Design fast AND and ADDER blocks.
- 4 × 4 bit multiplication requires 12 adder cells and 16 ANDs.

$$
\begin{array}{llll}
 & X_3 & X_2 & X_1 & X_0 & = X \\
 & Y_3 & Y_2 & Y_1 & Y_0 & = Y \\
\hline
 & X_3Y_0 & X_2Y_0 & X_1Y_0 & X_0Y_0 & \\
 & X_3Y_1 & X_2Y_1 & X_1Y_1 & X_0Y_1 & \\
 & X_3Y_2 & X_2Y_2 & X_1Y_2 & X_0Y_2 & \\
 & X_3Y_3 & X_2Y_3 & X_1Y_3 & X_0Y_3 & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0 & = P = X*Y
\end{array}
$$

(a)

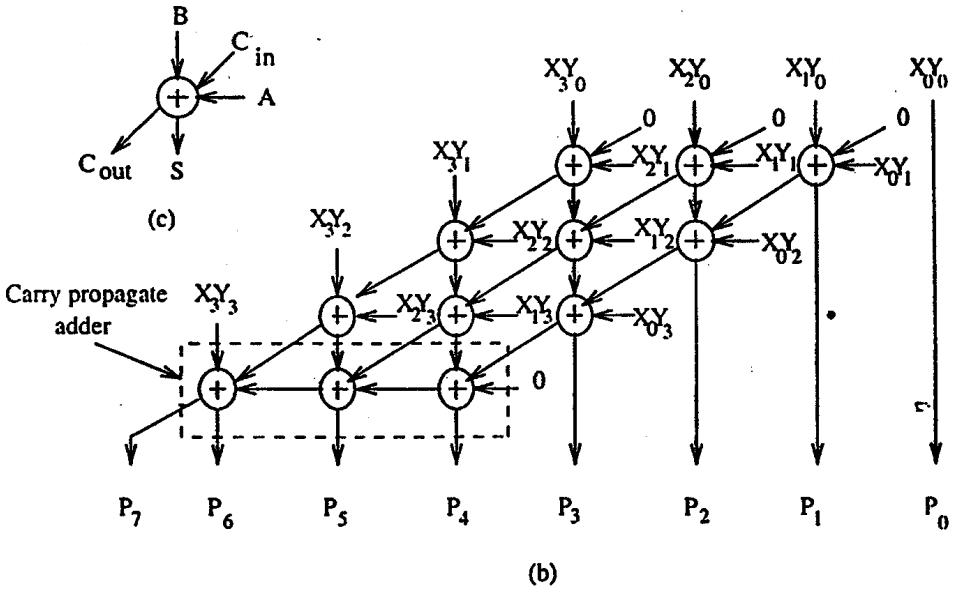


Figure 7.20 (a) Partial products of a 4 × 4 unsigned integer multiplication; (b) multiplier array; (c) full-adder schematic.

Multiplication Process

Consider multiplication of two 5 bits binary numbers.

					1	1	1	1	1
			×		1	1	1	1	1
					1	1	1	1	1
				1	1	1	1	1	X
			1	1	1	1	1	X	X
		1	1	1	1	1	X	X	X
	1	1	1	1	1	X	X	X	X
	1	1	1	1	1		1		
	1		1	1	1				
1	1	1	1	0	0	0	0	0	1

Which gives $31 \times 31 = 961 = 2^9 + 2^8 + 2^7 + 2^6 + 2^0$

Another example

					1	1	0	1	0
			×		1	0	1	1	0
					0	0	0	0	0
				1	1	0	1	0	X
			1	1	0	1	0	X	X
		0	0	0	0	0	X	X	X
	1	1	0	1	0	X	X	X	X
1	0	0	0	1	1	1	1	0	0

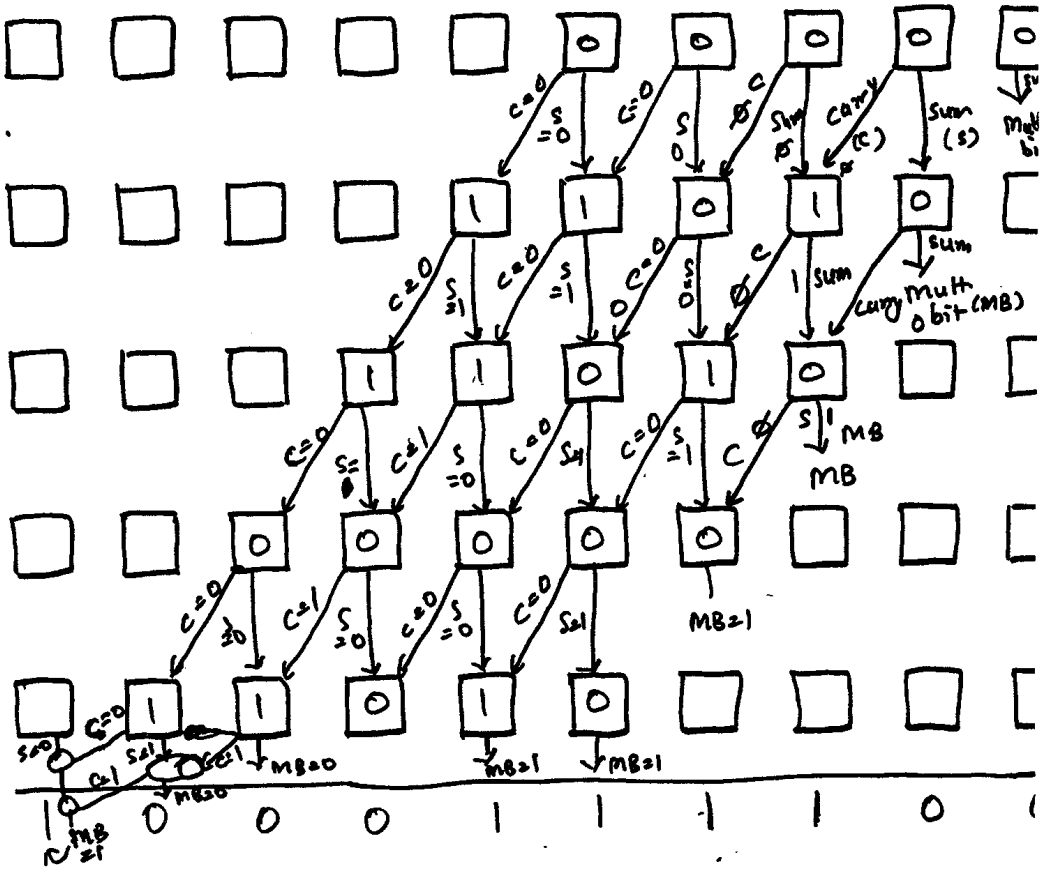
Which of course gives

$$26 \times 22 = 572 = 1 \times 2^9 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2$$

These can be implemented by noticing the next "procedure" diagram.

- For n bits, n(n-1) adders and n^2 AND gates are required in this implementation.
- Works for unsigned or positive numbers.
- To handle –ve numbers in 2's complement form, Baugh-Wooley multiplier is proposed which will not be discussed.

$$
\begin{array}{r}
1\ 1\ 0\ 1\ 0 \\
\times \\
1\ 0\ 1\ 1\ 0 \\
\hline
\end{array}
$$



which of course gives

$26 \times 22 = 572 = 1\times 2^9 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 1\times 2$

○ ○ adder cell □ multiplier cell.