Module 3 contd.. - Subsystem Design processes

This topic covers the design of a digital system using a top- approach. The complete system environment is that of a **4-bit microprocessor** which is readily envisaged as an interconnection of four major **Archi- -tectural blocks-ALU, control Unit, I/O Unit and Memory.**

The first part of the question—'What's in it for me?'—may be quite simply answered as: Providing better ways of tackling some problems, providing a way of designing and realizing systems that are too large, too complex, or just not suited to 'off-the-shelf' components and providing an appreciation and understanding of IC technology.

'Better' may include:

1. *Lower unit cost* compared with other approaches to the same requirement. Quantity plays a part here but even small quantities, if realized through cooperative ventures such as the multiproject chip (MPC) or multiproduct wafer (MPW), can be fabricated for as little as \$200 (MPC) or \$500 (MPW) per square millimetre of silicon, including the bonding and packaging of five or six chips per customer.
2. *Higher reliability.* High levels of system integration usually greatly reduce interconnections—a weak spot in any system.
3. *Lower power dissipation, lower weight, and lower volume* compared with most other approaches to a given system.
4. *Better performance*—particularly in terms of speed power product.
5. *Enhanced repeatability.* There are fewer processes to control if the whole system or a very large part of it is realized on a single chip.
6. *The possibility of reduced design/development periods* (particularly for more complex systems) if suitable design procedures and design aids are available.

7.1.1 Some Problems

Some of the problems associated with VLSI design are:

1. How to design large complex systems in a reasonable time and with reasonable effort. This is a problem shared with other approaches to system design.
2. The nature of architectures best suited to take full advantage of VLSI and the technology.
3. The testability of large/complex systems once implemented in silicon.

Problems 1 and 3 are greatly reduced if two aspects of standard practice are accepted:

- Approach the design in a top-down manner and with adequate computer-aided tools to do the job. Partition the system sensibly, aiming for simple interconnection between subsystems and high regularity within subsystems. Generate and then verify each section of the design.
- Design testability into the system from the outset and be prepared to devote a significant proportion (e.g. up to 30%) of the total chip area to test and diagnostic facilities.

These problems are the subject of considerable research and development activity at this time.

7.2 AN ILLUSTRATION OF DESIGN PROCESSES

- Structured design begins with the concept of hierarchy.
- It is possible to divide any complex function into less complex subfunctions. These may be subdivided further into even simpler subfunctions and so on—the bottom level being commonly referred to as 'leaf-cells'.
- This process is known as top-down design.
- As a system's complexity increases, its organization changes as different factors become relevant to its creation.
- Coupling can be used as a measure of how much submodules interact. Clever systems partitioning aims at reducing implicit complexity by minimizing the amount of interaction between subparts; thus independence of design becomes a reality.
- It is crucial that components interacting with high frequency be physically proximate, since one may pay severe penalties for long, high-bandwidth interconnects.
- Concurrency should be exploited—it is desirable that all gates on the chip do useful work most of the time.
- Because technology changes so fast, the adaptation to a new process must occur in a short time. Thus a technology-independent description becomes important.

In representing a design, several approaches may be used at different stages of the design process; for example:

- conventional circuit symbols;
- logic symbols;
- stick diagrams;
- any mixture of logic symbols and stick diagrams that is convenient at a particular stage;
- mask layouts;
- architectural block diagrams;
- floor plans.

The General Arrangement of a 4-bit Arithmetic Processor

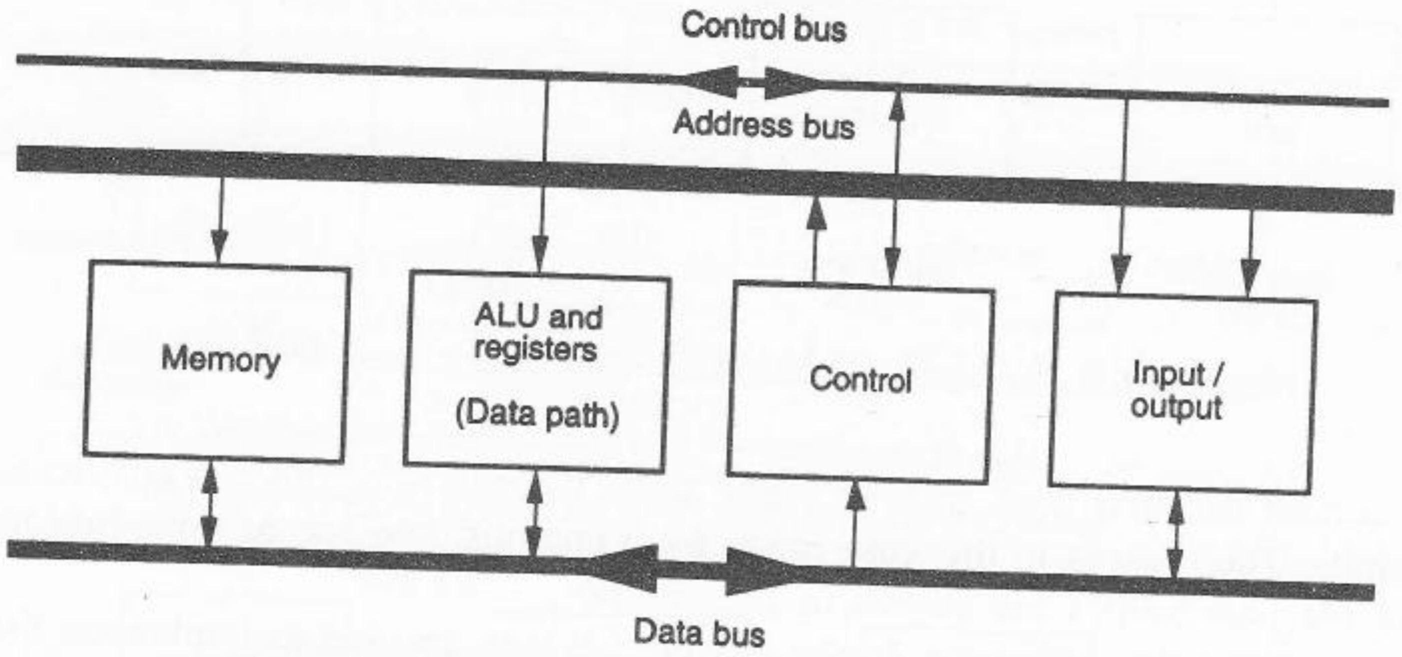
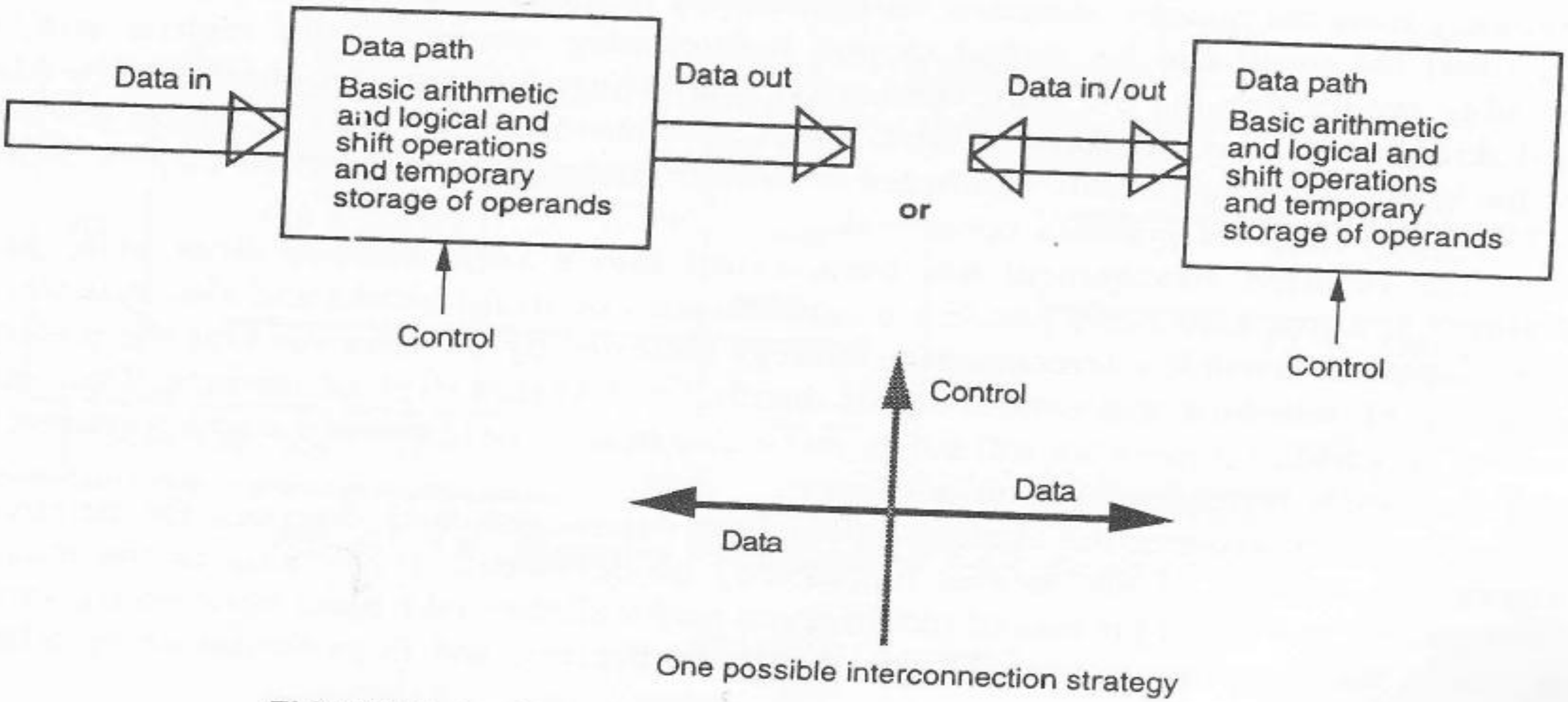


FIGURE 7.1 Basic digital processor structure.

The data path has been separated out in Figure 7.2



Data in
Data path
Basic arithmetic and logical and shift operations and temporary storage of operands
Data out
Control
or
Data in/out
Data path
Basic arithmetic and logical and shift operations and temporary storage of operands
Control
Control
Data
Data
One possible interconnection strategy

FIGURE 7.2 Communications strategy for data path.

Now we will decompose the data path into a block diagram showing the main subunits. In doing this it is useful to anticipate a possible *floor plan* to show the planned relative disposition of the subunits on the chip and thus on the mask layouts. A block diagram is presented in Figure 7.3.

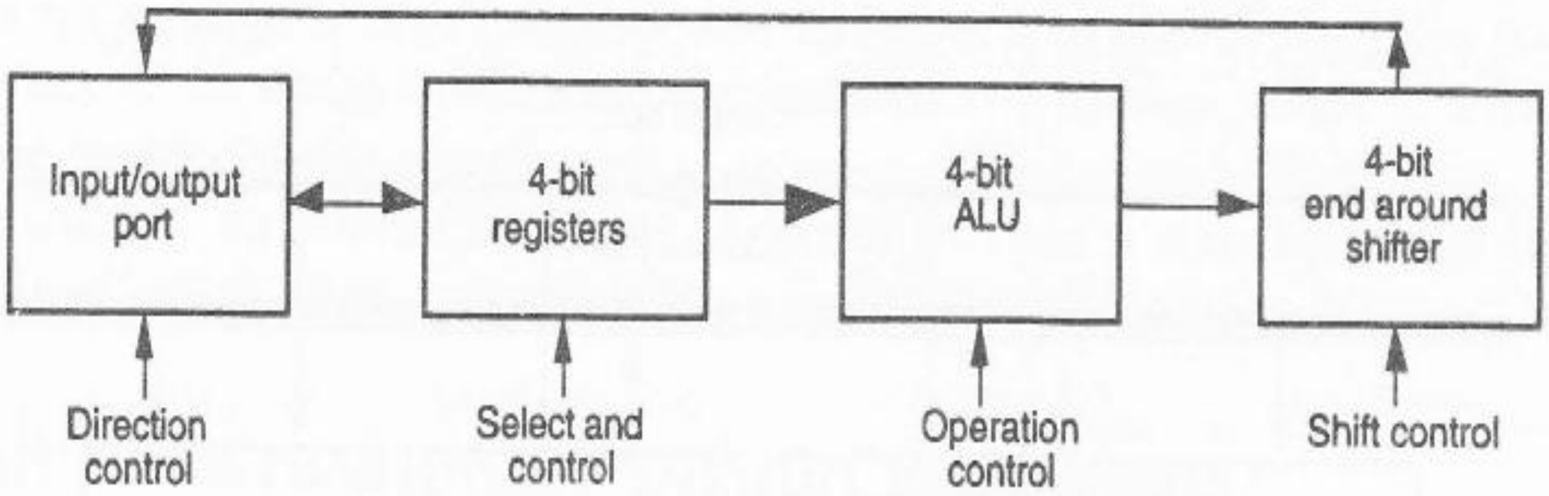
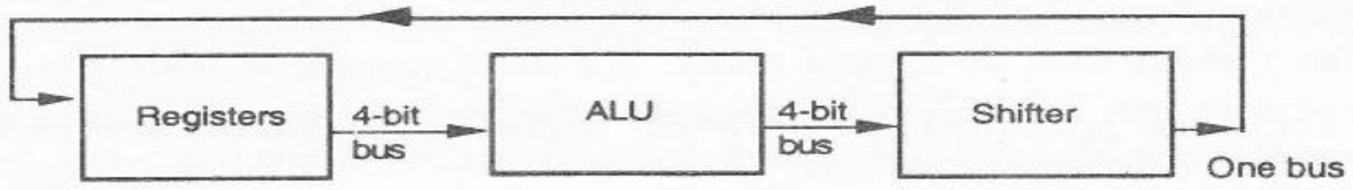


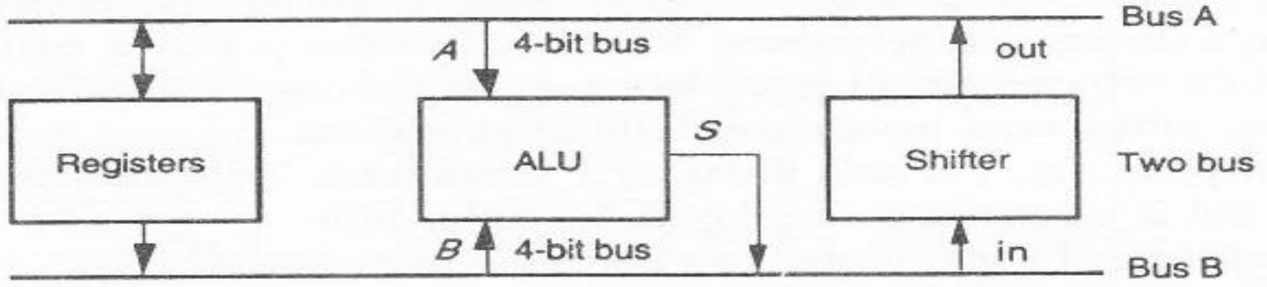
FIGURE 7.3 Subunits and basic interconnections for data path.

A further decision must then be made about the nature of the bus architecture linking the subunits. The choices in this case range from one-bus, two-bus or three-bus architecture.



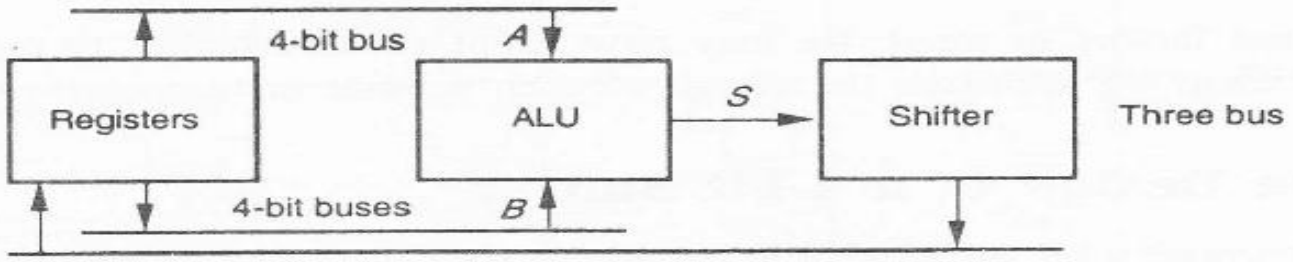
Sequence:

(i) First operand from registers to ALU. Operand is stored there.
(ii) Second operand from registers to ALU. Operands are added (etc.) and the result is, say, stored in the ALU.
(iii) The result is passed through shifter and stored in the registers.



Sequence:

(i) Two operands (*A* and *B*) are sent from register(s) to ALU and are operated upon and the result (*S*) is stored in ALU.
(ii) Result is passed through the shifter and stored in the registers.



Sequence:

The two operands (*A* and *B*) are sent from the registers, operated upon, and the shifted result (*S*) returned to another register *all in the same clock period.*

FIGURE 7.4 Basic bus architectures.

The required arrangement has been turned into a very tentative floor plan, as in Figure 7.5, which indicates a possible relative disposition of the blocks and also indicates an acceptable and sensible interconnection strategy indicated by the lines showing the preferred direction of data flow and control signal distribution. At this stage of learning, floor plans

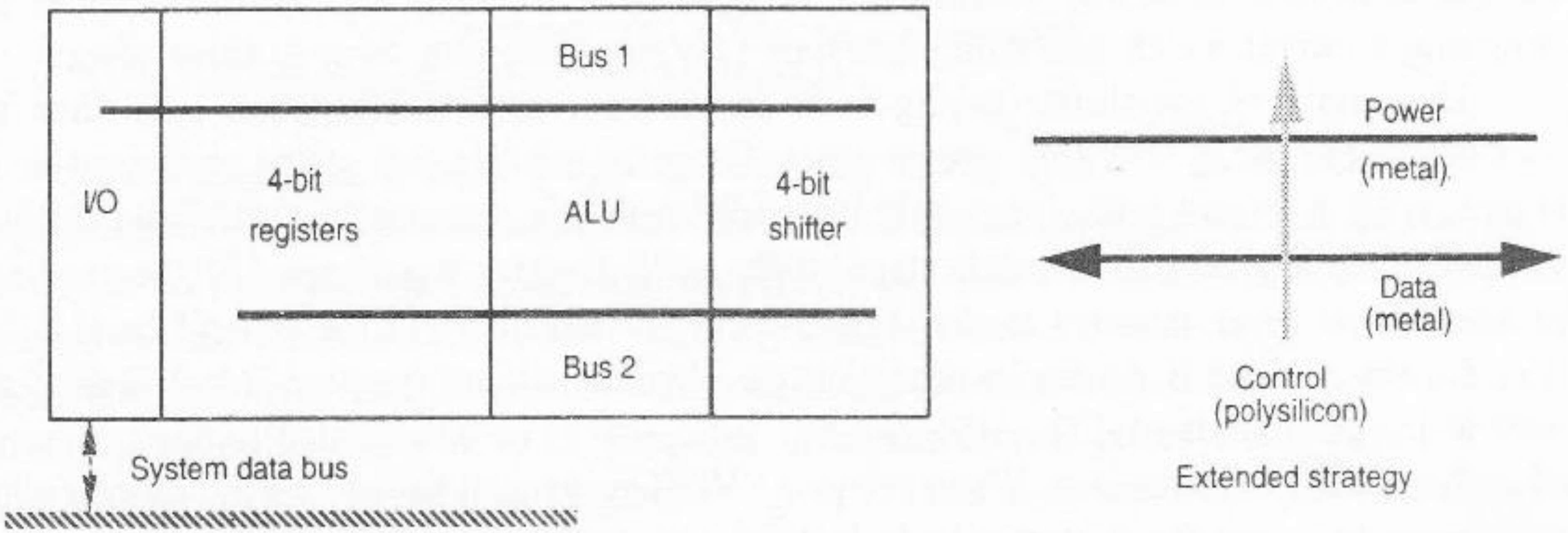


FIGURE 7.5 Tentative floor plan for 4-bit data path.

7.2.2 The Design of a 4-bit Shifter

Any general purpose n-bit shifter should be able to shift incoming data by up to $n - 1$ places in a right-shift or left-shift direction. If we now further specify that all shifts should be on an 'end-around' basis, so that any bit shifted out at one end of a data word will be shifted in at the other end of the word, then the problem of right shift or left shift is greatly eased.

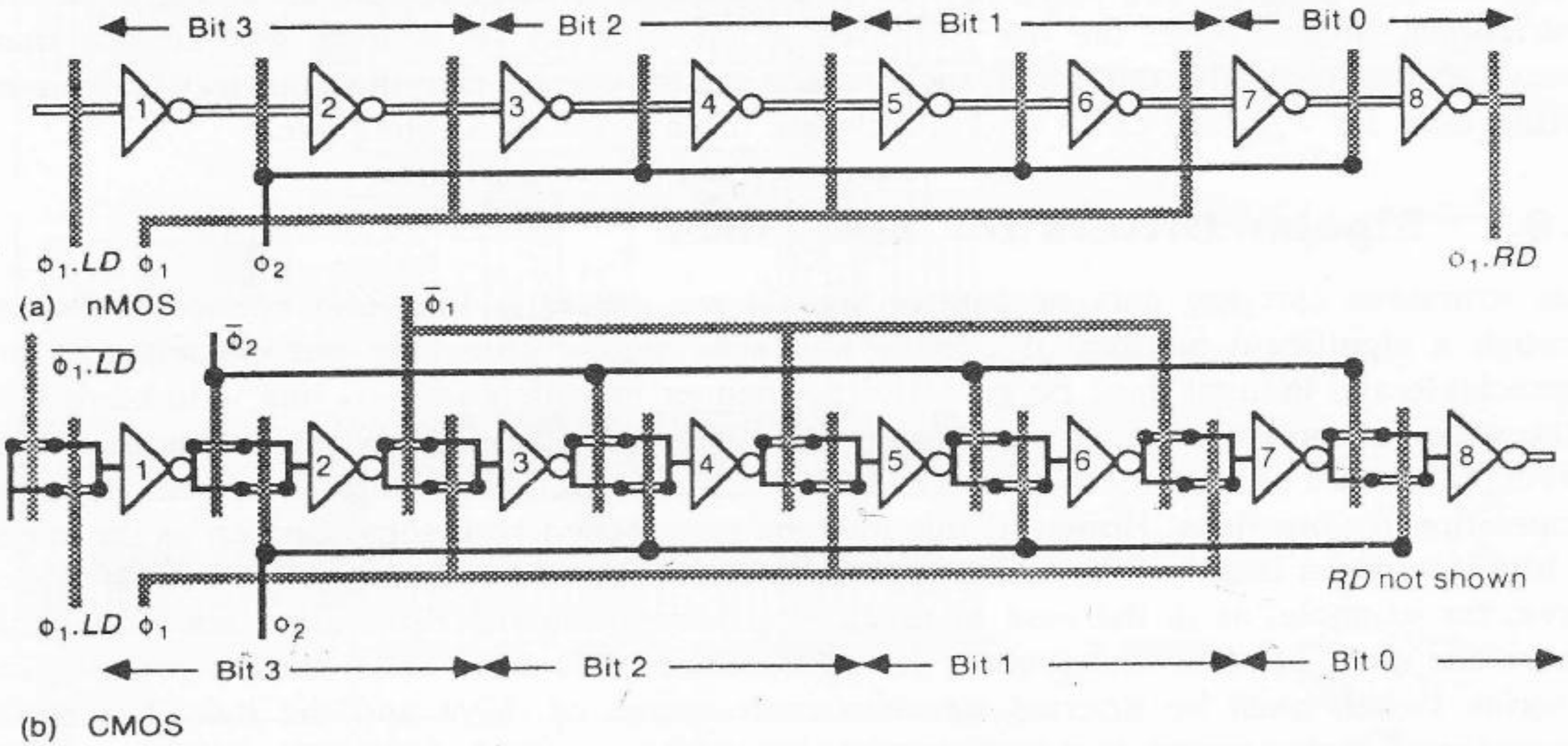


FIGURE 6.38 Four-bit dynamic shift registers (nMOS and CMOS).

of such standard arrangements. When designing VLSI systems, it pays to set out exactly what is required to assess the best approach. In this case, the shifter must have:

- input from a four-line parallel data bus;
- four output lines for the shifted data;
- means of transferring input data to output lines with any shift from zero to three bits inclusive.

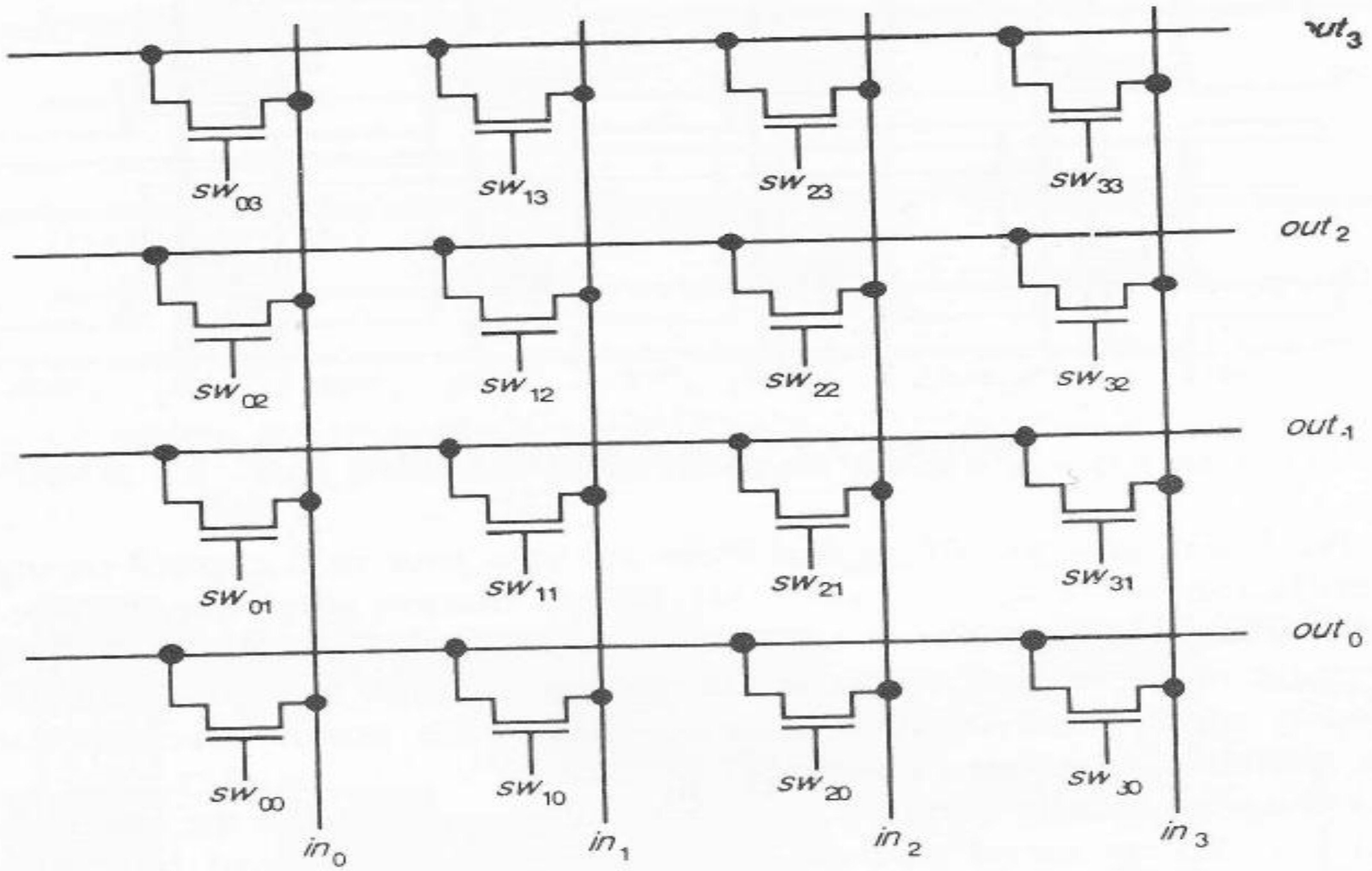


FIGURE 7.6 4 × 4 crossbar switch.

4 × 4 barrel shifter.

The interbus switches have their gate inputs connected in a staircase fashion in groups of four and there are now four shift control inputs which must be mutually exclusive in the active state. CMOS transmission gates may be used in place of the simple pass transistor switches if appropriate.

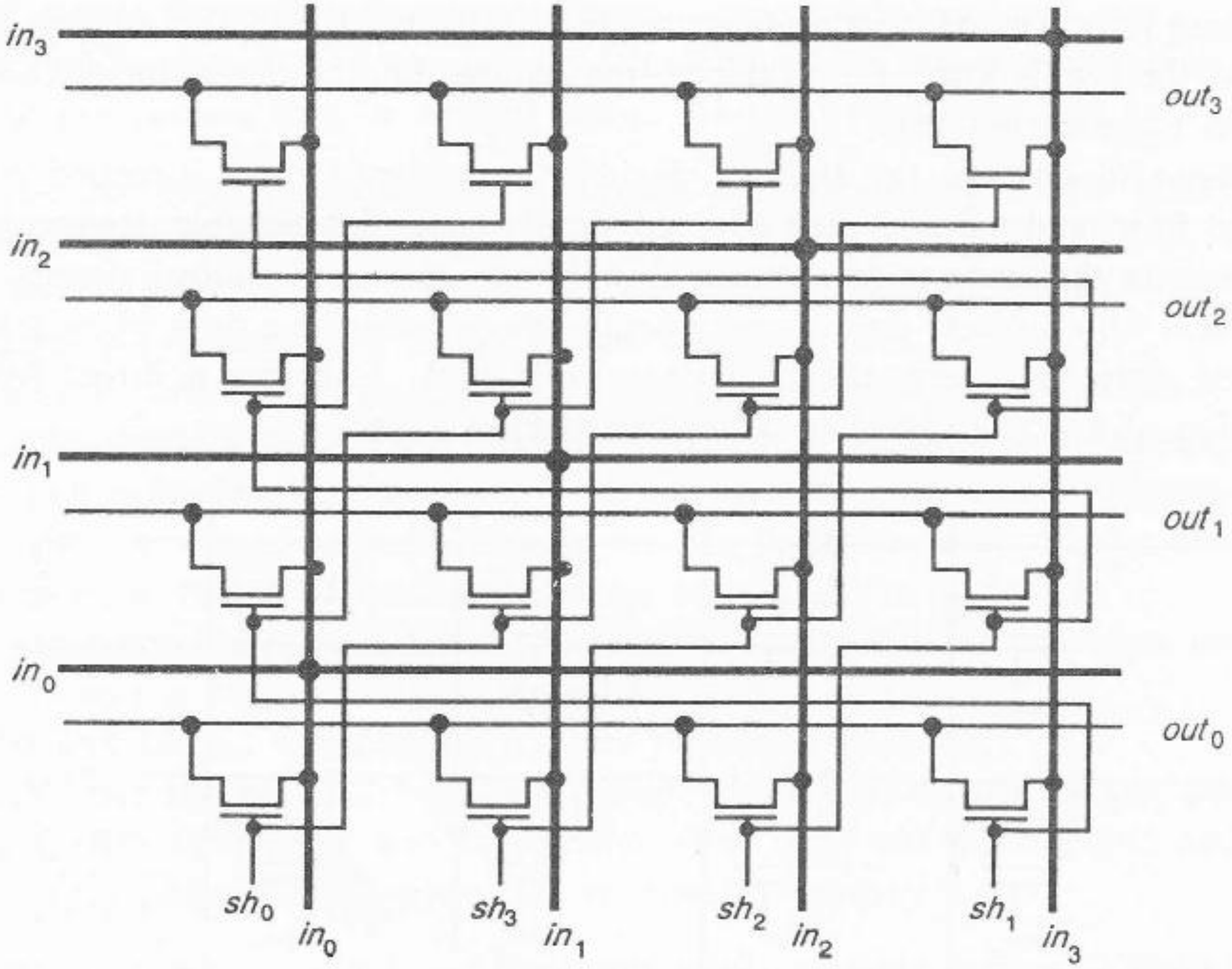


FIGURE 7.7 4 × 4 barrel shifter.

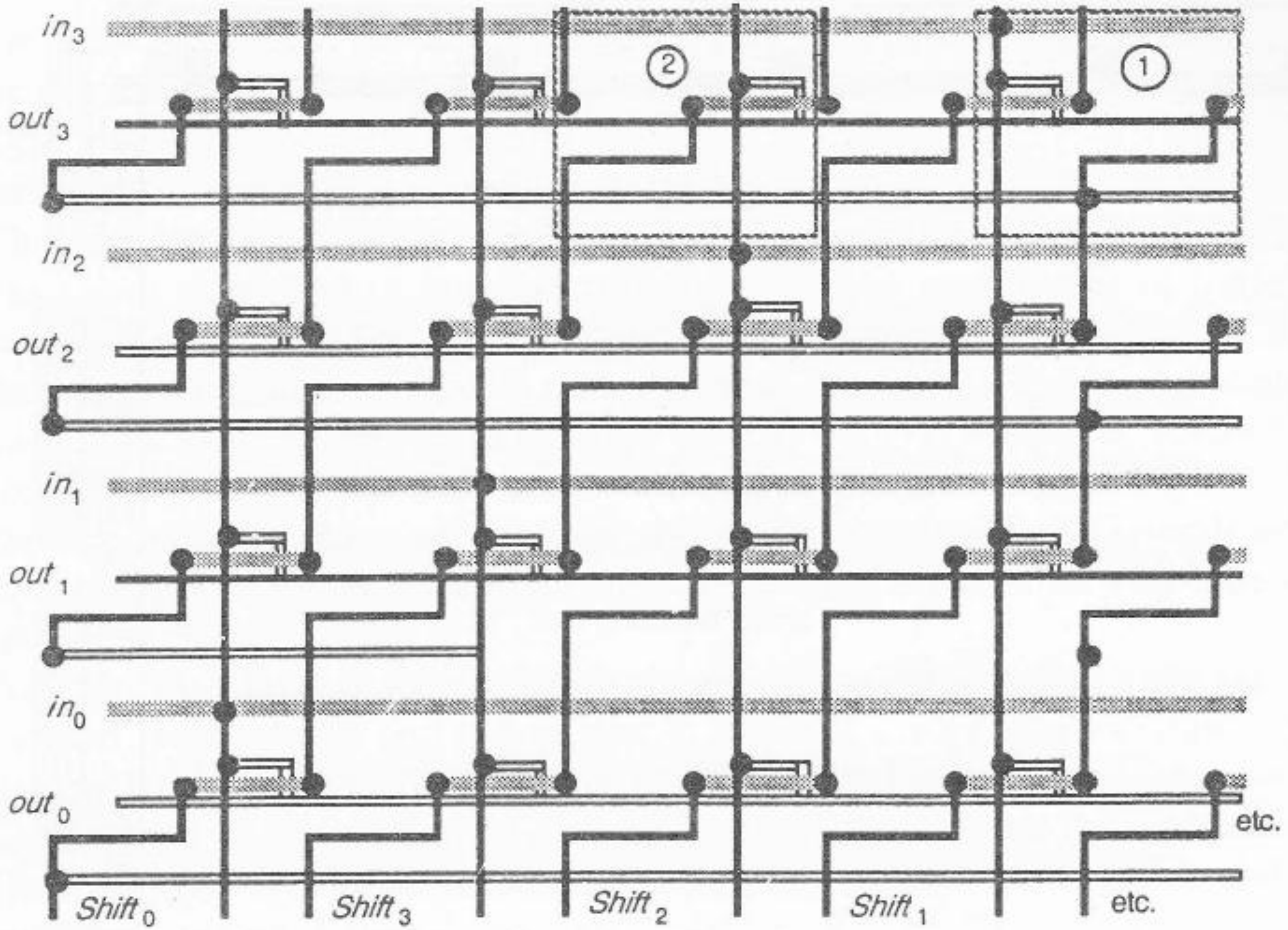


FIGURE 7.8 One possible stick diagram for a 4 × 4 barrel shifter.

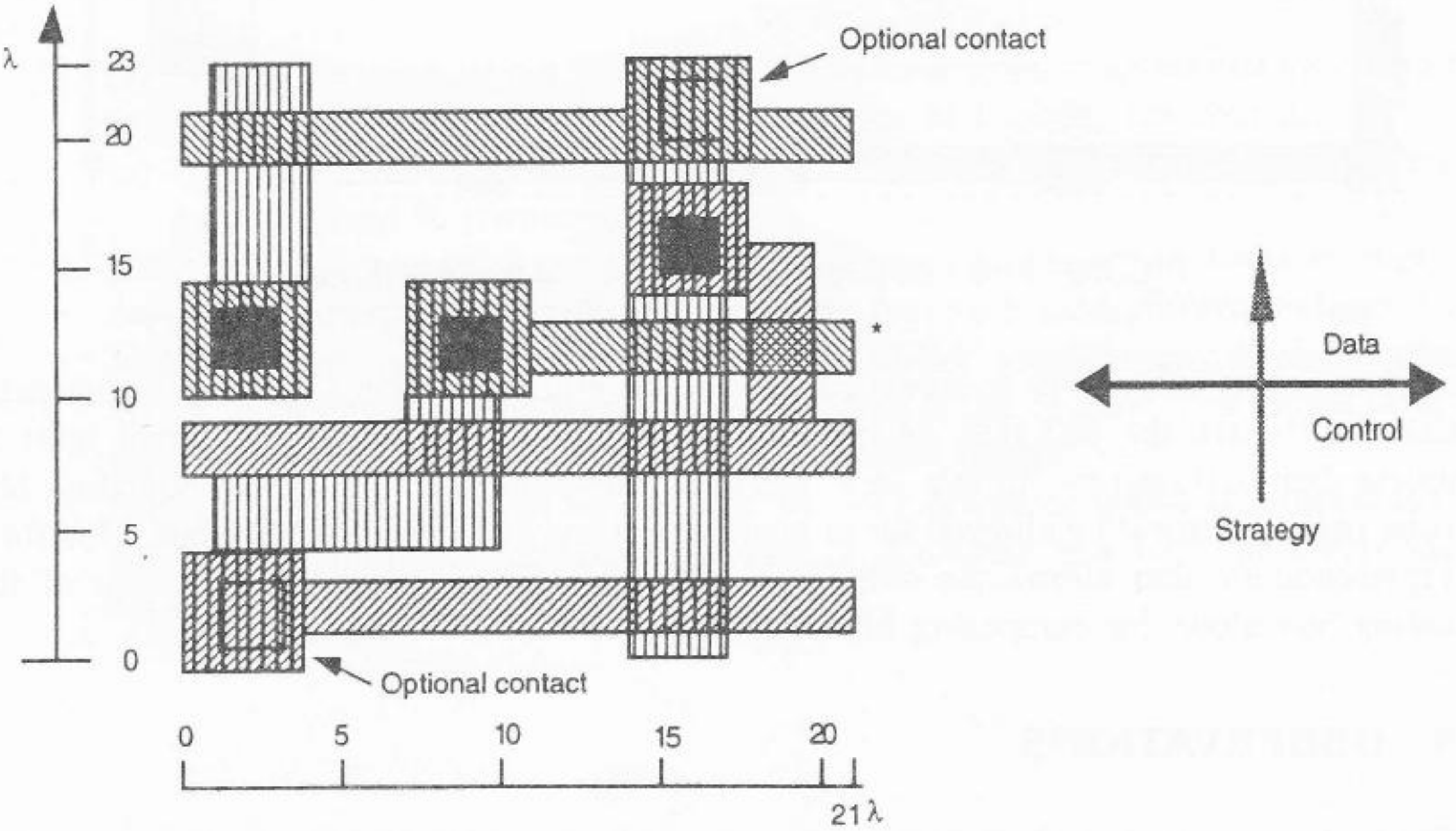


FIGURE 7.9 Barrel shifter standard cell 2—mask layout.

8.2 REGULARITY

So far we have used regularity as a qualitative parameter. Regularity should be as high as possible to minimize the design effort required for any system. The level of any particular design as far as this aspect is concerned may be measured by quantifying regularity as follows:

$$\text{Regularity} = \frac{\text{Total number of transistors on the chip}}{\text{Number of transistor circuits that must be designed in detail}}$$

For the 4 × 4-bit barrel shifter just designed, the regularity factor is given by

$$\text{Regularity} = \frac{16}{1} = 16$$

8.3 DESIGN OF AN ALU SUBSYSTEM

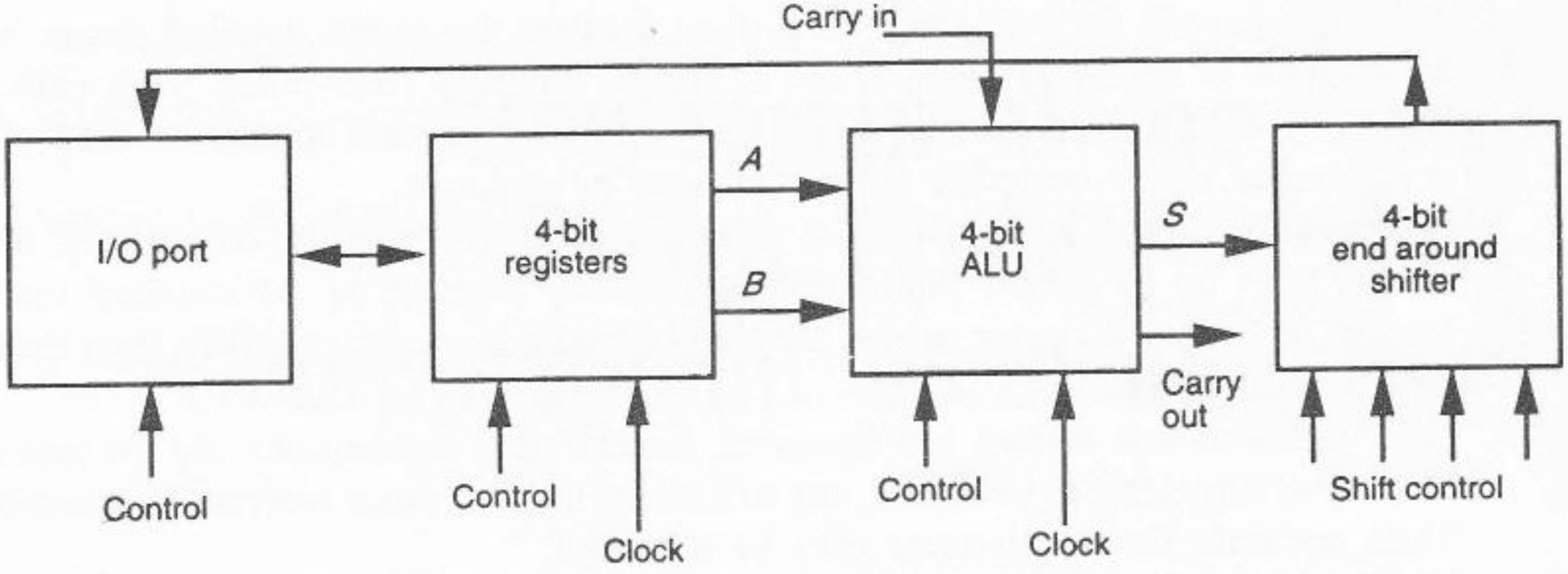


FIGURE 8.1 4-bit data path for processor (block diagram).

The heart of the ALU is a 4-bit adder circuit and it is this which we will actually design, indicating later how it may be readily adapted to subtract and perform logical operations.

8.3.1 Design of a 4-bit Adder

In order to derive the requirements for an n-bit adder, let us first consider the addition of two binary numbers A + B as follows:

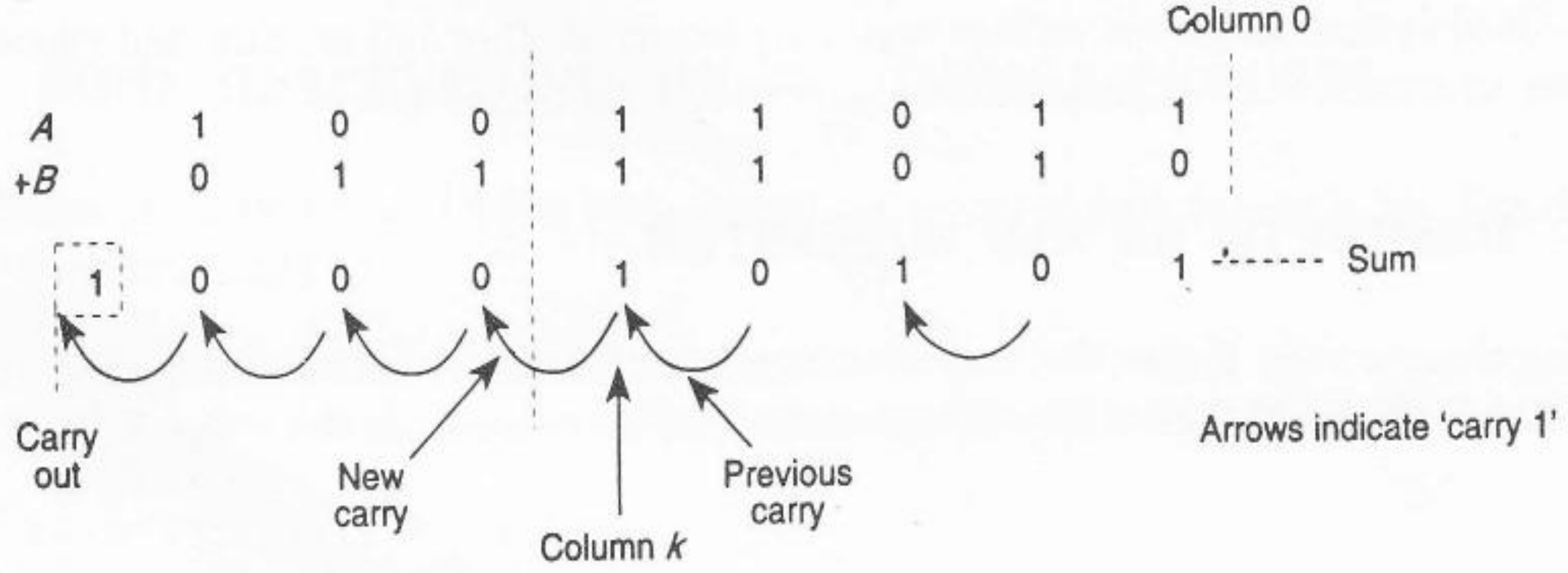


TABLE 8.1 Truth table for binary adder

Inputs			*Outputs*	
A_k	B_k	*Previous carry* C_{k-1}	*Sum* S_k	*New carry* 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

Conventionally, and assuming that we are not implementing a 'carry look ahead' facility, we may write *standard adder equations*, which fully describe the entries in Table 8.1.

One form of these equations is:

Sum $$S_k = H_k\bar{C}_{k-1} + \bar{H}_k C_{k-1}$$

New carry $$C_k = A_k B_k + H_k C_{k-1}$$

where

Half sum $$H_k = \bar{A}_k B_k + A_k \bar{B}_k$$

Previous carry is indicated as C_{k-1} and $0 \le k \le n - 1$ for n-bit numbers.

8.3.1.1 Adder element requirements

Table 8.1 reveals that the *adder requirements* may be stated thus:

If $A_k = B_k$ then $S_k = C_{k-1}$

else $S_k = \overline{C}_{k-1}$

and for the carry C_k

If $A_k = B_k$ then $C_k = A_k = B_k$*

else $C_k = C_{k-1}$

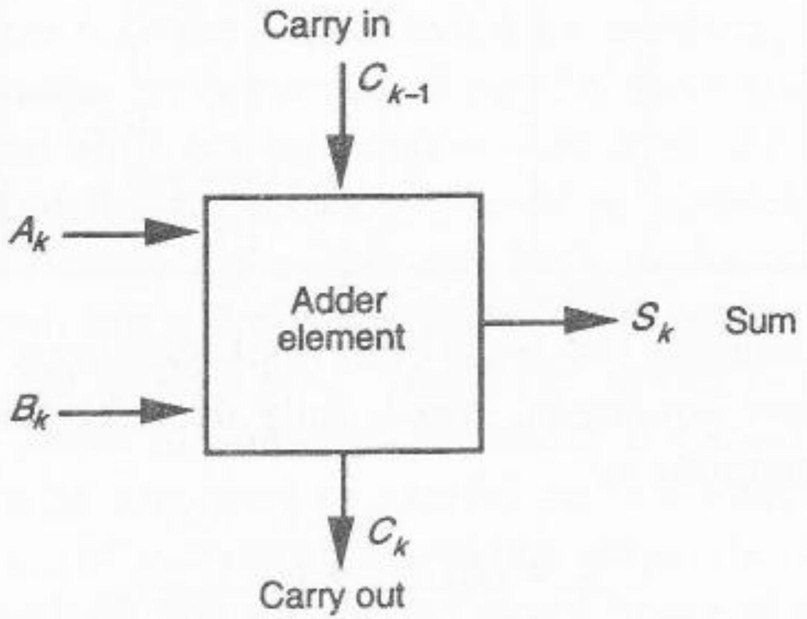
*This relationship could also have been stated as:

Carry $C_k = 1$ when $A_k = B_k = 1$

or $C_k = 0$ when $A_k = B_k = 0$

8.3.1.2 A standard adder element

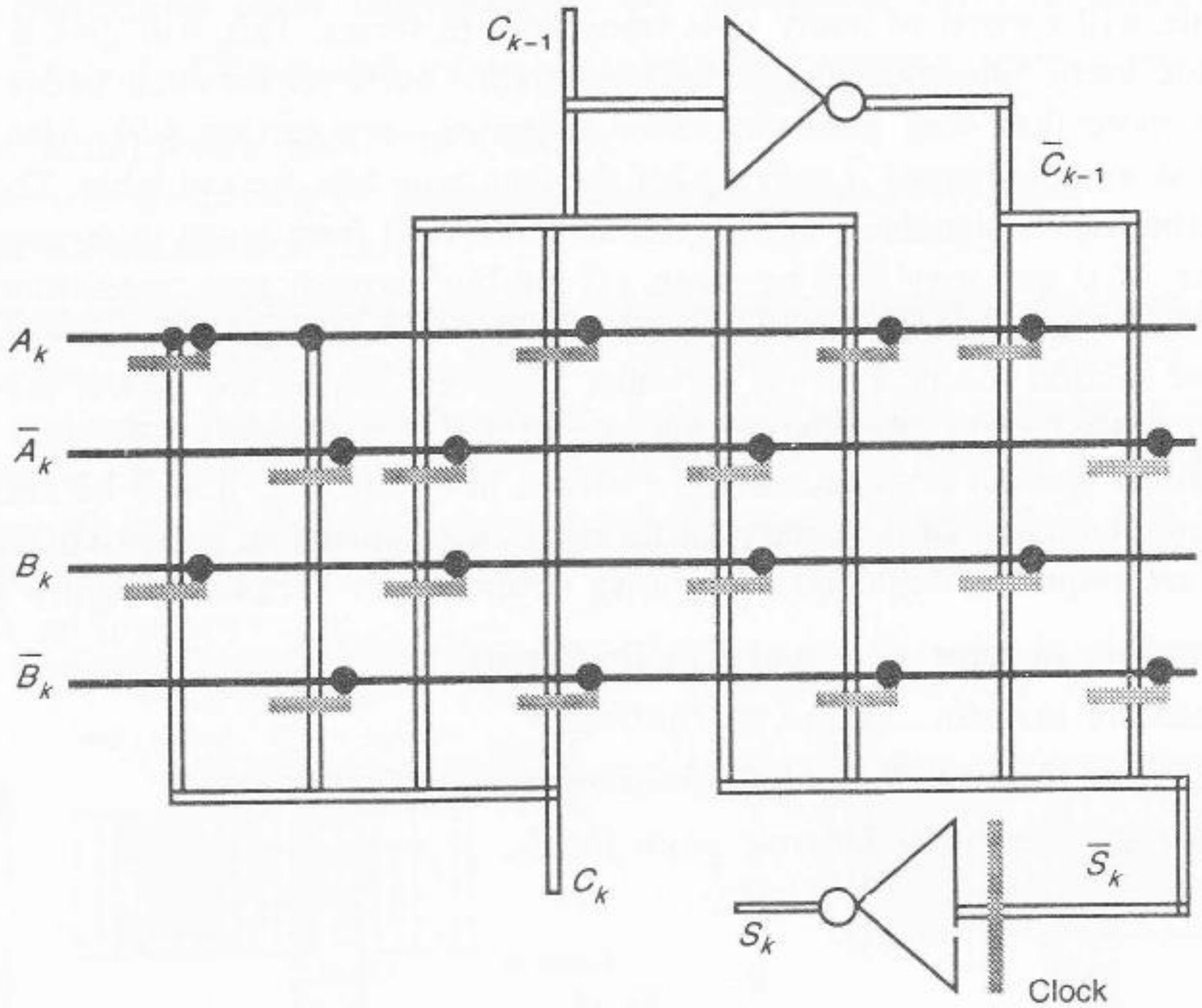
A 1-bit adder element may now be represented as in Figure 8.2. Note that any number of such elements may be cascaded to form any size of adder and that the element is quite general.



n such elements would be cascaded to form an *n*-bit adder.

FIGURE 8.2 Adder element.

Note also that this standard adder element may itself be composed from a number of replicated subcells. Regularity and generality must be aimed at in all levels of the architecture.



GURE 8.3 Multiplexer (n-switches)-based adder logic with stored and buffered sum output.

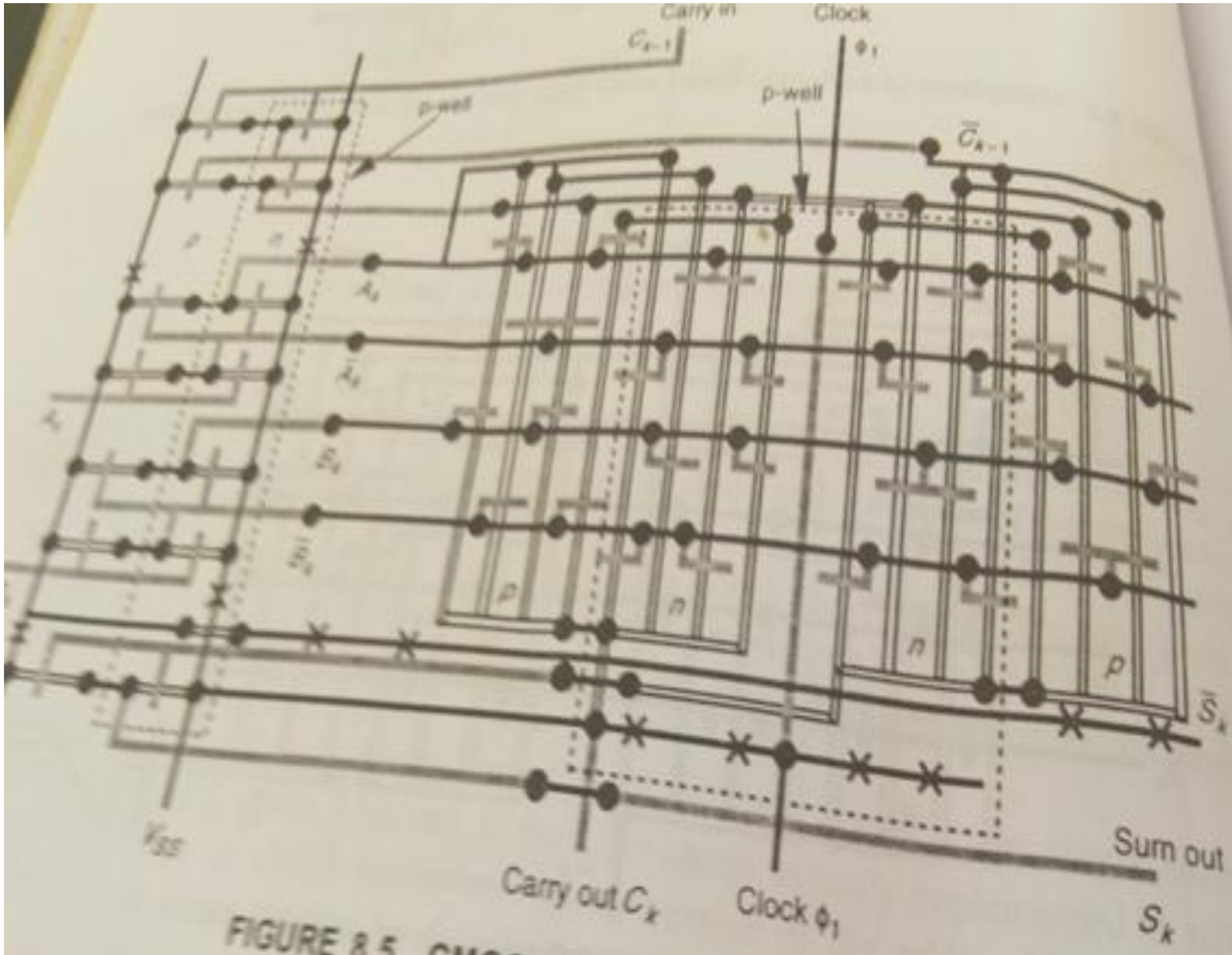
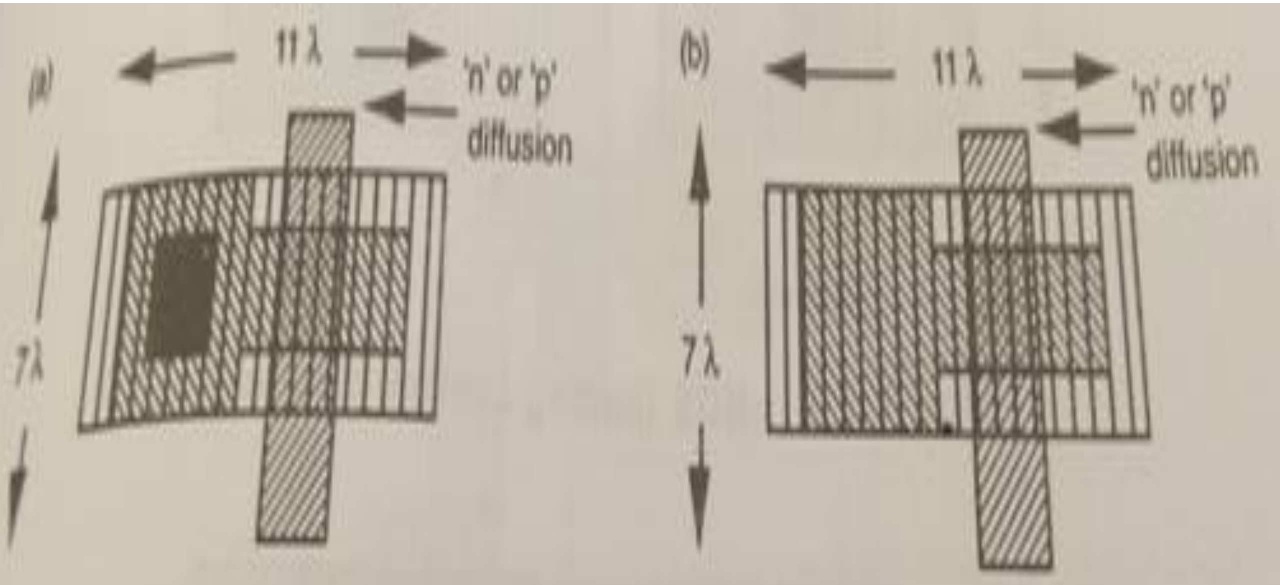
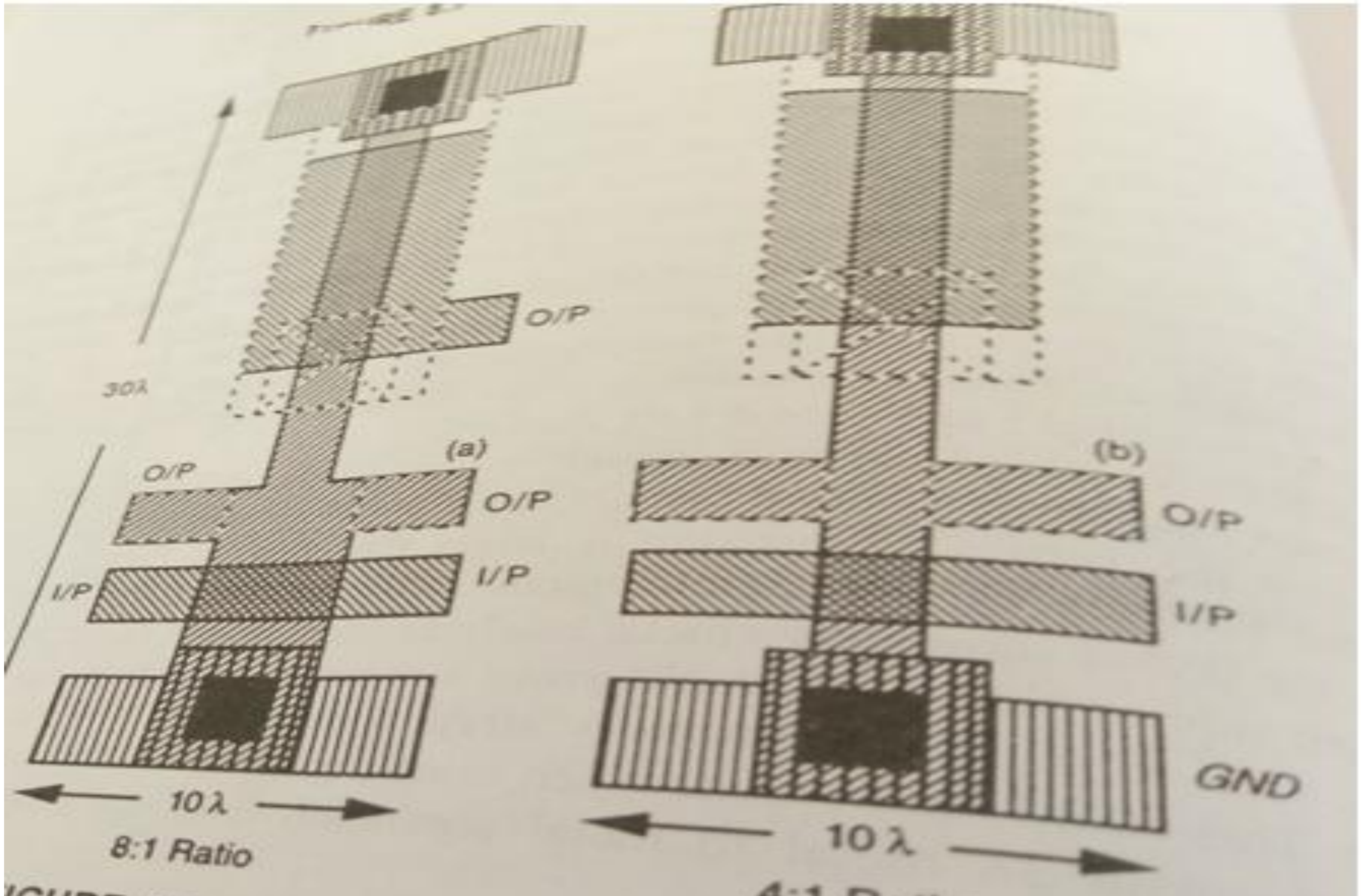


FIGURE 8.5 CMOS adder element.

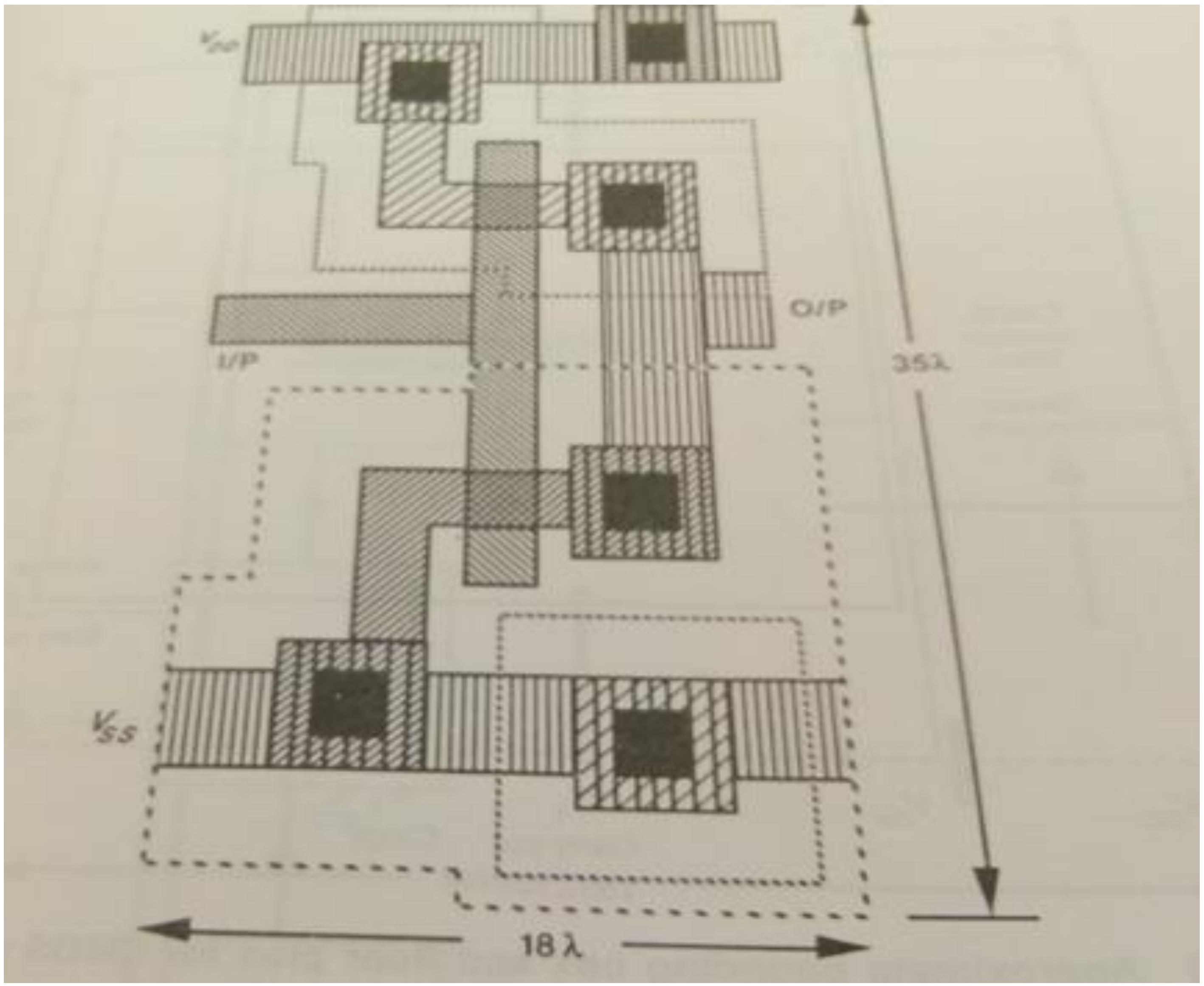
t only one in



FIGURES 8.6 Multiplexer cell with or without cut.



IGURE 8.8 nMOS (buried


V_{DD}
O/P
I/P
35λ
V_{SS}
18 λ

8.3.1.3 Standard cells required to be designed for the adder element

The stick diagram of Figure 8.5 shows that the adder consists of three parts:

1. the multiplexers (nMOS or CMOS);
2. the inverter circuits (4:1 and 8:1 ratio nMOS or CMOS);
3. the communication paths.

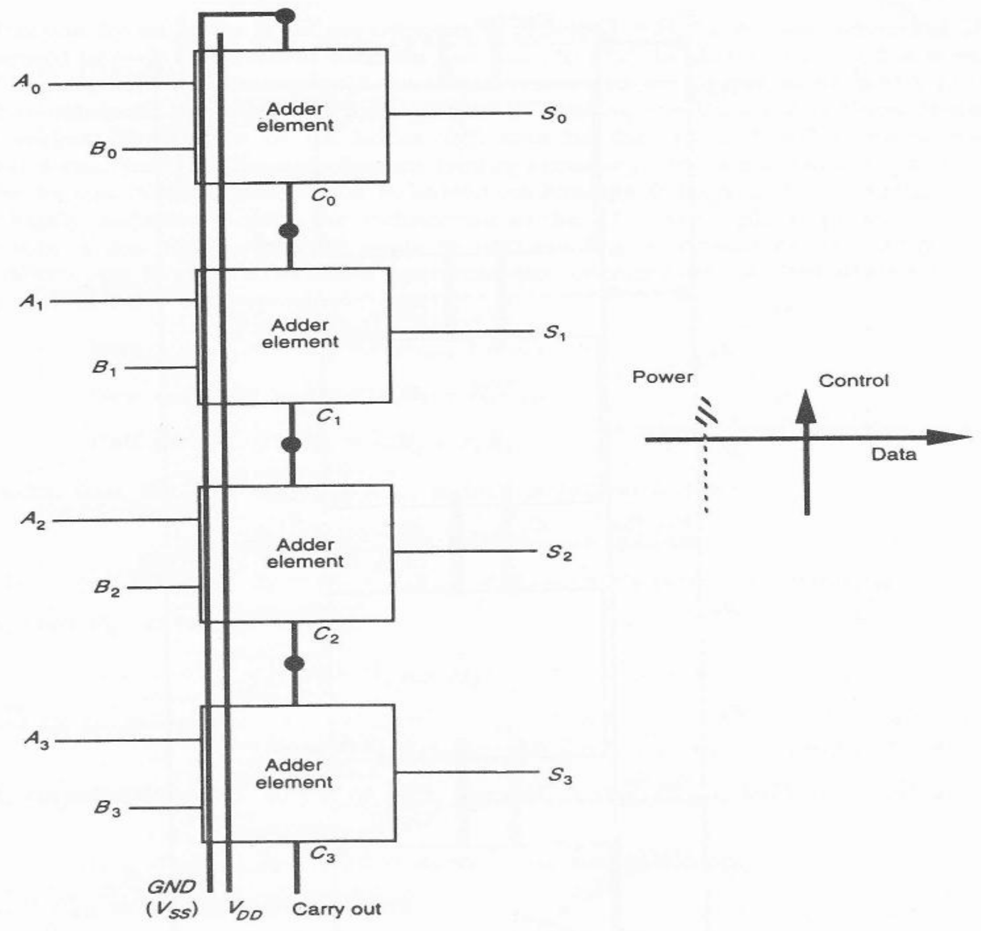


FIGURE 8.11(a) 4-bit adder.

8.3.2 Implementing ALU Functions with an Adder

An arithmetic and logical operations unit (ALU) must, obviously, be able to *add* two binary numbers ($A + B$), and must also be able to *subtract* ($A - B$).

From the point of view of logical operations it is essential to be able to *And* two binary words (A.B). It is also desirable to *Or* ($A + B$) and perhaps also detect *Equality*, and of course we also need an *Exclusive-Or* function.

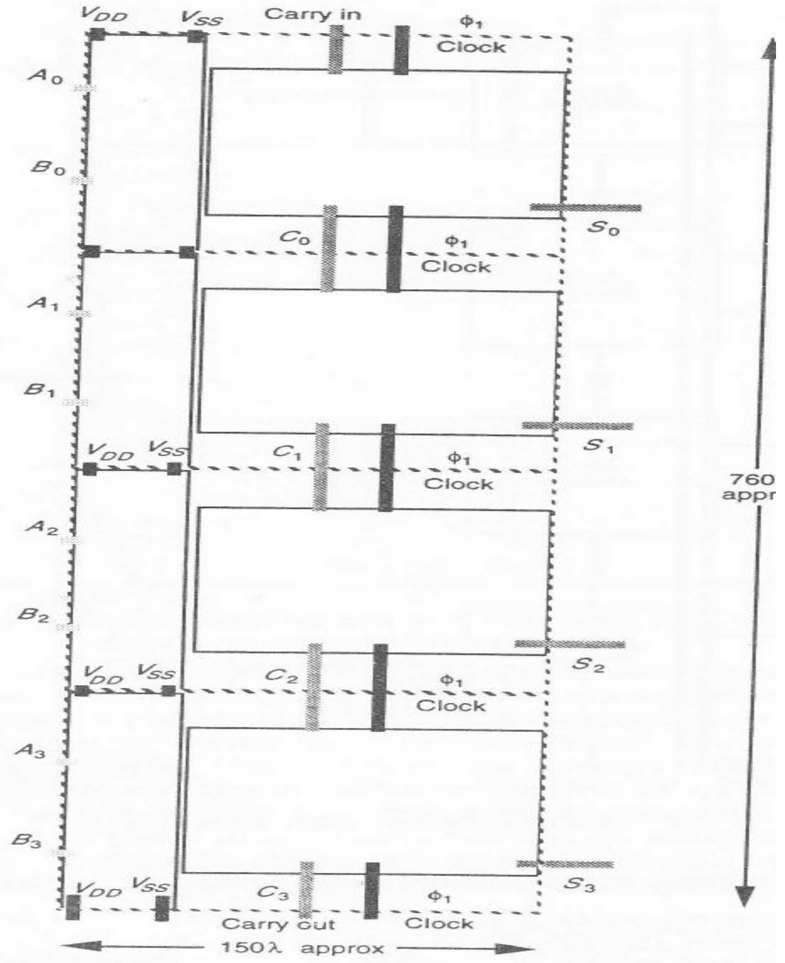


FIGURE 8.11(b) 4-bit adder outline.

Sum $S_k = \overline{H}_k C_{k-1} + H_k \overline{C}_{k-1}$

New carry $C_k = A_k B_k + H_k C_{k-1}$

where Half sum $H_k = \overline{A}_k B_k + A_k \overline{B}_k$

Consider, first, the *Sum* output if C_{k-1} is held at logical 0, then

$$S_k = H_k .1 + \overline{H}_k .0 = H_k$$

that is $S_k = H_k = A_k B_k + A_k B_k$—An *Exclusive-Or* operation

Now, hold C_{k-1} at logical 1, then

$$S_k = H_k .0 + \overline{H}_k .1 = \overline{H}_k$$

that is

$S_k = \overline{H}_k = \overline{A}_k \overline{B}_k + A_k B_k$—An *Exclusive-Nor* (*Equality*) operation

Next, consider the *carry* output of each element, first if C_{k-1} is held at logical 0.

Then

$C_k = A_k.B_k + H_k.0 = A_k.B_k$— An *And* operation

Now, if C_{k-1} is held at logical 1, then

$$C_k = A_k.B_k + H_k.1 = A_k.B_k + \overline{A}_k .B_k + A_k. \overline{B}_k$$

Therefore

$C_k = A_k. + B_k$ — An *Or* operation

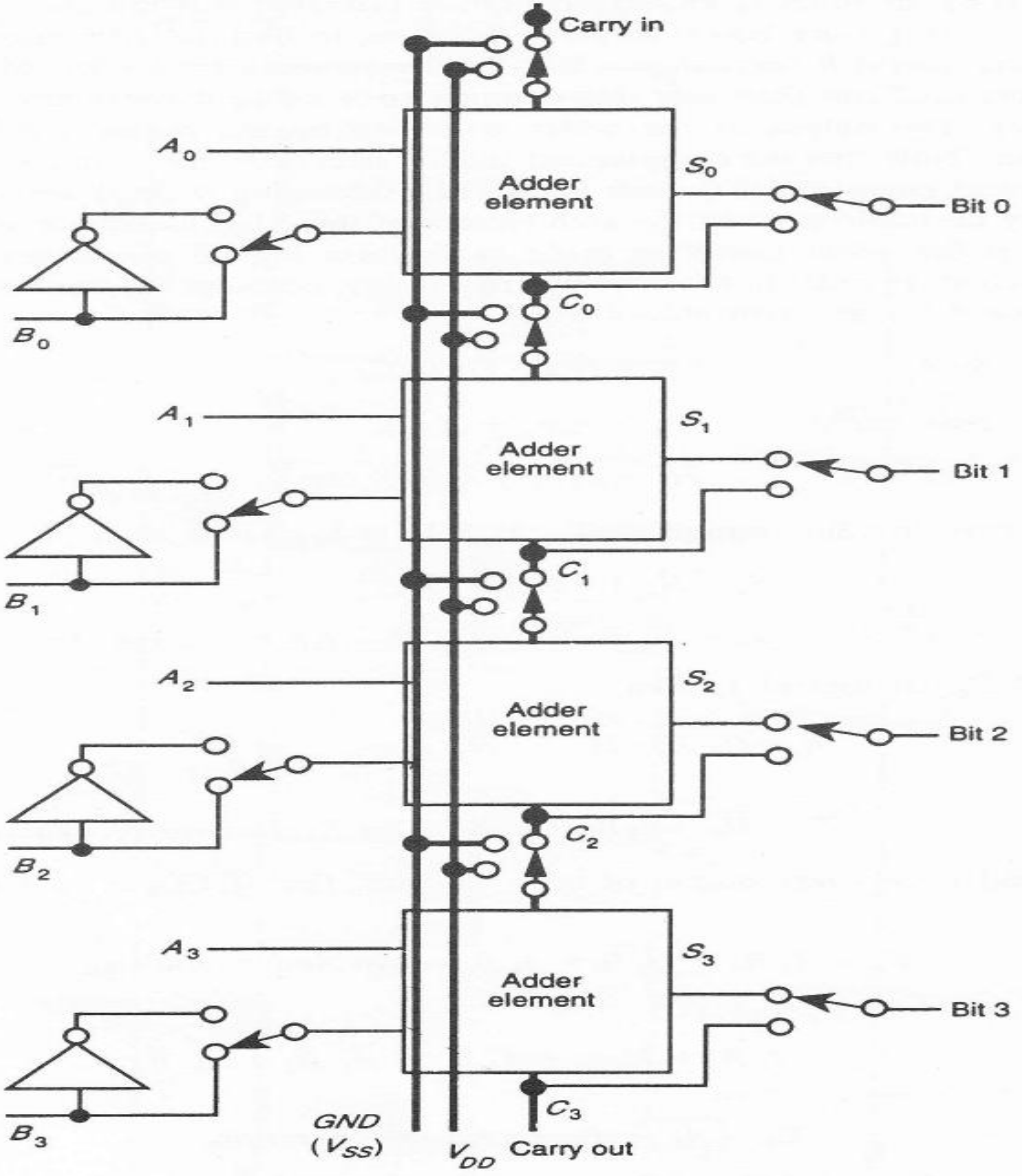


FIGURE 8.12 4-bit ALU.

In or... commonly used an... repeated here for convenience.

Sum $S_k = \overline{H}_k C_{k-1} + H_k \overline{C}_{k-1}$

New carry $C_k = A_k B_k + H_k C_{k-1}$

where

Half sum $H_k = \overline{A}_k B_k + A_k \overline{B}_k$

The expressions may also make use of lowercase letters. New carry may also be expressed in terms of the previous carry c_{k-1} with a *propagate* signal p_k and *generate* signal g_k, where

$$p_k(=H_k) = a_k \oplus b_k \text{ and } g_k = a_k . b_k$$

Then we may write,

new carry $c_k = p_k \cdot c_{k-1} + g_k$

$c_k = (a_k + b_k)c_{k-1} + a_k.b_k$

or

and sum $s_k = a_k \oplus b_k \oplus c_{k-1}$

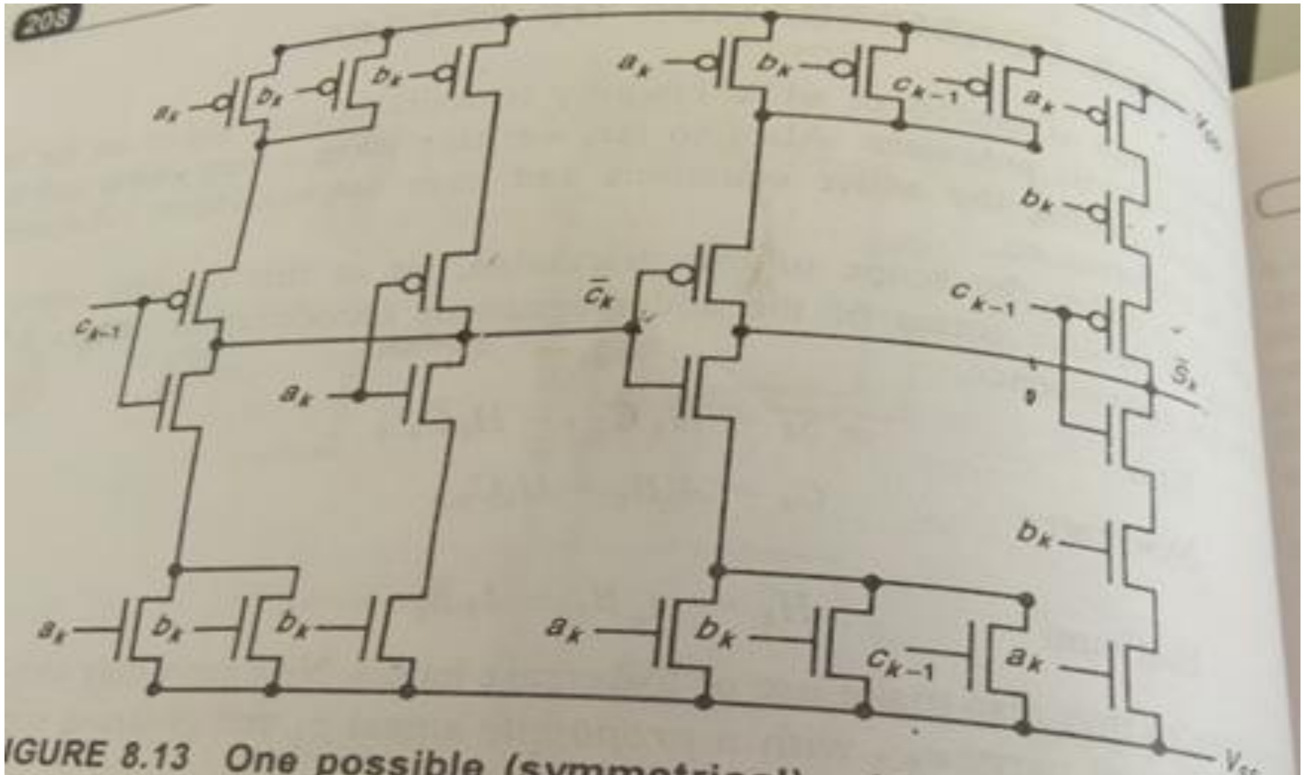
The sum may also be expressed in terms of the carry in c_{k-1} and carry out signals c_k together with the input bits a_k and b_k as follows:

$$s_k = \overline{c}_k .(a_k + b_k + c_{k-1.}) + a_k.b_k.c_{k-1}$$

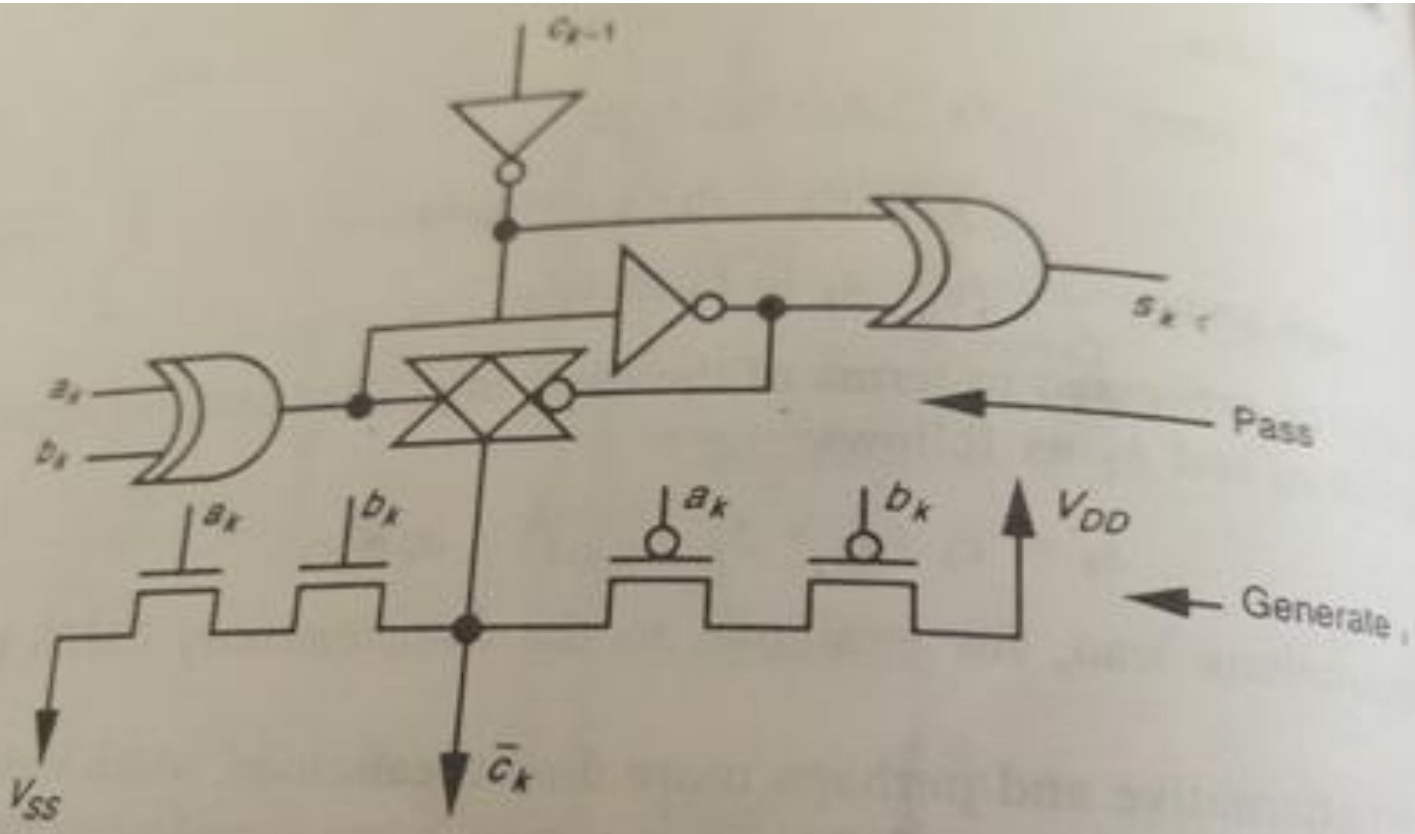
Such manipulations lead, for example, to the complementary CMOS logic circuit in Figure 8.13.

However, an alternative and perhaps more direct realization, which leads to the concept of a carry chain, is set out in Figure 8.14. This in turn, when considering carry circuits alone, leads to a popular arrangement known as the *Manchester carry-chain*.

Manchester Carry Chain

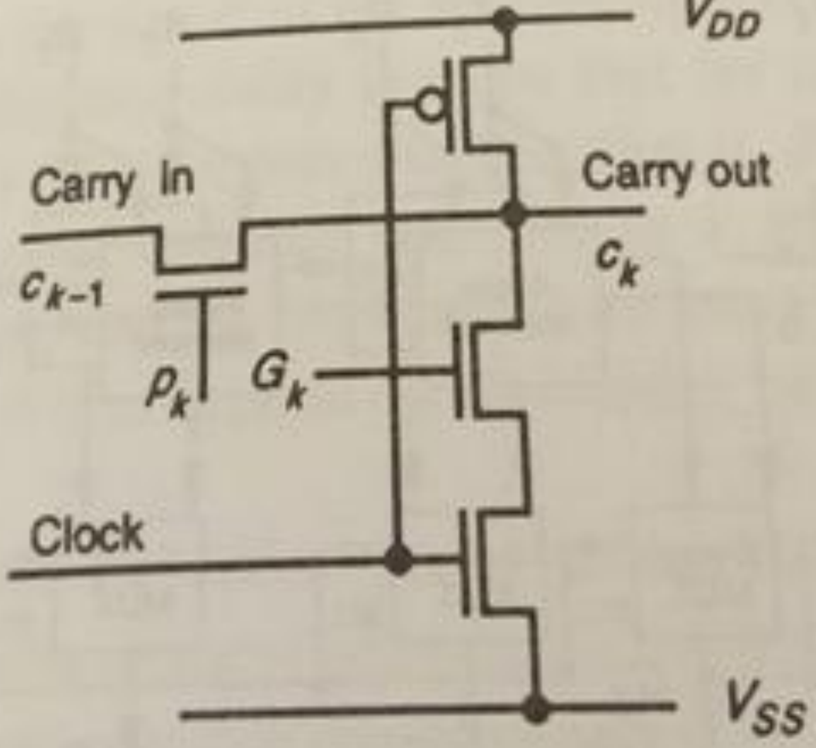


IGURE 8.13 One possible (symmetrical) adder cell arrangement.



URE 8.14 **An adder element based on the pass/generate concept.**

OS technology it in

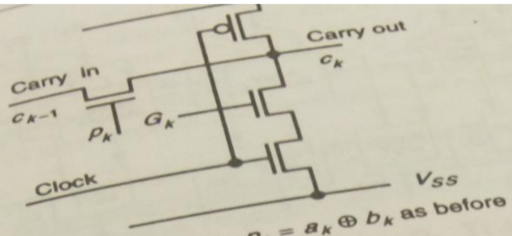


Note in this case, $p_k = a_k \oplus b_k$ as before

but $G_k = \bar{a}_k . \bar{b}_k$

FIGURE 8.15 Manchester carry-chain element.

Illustration



Note in this case, $P_k = a_k \oplus b_k$ as before
but $G_k = \overline{a}_k . \overline{b}_k$

FIGURE 8.15 **Manchester carry-chain element.**

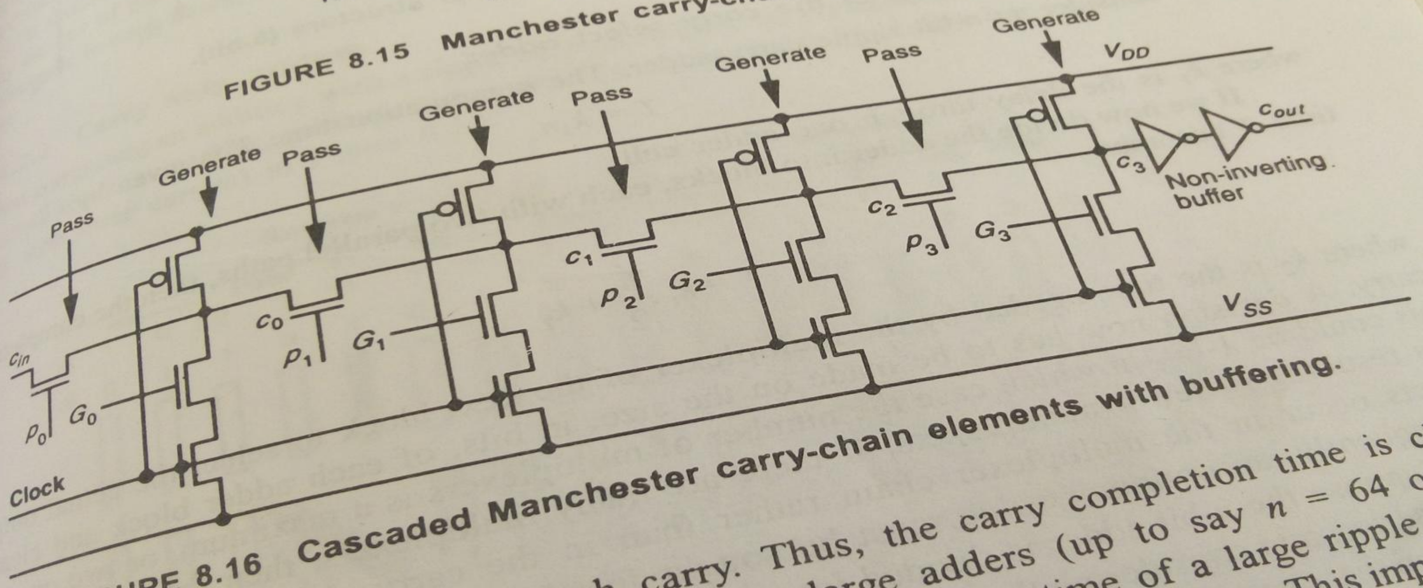


FIGURE 8.16 **Cascaded Manchester carry-chain elements with buffering.**

simple hardware of the ripple through carry. Thus, the carry completion time is clearly
proportional to n. On the other hand, large adders (up to say n = 64 or even
afford to wait for the long completion time of a large ripple through
must be adopted to improve addition time. This improvement
mplexity and, in consequence, at the expense of
three techniques for effecting faster
area/performance ratio.

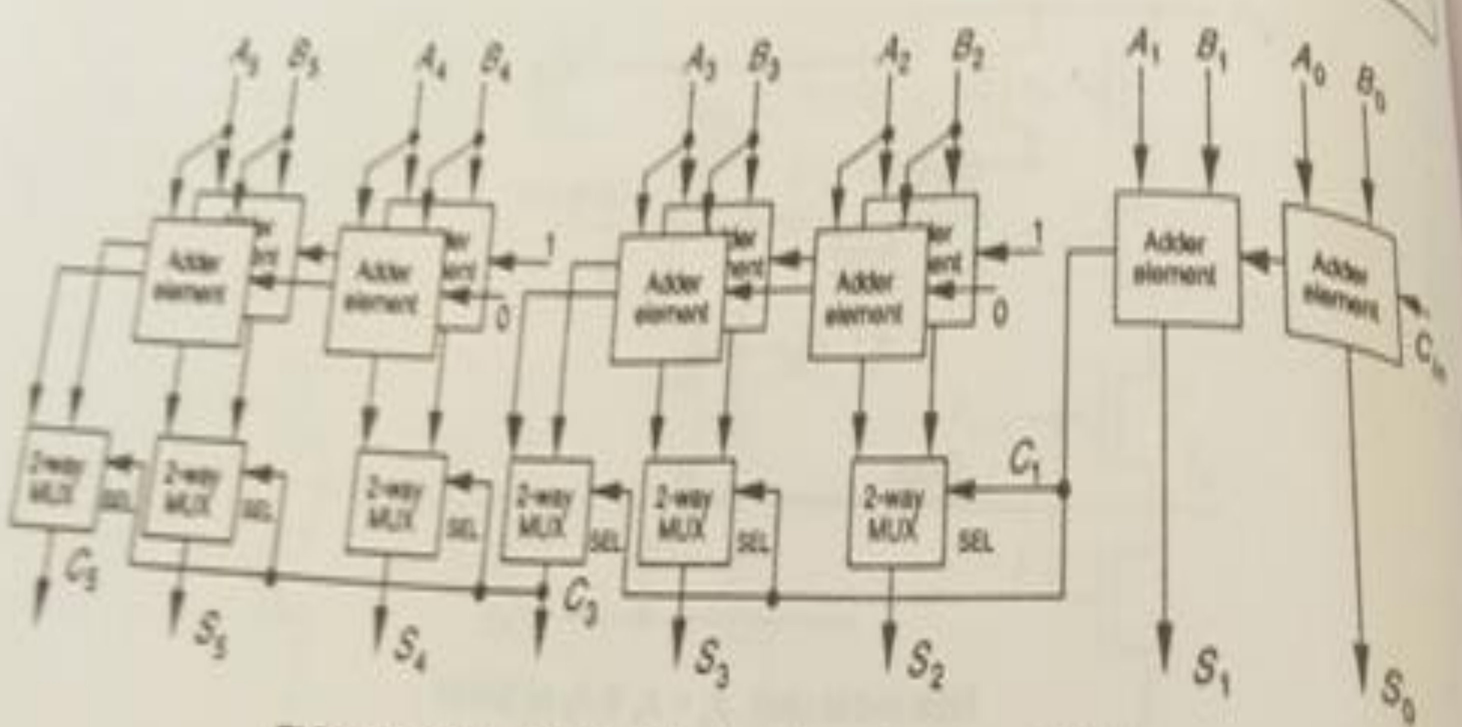
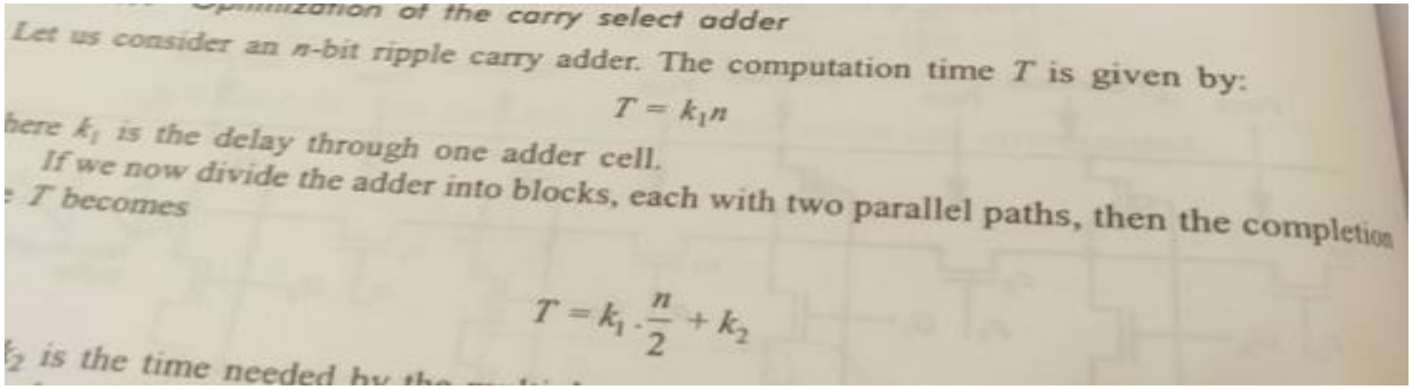


FIGURE 8.17 Carry select adder structure (6-bit).



imization of the carry select adder

Let us consider an n-bit ripple carry adder. The computation time T is given by:

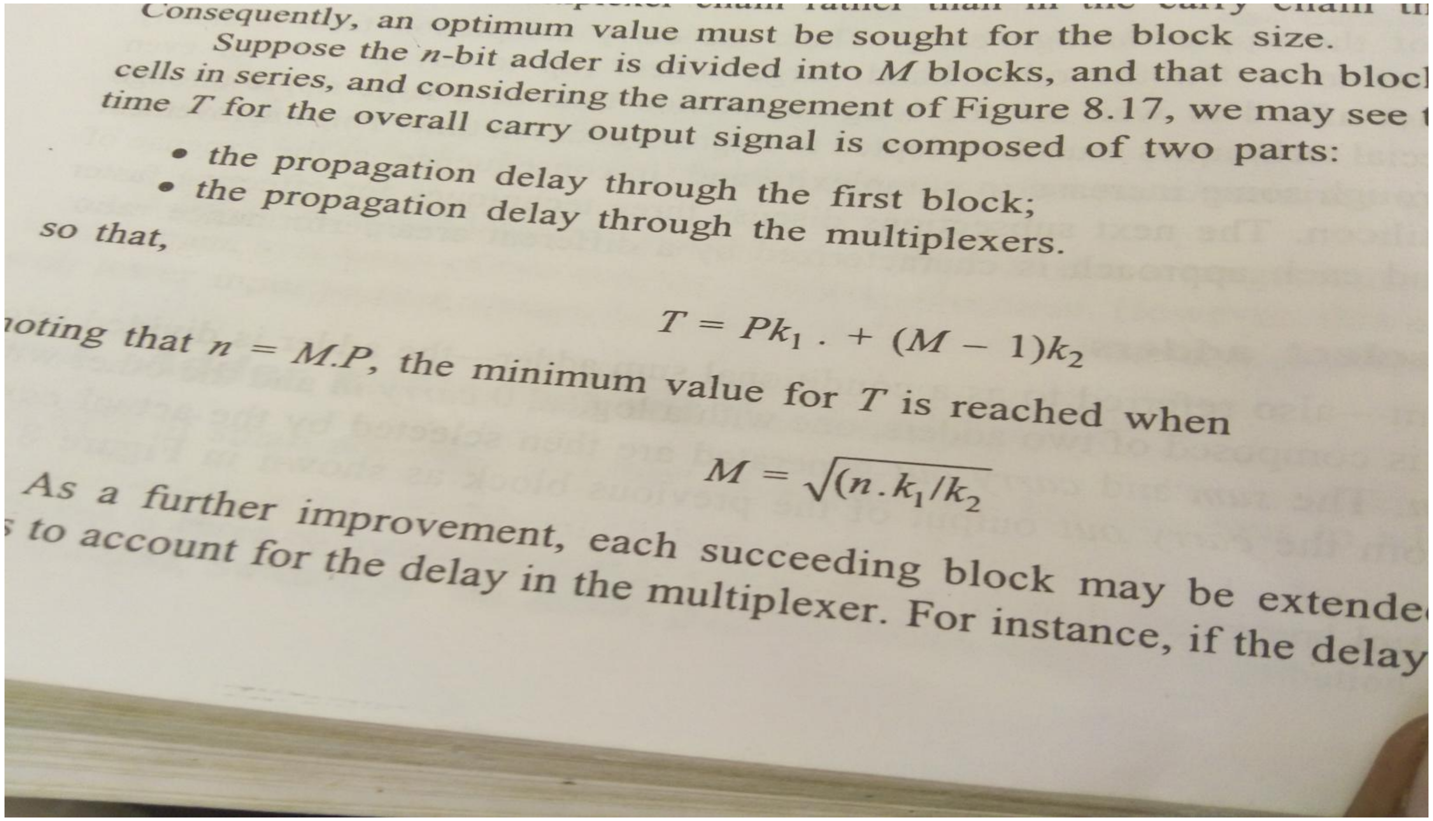
$$T = k_1 n$$

here k_1 is the delay through one adder cell.

If we now divide the adder into blocks, each with two parallel paths, then the completion
T becomes

$$T = k_1 . \frac{n}{2} + k_2$$

$_2$ is the time needed by th



[illegible]
Consequently, an optimum value must be sought for the block size.
Suppose the n-bit adder is divided into M blocks, and that each bloc
cells in series, and considering the arrangement of Figure 8.17, we may see
time T for the overall carry output signal is composed of two parts:

- the propagation delay through the first block;
- the propagation delay through the multiplexers.

so that,

$$T = Pk_1 . + (M - 1)k_2$$

noting that $n = M.P$, the minimum value for T is reached when

$$M = \sqrt{(n.k_1/k_2}$$

As a further improvement, each succeeding block may be extende
s to account for the delay in the multiplexer. For instance, if the delay

en by:

the completion

tual output
nd clearly
or more
through
blocks.

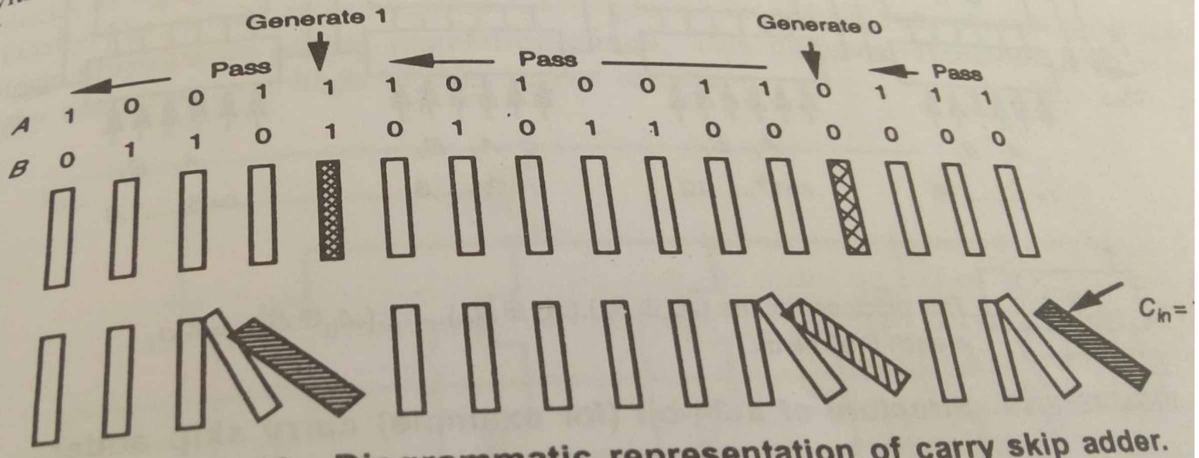
adder
letion

one b... depends on ... the multiplexer de... puts, that is, on P, increasing ... allow for as P increases.

It should also be noted that the adder blocks do not have to be ripple carry adders but may use any of the available enhancement techniques, such as carry look-ahead or carry skip techniques. In such cases, the optimization requirements may be different from those discussed here.

8.4.2.2 Carry skip adders

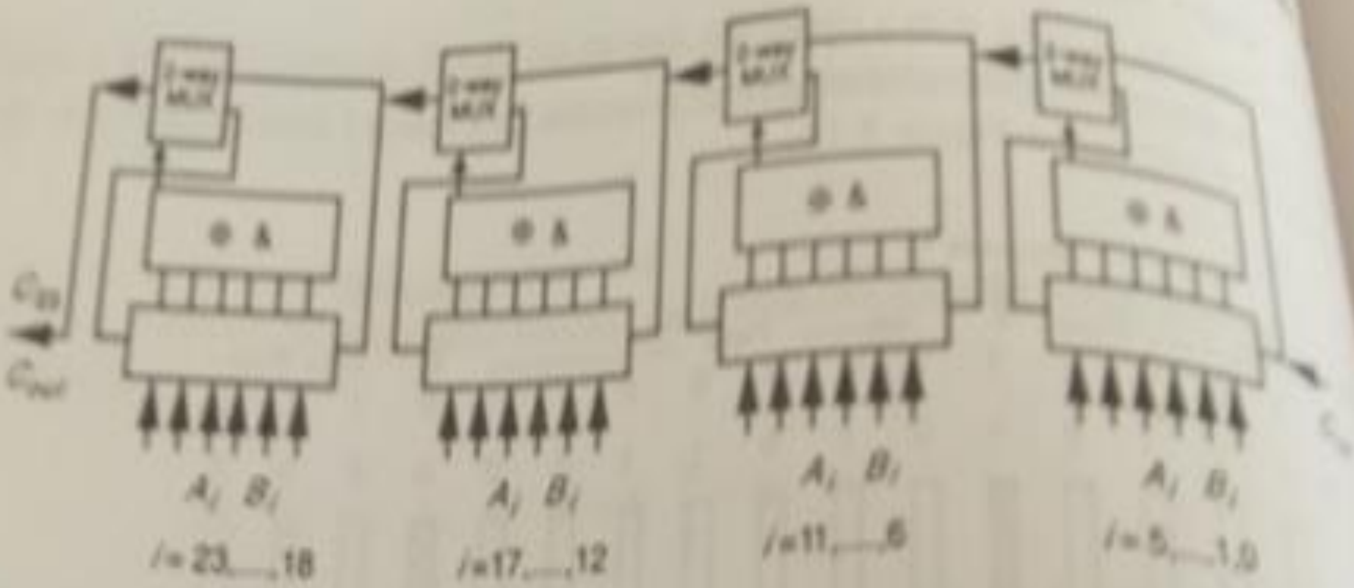
When computing an addition with a ripple through adder, the completion time will sometimes be small since the carries, generated at several positions, are formed simultaneously as shown (e.g. with three carries) in Figure 8.18.

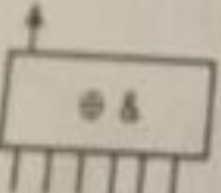


Generate 1
Generate 0
Pass
Pass
Pass
A 1 0 0 1 1 1 0 1 0 0 1 1 0 1 1 1
B 0 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0
C_{in}=

FIGURE 8.18 Diagrammatic representation of carry skip adder.

In this case, the carry propagation may be likened to the domino principle, falls, then each successive stage is knocked over in turn up to the next poi different carry is formed. in = 1, three simultaneous c





This block computes $(A_5 \oplus B_5).(A_4 \oplus B_4) \ldots (A_0 \oplus B_0). = \pi p_i$ in each 6-bit block.

FIGURE 8.19 **Structure of a 24-bit (for example) carry skip adder.**

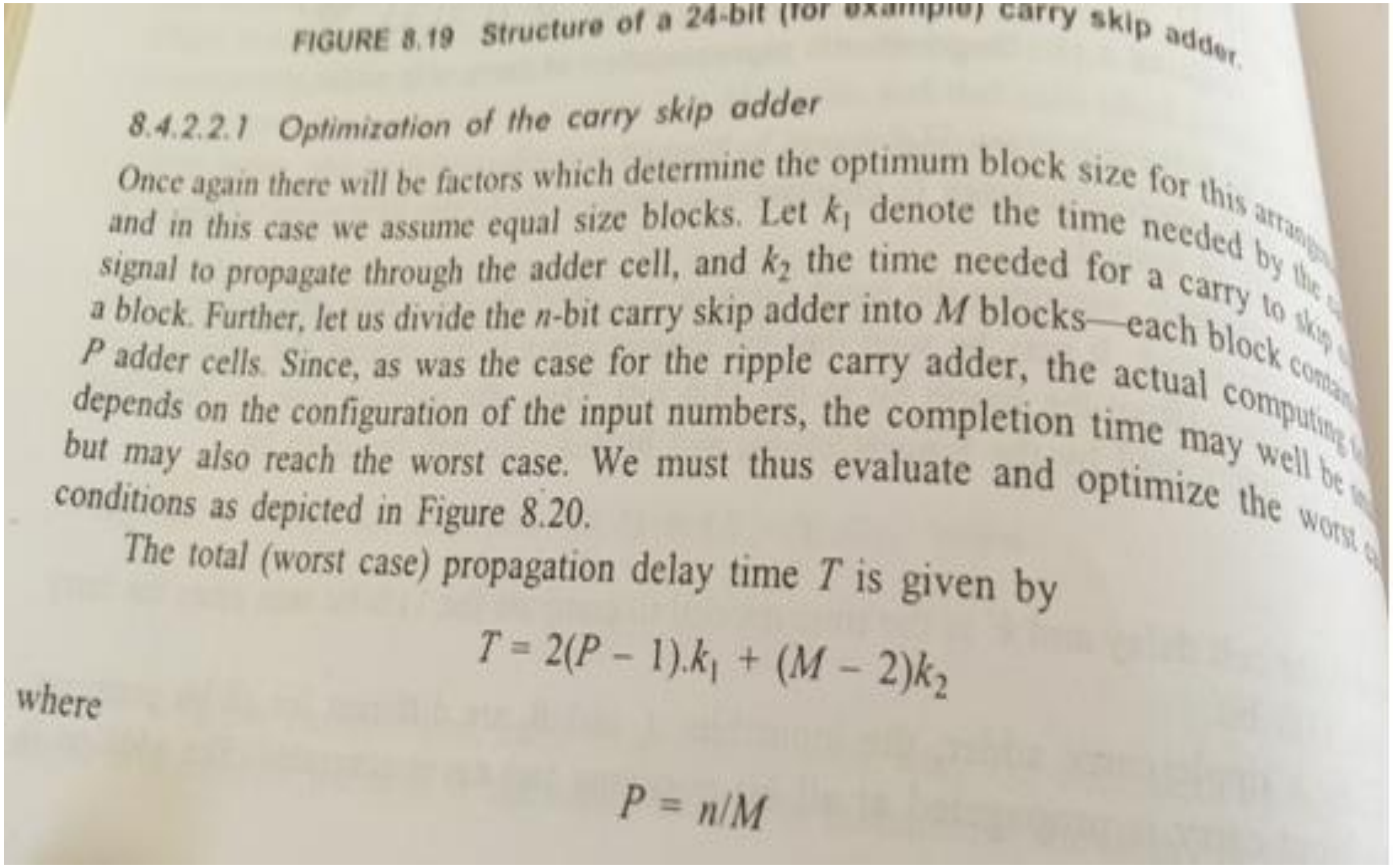


FIGURE 8.19 Structure of a 24-bit (for example) carry skip adder.

8.4.2.2.1 *Optimization of the carry skip adder*

Once again there will be factors which determine the optimum block size for this arrang and in this case we assume equal size blocks. Let k_1 denote the time needed by the signal to propagate through the adder cell, and k_2 the time needed for a carry to skip a block. Further, let us divide the n-bit carry skip adder into M blocks—each block contain P adder cells. Since, as was the case for the ripple carry adder, the actual computing depends on the configuration of the input numbers, the completion time may well be but may also reach the worst case. We must thus evaluate and optimize the worst c conditions as depicted in Figure 8.20.

The total (worst case) propagation delay time T is given by

$$T = 2(P - 1).k_1 + (M - 2)k_2$$

where

$$P = n/M$$

noting that $H_k = \overline{A_k}B_k + A_k\overline{B_k}$ the expression can be rearranged into the form

$$C_k = A_k.B_k + (A_k \oplus B_k).C_{k-1}$$

Thus for C_0 we may write

$$C_0 = A_0.B_0 + (A_0 \oplus B_0)C_{in}$$

which allows for an input carry; and, therefore

$$C_1 = A_1.B_1 + (A_1 \oplus B_1)C_0$$

may then be written as

$$C_1 = A_1.B_1 + (A_1. \oplus B_1).A_0.B_0. + (A_1. \oplus B_1.).(A_0 + B_0).C_{in}$$

and, similarly

$$C_2. = A_2.B_2. + (A_2 + B_2).A_1.B_1 + (A_2. + B_2).(A_1 + B_1).A_0.B_0. + (A_2 + B_2).(A_1 + B_1).(A_0 + B_0).C_{in}$$

The next stage would be

$$C_3 = A_3.B_3 + (A_3 + B_3).A_2.B_2 + (A_3 + B_3).(A_2 + B_2).A_1.B_1 + (A_3 + B_3).(A_2 + B_2).(A_1 + B_1).A_0.B_0 + (A_3 + B_3).(A_2 + B_2).(A_1. + B_1).(A_0 + B_0).C_{in}$$

so on for further stages.

f there is no input carry, then C_{in} becomes 0 and the last term in each expression for
vill be eliminated.

hough these expressions become very lengthy as the bit signif
n is only three logic levels deep, so th
of bit position. How

$$).(A_0 + B_0).C_{in}$$

and so on for further stages.

If there is no input carry, then C_{in} becomes 0 and the last term in each expression for carry will be eliminated.

Although these expressions become very lengthy as the bit significance increases, each expression is only three logic levels deep, so the delay in forming the carry is constant irrespective of bit position. However, the logic does rapidly become over-cumbersome and also presents problems in 'fan-out' and 'fan-in' requirements on the gates used. A compromise, usually adopted, is a combination of 'carry look-ahead' and 'ripple through' as indicated in Figure 8.22. The 3-bit groups shown were arbitrarily chosen to illustrate the approach.

Following this particular approach, we may now write carry look-ahead expressions in terms of the generate g_k and propagate p_k signals defined earlier. The general form for the carry signal c_k thus becomes

$$c_k = g_k + p_k \cdot g_{k-1} + p_k \cdot p_{k-1} g_{k-2} + \ldots\ldots + p_k \ldots\ldots p_1 \cdot g_0 + p_k \ldots\ldots p_0 \cdot c_{in}$$

Considering a CLA-based adder divided into blocks of 4-bits, as in Figure 8.23, we may write the expressions for the carry circuits in one block as follows:

$$c_0 = g_0 + p_0 \cdot c_{in}$$

$$c_1 = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_{in}$$

$$c_2 = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_{in}$$

$$c_3 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_{in}$$

and so on for further stages. $\ldots).(A_0 + B_0).C_{in}$

If there is no input carry, then C_{in} becomes 0 and the last term in each expression carry will be eliminated.

Although these expressions become very lengthy as the bit significance increases, each expression is only three logic levels deep, so the delay in forming the carry is constant irrespective of bit position. However, the logic does rapidly become over-cumbersome and also presents problems in 'fan-out' and 'fan-in' requirements on the gates used. A compromise, usually adopted, is a combination of 'carry look-ahead' and 'ripple through' as indicated in Figure 8.22. The 3-bit groups shown were arbitrarily chosen to illustrate the approach.

Following this particular approach, we may now write carry look-ahead expressions in terms of the generate g_k and propagate p_k signals defined earlier. The general form for the carry signal c_k thus becomes

$$c_k = g_k + p_k \cdot g_{k-1} + p_k \cdot p_{k-1} g_{k-2} + \ldots\ldots + p_k \ldots\ldots p_1 \cdot g_0 + p_k \ldots\ldots p_0 \cdot c_{in}$$

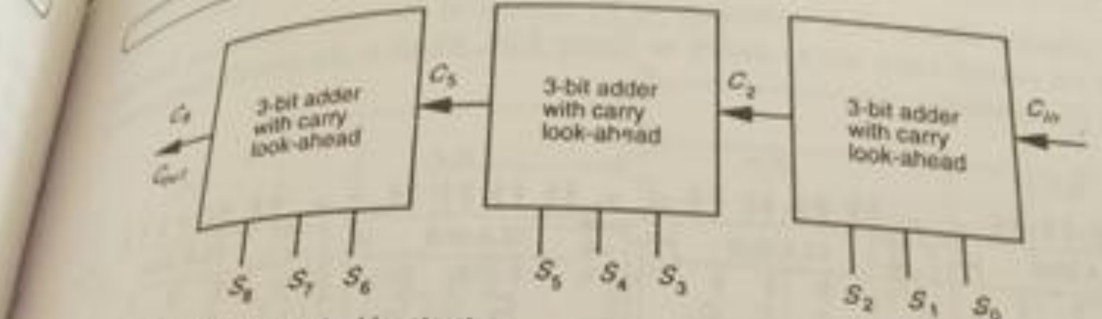
Considering a CLA-based adder divided into blocks of 4-bits, as in Figure 8.23, we may the expressions for the carry circuits in one block as follows:

$$c_0 = g_0 + p_0 \cdot c_{in}$$

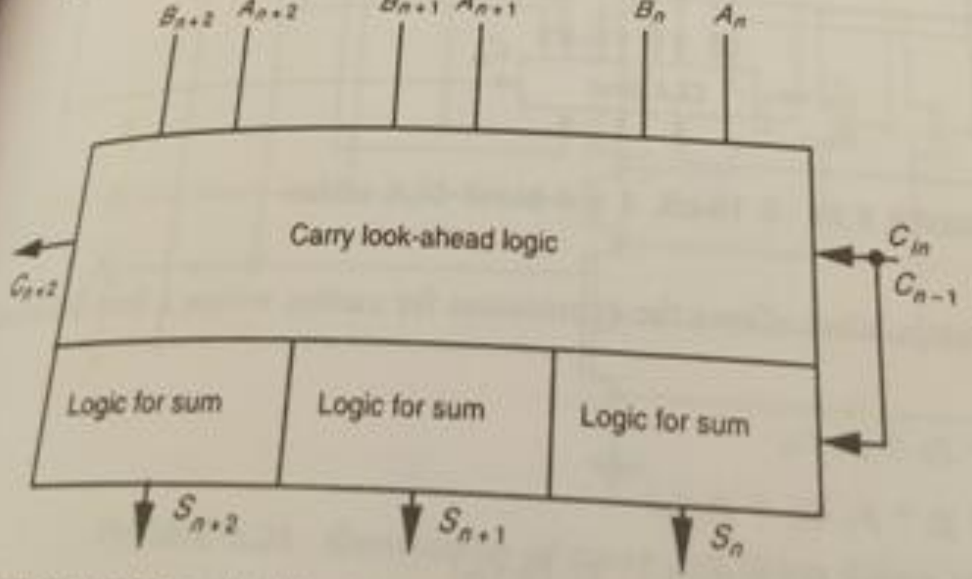
$$c_1 = g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_{in}$$

$$c_2 = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_{in}$$

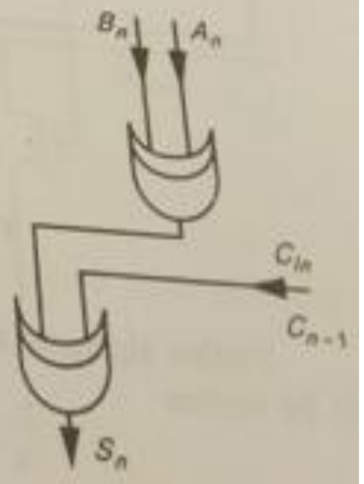
$$c_3 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0 + p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_{in}$$


3-bit adder with carry look-ahead
3-bit adder with carry look-ahead
3-bit adder with carry look-ahead
C_8
C_{out}
C_5
C_2
C_{in}
S_8 S_7 S_6
S_5 S_4 S_3
S_2 S_1 S_0

(a) Partial carry look-ahead adder structure

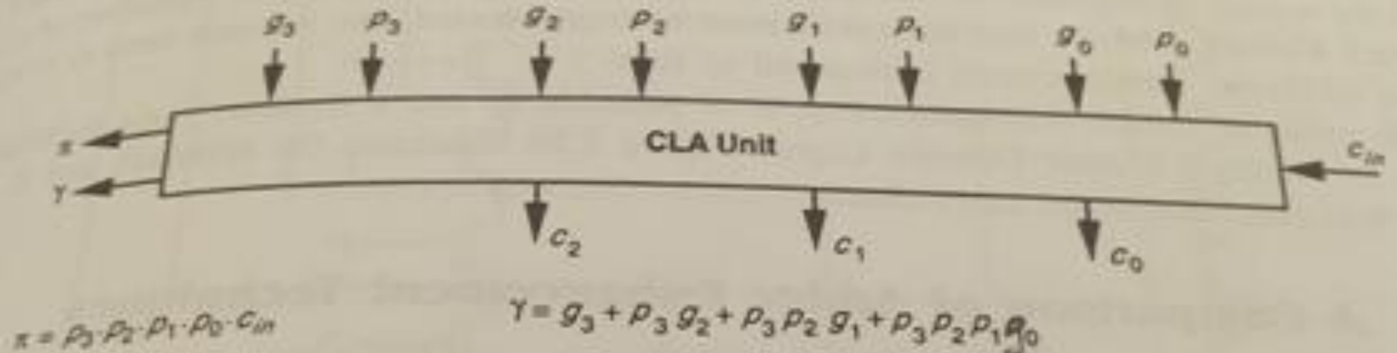


(b) Basic 3-bit adder cell with look-ahead



(c) Logic for sum

FIGU... ...through compromise.



g_3
p_3
g_2
p_2
g_1
p_1
g_0
p_0
CLA Unit
c_{in}
π
γ
c_2
c_1
c_0
$\pi = p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_{in}$
$\gamma = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

FIGURE 8.23 4-bit block CLA unit.

In order to avoid a sequential propagation of carry signals between the blocks, we may rate additional signals π and γ such that

$$\pi = p_3 \cdot p_2 \cdot p_1 \cdot p_0 \cdot c_{in} \text{ and } \gamma = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot g_0$$

An important property of these signals is that c_3, the *carry out* of the block, is

$$c_3 = \gamma + \pi$$

This concept allows CLA techniques to be applied to the carry generation betw blocks and for overall carry out as shown in Figure 8.24, which is the overall arrangem of a 16-bit CLA adder.

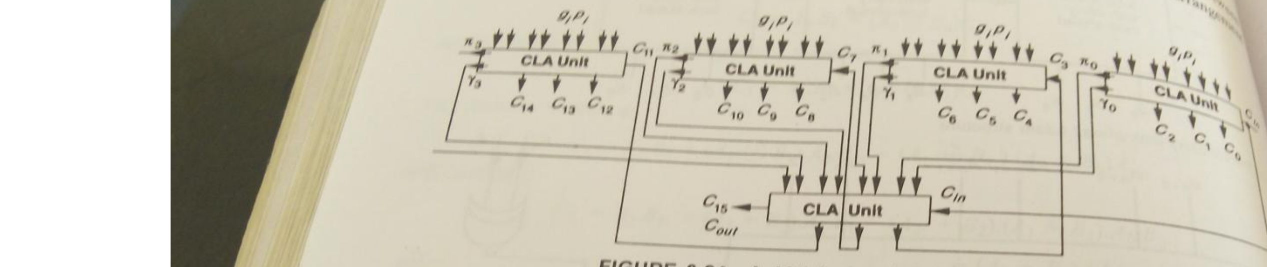


FIGURE 8.24 A 16-bit, 4 × 4 block CLA adder.

Further algebraic manipulation allows the expressions for carries within a four-bit block to be written

$$c_0 = g_0 + p_0 \cdot c_{in}$$
$$c_1 = g_1 + p_1 \cdot (g_0 + p_0 c_{in})$$
$$c_2 = g_2 + p_2 \cdot (g_1 + p_1 \cdot g_0 + p_1 \cdot p_0 \cdot c_{in})$$
$$c_3 = g_3 + p_3 \cdot (g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot g_0 + p_2 \cdot p_1 \cdot p_0 \cdot c_{in})$$

When implementing these circuits in silicon, each carry may be formed by one simple and very regular arrangement as indicated by Figure 8.25, which shows the formation of c_3. For each 4-bit CLA block, four such cells must be implemented, one for each carry c_0 to c_3, and an additional similar circuit is required to form γ.

In order to reduce this complexity, it is possible to use a dynamic logic technique known as 'Multiple Output Domino Logic'. Figure 8.26 illustrates the approach and is in fact, a four-cell Manchester carry-chain.