# 5 Scaling of MOS circuits

*Little things are pretty.*

Proverb

*Good things come in small packages.*

Proverb

## Objectives

**VLSI fabrication technology is still in the process of evolution which is leading to smaller line widths and feature size and to higher packing density of circuitry on a chip.**

**The scaling down of feature size generally leads to improved performance and it is important therefore to understand the effects of scaling. There are also future limits to scaling down which may well be reached in the next decade.**

**Although this chapter may be seen by some to interrupt the flow of the text toward actual VLSI design, the authors considered this an appropriate topic following the previous chapters dealing with basic parameters and characteristics which, of course, are all affected by scaling.**

Microelectronic technology may be characterized in terms of several indicators, or figures of merit. Commonly, the following are used:

- Minimum feature size
- Number of gates on one chip
- Power dissipation
- Maximum operational frequency
- Die size
- Production cost.

Many of these figures of merit can be improved by shrinking the dimensions of transistors, interconnections and the separation between features, and by adjusting the doping levels and supply voltages. Accordingly, over the past decade, much effort has been directed toward the upgrading of process technology and the resultant scaling down of devices and feature size.

In the design processes postulated by Mead and Conway and used for most examples in this text, it has been the practice to dimension all layouts in terms of $\lambda$. A value may then be allocated to $\lambda$, prior to manufacture, which is in line with the capabilities of the silicon foundry or is determined by current technology and/or meets the specifications which have been set out for the circuit. One benefit of this approach lies in the fact that the design rules have been formulated in such a way as to allow *limited* direct scaling of the dimensions of circuits, so that today's design is not automatically outdated when line widths are reduced (i.e. the value allocated to $\lambda$ is reduced) by advances in tomorrow's technology.

Scaling is therefore an important factor, and it is essential for the designer to understand the implementation and the effects of scaling. In writing this chapter, the authors gratefully acknowledge the useful contributions made by Dr A. Osserain and Dr B. Hochet, both of the Swiss Federal Institute of Technology, Lausanne, Switzerland.

This chapter discusses scaling and its effect on performance and indicates some problems and ultimate limitations.

## 5.1 Scaling models and scaling factors

The most commonly used models are the constant electric field scaling model and the constant voltage scaling model. They both present a simplified view, taking only first degree effects into consideration, but are easily understood and well suited to educational needs. Recently, a combined voltage and dimension scaling model has been presented (Bergmann, 1991).

In this chapter, the application of each of the three models will be illustrated. To assist in visualization, it is useful to refer to Figure 5–1 which indicates the

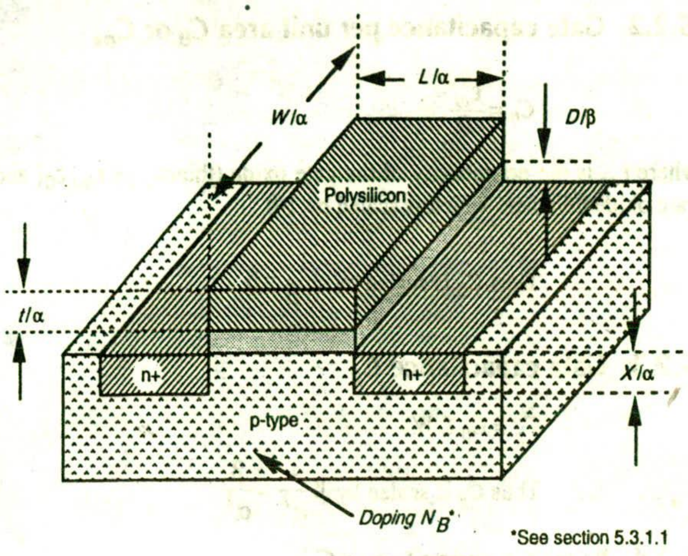


**Figure 5–1** Scaled nMOS transistor (pMOS similar)

device dimensions and substrate doping level which are associated with the scaling of a transistor.

In order to accommodate the three models, two scaling factors — $1/\alpha$ and $1/\beta$ — are used. $1/\beta$ is chosen as the scaling factor for supply voltage $V_{DD}$ and gate oxide thickness $D$, and $1/\alpha$ is used for all other linear dimensions, both vertical and horizontal to the chip surface. For the constant field model and the constant voltage model, $\beta = \alpha$ and $\beta = 1$ respectively are applied.

## 5.2 Scaling factors for device parameters

In this section, simple derivations and calculations reveal the effects of scaling.

### 5.2.1 Gate area $A_g$

$$A_g = L.W.$$

where $L$ and $W$ are the channel length and width respectively. Both are scaled by $1/\alpha$.

Thus $A_g$ is scaled by $1/\alpha^2$

### 5.2.2 Gate capacitance per unit area $C_0$ or $C_{ox}$

$$C_0 = \frac{\varepsilon_{ox}}{D}$$

where $\varepsilon_{ox}$ is the permittivity of the gate oxide (thinox) [$= \varepsilon_{ins}.\varepsilon_0$] and $D$ is the gate oxide thickness which is scaled by $1/\beta$

$$\text{Thus } C_0 \text{ is scaled by } \frac{1}{1/\beta} = \beta$$

### 5.2.3 Gate capacitance $C_g$

$$C_g = C_0 L.W.$$

$$\text{Thus } C_g \text{ is scaled by } \beta \frac{1}{\alpha^2} = \frac{\beta}{\alpha^2}$$

### 5.2.4 Parasitic capacitance $C_x$

$$C_x \text{ is proportional to } \frac{A_x}{d}$$

where $d$ is the depletion width around source or drain which is scaled by $1/\alpha$, and $A_x$ is the area of the depletion region around source or drain which is scaled by $1/\alpha^2$.

$$\text{Thus } C_x \text{ is scaled by } \frac{1}{\alpha^2} \cdot \frac{1}{1/\alpha} = \frac{1}{\alpha}$$

### 5.2.5 Carrier density in channel $Q_{on}$

$$Q_{on} = C_0 . V_{gs}$$

where $Q_{on}$ is the average charge per unit area in the channel in the 'on' state. Note that $C_0$ is scaled by $\beta$ and $V_{gs}$ is scaled by $1/\beta$.

Thus $Q_{on}$ is scaled by 1

### 5.2.6 Channel resistance $R_{on}$

$$R_{on} = \frac{L}{W} \frac{1}{Q_{on}\mu}$$

where $\mu$ is the carrier mobility in the channel and is assumed constant.

$$\text{Thus } R_{on} \text{ is scaled by } \frac{1}{\alpha}\frac{1}{1/\alpha}1 = 1$$

### 5.2.7 Gate delay $T_d$

$$T_d \text{ is proportional to } R_{on}.C_g$$

$$\text{Thus } T_d \text{ is scaled by } \frac{1.\beta}{\alpha^2} = \frac{\beta}{\alpha^2}$$

### 5.2.8 Maximum operating frequency $f_0$

$$f_0 = \frac{W}{L}\frac{\mu C_0 V_{DD}}{C_g}$$

or, $f_0$ is inversely proportional to delay $T_d$.

$$\text{Thus } f_0 \text{ is scaled by } \frac{1}{\beta/\alpha^2} = \frac{\alpha^2}{\beta}$$

### 5.2.9 Saturation current $I_{dss}$

$$I_{dss} = \frac{C_0\mu}{2}\frac{W}{L}(V_{gs} - V_t)^2$$

noting that both $V_{gs}$ and $V_t$ are scaled by $1/\beta$, we have

$$I_{dss} \text{ is scaled by } \beta\,(1/\beta)^2 = 1/\beta$$

Current density ↑

### 5.2.10 Current density $J$

$$J = \frac{I_{dss}}{A}$$

where $A$ is the cross-sectional area of the channel in the 'on' state which is scaled by $1/\alpha^2$

$$\text{So, } J \text{ is scaled by } \frac{1/\beta}{1/\alpha^2} = \frac{\alpha^2}{\beta}$$

### 5.2.11 Switching energy per gate $E_g$

$$E_g = \frac{1C_g}{2}(V_{DD})^2$$

$$\text{So, } E_g \text{ is scaled by } \frac{\beta}{\alpha^2}\cdot\frac{1}{\beta^2} = \frac{1}{\alpha^2\beta}$$

### 5.2.12 Power dissipation per gate $P_g$

$P_g$ comprises two components such that

$$P_g = P_{gs} + P_{gd}$$

where the static component

$$P_{gs} = \frac{(V_{DD})^2}{R_{on}}$$

and the dynamic component

$$P_{gd} = E_g f_0$$

It will be seen that both $P_{gs}$ and $P_{gd}$ are scaled by $1/\beta^2$

So, $P_g$ is scaled by $1/\beta^2$

### 5.2.13 Power dissipation per unit area $P_a$

$$P_a = \frac{P_g}{A_g}$$

$$\text{So, } P_a \text{ is scaled by } \frac{1/\beta^2}{1/\alpha^2} = \alpha^2/\beta^2$$

### 5.2.14 Power-speed product $P_T$

$$P_T = P_g . T_d$$

$$\text{So, } P_T \text{ is scaled by } \frac{1}{\beta^2}\cdot\frac{\beta}{\alpha^2} = \frac{1}{\alpha^2\beta}$$

### 5.2.15 Summary of scaling effects

It is useful to summarize the scaling effects in a convenient form. Table 5–1 sets out scaling effect for various key parameters of MOS FET devices and for the three scaling models mentioned earlier.

**Table 5–1** Scaling effects

| *Parameters* | | *Combined V and D* | *Constant E* | *Constant V* |
|---|---|---|---|---|
| $V_{DD}$ | Supply voltage | $1/\beta$ | $1/\alpha$ | 1 |
| $L$ | Channel length | $1/\alpha$ | $1/\alpha$ | $1/\alpha$ |
| $W$ | Channel width | $1/\alpha$ | $1/\alpha$ | $1/\alpha$ |
| $D$ | Gate oxide thickness | $1/\beta$ | $1/\alpha$ | 1 |
| $A_g$ | Gate area | $1/\alpha^2$ | $1/\alpha^2$ | $1/\alpha^2$ |
| $C_0$ (or $C_{ox}$) | Gate C per unit area | $\beta$ | $\alpha$ | 1 |
| $C_g$ | Gate capacitance | $\beta/\alpha^2$ | $1/\alpha$ | $1/\alpha^2$ |
| $C_x$ | Parasitic capacitance | $1/\alpha$ | $1/\alpha$ | $1/\alpha$ |
| $Q_{on}$ | Carrier density | 1 | 1 | 1 |
| $R_{on}$ | Channel resistance | 1 | 1 | 1 |
| $I_{dss}$ | Saturation current | $1/\beta$ | $1/\alpha$ | 1 |
| $A_c$ | Conductor X-section area | $1/\alpha^2$ | $1/\alpha^2$ | $1/\alpha^2$ |
| $J$ | Current density | $\alpha^2/\beta$ | $\alpha$ | $\alpha^2$ |
| $V_g$ | Logic 1 level | $1/\beta$ | $1/\alpha$ | 1 |
| $E_g$ | Switching energy | $1/\alpha^2.\beta$ | $1/\alpha^3$ | $1/\alpha^2$ |
| $P_g$ | Power dispn per gate | $1/\beta^2$ | $1/\alpha^2$ | 1 |
| $N$ | Gates per unit area | $\alpha^2$ | $\alpha^2$ | $\alpha^2$ |
| $P_a$ | Power dispn per unit area | $\alpha^2/\beta^2$ | 1 | $\alpha^2$ |
| $T_d$ | Gate delay | $\beta/\alpha^2$ | $1/\alpha$ | $1/\alpha^2$ |
| $f_0$ | Max. operating frequency | $\alpha^2/\beta$ | $\alpha$ | $\alpha^2$ |
| $P_T$ | Power-speed product | $1/\alpha^2.\beta$ | $1/\alpha^3$ | $1/\alpha^2$ |

Constant E: $\beta = \alpha$; Constant V: $\beta = 1$

## 5.3 Some discussion on and limitations of scaling

Although scaling down does have many desirable effects, some of the associated effects may cause problems which eventually become severe enough to prevent further miniaturization.

### 5.3.1. Substrate doping

So far, in discussing the various effects, we have neglected the built-in (junction) potential $V_B$, which in turn depends on the substrate doping level, and this is acceptable so long as $V_B$ is small compared with $V_{DD}$. However, when this no longer holds, then the effects of $V_B$ must be included.

Furthermore, substrate doping impinges on many of the characteristics of transistors fabricated on it. Thus further discussion is warranted.

### 5.3.1.1 *Substrate doping scaling factors*

As the channel length of a MOS transistor is reduced, the depletion region widths must also be scaled down to prevent the source and drain depletion regions from meeting. Depletion region width $d$ for the junctions is given by

$$d = \sqrt{\frac{2\varepsilon_{si}\varepsilon_0 V}{qN_B}}$$

where

$\varepsilon_{si}$ = relative permittivity of silicon ($\doteqdot 12$)
$\varepsilon_0$ = permittivity of free space ( $= 8.85 \times 10^{-14}$ F/cm)
$V$ = effective voltage across the junction $\doteqdot V_a + V_B$
$q$ = electron charge
$N_B$ = doping level of substrate
$V_a$ (maximum value $= V_{DD}$) = applied voltage
$V_B$ = built-in (junction) potential

and

$$V_B = \frac{kT}{q}\ln\left(\frac{N_B N_D}{n_i^2}\right)$$

where $N_D$ is the source or drain doping, and $n_i$ is the intrinsic carrier concentration in silicon.

In, say, 5 μm technology, $V_B$ is in the region of 500 mV whilst applied voltage $V_a$ ($= V_{DD}$) is commonly 5 V so that $V_B$ may be neglected for scaling considerations. Under these circumstances,

$$d = \sqrt{\frac{2\varepsilon_{si}\varepsilon_0 V_{DD}}{qN_B}}$$

If $V_{DD}$ is scaled by $1/\beta$ and $d$ by $1/\alpha$, then $N_B$ can be scaled by $\alpha^2/\beta$ (Bergmann, 1991).

For some more recent technologies, $N_B$ is increased to reduce $d$ so that $V_B$ is also enlarged. (For example, if $N_B = 10^{15}$ cm$^{-3}$ and $N_D = 10^{20}$ cm$^{-3}$ then $V_B =$ 0.88 V). At the same time, $V_{DD}$ is also scaled down, and is thus no longer large compared with $V_B$ so that $V_B$ must be taken account of in scaling.

Thus, for the combined voltage and dimension scaling model applied to a transistor for which we have a known $V_a$, we may write

$$V_a = mV_B$$

where $m$ is a real number, so that

$$V = V_a + V_B = mV_B + V_B$$

Now if we scale $V_a$ by $1/\beta$ we have

$$V_s = \frac{mV_B}{\beta} + V_B \text{ so that scaling factor} = \frac{\beta + m}{\beta(m+1)}$$

where $V_s$ is the effective scaled voltage across the depletion region. Consequently, $N_B$ should be scaled by

$$\frac{\alpha^2(\beta+m)}{(m+1)}$$

so that $d$ scales by $1/\alpha$.

This model not only expresses the effects of the relationship between $V_a$ and $V_B$, but also shows their relation to the scaling factor $\beta$. Where $m$ is large and $\beta$ is small, the scaling factor for $N_B$ reverts to $\alpha^2/\beta$, but in other cases this model becomes significant.

### 5.3.1.2 *Depletion width*

In the previous discussion, $N_B$ is increased to reduce the depletion width, but this also increases the threshold voltage $V_t$ which is against the required trends for scaling down.

In [Hoen] $N_B$ must be kept below $1.3 \times 10^{19}\,\text{cm}^{-3}$. At higher values of $N_B$, the maximum electric field which can be applied to the gate oxide is insufficient to invert the substrate so that no channel can be formed.

However, the technology of deep channel implantation increases $N$ only near the source and drain to substrate junctions. Thus, $N_B$ can be maintained at a satisfactory level in the channel region and this problem is thus reduced. Nonetheless, depletion width $d$ and built-in potential $V_B$ will impose limitations on scaling.

It can be shown (Grove, 1967) that

$$E_{max} = \frac{2V}{d}$$

where $E_{max}$ is the maximum electric field induced in the one-sided step junction.

When $N_B$ is increased by $\alpha$ and if $V_a = 0$, then $V_B$ is increased by $\ln\alpha$ and $d$ is decreased by

$$\sqrt{\frac{\ln\alpha}{\alpha}}$$

Therefore, the electric field $E$ across the depletion region is increased by $\sqrt{\alpha/\ln\alpha}$ and will thus reach the critical level $E_{crit}$ with increasing $N_B$.

Figure 5–2(a) shows the depletion width $d$ as a function of substrate concentration $N_B$ and supply voltage $V_{DD}$. The dashed line indicates the maximum depletion width for $E_{max} = E_{crit}$. Substituting into the equation for $d$, we have

$$d = \sqrt{\frac{2\varepsilon_{si}\varepsilon_0}{qN_B}\left(\frac{E_{crit}.d}{2}\right)}$$

whence

$$d = \frac{\varepsilon_{si}\varepsilon_0(E_{crit})}{qN_B}$$

The area of Figure 5–2(a) above the dashed line is the region where the increased electric field will induce breakdown. Thus, the point at which the dashed line and the $V_a = 0$ line intersect indicates the maximum allowable substrate doping level, which is about $N_B = 3 \times 10^{17}\text{cm}^{-3}$ (for $N_D = 3 \times 10^{20}\text{cm}^{-3}$). At higher values of $N_B$ junction tunneling will occur. Therefore allowable values for $d$ fall below the dashed line and above the $V_a = 0$ line.

Figure 5–2(b) shows the maximum electric field in the depletion layer versus $N_B$. Any applied voltage greater than $V_a = 0$ will cause breakdown to occur at lower values of $N_B$.

In the foregoing discussions, the effects of $N_D$ have been assumed to be negligible.

## 5.3.2 Limits of miniaturization

The minimum size of a transistor is determined by both process technology and the physics of the device itself. The reduction of device geometry currently depends mainly on alignment accuracy and on the resolution of photolithographic technology; the limit on feature size is now at 0.3 μm, but the increasing availability of direct write E-beam technology will allow this limit to be further reduced.

The size of a transistor is usually defined in terms of its channel length $L$. As the channel length is scaled down, the edge of the depletion region around the source comes closer to that around the drain. In order to prevent punch-through and maintain transistor action, it can be shown that the channel length $L$ must be at least $2d$. Therefore. $L$ is in turn determined by the substrate concentration $N_B$ and supply voltage $V_{DD}$ (which determines $V_a$).

Applying the conclusions from the previous section, we may estimate the minimum possible channel length as 0.14 μm. The minimum transit time for an electron to travel from source to drain can also be calculated. From (Sze, 1985),

$$v_{drift} = \mu E$$

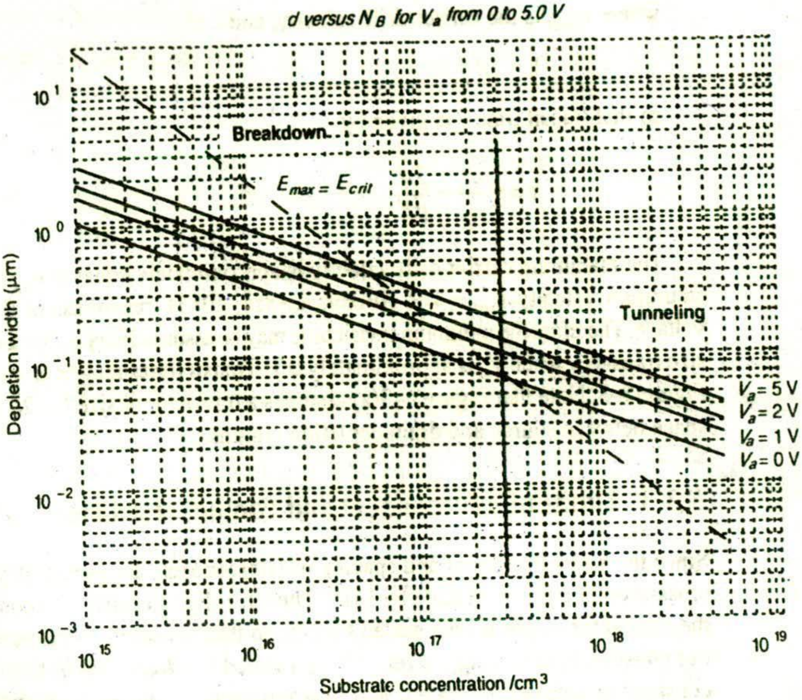


**Figure 5–2(a)** Substrate concentration/cm$^3$

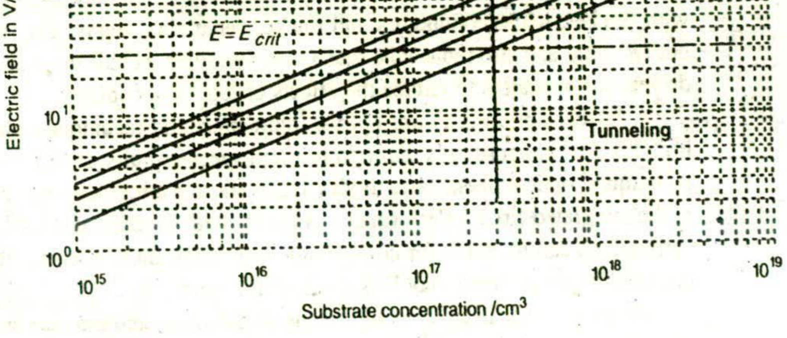


**Figure 5–2(b)** Depletion width *d* and electric field *E* versus substrate doping $N_B$

where $v_{drift}$ is the carrier drift velocity, and

$$L = 2d$$

so that transit time $\tau$ is given by

$$\tau = \frac{L}{v_{drift}} = \frac{2d}{\mu E}$$

The maximum carrier drift velocity is approximately equal to $v_{sat}$ where the saturation velocity $v_{sat} = 1 \times 10^7$cm/sec (Sze, 1985), regardless of the supply voltage. Therefore the minimum transit time may be assumed to occur for a minimum size transistor when $V_a$ is approximately 0 V. Transit times may be assessed from Figure 5–3. Note that Figure 5–3(a) assumes a transistor of size $L = 2d$ with zero space between source and drain depletion regions.

## 5.3.3 Limits of interconnect and contact resistance

Since the width, thickness and spacing of interconnects are each scaled by $1/\alpha$, cross-section areas must scale by $1/\alpha^2$. Thus, for short distance interconnections the conductor length is also scaled by $1/\alpha$, so that resistance is increased by $\alpha$. For constant field scaling, current $I$ is also scaled by $1/\alpha$ so that $IR$ drop remains constant as a device is scaled, and thus represents a higher proportion of the supply voltage $V_{DD}$ which is also scaled by $1/\alpha$. Thus driving capability and noise margins are degraded.

With decreasing device dimensions, we are also seeing further increases in the levels of integration and consequent increases in die size. This lengthens the interconnections from one side of the chip to the other and, therefore, both resistance and capacitance of the interconnects are increased, producing much larger time constant values. Thus the effects of increased propagation delays, signal decay, and clock skew will decrease maximum achievable operating frequency, even though the smaller transistors produce gates with less delay.

One solution to this problem has been to make use of multilayer interconnections with thicker, wider conductors and thicker separating layers. This will reduce both $R$ and $C$ and also reduce die size. Other measures include the use of cascaded drivers and repeaters to reduce the effects of long interconnects.

A further option is to use optical interconnection techniques where a very high level of integration is required for high speed circuits. In order to use such techniques, optical fibers, laser diodes, receivers, and amplifiers must be included in the integrated circuit. Performance will vary with the materials used, but rough estimations can be made for comparison with metal interconnects. To start our considerations, a model may be set out as in Figure 5–4.

The propagation delay $T_p$ along a single aluminum interconnect can be calculated from the following approximate equation (Sakurai, 1983):

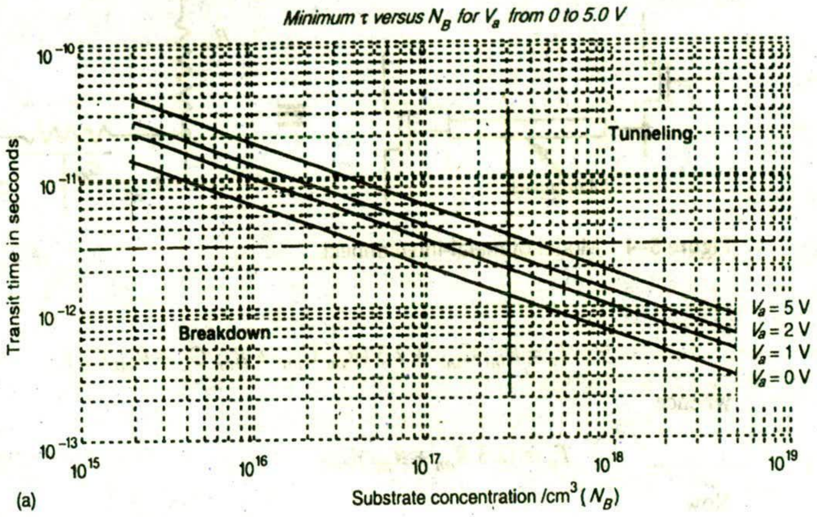


**Figure 5–3(a)** Transit time τ versus substrate concentration/cm$^3$

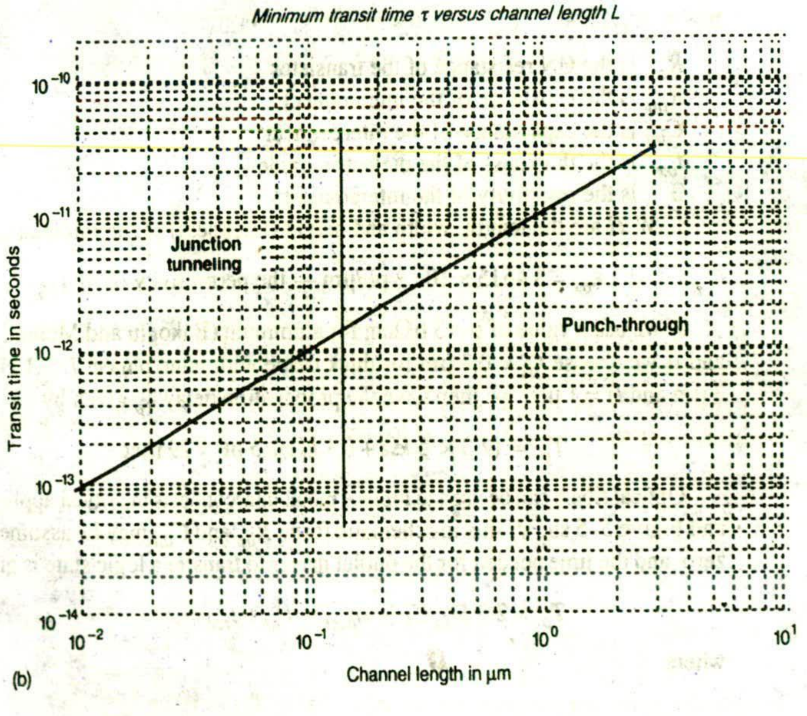


**Figure 5–3(b)** Transit time τ versus *L*

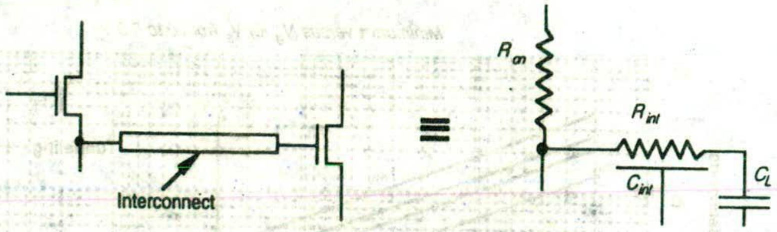


**Figure 5–4** Model of metal interconnect

$$T_p = R_{int}\,C_{int} + 2.3\,(R_{on}\,C_{int} + R_{on}\,C_L + R_{int}\,C_L)$$

whence

$$T_p \doteqdot (2.3\,R_{on} + R_{int})C_{int}$$

Now

$$R_{int} = \rho\frac{L}{HW}$$

$$C_{int} = \varepsilon_{ox}\,[1.15.W/t_{ox} + 2.28\,(H/t_{ox})^{0.222}]L$$

where

| | |
|---|---|
| $R_{on}$ | is the ON resistance of the transistor |
| $R_{int}$ | is the resistance of the interconnect |
| $C_{int}$ | is the capacitance of the interconnect |
| $t_{ox}$ | is the thickness of the dielectric oxide |
| $\rho$ | is the resistivity of the interconnect |
| $L, W, H$ | are the length, width and height (thickness) of the interconnect |

$$\varepsilon_{ox} = 3.4515 \times 10^{-5}\ \text{pF/µm — the permittivity of SiO}_2$$

If we use a value of $\rho = 3$ µΩcm for aluminum (Bakoglu and Meindl, 1985), and if we choose $t_{ox} = 0.8$ µm for thick oxide with interconnect $L = 1$ cm, $W = 3$ µm and $H = 1$ µm, we then have the propagation delay $T_p$ given by

$$T_p = (2.3 \times 5\ k\Omega + 0.1\ k\Omega)2.5\ \text{pF} = 29\ \text{nsec}$$

Optical fibers can be used to replace metal interconnects in critical applications, and Figure 5–5 shows this in schematic form. $R_{int}$ and $C_{int}$ may be assumed to be zero, and the time needed for the output driver to transfer a logic state is given by

$$T_p = 2.3\,R_{on}\,C_L + t_{laser} + t_{int} + t_{rec}$$

where

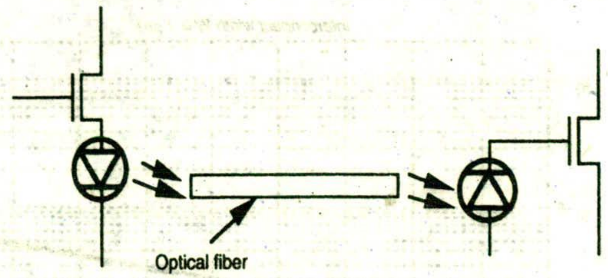


**Figure 5–5** Electro-optical interconnection

$C_L$ is the input capacitance of the laser diode
$t_{laser}$ is the delay time through the laser diode
$t_{int}$ is the propagation delay along the optical fiber interconnnect
$t_{rec}$ is the receiver delay time

and

$$t_{int} = \frac{nL}{c}$$

where

$n$ is the retractive index for the optic fiber material
$L$ is the length of the fiber
$c$ is the free space speed of light ($c = 3 \times 10^8$ m/sec).

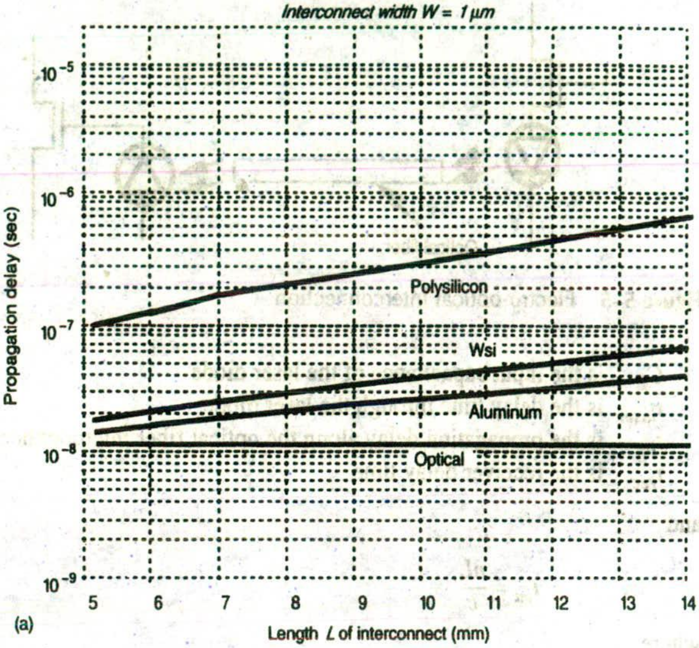
Since laser diodes and receivers can work at frequencies above 10 GHz, each of them presents a relatively short delay — typically around 100 psec. The capacitance of a discrete laser diode is about 1 pF (Hutcheson, 1987) and the refractive index of commonly used material for fiber optics is between 1.5 and 2.0.

Evaluating the propagation delay we have

$$T_p = 2.3 \times 5 \times 10^3 \times 1 \times 10^{-12} + 1 \times 10^{-10} + \frac{2 \times 10^{-4}}{3 \times 10^8} + 1 \times 10^{-10} = 11.7 \text{ nsec}$$

Delay time versus line length and width may be assessed from Figure 5–6. It is obvious that the longer the interconnect, the more speed advantage arises from the use of fiber optics. In considering delay time versus line width, it may be shown that $R_{on}$ is the dominant factor for aluminum whilst $R_{int}$ contributes the major component for poly.

The performance of laser diodes and receivers can be improved if they are formed as part of an integrated circuit. GaAs is a material which allows this integration since it can accommodate both electronic components and optical interconnections in the one chip.



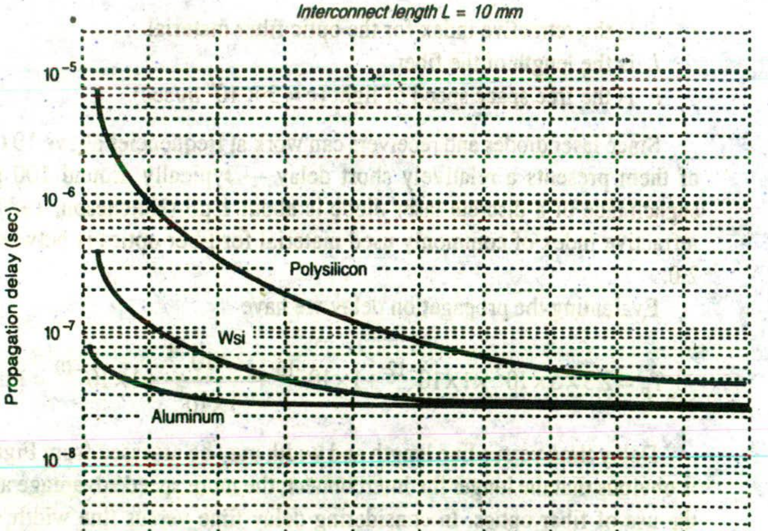


**Figure 5–6** Interconnect delay versus width and length

$R_{on} = 5\ \text{k}\Omega$; $H = W/3$; $t_{ox} = W/3$; $\rho_{Al} = 3\ \mu\Omega\text{cm}$; $\rho_{WsI} = 30\ \mu\Omega\text{cm}$; $\rho_{Poly.} = 500\ \mu\Omega\text{cm}$.

## 5.4 Limits due to subthreshold currents

One of the major concerns in the scaling of devices is the effect on subthreshold current $I_{sub}$ which is directly proportional to $exp(V_{gs} - V_t)q/kT$.

When a transistor is in the off state, then the value of $V_{gs} - V_t$ is negative and should be as large as possible to minimize $I_{sub}$. As voltages are scaled down, the ratio of $V_{gs} - V_t$ to $kT$ will reduce so that subthreshold current increases quite dramatically. For this reason it may be desirable to scale both $V_{gs}$ and $V_t$ together with $V_{DD}$ by factor 1/b rather than 1/a, since a is generally greater than b. However, this increases electric field strengths and thus lowers breakdown voltages.

The maximum electric field across a depletion region is given by (Grove, 1967):

$$E_{max} = \frac{2(V_a + V_B)}{d}$$

This applies to a one-sided step junction.

As discussed previously, $(V_a + V_B)$ is scaled by $(\beta + m)/\beta\ (m + 1)$ and $d$ is scaled by $1/\alpha$. Therefore, $E_{max}$ is scaled by $\alpha\,(\beta + m)/\beta\ (m + 1)$. Again, if $\alpha$ is greater than $\beta$, then more electric field stress will be applied across depletion regions of scaled-down transistors.

At the same time, the junction breakdown voltage $BV$ must be considered. $BV$ is given by (Grove, 1967):

$$BV = \frac{\varepsilon_{si}\varepsilon_0 (E_{crit})^2}{2qN_B}$$

It will be seen that $BV$ is thus scaled by $\beta(m + 1)/\alpha^2\,(\beta + m)$ and will decrease. Extra care is therefore required in estimating the breakdown voltage for scaled devices. It should be noted that electric fields are greater and $BV$ is greater at the corners of diffusion regions underlying or abutting silicon dioxide.

## 5.5 Limits on logic levels and supply voltage due to noise

Major advantages in the scaling of devices are smaller gate delay time, that is, higher operating frequencies and lower power dissipation. However, the decreased inter-feature spacing and greater switching speeds inevitably result in noise problems. Noise may also be amplified and is thus a major concern.

The mean square current fluctuation in the channel is given by

$$(i^2) = 4kTR_n g_m \Delta f$$

where $R_n$ is the equivalent noise resistance at the input and $\Delta f$ is the bandwidth.

F. M. Klaassen and J. Prins (1966) have investigated the thermal noise in a MOS transistor over a range of substrate doping levels $N_B$ from $10^{14}$ to $10^{17}$ cm$^{-3}$. When a transistor works in saturation, $g_m$ is no longer proportional to the gate voltage $V_g$, and can be expressed as

$$g_m \doteq BV_p$$

where

$$B = \frac{\mu W C_{ox}}{L}$$

and $V_p$ is the pinch off voltage given by

$$V_p = V'_g - \frac{1}{2}\left(\frac{a}{C_{ox}}\right)^2 \left[\frac{(1+4V'_g\, C_{ox})^{1/2}}{a} - 1\right]$$

where

$$V'_g = V_g - V_t + V_B$$
$$a = (2\varepsilon_{si} q N_B)^{1/2}$$

$V_B$ is the junction (built-in) potential.

Then, the equivalent noise resistance $R_n$ is given by

$$R_n = \left(\frac{1V'_g}{2V'_p} + \frac{1}{6}\right) g_m^{-1}$$

where

$$V'_p = V_p + V_B$$

Since $V_p$ is a monotonically decreasing function of the gate oxide thickness $t_{ox}$ and substrate doping $N_B$, $R_n$ is also a monotonically decreasing function of the same parameters.

Consequently, the main factor of the thermal noise $R_n\, g_m$ is given by

$$R_n g_m = \frac{1}{2}\left(\frac{V_g - V_t + V_B}{V_p + V_B}\right) + \frac{1}{6}$$

This indicates that $R_n\, g_m$ is strongly and directly dependent on $t_{ox}$ and $N_B$ and also, to a lesser extent, on $V_g$. Experimental results, as in Figure 5–7, support this theory (Klaassen and Prins, 1966).

Considering current technology in which $t_{ox}$ is scaled, the effect of $N_B$ is smaller (Sah, Wu and Hielscher, 1966), and $V'_p$ and $V'_g$ are replaced by $V_p$ and $V_g$ respectively. Thus, the expression for $R_n\, g_m$ becomes

$$R_n g_m = \frac{1}{2}\left[\frac{V_g}{V_g - \frac{1}{2}\left(\frac{a}{C_{ox}}\right)^2\left(\left(\frac{1+4V_g'C_{ox}}{a}\right)^{1/2} - 1\right)}\right] + \frac{1}{6}$$

When constant field scaling is applied, $V_g$ is scaled by $1/\alpha$, $C_{ox}$ and $N_B$ are scaled by $\alpha$. Consequently, $R_n g_m$ is only slightly reduced owing to the increased value of $C_{ox}$. Thus, the ratio of logic level to thermal noise is degraded by almost the same factor.

Flicker noise has been the subject of many studies since A. L. McWhorter introduced his model in 1956. The noise was observed and explained as the result of fluctuations of carriers trapped in the channel by surface states.

As a conclusion of the investigations by F. M. Klaassen, the change in the number of trapped carriers $dn_t$ due to the change in the number of induced free carriers $d_n$ presents a current fluctuation $\Delta i$ at the output, such that

$$\Delta i^2 \approx \frac{q\mu s I V_d}{L^2 f}$$

where

$s = dn_t / d_n$ — the surface state efficiency
$I$ = the DC drain current
$f$ = the frequency
$V_d$ = the applied drain voltage.

Usually the output noise is represented by an equivalent noise voltage source $\Delta V$ at the input (Klaassen, 1971), such that

$$\Delta V^2 = \frac{qs(V_d - 0.5V_d)}{C_g f}$$

When the transistor operates in saturation, then $V_d \doteqdot V_g$, so that

$$\Delta V^2 = \frac{1}{2}\frac{qsV_g}{C_g f}$$

where

$V_g$ = the effective applied gate voltage
$C_g$ = the gate capacitance.

Since $s$ is a process dependent factor, the flicker noise is scaled by one for constant field scaling or by $\alpha^2 / \beta^2$ for the combined scaling model.

In addition to noise sources already considered, other noise inputs are due to mutual inductive and mutual capacitive coupling, and these alone could impose practical limitations on the lowest usable operating voltages.

Considering the cross-talk between two parallel signal lines on a chip, the coupling model presented (Watts, 1989) shows that capacitive noise is proportional to $C.dV/dt$, where $C$ is the inter-line capacitance, and $dV/dt$ is approximately equal to $V_a/t_r$, where $t_r$ is the rise time of the coupled signal. The inductive noise is related to $LdI/dt$, where $L$ is the mutual inductance and $dI/dt \approx I_{sat}/t_r$. Therefore, cross-talk noise increases as operating frequency increases and $t_r$ is reduced.

There are also other noise sources due to external influences, such as radio frequency signals, voltage spikes or voltage drops on power lines or ground connections, unterminated signal lines and lines with non-uniform impedance characteristics.

A typical peak to peak noise of at least 100 mV may be observed on the power and ground lines of even well-designed multilayer PC boards (Long & Butner, 1989).

Scaling down exacerbates the effect of both internally and externally generated noise and this degrades both the production yield and the reliability of high density chip layouts.

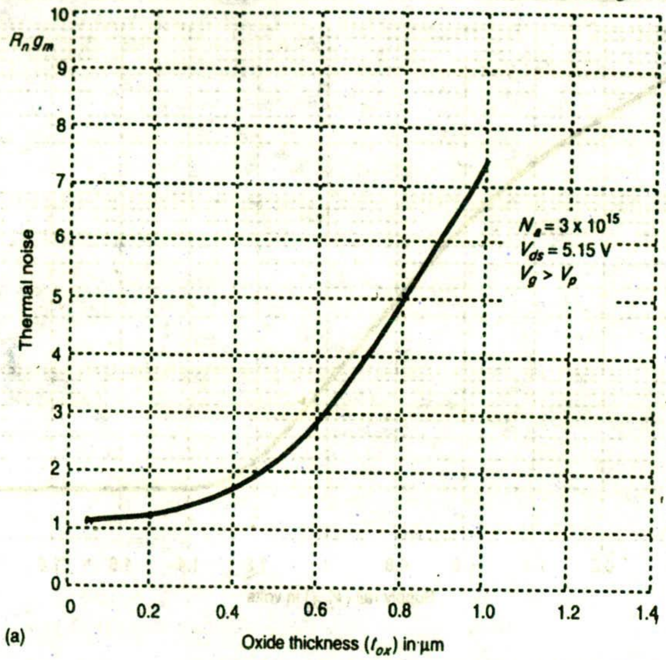
In order to assess the effects of noise on the probability of failure $P_F$ in a circuit, we may consider a situation where, as in Figure 5–7, the minimum signal to noise ratio *(SNR)* for satisfactory operation is assumed to be four, and the mean noise is assumed to be zero with a standard deviation $\Sigma = 100$ mV. $P_F$ may be estimated by using the Gaussian distribution

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \left( \frac{e^{-\mu^2/2}}{2} \right) du$$

The $P_F$ values for a range of supply rail voltages and for the conditions specified are derived by integrating the appropriate area at one end of the Gaussian curve and are shown in Figure 5–8.

## 5.7 Limits due to current density

High purity aluminum seems the most attractive, and is thus the most widely used, material for forming interconnections in VLSI chips. However, the scaling down of dimensions also increases the current density in interconnects by the same factor if constant field scaling is applied. When the current density in aluminum approaches $10^6$ Amps/cm$^2$ (10 mA/μm$^2$), the interconnects are likely to be burned off owing to metal migration. Thus, allowable current densities are set well below this limit and figures of J = 1 to 2 mA/μm$^2$ are commonly used.



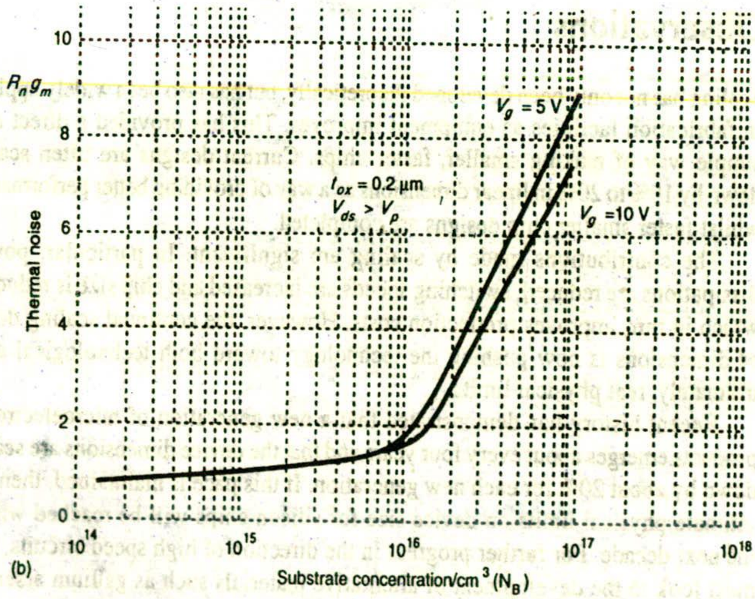


**Figure 5–7** Thermal noise versus oxide thickness (a) and substrate doping (b)

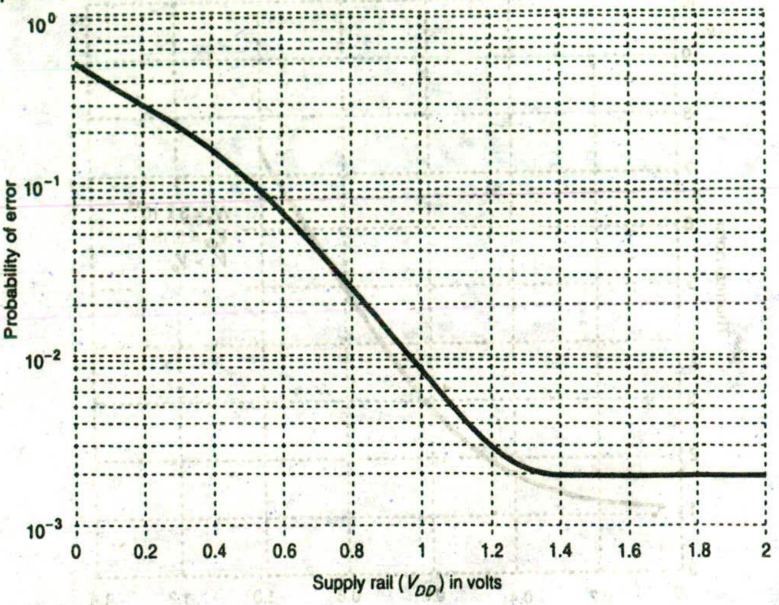


**Figure 5–8** Probability of error versus supply voltage

## 5.8 Observations

Scaling has not only been developed theoretically, but has also been widely applied in fabrication facilities as equipment improves. This has provided a direct and simple way of making smaller, faster chips. Current designs are often scaled down by 10% to 20% in linear dimensions as a way of providing better performance whilst faster smaller chip designs are completed.

The contributions made by scaling are significant. In particular, power dissipations are reduced, switching speeds are increased and chip size is reduced, which in turn improves production costs. However, the continual scaling down of dimensions is now pushing the technology toward both technological and, ultimately, real physical limits.

Recent history has demonstrated that a new generation of microelectronic products emerges about every four years and that the device dimensions are scaled down by about 20% for each new generation. If this pace is maintained, then the ultimate physical limits on device size for silicon chips will be reached within the next decade. For further progress in the direction of high speed circuits, one must look to the development of alternative materials such as gallium arsenide (GaAs) and to super conductors.

## 9 References

Bakoglu, H. B., and Meindl, J. D. (1985, May) 'Optimal interconnection circuits for VLSI', *IEEE Transactions on Electronic Devices*, Vol. ED-32, No. 5.

Bergmann, N. W. (1991) 'A combined voltage and dimension scaling model for VLSI circuits', Proceedings of Microelectronics Conference, Melbourne, 24–25 June 1991, 101–4.

Grove, A. S. (1967) *Physics and Technology of Semiconductor Devices*, John Wiley and Sons Inc., New York.

Hutcheson, L. D. (1987) *Integrated Optical Circuits and Components: Design and Applications*, Marcel Dekker, New York.

Klaassen, F. M., and Prins, J. (1966) 'Thermal noise in MOS Transistors', *Philips Research Report*, Vol. 22.

Klaassen, F. M. (1971) 'Characterisation of Low 1/f Noise in MOS Transistors', *IEEE Transactions on Electronic Devices*, Vol. ED-18, No. 10, 887–91.

Long, S. I., and Butner, S.E. (1989) *Gallium Arsenide Digital Integrated Circuit Design*, McGraw-Hill, USA.

Meindl, J. D. (1986) 'Interconnection limits on ultra large scale integration', *Proceedings VLSI '85*, North Holland, 13–19.

Sah, C. T., Wu, S. Y., and Hielscher, F. H. (1966) 'The effects of fixed bulk charge on the thermal noise in metal-oxide-semiconductor transistors', *IEEE Transactions on Electronic Devices*, Vol. ED-13, 410–14.

Sakurai, T. (1983, August) 'Approximation of wiring delay in MOSFET LSI', *IEEE Journal of Solid State Circuits*, Vol. SC-18, 418–26.

Sze, S. M. (1985) *Semiconductor Devices: Physics and Technology*, Bell Telephone Laboratories, USA.

Watts, R. K. (1989) *Submicron Integrated Circuits*, John Wiley and Sons Inc., New York.

# 6 Subsystem design and layout

*Logic is simply the architecture of human reason.*

Evelyn Waugh

*Tall oaks from little acorns grow.*

David Everett

## Objectives

Having now covered the basic MOS and BiCMOS technologies, the behavior of components formed by MOS layers, the basic units which help to characterize behavior, and a set of design rules, we are in a position to undertake the design of some of the subsystems (leaf-cells) from which larger systems are composed.

The most basic leaf-cells are the common logic gate arrangements and these are dealt with in nMOS, CMOS and BiCMOS forms. In the case of CMOS gates there are several ways in which the logic may be configured, most of which are dealt with in this chapter.

The concepts of structured design, which leads to system designs of high 'regularity', are introduced through examples. A highly regular design is to be sought for VLSI systems since high regularity implies the detailed design of relatively few leaf-cells which are then replicated many times and interconnected to form the system.

Other commonly applied concepts, such as two-phase clocks and buses for the interconnection paths between leaf-cells and subsystems, are also introduced and illustrated. Important aspects of power distribution on chip and associated limitations are discussed.

The chapter also includes an introduction to common subsystem designs such as multiplexers and shift registers.

## 6.1 Some architectural issues

In all design processes, a logical and systematic approach is essential. This is particularly so in the case of the design of a VLSI system which could otherwise take so long as to render the whole system obsolete before it is off the drawing board. Take, for example, the case of a relatively straightforward MSI logic circuit comprising, say, 500 transistors. A reasonable time to allocate to the design and proving of such a circuit could be some two engineer-months. Consider now the design of a 500,000 transistor VLSI system. Even if a linear relationship exists between complexity and design time, the required design time would be 2000 engineer-months or 170 engineer-years. In fact, design time tends to rise exponentially with increased complexity. Obviously, then, we must adopt design methods which allow the handling of complexity in reasonable periods of time and with reasonable amounts of labor.

Certainly we are not about to tackle 500,000 transistor designs in this text, but some sensible concepts applied even at the subsystem (leaf-cell) level can be most worthwhile and can also be directly compatible with larger system design requirements. Guidelines may be set out as follows:

1. Define the requirements (properly and carefully).
2. Partition the overall architecture into appropriate subsystems.
3. Consider communication paths carefully in order to develop sensible interrelationships between subsystems.
4. Draw a floor plan of how the system is to map onto the silicon (and alternate between 2, 3 and 4 as necessary).
5. Aim for regular structures so that design is largely a matter of replication.
6. Draw suitable (stick or symbolic) diagrams of the leaf-cells of the subsystems.
7. Convert each cell to a layout.
8. Carefully and thoroughly carry out a design rule check on each cell.
9. Simulate the performance of each cell/subsystem.

The whole design process will be greatly assisted if considerable care is taken with:

1. the *partitioning* of the system so that there are clean and clear subsystems with a minimum interdependence and complexity of interconnection between them;

2. the *design simplification* within subsystems so that architectures are adopted which allow the exploitation of a cellular design concept. This allows the system to be composed of relatively few standard cells which are replicated to form highly regular structures.

In designing digital systems in MOS technology there are two basic ways of building logic circuits, which will now be discussed.

## 6.2 Switch logic

Switch logic is based on the 'pass transistor' or on transmission gates. This approach is fast for small arrays and takes no static current from the supply rails. Thus, power dissipation of such arrays is small since current only flows on switching.

Switch (pass transistor) logic is similar to logic arrays based on relay contacts in that the path through each switch is isolated from the signal activating the switch. In consequence, the designer has a considerable amount of freedom in implementing architectural features compared with bipolar logic-based designs.

A number of texts on switching theory, some dating from the 1950s and 1960s, have sections on relay/switch logic and the reader is referred to such material for generating ideas for implementation in MOS switch logic. An example is Marcus, *Switching Circuits for Engineers*, Prentice Hall, 1962; 3rd edn, 1975.

Basic *And* and *Or* connections are set out in Figure 6–1, but many combinations of switches are possible.

### 6.2.1 Pass transistors and transmission gates

Switches and switch logic may be formed from simple n- or p-pass transistors or from transmission gates (complementary switches) comprising an n-pass and a p-pass transistor in parallel as shown in Figure 6–2. The reason for adopting the apparent complexity of the transmission gate, rather than using a simple n-switch or p-switch in most CMOS applications, is to eliminate the undesirable threshold voltage effects which give rise to the loss of logic levels in pass transistors as indicated in Figure 6–2. No such degradation occurs with the transmission gate, but more area is occupied and complementary signals are needed to drive it. 'On' resistance, however, is lower than that of the simple pass transistor switches.

When using nMOS switch logic, there is one restriction which must always be observed: *no* pass transistor gate input may be driven through *one or more pass transistors* (see Figure 6–2). As shown, logic levels propagated through pass transistors are degraded by threshold voltage effects. Since the signal out of pass transistor $T_1$ does not reach a full logic 1, but rather a voltage one transistor

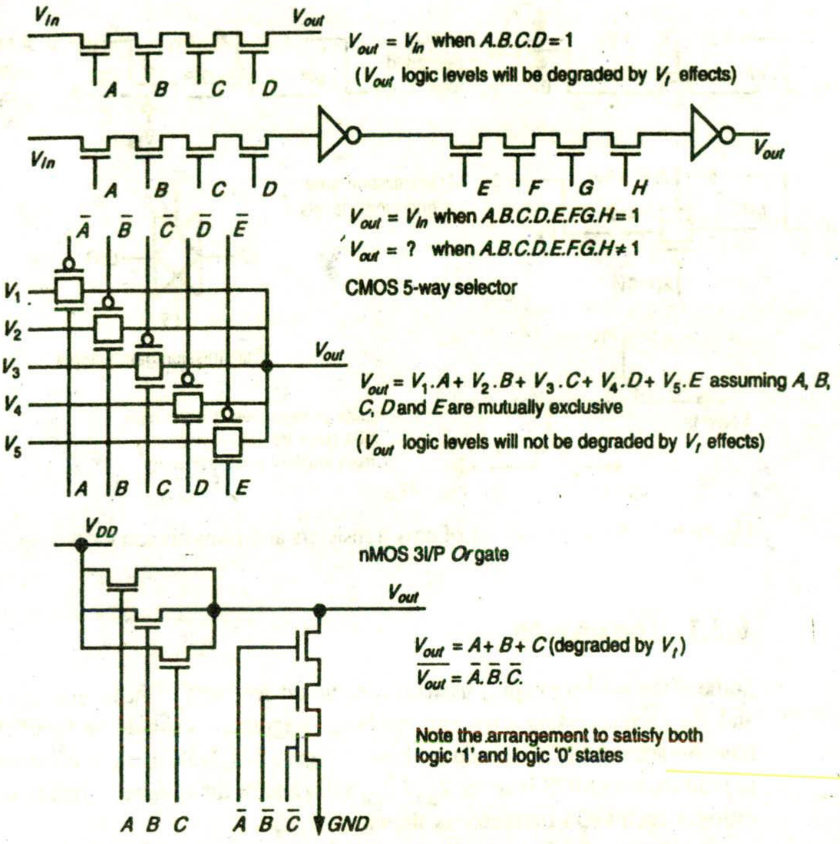

$V_{in}$
$V_{out}$
A B C D
$V_{out} = V_{in}$ when $A.B.C.D = 1$
($V_{out}$ logic levels will be degraded by $V_t$ effects)
$V_{in}$
A B C D E F G H
$V_{out}$
$V_{out} = V_{in}$ when $A.B.C.D.E.F.G.H = 1$
$V_{out}$ = ? when $A.B.C.D.E.F.G.H \neq 1$
CMOS 5-way selector
$\bar{A}$ $\bar{B}$ $\bar{C}$ $\bar{D}$ $\bar{E}$
$V_1$ $V_2$ $V_3$ $V_4$ $V_5$
$V_{out}$
$V_{out} = V_1.A + V_2.B + V_3.C + V_4.D + V_5.E$ assuming A, B, C, D and E are mutually exclusive
($V_{out}$ logic levels will not be degraded by $V_t$ effects)
A B C D E
$V_{DD}$
nMOS 3I/P Or gate
$V_{out}$
$V_{out} = A + B + C$ (degraded by $V_t$)
$\overline{V_{out}} = \bar{A}.\bar{B}.\bar{C}$
Note the arrangement to satisfy both logic '1' and logic '0' states
A B C
$\bar{A}$ $\bar{B}$ $\bar{C}$
GND

**Figure 6–1** Some switch logic arrangements

threshold below a true logic 1, this degraded voltage would not permit the output of $T_2$ to reach an acceptable logic 1 level.

## 6.3 Gate (restoring) logic

Gate logic is based on the general arrangement typified by the inverter circuits (the inverter being the simplest gate).

Both *Nand* and *Nor* and, with CMOS, *And* and *Or* gate arrangements are available. Inverters are also employed to complement and restore logic levels that have been degraded (e.g. because they have passed through pass transistors).

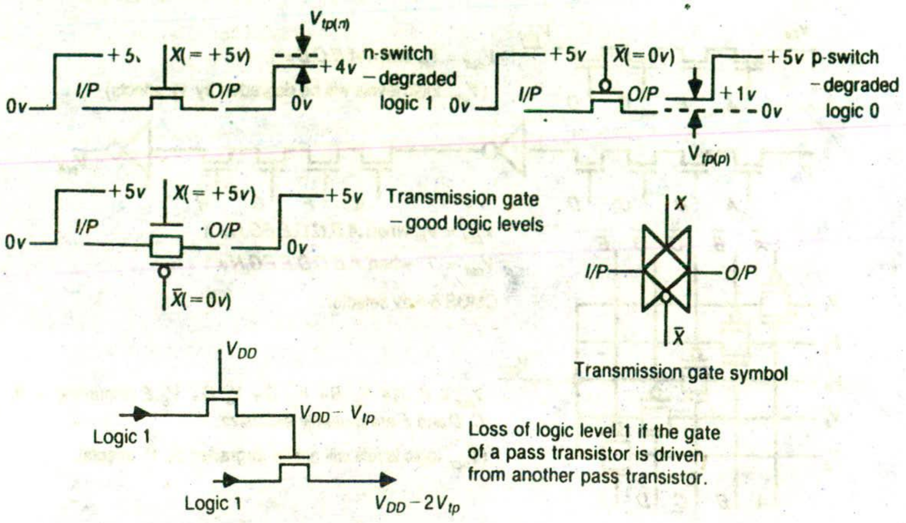


**Figure 6–2** Some properties of pass transistors and transmission gates

### 6.3.1 The inverter

Some of the most commonly used inverter circuit diagrams — the inverter symbol, and the corresponding stick and symbolic diagrams — should be familiar by now. An assortment is reproduced here in Figure 6–3. Note that it is often useful to indicate the nMOS inverter $Z_{p.u.}/Z_{p.d.}$ ratio and/or the channel length to width ratio for each MOS transistor as shown.

In achieving the desired pull-up to pull-down ratio, several possibilities emerge, two of which are illustrated in Figures 6–4 and 6–5 for an 8:1 nMOS inverter. Note the effect that the different approaches have on power dissipation $P_d$ and on the area occupied by the inverter. Also note the resistance and capacitance values. The CMOS inverter carries no static current and thus has no power dissipation unless switching. The reader must not, however, imagine that CMOS circuits have no dissipation problems. The switching dissipation for fast CMOS logic circuits will be considerable.

### 6.3.2 Two-input nMOS, CMOS and BiCMOS *Nand* gates

Two-input *Nand* gate arrangements are given in Figure 6–6. The nMOS (and pseudo-nMOS) *L*:*W* ratios should be carefully noted since they must be chosen to achieve the desired overall $Z_{p.u.}/Z_{p.d.}$ ratio (where $Z_{p.d.}$ is contributed in this case by *both* input transistors in series).

In order to arrive at the required *L*:*W* ratios for an nMOS (or pseudo-nMOS)

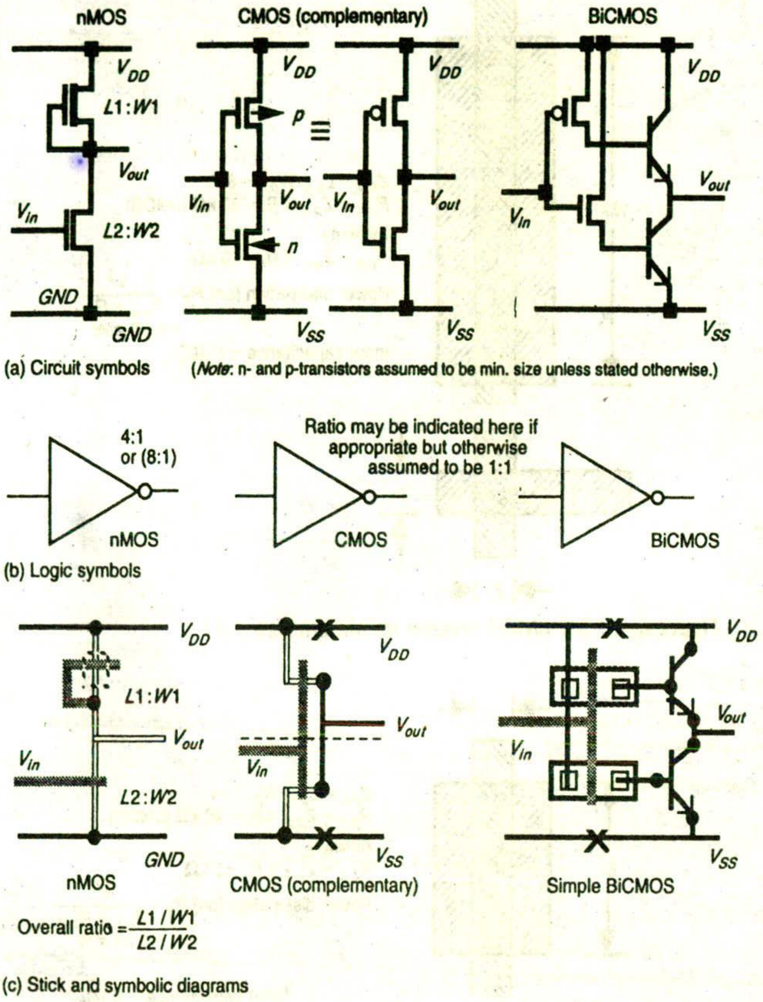
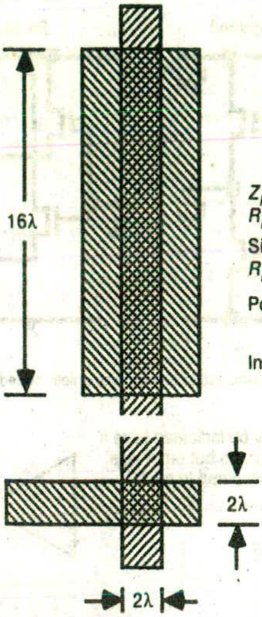

nMOS
CMOS (complementary)
BiCMOS
$V_{DD}$
$L1:W1$
$V_{out}$
$V_{in}$
$L2:W2$
GND
p
n
≡
$V_{SS}$
(a) Circuit symbols
(Note: n- and p-transistors assumed to be min. size unless stated otherwise.)
4:1 or (8:1)
Ratio may be indicated here if appropriate but otherwise assumed to be 1:1
nMOS
CMOS
BiCMOS
(b) Logic symbols
nMOS
CMOS (complementary)
Simple BiCMOS
Overall ratio = $\frac{L1/W1}{L2/W2}$
(c) Stick and symbolic diagrams

**Figure 6–3** nMOS, CMOS and BiCMOS inverters

*Nand* gate with *n* inputs, it is only necessary to consider the very simple circuit model of the gate in the condition when all *n* pull-down transistors are conducting as in Figure 6–7.

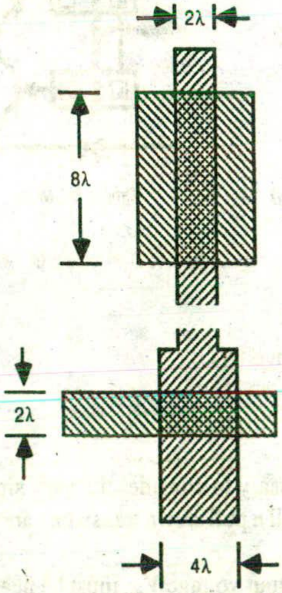
The critical factor here is that the output voltage $V_{out}$ must be near enough to ground to turn off any following inverter-like stages, that is

$$V_{out} \leq V_t = 0.2V_{DD}$$



$Z_{p.u.} = L_{p.u.}/W_{p.u.} = 8$
$R_{p.u.} = Z_{p.u.} \times R_s = 80\ \text{k}\Omega$ (nMOS)
Similarly,
$R_{p.d.} = Z_{p.d.} \times R_s = 10\ \text{k}\Omega$
Power dissipation (on) $P_d = \dfrac{V^2}{R_{p.u.} + R_{p.d.}}$
$= 0.28$ mW
Input capacitance $= 1\square C_g$

**Figure 6–4** 8:1 nMOS inverter (minimum size p.d.)

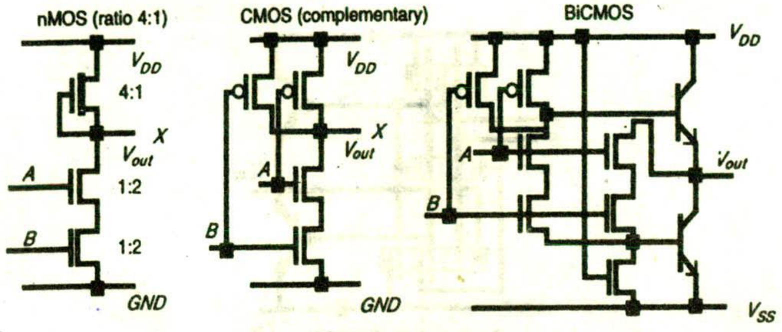


$Z_{p.u.} = L_{p.u.}/W_{p.u.} = 4$
$R_{p.u.} = Z_{p.u.} \times R_s = 40\ \text{k}\Omega$ (nMOS)
Similarly,
$R_{p.d.} = Z_{p.d.} \times R_s = 5\ \text{k}\Omega$
Power dissipation (on) $P_d = \dfrac{V^2}{R_{p.u.} + R_{p.d.}}$
$= 0.56$ mW

Input capacitance $= 2\square C_g$

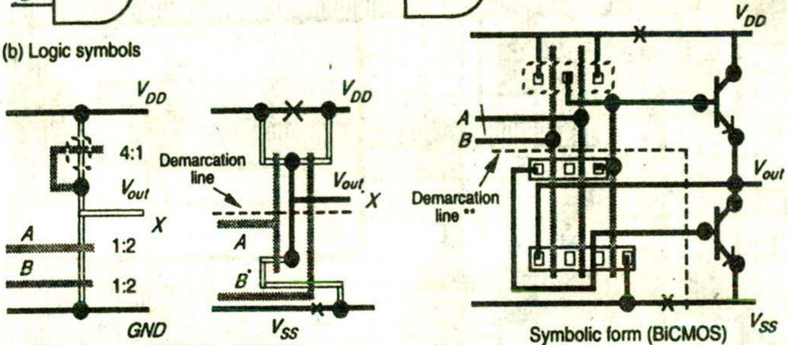
*Note:* A 4:1 inverter is formed if the p.d. width is halved.

**Figure 6–5** An alternative 8:1 nMOS inverter



(a) Circuit diagrams *Note*: n- and p- transistors assumed to be minimum size unless stated otherwise.

(b) Logic symbols



(c) Stick diagrams (nMOS and CMOS)

*Note*: The natural 2.5:1 asymmetry of the CMOS inverter is improved to 1.25:1 (or better) owing to the two n-type pull-down transistors in series for the two I/P *Nand*.

** Demarcation line (edge of n-well) may be shown if required.

**Figures 6–6(a)–(c)** nMOS, CMOS and BiCMOS 2-input *Nand* gates

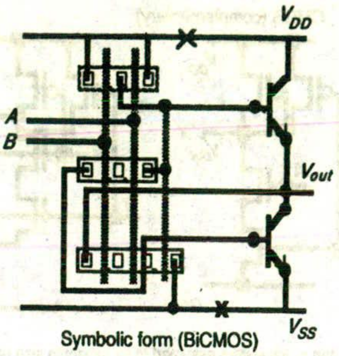
Thus

$$\frac{V_{DD}\times nZ_{p.d.}}{nZ_{p.d.}+Z_{p.u.}} \leqslant 0.2V_{DD}$$

where $Z_{p.d.}$ applies for any one pull-down transistor. The boundary condition then is

$$\frac{nZ_{p.d.}}{nZ_{p.d.}+Z_{p.u.}} = 0.2$$

whence nMOS *Nand* ratio $= \dfrac{nZ_{p.d.}}{nZ_{p.d.}} = \dfrac{4}{1}$



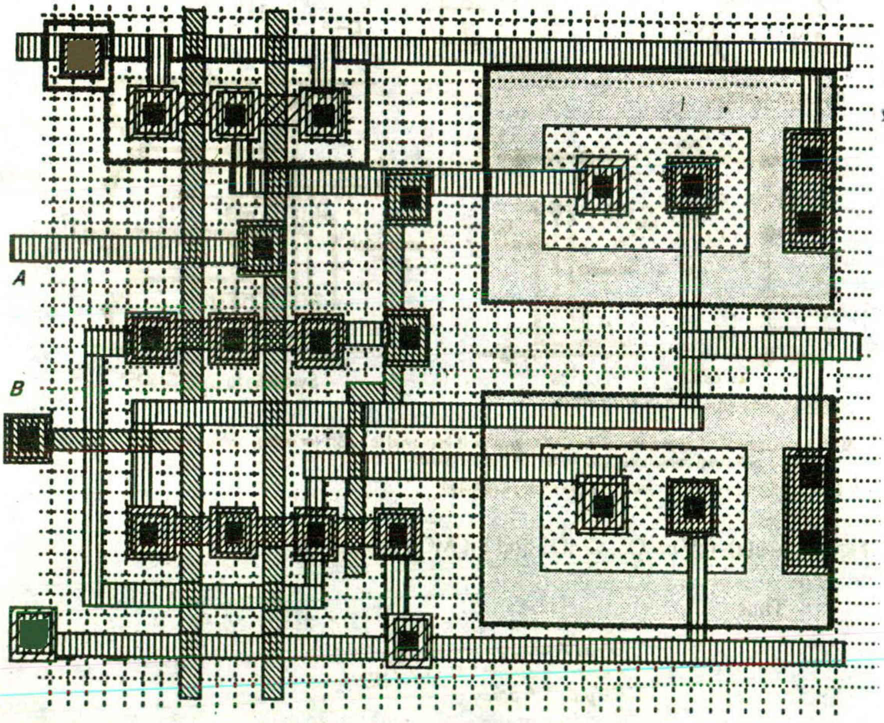


Figure 6–6(d) A BiCMOS two-input *Nand* gate

that is, the ratio between $Z_{p.u.}$ and the sum of all the pull-down $Z_{p.d.}$s must be 4:1 (as for the nMOS inverter).

This ratio must be adjusted appropriately if input signals are derived through pass transistors.

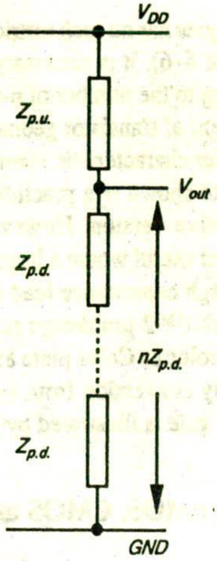


**Figure 6–7** nMOS *Nand* ratio determination

Further consideration of the nMOS *Nand* gate geometry reveals two significant factors:

1. nMOS *Nand* gate *area requirements* are considerably greater than those of a corresponding nMOS inverter, since not only must pull-down transistors be added in series to provide the desired number of inputs, but, as inputs are added, so must there be a corresponding adjustment of the length of the pull-up transistor channel to maintain the required overall ratio.
2. nMOS *Nand* gate *delays* are also increased in direct proportion to the number of inputs added. If each pull-down transistor is kept to minimum size ($2\lambda \times 2\lambda$), then each will present 1 $\square C_g$ at its input, but if there are *n* such inputs, then the length and *resistance* of the pull-up transistor must be increased by a factor of *n* to keep the correct ratio. Thus, delays associated with the nMOS *Nand* are

$$\tau_{Nand} = n\tau_{inv}$$

where *n* is the number of inputs and $\tau_{inv}$ is the corresponding nMOS inverter delay. (The alternative approach of keeping $Z_{p.u.}$ constant and widening the pull-down channels has the same effect, since in this case $C_g$ for each pull-down transistor will be increased to $n\square Cg$.)

Furthermore, the rise time of the nMOS *Nand* output is dependent on the actual input(s) on which the ∇ transition takes place.

In consequence of these properties, the nMOS *Nand* gate is used *only* where absolutely necessary and the number of inputs is restricted.

The CMOS *Nand* gate has no such restrictions but, bearing in mind the remarks on asymmetry (Figure 6–6), it is necessary to allow for extended fall-times on capacitive loads owing to the number of n-transistors in series forming the pull-down. Some adjustment of transistor geometry may be necessary for this reason and to keep the transfer characteristic symmetrical about $V_{DD}/2$.

The BiCMOS gate shown is a practical version and is thus more complex than the simple intuitive version. However, it has considerable load-driving capabilities and is most useful where a large fan-out is required or where there is some other form of high capacitance load on the output. A typical mask layout for this gate, using Orbit™ 2 µm design rules, is given in monochrome form in Figure 6–6(d) and in color as Color plate 8(a).

The relatively easy conversion from symbolic form to mask layout for the BiCMOS 2-input *Nor* gate is illustrated by Figure 6–6(d) and Color plate 8(a).

## 6.3.3 Two-input nMOS, CMOS and BiCMOS *Nor* gates

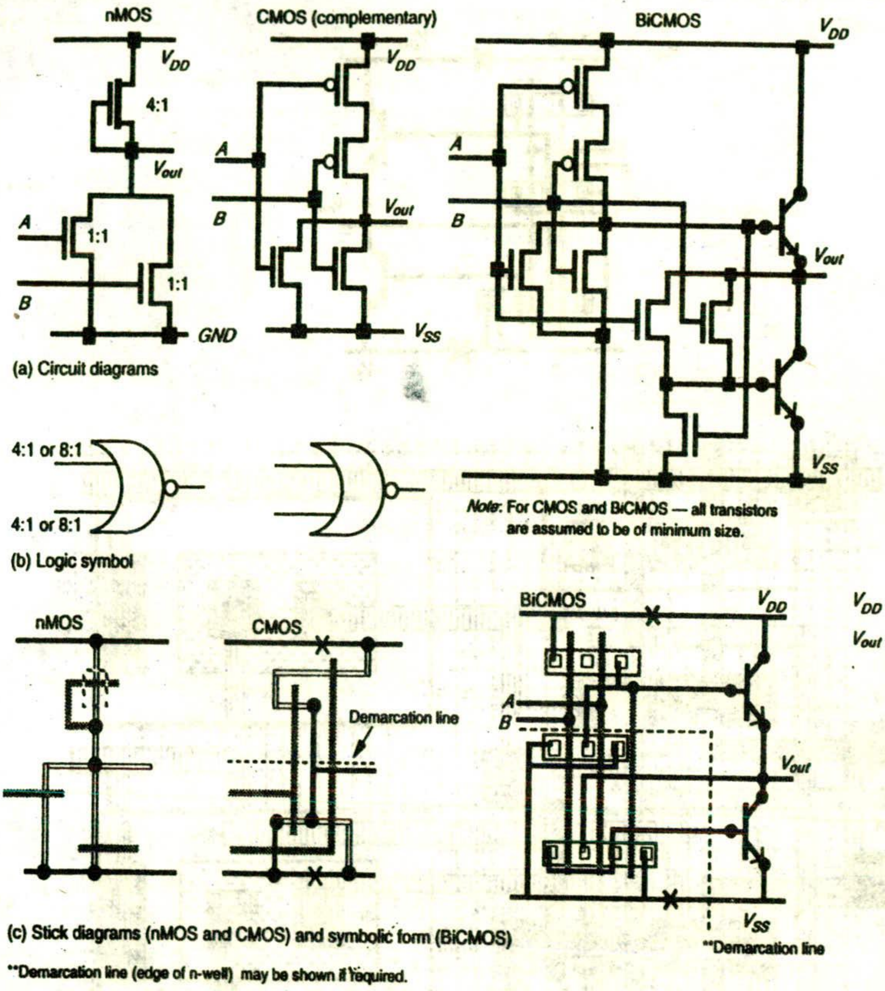
Two-input *Nor gate* arrangements are given in Figure 6–8; note here that the nMOS (or pseudo nMOS) form of *Nor* gate can be expanded to accommodate any reasonable number of inputs (e.g. see Color plate 9) and, in those technologies, is preferred to the *Nand* gate when there is a choice (which is usually the case if logical expressions are suitably manipulated).

Since both 'legs' of the two-input nMOS *Nor* gate provide a path to ground from the pull-up transistor, the ratios must be such that any one conducting pull-down leg will give the appropriate inverter-like transfer characteristic. Thus, each leg has the same ratio as would be the case for an nMOS inverter. This applies irrespective of the number of inputs accommodated.

The area occupied by the nMOS (or pseudo-nMOS) *Nor* gate is reasonable since the pull-up transistor dimensions are unaffected by the number of inputs accommodated. In consequence, the *Nor* gate is as fast as the corresponding inverter and is the preferred inverter-based nMOS (or pseudo-nMOS) logic gate when a choice is possible.

Obviously, the ratio between $Z_{p.u.}$ and $Z_{p.d.}$ of any one leg must be appropriate to the source from which that input is driven for nMOS designs (namely 4:1 driven from another inverter-based circuit or 8:1 if driven via one or more pass transistors) but will be uniformly 3:1 for a pseudo-nMOS design where any series switching is by transmission gate.

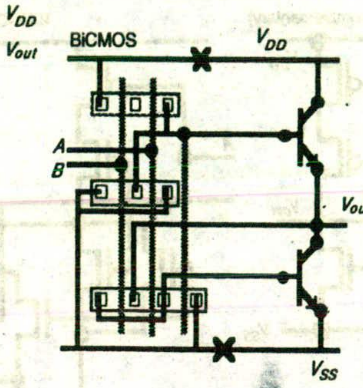
The CMOS *Nor* gate (see Color plate 9) consists of a pull-up p-transistor-based structure, which implements the logic 1 conditions and a complementary n-transistor arrangement to implement the logic 0 conditions at the output. In the case of the *Nor* gate, the p-structure consists of transistors in series, one for each input, while the n pull-down arrangement has as many transistors in parallel as there are inputs to the *Nor* gate. Thus, the already predominant resistance of the p-


nMOS
CMOS (complementary)
BiCMOS
$V_{DD}$
4:1
$V_{out}$
A
B
1:1
GND
$V_{SS}$
(a) Circuit diagrams
4:1 or 8:1
(b) Logic symbol
*Note*: For CMOS and BiCMOS — all transistors are assumed to be of minimum size.
Demarcation line
(c) Stick diagrams (nMOS and CMOS) and symbolic form (BiCMOS)
**Demarcation line (edge of n-well) may be shown if required.
**Demarcation line

**Figures 6–8(a)–(c)** nMOS, CMOS and BiCMOS two-input *Nor* gate

devices is aggravated in its effect by the number connected in series. Rise-time and fall-time asymmetry on capacitive loads is thus increased and there will also be a shift in the transfer ($V_{in}$ vs $V_{out}$) characteristic which will reduce noise immunity. For these reasons, CMOS (complementary logic) *Nor* gates with more than two inputs may require adjustment of the p- and/or n- transistor geometries (*L*:*W* ratios).

The CMOS *Nand* gate, on the other hand, benefits from the connection of p-transistors in parallel, but once again the geometries may require thought when several inputs are required.



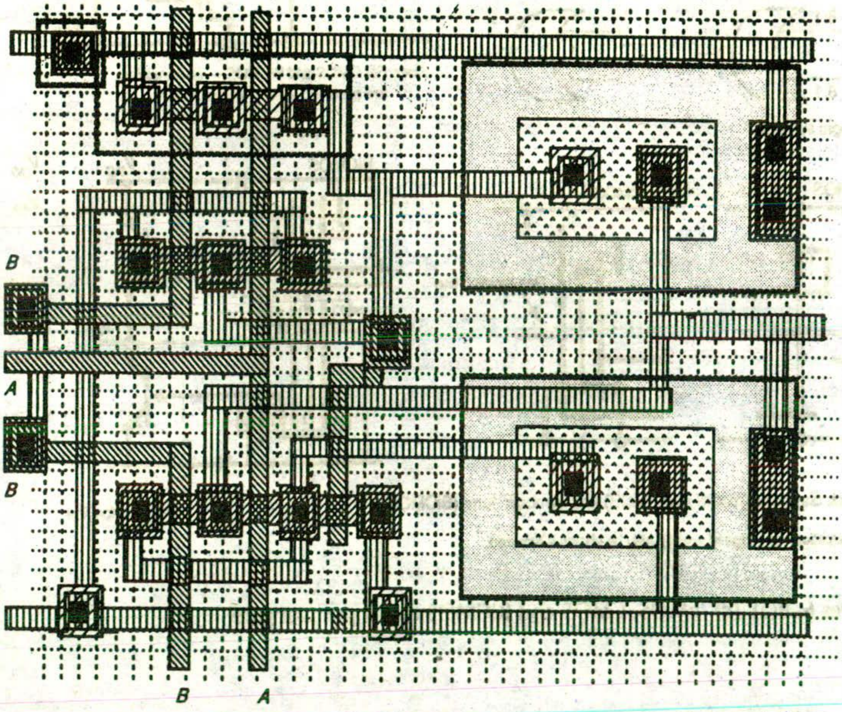


Figure 6–8(d) A BiCMOS two-input *Nor* gate

The BiCMOS *Nor* gate shown is a practical version and, as for the BiCMOS *Nand,* is more complex than a simple intuitive version. However, it also has considerable capacitive load-driving capabilities and is most useful where a large fan-out is required or where there is some other form of high capacitance load on the output such as in the I/O region of a chip.

The relatively easy conversion from symbolic form to mask layout is illustrated for the BiCMOS two-input *Nor* gate in Figure 6–8(d) and Color plate 8(b). The mask layout has been drawn using the Orbit™ 2 μm BiCMOS rule set.

## 6.3.4 Other forms of CMOS logic

The availability of both n- and p-transistors makes it possible for the CMOS designer to explore and exploit various alternatives to inverter-based CMOS logic.

### *6.3.4.1 Pseudo-nMOS logic*

Clearly, if we replace the depletion mode pull-up transistor of the standard nMOS circuits with a p-transistor with gate connected to $V_{SS}$, we have a structure similar to the nMOS equivalent. This approach to logic design is illustrated by the three-input *Nand* gate in Figure 6–9. The circuit arrangements look and behave much like nMOS circuits and appropriate ratio rules must be applied.

In order to determine the required ratio, we consider the arrangement of Figure 6–10 in which a pseudo-nMOS inverter is being driven by another similar inverter, and we consider the conditions necessary to produce an output voltage of $V_{inv}$ for an identical input voltage. As for the nMOS analysis, we consider the conditions for which $V_{inv} = V_{DD}/2$.

At this point the n-device is in saturation (i.e. $0 < V_{gsn} - V_{tn} < V_{dsn}$) and the p-device is operating in the resistive region (i.e. $0 < V_{dsp} < V_{gsp} - V_{tp}$). Equating currents of the n-transistor and the p-transistor, and by suitable rearrangement of

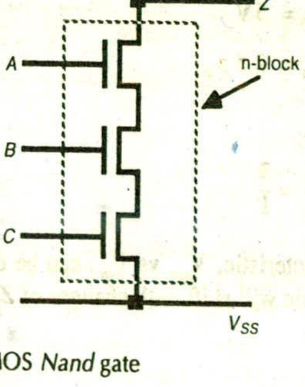


**Figure 6–9** pseudo-nMOS *Nand* gate

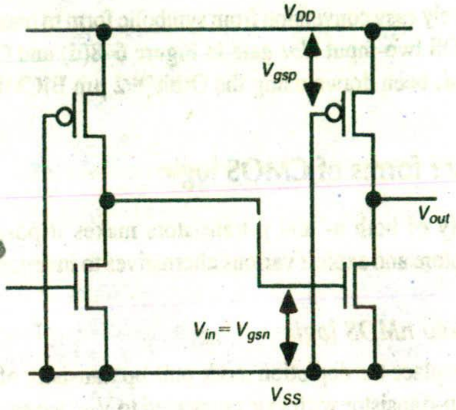


**Figure 6–10** Pseudo-nMOS inverter when driven from a similar inverter

the resultant expression, we obtain

$$V_{inv} = V_{tn} + \frac{(2\mu_p/\mu_n)^{1/2}[(-V_{DD} - V_{tn})V_{dsp} - V_{dsp}^{\ 2}]^{1/2}}{(Z_{p.u.}/Z_{p.d.})^{1/2}}$$

where

$$Z_{p.u.} = L_p/W_p$$

and

$$Z_{p.d.} = L_n/W_n$$

With

$$V_{inv} = 0.5\ V_{DD}$$

$$V_{tn} = |V_{tp}| = 0.2\ V_{DD}$$

$$V_{DD} = 5\text{V}$$

$$\mu_n = 2.5\ \mu_p$$

we obtain

$$\frac{Z_{p.u.}}{Z_{p.d.}} = \frac{3}{1}$$

A transfer characteristic, $V_{out}$ vs $V_{in}$, can be drawn and, as for the nMOS case, the characteristic will shift with changes of $Z_{p.u.}/Z_{p.d.}$ ratio.

Two points require comment:

1. Since the channel sheet resistance of the p-pull-up is about 2.5 times that of the n-pull-down, and allowing for the ratio of 3:1, the pseudo-nMOS inverter presents a resistance between $V_{DD}$ and $V_{SS}$ which is, say, 85 kΩ compared with 50 kΩ for a comparable 4:1 nMOS device. Thus, power dissipation is reduced to about 60% of that associated with the comparable nMOS device.
2. Owing to the higher pull-up resistance, the inverter pair delay is larger by a factor of 8.5:5 than the 4:1 minimum size nMOS inverter.

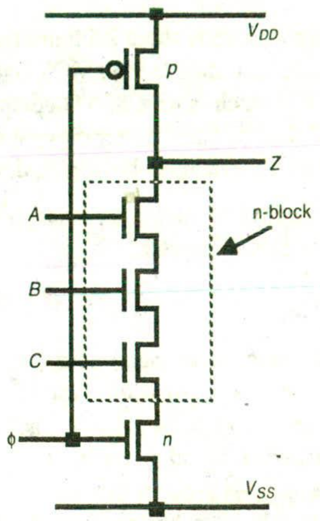
### *6.3.4.2 Dynamic CMOS logic*

The actual logic (see Figure 6–11(a) for the schematic arrangement) is implemented in the inherently faster nMOS logic (the n-block); a p-transistor is used for the non-time-critical precharging of the output line 'Z' so that the output capacitance is charged to $V_{DD}$ during the off period of the clock signal $\phi$. During this same period the inputs are applied to the n-block and the state of the logic is then evaluated during the on period of the clock when the bottom n-transistor is turned on. Note the following:

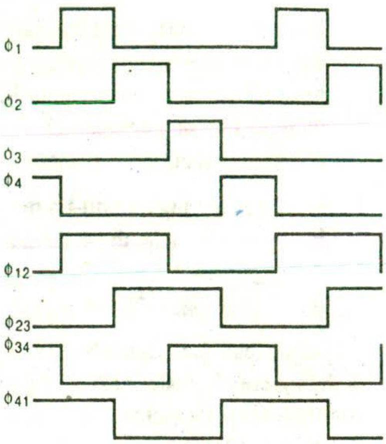
1. Charge sharing may be a problem unless the inputs are constrained not to change during the on period of the clock.
2. *Single phase dynamic logic structures cannot be cascaded* since, owing to circuit delays, an incorrect input to the next stage may be present when evaluation begins, so that its output is inadvertently discharged and the wrong output results.

One remedy is to employ a four-phase clock in which the actual signals used are the derived clocks $\phi_{12}$, $\phi_{23}$, $\phi_{34}$, and $\phi_{41}$, as illustrated in Figure 6–11(b).

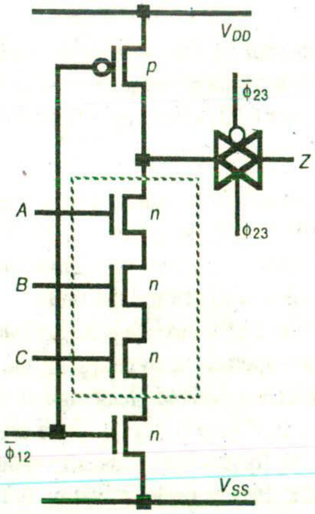
The basic circuit of Figure 6–11(a) is modified by the inclusion of a transmission gate as in Figure 6–11(c), the function of which is to sample the output during the 'evaluate' period and to hold the output state while the next stage logic evaluates. For this strategy to work, the next stage must operate on overlapping but later clock signals. Clearly, since there are four different derived clock signals which are used in sequential pairs (e.g. $\phi_{12}$ and $\phi_{23}$ in Figure 6–11(c)), there are four different gate clocking configurations. These configurations are usually identified by a type number which reflects the last of the clock periods activating the gate. For example, the gate shown would be identified as 'type 3' since the output Z is precharged during $\phi_2$ and is evaluated during $\phi_3$ (the transmission gate is clocked by $\phi_{23}$). In order to avoid erroneous evaluations, the gates must be connected in allowable sequences as set out in Table 6–1.



(a) Schematic



(b) Possible 4ϕ and derived clocks



(c) Type 3 arrangement

**Figure 6–11** Dynamic CMOS logic three-input *Nand* gate

**Table 6–1** Dynamic logic types and sequences

| *Gate type* | *Evaluate clock* | *Transmission gate clock* | *Allowable next types* |
|---|---|---|---|
| Type 1 | $\bar{\phi}_{34}$ | $\phi_{41}$ | Types 2 or 3 |
| Type 2 | $\bar{\phi}_{41}$ | $\phi_{12}$ | Types 3 or 4 |
| Type 3 | $\bar{\phi}_{12}$ | $\phi_{23}$ | Types 4 or 1 |
| Type 4 | $\bar{\phi}_{23}$ | $\phi_{34}$ | Types 1 or 2 |

### *6.3.4.3 Clocked CMOS (C²MOS) logic*

The general arrangement may be made clearer by Figure 6–12. The logic is implemented in both n- and p-transistors in the form of a pull-up p-block and a complementary n-block pull-down structure (Figure 6–12(a)), as for the inverter-based CMOS logic discussed earlier. However, the logic in this case is evaluated (connected to the output) only during the on period of the clock. As might be expected, a clocked inverter circuit forms part of this family of logic as shown in Figure 6–12(b). Owing to the extra transistors in series with the output, slower rise-times and fall-times can be expected.

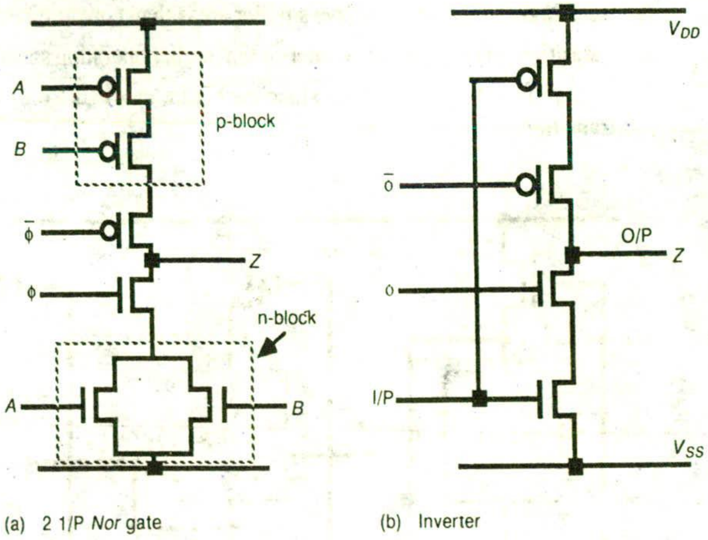


**Figure 6–12** Clocked CMOS (C²MOS) logic

### *6.3.4.4 CMOS domino logic*

An extension to the dynamic CMOS logic discussed earlier is set out in Figure 6–13. This modified arrangement allows for the cascading of logic structures

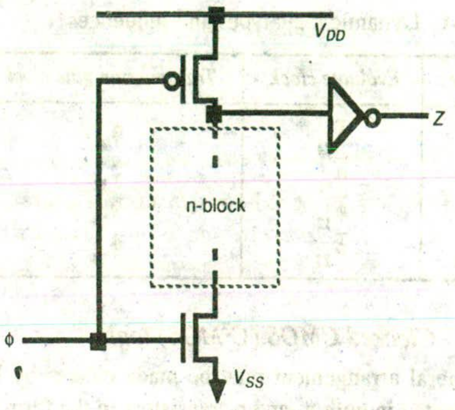


**Figure 6–13** CMOS domino logic

using only a single phase clock. This requires a static CMOS buffer in each logic gate.

The following remarks will help to place this type of logic in the scheme of things:

1. Such logic structures can have smaller areas than conventional CMOS logic.
2. Parasitic capacitances are smaller so that higher operating speeds are possible.
3. Operation is free of glitches since each gate can make only one '1' to '0' transition.

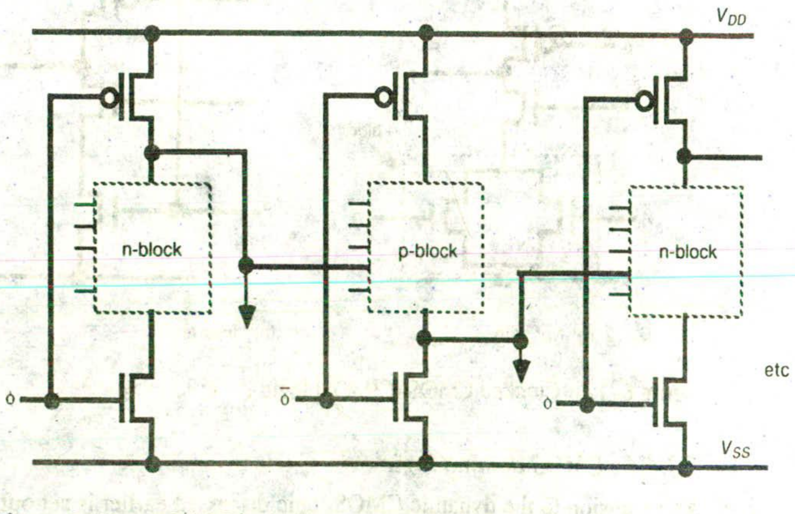


**Figure 6–14** n-p CMOS logic

4. Only non-inverting structures are possible because of the presence of the inverting buffer.
5. Charge distribution may be a problem and must be considered.

#### 6.3.4.5 n-p CMOS logic

This is another variation of basic dynamic logic arrangement, in which the actual logic blocks are alternately 'n' and 'p' in a cascaded structure as in Figure 6–14. The precharge and evaluate transistors are fed from the clock $\phi$ and clockbar $\bar{\phi}$ alternately, and clearly the functions of the top and bottom transistors also alternate, between precharge and evaluate.

Other forms of CMOS logic are also possible, but this text does not attempt to give an exhaustive treatment.

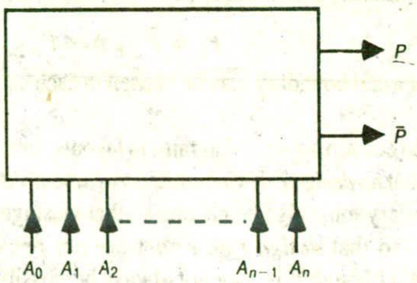
## 6.4 Examples of structured design (combinational logic)

The best way to illustrate the nature of and approach to structured design is to work through some examples.

### 6.4.1 A parity generator

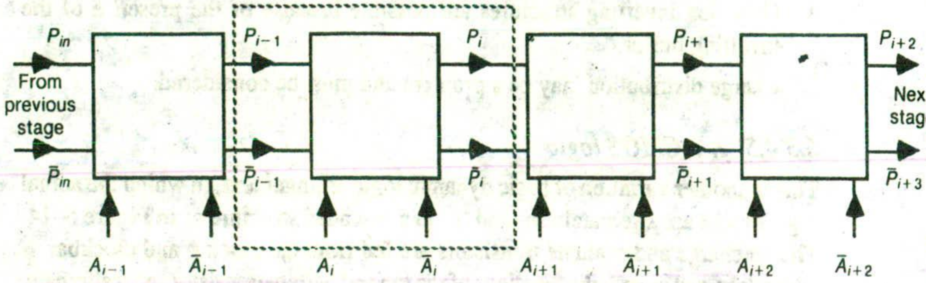
A circuit is to be designed to indicate the parity of a binary number or word. The requirement is indicated in Figure 6–15 for an $(n + 1)$-bit input.

Since the number of bits is undefined, we must find a general solution on a cascadable bit-wise basis so that $n$ can have any value. A suitably regular structure is set out in Figure 6–16. From this, we may recognize a standard or basic one-bit cell from which an n-bit parity generator may be formed. Such a cell is shown in Figure 6–17.



Note: $P = \begin{cases} 1 & \text{Even number of 1s at input} \\ 0 & \text{Odd number of 1s at input} \end{cases}$

**Figure 6–15** Parity generator basic block diagram



*Note*: Parity requirements are set at the left-most cell where $P_{in} = 1$ sets even and $P_{in} = 0$ sets odd parity.

**Figure 6–16** Parity generator — structured design approach

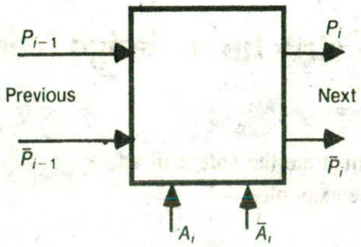


**Figure 6–17** Parity generator — basic one-bit cell

It will be seen that parity information is passed from one cell to the next and is modified or not by a cell, depending on the state of the input lines $A_i$ and $\overline{A}_i$.

A little reflection will readily reveal that the requirements are

$$A_i = 1 \text{ parity is changed, } P_i = \overline{P}_{i-1}$$
$$A_i = 0 \text{ parity is unchanged, } P_i = P_{i-1}$$

A suitable arrangement for such a cell is given in stick diagram form in Figure 6–18(a) (nMOS) and 6–18(b) (CMOS). The circuit implements the function

$$P_i = \overline{P}_{i-1} \cdot A_i + P_{i-1} \cdot \overline{A}_i$$

Note that a cell boundary may be chosen in each case so that cells may be cascaded at will.

When converting stick diagrams to layouts, care must be taken that the boundary is set so that *no design rule violations occur when cells are butted together.* Obviously, the boundary must also be chosen so that wastage of area is avoided and, where possible, so that *design rule errors are not present when a cell is checked in isolation,* although this may not always be possible.

Also, note that inlet and corresponding outlet points should match up both in *layer* and *position,* so that direct interconnection between cells is achieved when cells are butted.

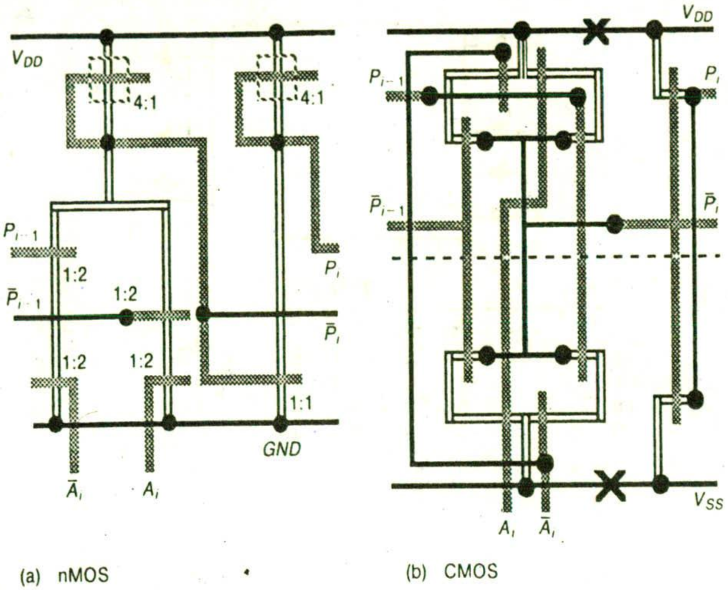


**Figure 6–18** Stick diagrams (parity generator)

## 6.4.2 Bus arbitration logic for n-line bus

(This example and its solutions are similar to an example accredited to Professor John Newkirk in VTI course material.)

The functional requirements of this circuit are given by Figure 6–19 and associated truth table. If the highest priority line $A_n$ is Hi (Logic 1), then output line $A_n^p$ will be Hi and all other output lines Lo (Logic 0), irrespective of the state of the other input lines $A_1$ --- $A_{n-1}$. Similarly, $A_{n-1}^p$ will be Hi only when $A_{n-1}$ is Hi and $A_n$ is Lo; again the state of all input lines of lower priority ($A_1$ ----$A_{n-2}$) will have no effect and all other output lines will be Lo.

This requirement can be expressed algebraically as follows:

$$A_n^p = A_n$$

$$A_{n-1}^p = \bar{A}_n . A_{n-1} \qquad [\overline{A}_{n-1}^p = A_n + \bar{A}_{n-1}]$$

$$A_{n-2}^p = \bar{A}_n . \bar{A}_{n-1} . A_{n-2} \qquad [\overline{A}_{n-2}^p = A_n + A_{n-1} + \bar{A}_{n-2}]$$

$$\vdots \qquad\qquad \vdots$$

$$A_n^p = \bar{A}_n . \bar{A}_{n-1} . \bar{A}_{n-2} \text{----------} \bar{A}_3 . \bar{A}_2 . A_1 \qquad \text{(etc.)}$$

$A_n$
$A_3$
$A_2$
$A_1$
$A_n^P$
$A_3^P$
$A_2^P$
$A_1^P$

Truth table

| $A_n$ | ..... | $A_3$ | $A_2$ | $A_1$ | $A_n^P$ | ..... | $A_3^P$ | $A_2^P$ | $A_1^P$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | ..... | 0 | 0 | 0 | 0 | ..... | 0 | 0 | 0 |
| 0 | | 0 | 0 | 1 | 0 | | 0 | 0 | 1 |
| 0 | | 0 | 1 | X | 0 | | 0 | 1 | 0 |
| 0 | | 1 | X | X | 0 | | 1 | 0 | 0 |
| ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ | | | | ⋮ |
| 1 | ..... | X | X | X | 1 | ..... | 0 | 0 | 0 |

*X = Don't care

**Figure 6–19** Bus arbitration logic and truth table

A direct but unstructured implementation of these expressions may be readily envisaged and a suitable arrangement of switch (pass transistor) logic is given in Figure 6–20.

This implementation seems the obvious one, but it does suffer from the fact that as the input line under consideration moves down in significance so the complexity of the logic grows. For example, we have shown only the top three lines in Figure 6–20, but it will be seen that:

| | |
|---|---|
| $A_n$ | requires one diffusion path and no switches |
| $A_{n-1}$ | requires two diffusion paths and two switches |
| $A_{n-2}$ | requires three diffusion paths and four switches |

and so on.

This is not a regular structure and is not well suited for VLSI implementation. Therefore, we must take a cellular approach by setting out the requirements in alternative fashion as in Figure 6–21.

A regular structure having been arrived at, the requirements for each cell may be expressed as follows:

$$A_i^P \begin{cases} = g_{i+1} \text{ if } A_i = 1 \\ \text{or } 0 \text{ otherwise} \end{cases}$$

$$g_i \begin{cases} = 0 \quad \text{if } A_i = 1 \\ \text{or } g_{i+1} \text{ otherwise} \end{cases}$$

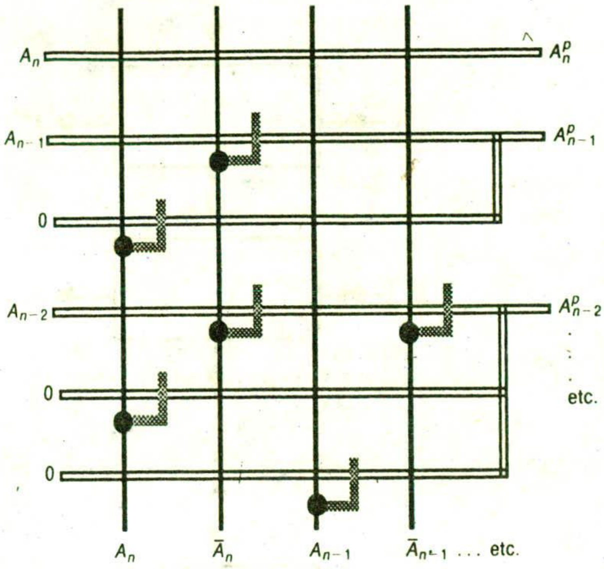


**Figure 6–20** Stick diagram — bus arbitration logic

These requirements may be met by the circuit of Figure 6–22, but care must be taken not to cascade more than four cells without buffering the grant line.

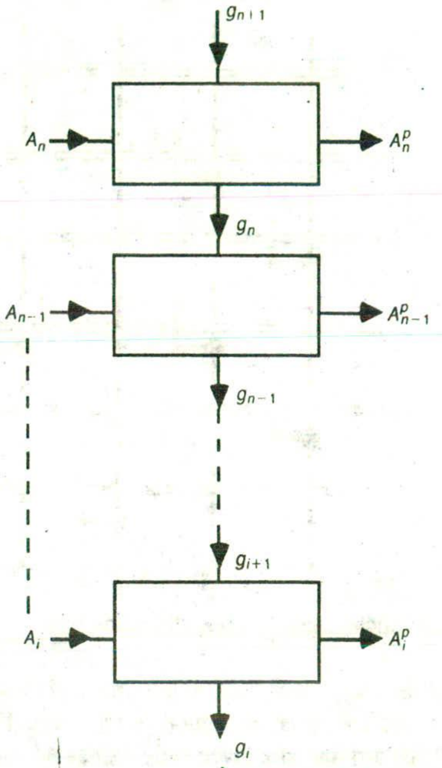
The art of arriving at conveniently expressed relationships which allow a sructured design is one which must be cultivated and it is often helped by adopting an 'if, then, else (or otherwise)' approach. The solution to the problem under consideration can be formulated after expressing the need of each cell in words:

$$\left.\begin{array}{r} \text{If } A_i = 1 \text{ then } A_i^p = g_{i+1}, \\ \text{else } A_i^p = 0 \text{ (if } A_i = 0) \\ \text{If } A_i = 0 \text{ then } g_i = g_{i+1} \\ \text{else } g_i = 0 \text{ (if } A_i = 1) \end{array}\right\} \quad \text{both } A_i^p \text{ and } g_i \text{ can be derived from } g_{i+1}$$

From which we could deduce

$$A_i^p = A_i \cdot g_{i+1}$$
$$g_i = \overline{A}_i \cdot g_{i+1}$$

However, there is a danger with expressions of the conventional Boolean type — a tendency to ignore the fact that MOS *switch* logic is such that not only must the logic 1 condition be satisfied, but it is also necessary to deliberately *satisfy* the logic 0 conditions. The TTL logic designer is used to working with logic circuits in which the output must be logic 0 if the logic 1 output conditions are not satisfied. However, some MOS switch-based logic circuits have the property



*Notes:* 1. $g$ = grant line.
2. If grant line is 1, none of the lines above it wants priority.
3. If $A_i = 0$, pass grant.

**Figure 6–21** Bus arbitration logic — structured design

that if the logic 1 output conditions are not met, then the output can be indeterminate or, if some storage capacitance is present (for example, input capacitance $C_g$ of an inverter), then the output can remain at logic 1 even after the conditions which caused it no longer exist. Thus, it is necessary to deliberately implement the 'else' conditions. We must, therefore, write expressions for both the logic 1 and logic 0 conditions of the output lines, thus

$$A_i^p = A_i \cdot g_{i+1};\ \overline{A}_i^p = \overline{A}_i + \overline{g}_{i+1}$$
$$g_i = \overline{A}_i g_{i+1};\ \overline{g}_i = A_i + \overline{g}_{i+1}$$

which is the circuit realized in Figure 6–22. This circuit is suitable for implementation in nMOS or in CMOS technology. Although in the CMOS case there is the possibility of replacing the n-type pass transistors by transmission gates, there is no advantage

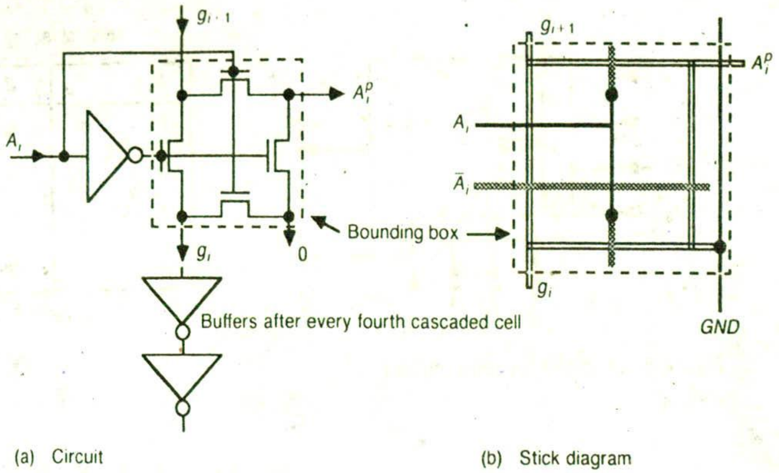


**Figure 6–22** Bus arbitration logic — structured design

to be gained from this as the degrading of logic 1 level is counteracted by the presence of buffers after every fourth cell. There is clearly an area advantage in using simple n-type pass transistors and the only difference, therefore, between an nMOS and a CMOS design will be the type of buffer (inverter) stages.

## 6.4.3 Multiplexers (data selectors)

Multiplexers are widely used and have many applications. They are also commonly available in a number of standard configurations in TTL and other logic families. In order to arrive at a standard cell for multiplexers, we will consider a commonly used circuit, the four-way multiplexer.

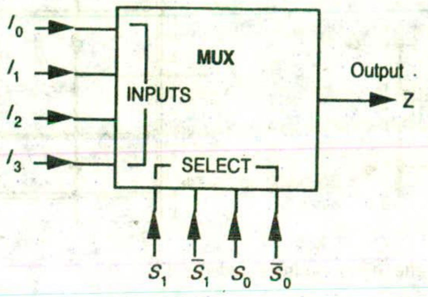
The requirements and general arrangement of a four-way multiplexer are set out in Figure 6–23, from which we may write

$$Z = I_0.\bar{S}_1.\bar{S}_0 + I_1.\bar{S}_1.S_0 + I_2.S_1.\bar{S}_0 + I_3.S_1.S_0$$

where $S_1$ and $S_0$ are the selector inputs. Note that in this case we do not need to be concerned about undefined ouput conditions since, if $S_1$ and $S_0$ have defined logic states, output Z must always be connected to one of $I_0$ to $I_3$.

Thus, a direct n-switch logic implementation follows which is given as Figure 6–24(a) in stick diagram form with a standard-cell-based mask layout following as Figure 6–25 and in Color plate 10.

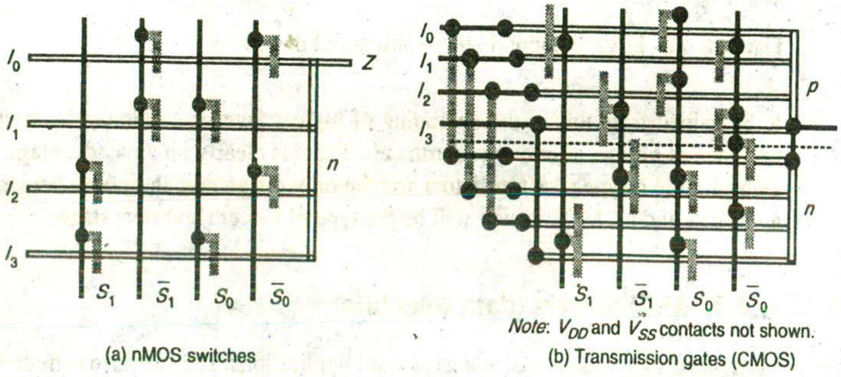
A transmission-gate-based CMOS stick diagram is given in Figure 6–24(b). A mask layout of this figure appears as Color plate 11 and it can be seen that all n-transistors are placed below the demarcation line and close to the $V_{SS}$ rail to allow ready configuration of the p-well and $V_{SS}$ contacts. The p-transistors are



Truth table

| $S_1$ | $S_0$ | $Z$ |
|---|---|---|
| 0 | 0 | $I_0$ |
| 0 | 1 | $I_1$ |
| 1 | 0 | $I_2$ |
| 1 | 1 | $I_3$ |

**Figure 6–23** Selector logic circuit



$I_0$
$I_1$
$I_2$
$I_3$
Z
n
$S_1$ $\bar{S}_1$ $S_0$ $\bar{S}_0$
(a) nMOS switches
$I_0$
$I_1$
$I_2$
$I_3$
p
n
$S_1$ $\bar{S}_1$ $S_0$ $\bar{S}_0$
Note: $V_{DD}$ and $V_{SS}$ contacts not shown.
(b) Transmission gates (CMOS)

**Figure 6–24** Switch logic implementations of a four-way multiplexer

similarly placed above the notional demarcation line and close to $V_{DD}$. Note that logic 1 levels will not be degraded by this arrangement as those in the nMOS version are.

Now, if we can establish standard cells from which a four-way multiplexer can be composed, then we will also cover the case of the two-way multiplexer. Such a cell will, by inference, also be suitable for constructing an 8-way or a 16-way multiplexer.

For the nMOS case a standard cell is illustrated in Figure 6–25. The standard cell in this case measured $7\lambda \times 11\lambda$ and is shown in the dotted outline. Note that two versions of the cell are needed to complete the network, one version with a pass transistor as shown and the other version without. If computer-aided design tools are used, the two versions may be designed as one cell suitably parameterized to include or exclude the pass transistor. Note that in Figure 6–25 the dimensions do not include the end connection to Z.

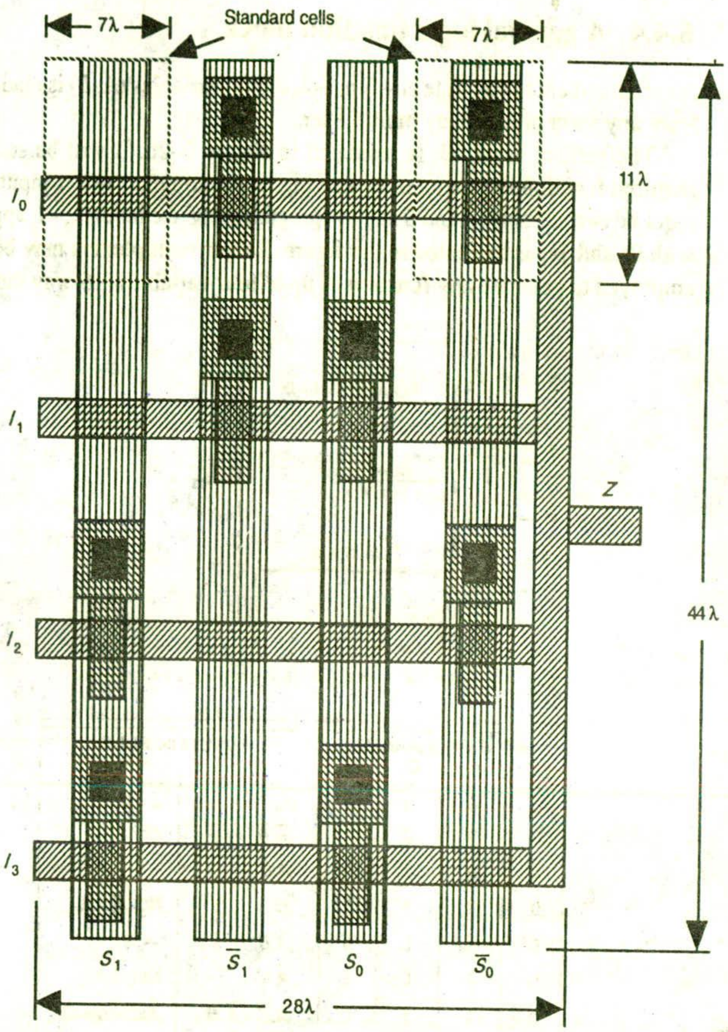


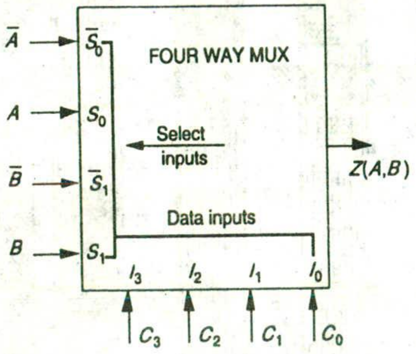
**Figure 6–25** Four-way n-switch based multiplexer (MUX) layout (see also Color plate 10)

Note also that this layout places the metal select lines over the top of pass transistors. This practice is acceptable in this situation where a transistor gate is actually driven from and connected to the particular metal line which runs across it. This method of economizing in area must be used with caution when locating transistors under metal layers to which they are not connected, and may not be acceptable when the underlying transistors are used as storage points to hold a charge and retain a logic level.

## 6.4.4 A general logic function block

An arrangement to generate any function of two variables ($A$, $B$) is readily formed from any form of four-way multiplexer.

The general approach is indicated in Figure 6–26. It will be seen that the required function is generated by driving the multiplexer select inputs from the required two variables $A$ and $B$ and by 'programming' the inputs $I_0 - I_3$ appropriately with 0s and 1s, as indicated in the figure. Larger multiplexers may be similarly employed to generate any function of up to four variables (16-way multiplexer).



| INPUT PROGRAMMING | | | | FUNCTION $Z(A,B)$ | |
|---|---|---|---|---|---|
| $C_3$ | $C_2$ | $C_1$ | $C_0$ | | |
| 0 | 0 | 0 | 0 | 0 | $Z = 0$ |
| 0 | 0 | 0 | 1 | $\bar{A}.\bar{B}; \overline{A+B}$ | *Nor* |
| 0 | 0 | 1 | 0 | $A.\bar{B}$ | |
| 0 | 0 | 1 | 1 | $\bar{B}$ | Not $B$ |
| 0 | 1 | 0 | 0 | $\bar{A}.B$ | |
| 0 | 1 | 0 | 1 | $\bar{A}$ | Not $A$ |
| 0 | 1 | 1 | 0 | $A.\bar{B}+\bar{A}.B$ | *Exclusive-Or* |
| 0 | 1 | 1 | 1 | $\bar{A}+\bar{B}; \overline{AB}$ | *Nand* |
| 1 | 0 | 0 | 0 | $A.B$ | *And* |
| 1 | 0 | 0 | 1 | $\bar{A}.\bar{B}+A.B$ | Comparator |
| 1 | 0 | 1 | 0 | $A$ | *O/P = A* |
| 1 | 0 | 1 | 1 | $A+\bar{B}$ | |
| 1 | 1 | 0 | 0 | $B$ | *O/P = B* |
| 1 | 1 | 0 | 1 | $\bar{A}+B$ | |
| 1 | 1 | 1 | 0 | $A+B$ | *Or* |
| 1 | 1 | 1 | 1 | 1 | $Z = 1$ |

**Figure 6–26** General logic function block (two variables)

## 6.4.5 A four-line Gray code to binary code converter

As a further exercise, which employs a very widely used logic arrangement (the *Exclusive-Or* gate), consider the requirement for code conversion from Gray to binary as set out in Table 6–2.

By inspecting (or mapping from) Table 6–2, it will be seen that the following expressions relate the two codes:

$$\left.\begin{aligned} A_0 &= \overline{G}_0.A_1 + G_0.\overline{A}_1 \\ A_1 &= \overline{G}_1.A_2 + G_1.\overline{A}_2 \\ A_2 &= \overline{G}_2.A_3 + G_2.\overline{A}_3 \end{aligned}\right\}\ \textit{Exclusive-Or}\text{ operations}$$

$$A_3 = G_3$$

A suitable arrangement is set out in Figure 6–27, and the only detailed design required is that of a two input *Exclusive-Or* gate. Many arrangements are possible to implement this operation, but let us consider an *Exclusive-Or* gate made up of standard logic gates, as in Figure 6–28.

Table 6–2 Gray to binary code conversion

| *Gray code* | | | | *Binary code* | | | |
|---|---|---|---|---|---|---|---|
| $G_3$ | $G_2$ | $G_1$ | $G_0$ | $A_3$ | $A_2$ | $A_1$ | $A_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

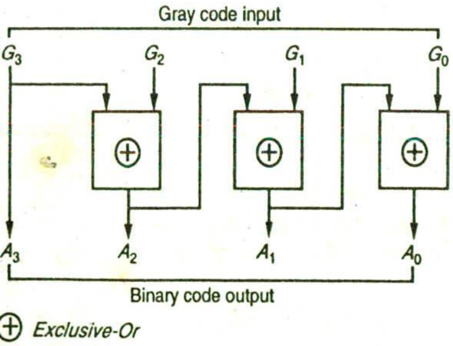
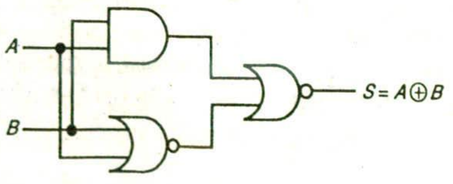


Figure 6–27 Gray to binary code converter



Truth table

| *Inputs* | | *Output* |
|---|---|---|
| *A* | *B* | *S* |
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

Figure 6–28 One possible arrangement for an *Exclusive-Or* gate

A mask layout for this arrangement is presented in Figure 6–29; note that the p-well and $p^+$ mask outlines are included in the layout together with the p-well and substrate contacts. Simulation of the design yields the results given in Figure 6–30. The simulator used was the ISD* program PROBE™ and the circuit extractor NET™. Likely circuit delays can be seen quite plainly.

### 6.4.6 The programmable logic array (PLA)

This arrangement provides a general, structured and regular way of mapping multiple output combinational logic expressions onto silicon. See Appendix C.

## 6.5 Some clocked sequential circuits

### 6.5.1 Two-phase clocking

The clocked circuits to be considered here will be based on a two-phase non-overlapping clock signal as defined by Figure 6–31.

A two-phase clock offers a great deal of freedom in sequential circuit design if the clock period and the duration of the signals $\phi_1$ and $\phi_2$ are correctly chosen. If this is the case, data is allowed to become stable before any further transfer takes place and there is no chance of race conditions occurring.

Clocked circuitry is considerably easier to design than the corresponding asynchronous sequential circuitry. It does, however, usually pay the penalty of being slower. However, at this stage of learning VLSI design we will concentrate on two-phase clocked sequential circuits alone and thus simplify design procedures. When studying Figure 6–31, it is necessary to recognize the fact that $\phi_1$ and $\phi_2$ do not need to be symmetrical as shown. For a given clock period, each clock phase period and its associated underlap period can be varied if the need arises in optimizing a design.

A number of techniques are used to generate the two clock phases. One popular method is illustrated in Figure 6–32 and it will be seen that the output frequency is one-quarter of that of the input clock.

A very simple arrangement using combinational logic and generating a two-phase clock at the frequency of a single-phase input clock is set out in Figure 6–33(a). The input clock signal $C$ is used to provide a delayed version of itself ($CD$) by passing it through an even number of inverters. The delay thus produced determines the underlap period for the two-phase clock. Waveforms are as shown in Figure 6–33(b). The phase 1 signal $\phi_1$ (PH1) is generated by *Anding* $C$ with $CD$ whilst the phase 2 signal $\phi_2$ (PH2) is produced by *Noring* $C$ with $CD$ (that is, *Anding* $C'$ with $CD'$). Clearly, the minimum underlap period will be that generated

* ISD refers to Integrated Silicon Design Pty Ltd of Adelaide, South Australia. The use of their design tools is acknowledged with thanks and appreciation.

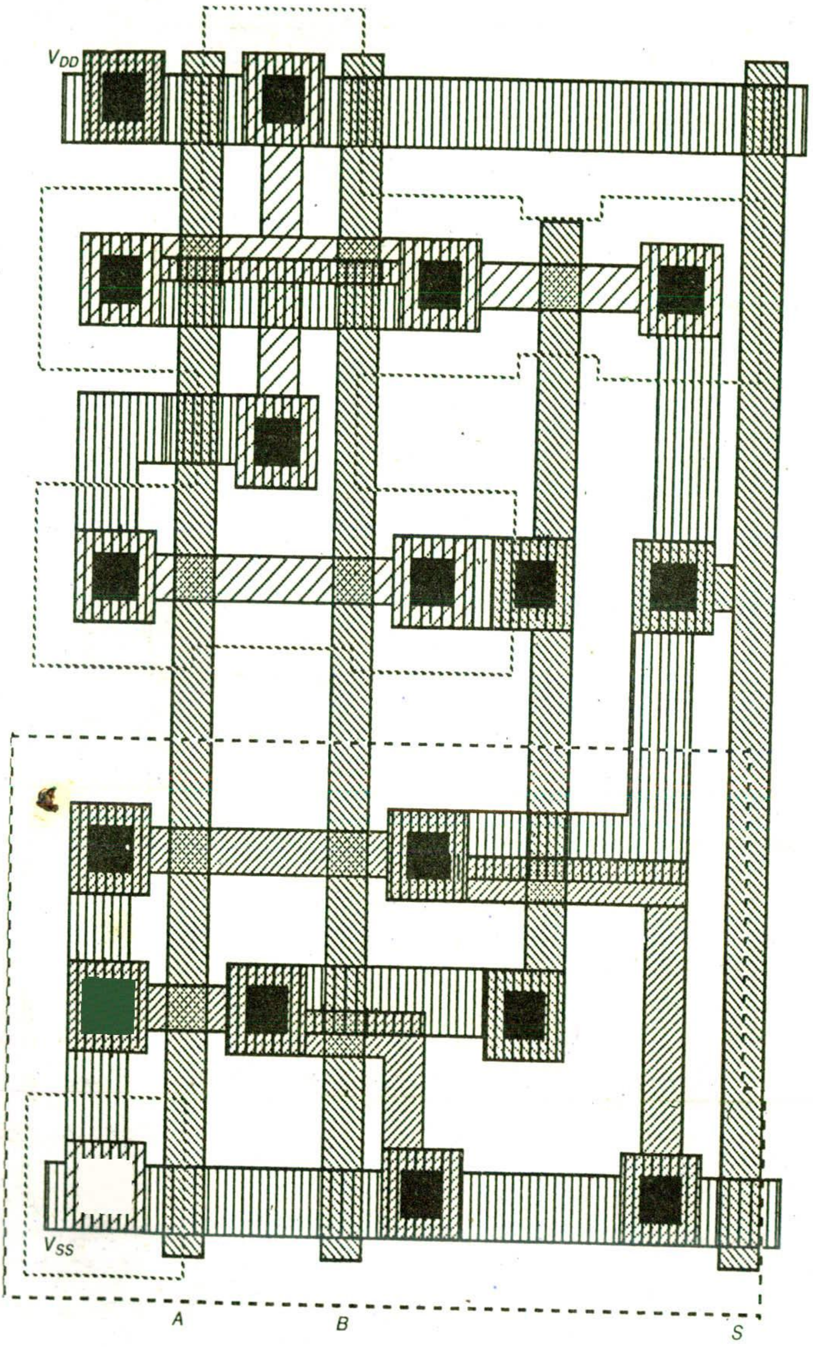


**Figure 6–29** Mask layout for *Exclusive-Or* gate of Figure 6–28

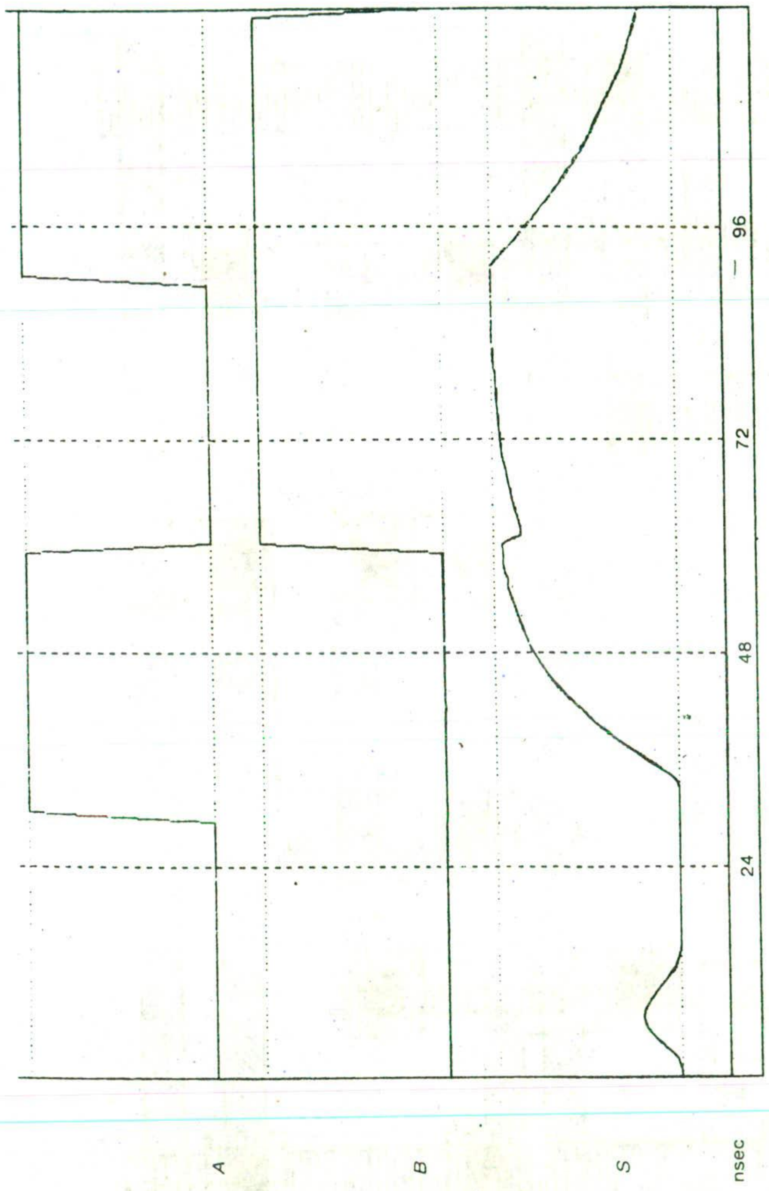
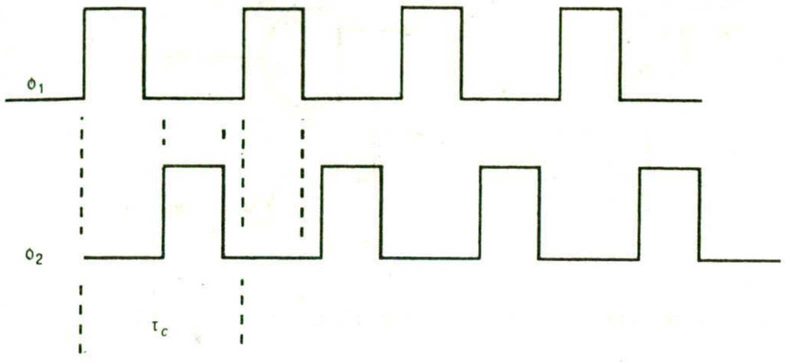


**Figure 6–30** Simulation results for *Exclusive-Or* gate



*Notes:* 1. $\tau_c$ = clock period
2. $\phi_1(t).\phi_2(t) = 0$: *all* $t$

**Figure 6–31** Two-phase clocking

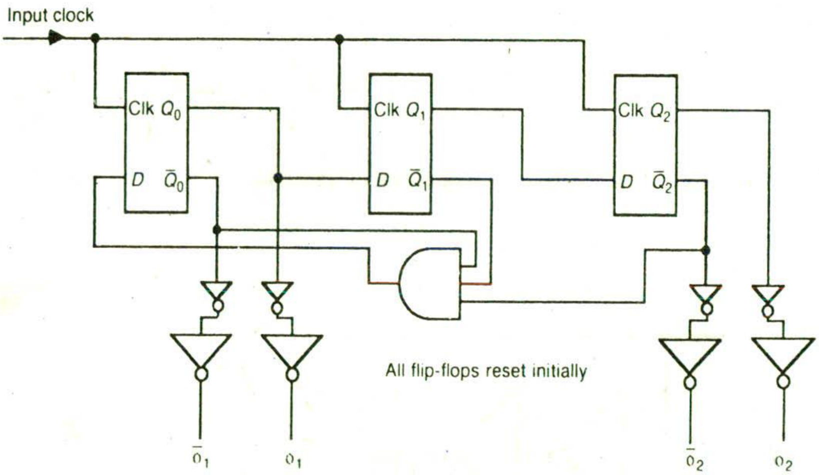


**Figure 6–32** Two-phase clock generator using D flip-flops

by the delay through two inverters and this is also the increment by which the delay may be increased by adding further inverter pairs.

Since clock lines often feed many stages and are associated with long bus lines, they often present quite considerable capacitance to the clock line drivers. Here then is a case where a bipolar capability can be used to advantage to drive the high capacitance load. This approach is demonstrated in Figure 6–34, which uses bipolar-based output stages and also produces the complements of the two phases since complementary clocks are almost invariably required. Simulation waveforms are given in Figure 6–34(b) and a possible mask layout is presented as Color plate 12.

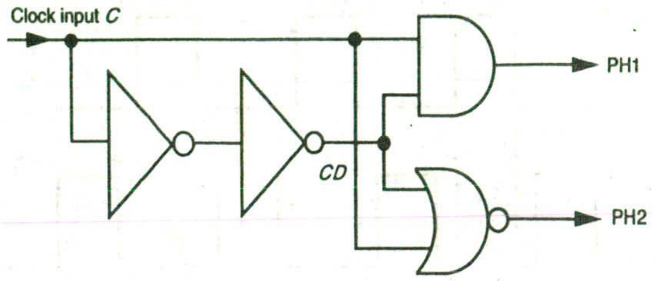


**Figure 6–33(a)** Simple two-phase clock generator circuit — basic form

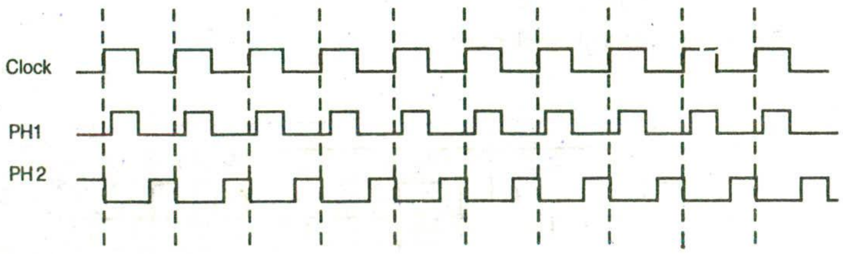


**Figure 6–33(b)** Waveforms for two-phase clock generator

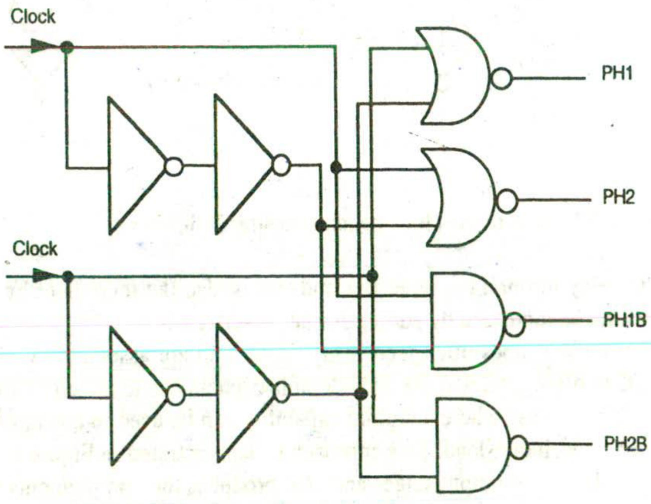


**Figure 6–34(a)** Two-phase clock generator (with complementary outputs) for BiCMOS logic implementation

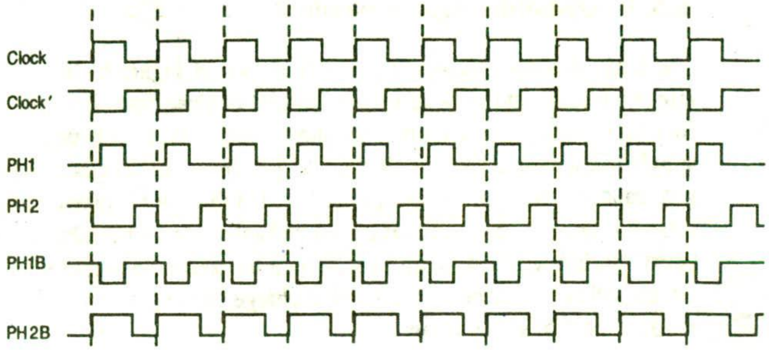


**Figure 6–34(b)** Waveforms for circuit of Figure 6–34(a)

## 6.5.2 Charge storage

A necessary feature of sequential circuits is a facility to remember or take account of previous conditions. An obvious area of application of such a facility is in memory elements, registers, finite state machines, etc.

MOS technology takes advantage of the excellent insulating properties of silicon dioxide layers on integrated circuits to store charges in capacitors, including the gate-channel capacitance of transistors. Such storage is known as *dynamic storage* since, in a reasonably short time, stored charges will leak away and will have to be refreshed if data/conditions are to be retained.

Considering charges on the gate capacitance, the leaking away of the charge is mainly due to leakage currents $I_s$ across the channel to substrate reversed biased diode. At room temperature and for typical 5 μm dimensions and typical voltages, this current is in the order of 0.1 nA, and so an approximate idea of holding time can be obtained from the simple circuit model of Figure 6–35 which considers 1□$Cg$ initially charged to 5 volts.

This simple model indicates storage times of up to, say, 0.25 msec to discharge from $V_{DD}$ to $V_{inv}$ (= 0.5 $V_{DD}$), but it should be noted that current $I_s$ doubles for every 10°C rise in temperature so that the storage time is halved.

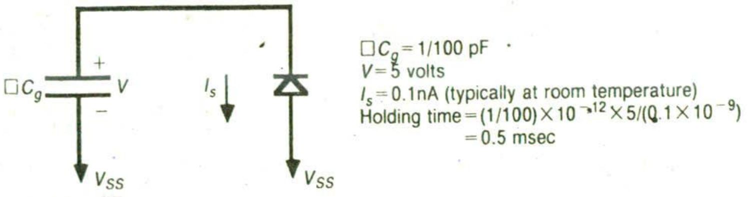

□$C_g$
+
V
−
$I_s$
$V_{SS}$
$V_{SS}$
□$C_g$ = 1/100 pF
$V$ = 5 volts
$I_s$ = 0.1nA (typically at room temperature)
Holding time = (1/100) × $10^{-12}$ × 5/(0.1 × $10^{-9}$)
= 0.5 msec

**Figure 6–35** Simple stored charge model

## 6.5.3 Dynamic register element

The basic dynamic register element is shown in Figure 6–36 in mixed stick/circuit notation and may be seen to consist of three transistors for nMOS and four for CMOS per stored bit in complemented form. The element's operation is simple to appreciate. $(V_{in})_t$ is clocked in by $\phi_1$ (or $\phi_2$) of the clock and charges the gate capacitance $C_g$ of the inverter to $V_{in}$. If subscript $t$ is taken to represent the time during which $\phi_1$ (say) is at logic 1 and subscript $t+1$ is taken to indicate the period during which $\phi_1$ is at logic 0, then the available output will be $(\overline{V}_{in})_{t+1}$ which will be maintained by the stored charge on the gate until $C_g$ discharges or until the next $\phi_1$ signal occurs.

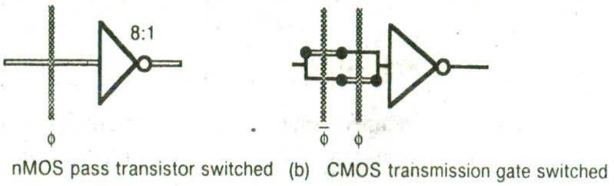

8:1
ϕ
ϕ̄
ϕ
(a) nMOS pass transistor switched
(b) CMOS transmission gate switched

**Figure 6–36** Basic inverting dynamic storage cells

If uncomplemented storage is essential, the basic element is modified as indicated in Figure 6–37 and will be seen to consist of six transistors for nMOS and eight for CMOS. Data clocked in on $\phi_1$ is stored on $C_{g1}$ and the corresponding output appears at the output of inverter 1. On $\phi_2$ this value is clocked into and stored by $C_{g2}$ and the output of inverter 2 then presents the 'true' form of the stored bit. Note that data read in on $\phi_1$ is not available at the output until sometime following the next positive edge of the clock signal $\phi_2$.

Dynamic storage elements and the corresponding register arrays are used in situations where signals are updated frequently (i.e. at < 0.25 msec intervals).

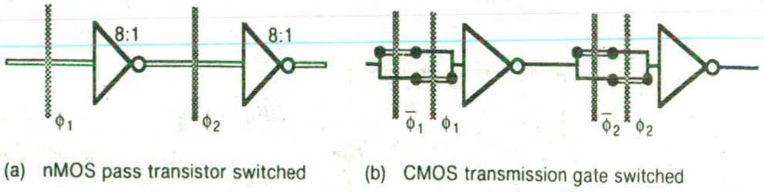

8:1
8:1
ϕ1
ϕ2
ϕ̄1
ϕ1
ϕ̄2
ϕ2
(a) nMOS pass transistor switched
(b) CMOS transmission gate switched

**Figure 6–37** Non-inverting dynamic storage cells

### 6.5.4 A dynamic shift register

Cascading the basic elements of Figure 6–37 gives a serial shift register arrangement which may be extended to *n* bits. A four-bit serial right shift nMOS register is illustrated in Figure 6–38(a). Data bits are shifted in when $\phi_1.LD$ is present, one bit being entered on each $\phi_1$ signal (provided that $LD$ is logic 1). Each bit is stored in $C_{g1}$ as it is entered, and then transferred complemented into $C_{g2}$ during the next $\phi_2$. Thus, after a $\phi_1$ followed by $\phi_2$ signal, the stored bit is present at the output of inverter 2. On the next $\phi_1$, the next input bit is stored in $C_{g1}$ and simultaneously the first bit stored is passed on to inverter pair 3 and 4 by being stored in $C_{g3}$, and so on. It will be seen that bits are thus clocked to the right along the shift register on each $\phi_1$ followed by $\phi_2$ sequence. Once four bits are stored, the data is available in parallel form at the outputs of inverters 2, 4, 6 and 8, and is also available in serial form from the output of inverter 8 when $\phi_1.RD$ is high as further clock sequences are received (where $RD$ is the serial read control signal). The operation of the CMOS version (Figure 6–38(b)) is similar, transmission gates replacing inter-stage pass transistors and $C_{in1}$ replacing $C_{g1}$, etc., as the storage capacitance.

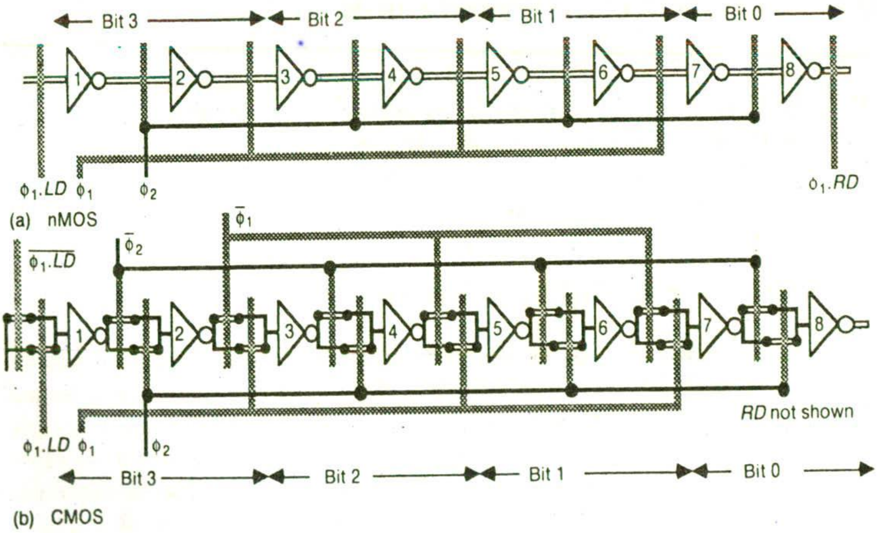
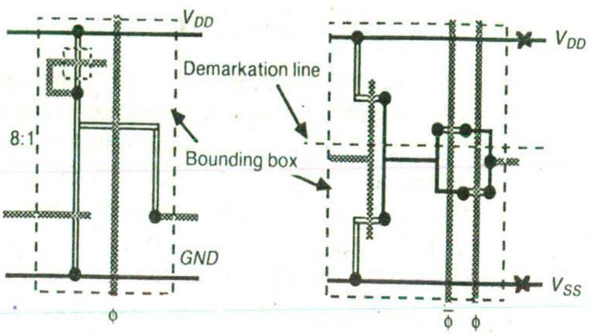


**Figure 6–38** Four-bit dynamic shift registers (nMOS and CMOS)



(a) nMOS (b) CMOS

**Figure 6–39** Stick diagrams for shift register cells

Many variations of this basic arrangement are possible, but in general they are all based on the basic cell consisting of an inverter and a pass transistor or a transmission gate. Suitable standard cells are shown in stick diagram form in Figure 6–39 with the corresponding mask layouts in Figure 6–40. Note that two nMOS layouts are given (using butting and buried contacts respectively) and one possible CMOS layout is suggested (see also Color plate 7).

## 6.6 Other system considerations

When designing at leaf-cell level, it is easy to lose sight of overall system requirements and restrictions. In particular, the use of buses to interconnect subsystems and circuits must always be most carefully considered; such matters and the current-carrying capacity of aluminum wiring used for $V_{DD}$ and $GND$ or $V_{SS}$ rails are often overlooked completely.

### 6.6.1 Bipolar drivers for bus lines

Bus structures carrying data or control signals are generally long and connected to and through a significant number of circuits and subsystems. Thus, the bus capacitances are appreciable and thought must be given to the manner in which any bus line is to be driven. Otherwise, the propagation of signals may be a slow process. Clearly, the capacitive load-driving properties of bipolar transistors in a BiCMOS process make bipolar drivers an attractive proposition for bus lines. However, this must be approached with some caution as the speed of bipolar drivers is only fully realized with bus lines for which there is only one source of drive, for example, as in the case of clock line drivers. Bipolar drivers are not so suitable where one or other of several sources drives a common bus since under

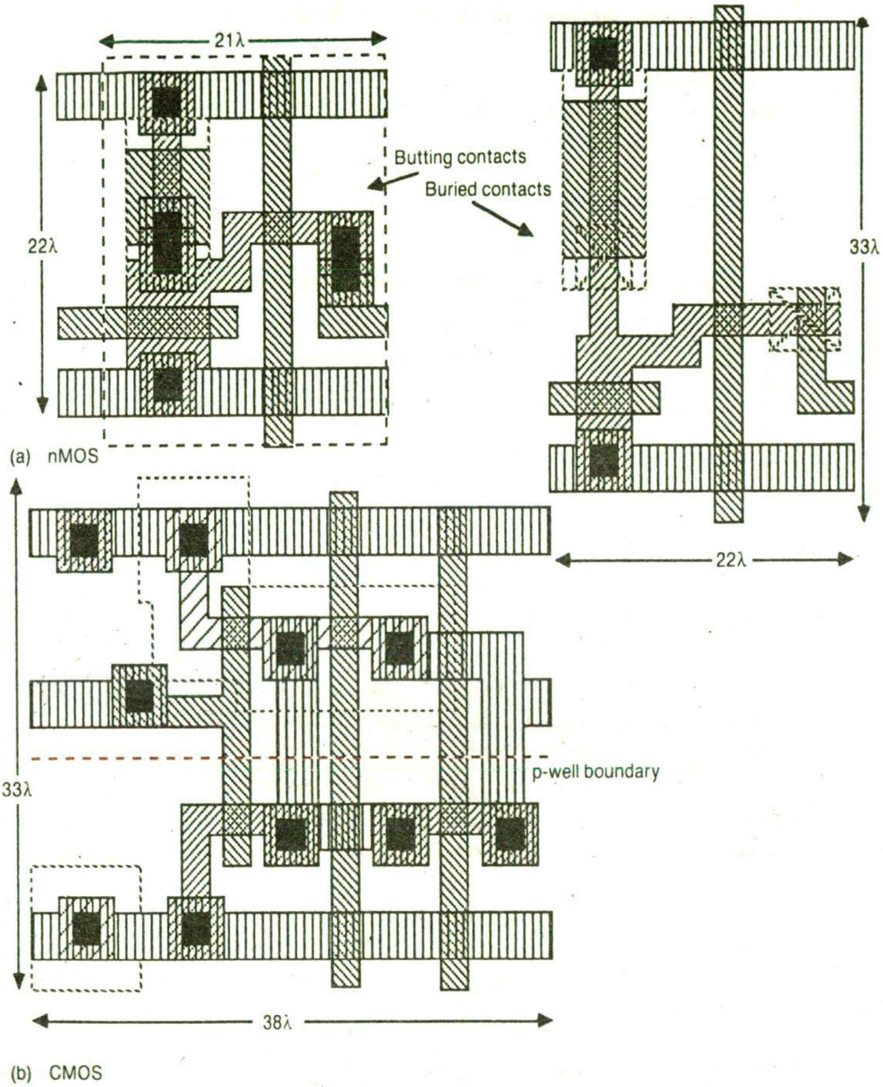
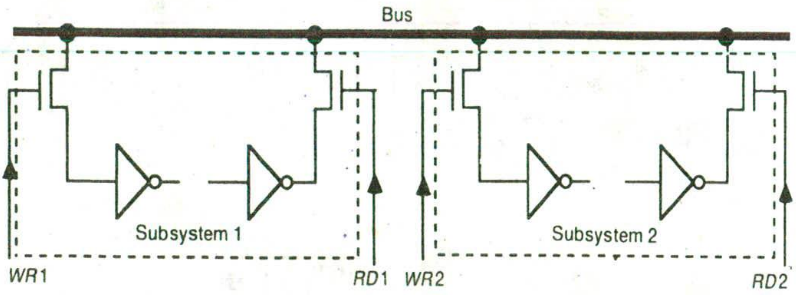


**Figure 6–40** Mask layouts for nMOS and CMOS shift register cells (see also Color plate 7)

those circumstances a series switch must be inserted between each source of drive and the bus. The series resistance of such a switch to a large extent negates the speed advantage. In such cases MOS transistor drivers are often used and the following basic approaches may be considered.

## 6.6.2 Basic arrangements for bus lines

There are three classes of bus — passive, active, and precharged. A *passive* bus rail is a floating rail to which signals may be connected from drivers through series switches, for example, pass transistors, to propagate along the bus and from which signals may be taken, also through pass transistors (see Figure 6–41).



*Note:* For CMOS the pass transistors could become transmission gates.

**Figure 6–41** Passive bus — nMOS or CMOS

A form of *active* bus is to treat the bus rail as a wired *Nor* connection which has a common pull-up $R_{p.u.}$ and n-type pull-down transistors or series n-type transistor logic pull-downs where there are circuits which must be selected to drive the bus. Signals are taken off the bus in a similar manner and the general arrangement is given as Figure 6–42. This arrangement is not suited to complementary CMOS logic-based designs since it is based on pull-down logic only.

The passive bus suffers from ratio problems in that, for any reasonable area restrictions on the bus driver circuits, the bus will be slow to respond, particularly for the $\Delta V$ (logic 0 to 1) transitions, because of the relatively high value pull-up resistance of the drivers and the associated series pass transistor or transmission gate.

The active bus is better in that more time is available for the bus to charge to $V_{DD}$, since $R_{p.u.}$ is always connected to the bus and there are no series pass transistors between $R_{p.u.}$ and the bus. However, there are still ratio problems which limit the speed of the bus if reasonable area is to be occupied.

## 6.6.3 The precharged bus concept

The *precharged* bus approach limits the effects of bus capacitance in that a single pull-up transistor which is turned on only during $\phi_2$ (say) provides for the bus to charge during the $\phi_2$ on period; the size of this transistor can be made relatively

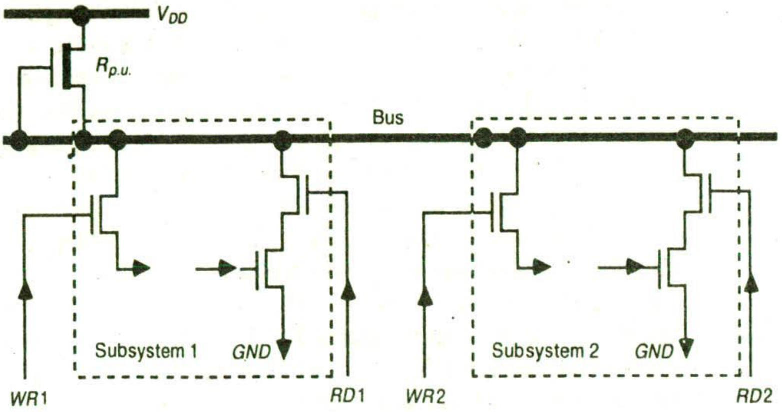


**Figure 6–42** Active bus (not CMOS)

large (i.e. a low $L{:}W$ ratio) and, therefore, have a low resistance. There are no ratio problems between it and the bus drivers since they are never turned on at the same time. The bus drivers merely pull down (or not) the precharged bus by discharging $C_{Bus}$. The arrangement is given at Figure 6–43 and, in effect, a ratioless precharged wired *Nor* circuit is formed by the bus system. However, care must be taken in nMOS systems when using logic 1 levels from the bus since the bus never reaches $V_{DD}$, due to threshold voltage effects in the precharging transistor.

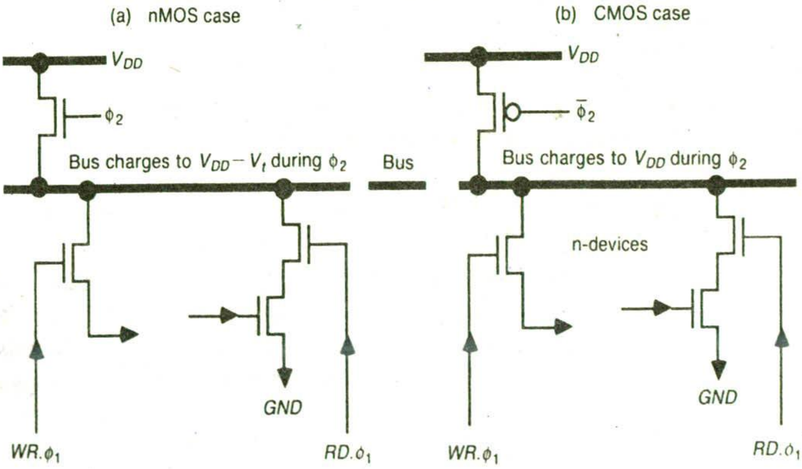


**Figure 6–43** Precharged bus — nMOS and CMOS

*Cross-talk and delay* factors are also of significance in bus design. For example, many signals on chip may be propagated for some considerable distance (in chip dimension terms) along metal buses. Now metal buses even of minimum width are relatively wide (e.g. $3\lambda = 7.5$ μm in 5 μm technology), and thus have significant area capacitance to substrate (almost $2.5 \times 10^{-4}$ pF per μm length for the example). This dces not give rise to serious delay-line effects since the metal exhibits a low resistance (approximately 0.01 Ω/μm for the example) but a metal bus of any length presents a significant capacitive load to the driver. For example, a bus $400\lambda$ (1000 μm) long will present a total $C = 0.25$ pF.

Since metal also has appreciable thickness — typically 1 μm — the edge of a long bus represents a significant area. For the $400\lambda$ long bus considered above the area of each edge will be 1000 μm$^2$. This may give rise to cross-talk noise between two or more buses which run side by side for any appreciable length. This problem is not as serious in silicon chip designs as in GaAs technology, for example, owing to the relatively low dielectric constant (approx. = 4) for the silicon dioxide which will form the dielectric between the edges of two parallel buses.

Bus structures are widely used and will be further discussed in following sections of this text.

## 6.6.4 Power dissipation for CMOS and BiCMOS circuits

For pseudo-nMOS type circuitry, current and power are readily determined in a manner similar to nMOS. However, for complementary inverter-based circuits we may proceed by first recognizing that the very short current pulses which flow when circuits of this type are switching between states are generally negligible in comparison with charge and discharge currents of circuit capacitances. Then we may see that overall dissipation is composed of two terms:

1. $P_1$ the dissipation due to the leakage current $I_1$ through an 'off' transistor. Consequently, for $n$ transistors, we have

$$P_1 = n.I_1 V_{DD}$$

where $I_1 = 0.1$ nA, typically at room temperature.

2. $P_s$ is the dissipation due to energy supplied to charge and discharge the capacitances associated with each switching circuit. Assuming that the output capacitance of a stage can be combined with the input capacitance(s) of the stage(s) it is driving and then represented collectively as $C_L$, then, for $n$ identical circuits switched by a square wave at frequency $f$ it may be shown that

$$P_s = C_L V_{DD}{}^2 f$$

The total power dissipation $P_T$ is thus

$$P_T = P_1 + P_s$$

from which the average current may be deduced.

Power dissipation for bipolar devices can be simply modeled by

$$P = V_{cc} \times I_c$$

where $V_{cc}$ is the supply voltage and $I_c$ is the current through the device.

It may be seen that BiCMOS switching devices will exhibit a constant value for power dissipation, not frequency-dependent like CMOS.

## 6.6.5 Current limitations for $V_{DD}$ and $GND$ ($V_{SS}$) rails

A problem often ignored is that of metal migration for high current densities in metal conductors. If the current density exceeds a threshold value then one finds that metal atoms begin to move in the direction of the current. For aluminum conductors this threshold value is

$$J_{th} \div 1 \text{ to } 2 \text{ mA}/\mu\text{m}^2$$

The danger points occur where there is a narrowing or constriction in the conductor. At these points the current density is at its highest and metal is transported from the constricted regions which, in consequence, become even more constricted and eventually may blow like a fuse. The actual mechanism of atomic transport of metal in a thin film carrying relatively high currents is well understood, but the science of predicting the location and the time of such occurrences is not well developed.

By way of example, we may consider the question of how many nMOS 8:1 inverters (as in a dynamic shift register) can be driven by a minimum size conductor assuming lambda-based rules and 5 μm technology. From the design rules, the metal is 3λ wide, which corresponds to 7.5 μm. The thickness of the conductor is about 1 μm as shown in Figure 6–44.

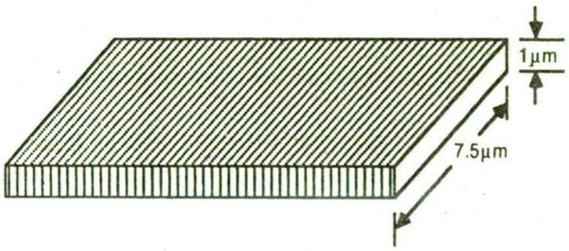


**Figure 6–44** A minimum size metal path or wire

For 8:1 inverter (e.g. 8:1 p.u. and 1:1 p.d.)

$$R = (8+1) \times 10^4\,\Omega$$
$$= 90\ \text{k}\Omega$$

Therefore

$$\text{Current } I = \frac{5}{90} = 0.06\ \text{mA per inverter.}$$

Now, with a wire cross-section of 7.5 $\mu\text{m}^2$, the current density limitation $J_{th} = 1\ \text{mA}/\mu\text{m}^2$ implies that a current of 7.5 mA can be supplied. Thus about 125 inverters can be driven.

One approach that may be pursued to allow some increase in the current density above the specified critical limit is to take advantage of the 'relaxation effect' that occurs in the metal when electron flow occurs in short pulses rather than at a steady state level.

However, the important factor here is that a standard (minimum) width metal conductor can only support a subsystem of quite modest size. Thus, in a design of any complexity we must ensure that this fact is not overlooked and power rail distribution becomes an important and often complex issue.

## 6.6.6 Further aspects of $V_{DD}$ and $V_{SS}$ rail distribution

Ideally, the power distibution rails (power distribution buses) for a chip should provide a constant and equal voltage supply to each and every device on the chip. Rails should also be able to supply the current required by every device. Clearly, these ideals are not achievable in practice and issues which determine the limitations are:

1. metal migration imposed current density restrictions — as already discussed in the preceding section;
2. the *IR* drop due to rail series resistance;
3. the series inductance of the rails.

The IR drops are readily calculated, provided that the currents in any bus section can be estimated since the metal bus cross-sectional area and length for that section are known.

For a parent bus supplying current to other uniformly distributed short bus branches along the length $L$ of the parent bus, then the current at any distance $x$ from the source is given by

$$I_x = I_L\left(\frac{1-x}{L}\right)$$

where

$I_L$ = the total load current supplied by the parent bus
$x$ = the distance from the source.

The voltage drop at, say, the far end of the bus can be estimated from

$$\Delta V = \rho . \frac{I_L}{A}\left(\text{integral from 0 to } L \text{ of} \left(1 - \frac{x}{L}\right) dx.\right)$$
$$= \rho . I_L . \frac{L}{2A}$$

where

$\rho$ = resistivity of metal
$A$ = cross-sectional area of metal bus.

However, the bus structure is not usually as regular as envisaged here so that estimation of the voltage drop at any point is not as simple a matter as implied above if accurate determination is required.

The transmission line nature of any wiring introduces the possibility of voltage transients due to its self-inductance $L_0$. The transient changes in voltage due to the presence of self-inductance can be modeled by

$$\Delta V = L_0 \frac{dI}{dt}$$

where

$dI/dt$ is the rate of change of line current.

Regarding the bus/oxide/substrate structure as a microstrip, the inductance $L_0$ is given by

$$L_0 = Z_0 \sqrt{\frac{\varepsilon_{eff}}{c}}$$

where

$Z_0$ is the characteristic impedance
$c$ is the velocity of light
$\varepsilon_{eff}$ is the effective dielectric constant.

In general terms, line impedance $Z_0$ is given by

$$Z_0 = \left(\frac{L}{C}\right)^{1/2}$$

where

$L$ and $C$ are the values per unit length of the bus.

Clearly, transient voltages induced in either the $V_{DD}$ or the *GND* ($V_{SS}$) rail may lead to noise margin problems for inverters and gates.

*IR* drops generally can give rise to deterioration in the noise margins. This can be visualized with the aid of Figure 6–45.

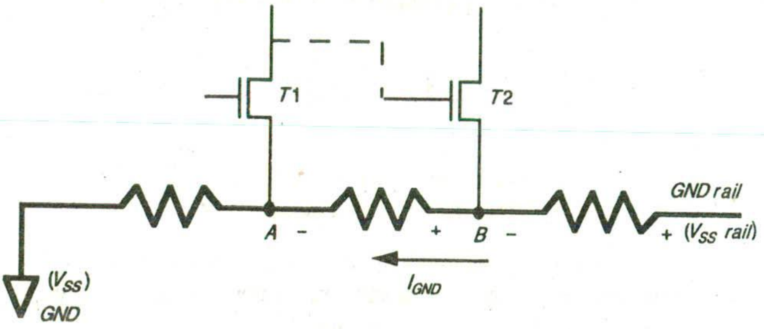


**Figure 6–45** Ground ($V_{SS}$) rail noise

It may be seen that if *T1* is switched on, then any transient at point *A* and/or DC voltage induced in the *GND* ($V_{SS}$) rail from point *B* to *GND* will affect the noise margins for the next stage, *T2*.

## 6.7 Observations

The material presented up to this point has provided a basic toolkit and has introduced techniques with which to approach the task of VLSI system design. In this context it is most important to learn by doing and the tutorial exercises included in the text are designed for that purpose. Instructors may wish to introduce further exercises from which students will undoubtedly benefit greatly. The best (perhaps the only) way to learn to design VLSI systems is to do it.

The next three chapters tackle the design of a four-bit data path system as part of a four-bit microprocessor.

## 6.8 Tutorial exercises

1. (a) Construct a color-coded stick diagram to represent the design of the following integrated nMOS and CMOS structures and indicate pull-up/pull-down ratios in each case:

(i) three-input *Nand* gate;

(ii) three-input *Nor* gate;

(iii) 8:1 multiplexer circuit incorporating an enable control line;

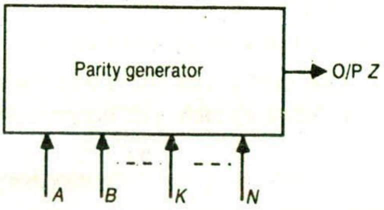
(iv) a dual-serial shift register capable of holding and shifting (right) two 4-bit words;

(v) a selectively loadable dynamic register to hold one four-bit word (parallel).

(b) For question l(a)(iv) draw the corresponding transistor circuit diagrams.

2. Construct a stick diagram for an nMOS or CMOS parity generator as in Figure 6–46. The required response is such that Z = 1 if there is an even number (including zero) of 1s on the inputs and Z = 0 if there is an odd number. (Use simple color coding for stick diagrams.)

Configure your design in a modular expandable fashion so that the inputs could be increased to five or more quite readily, using the basic cell suggested in Figure 6–47.



igure 6–46 Parity generator outline

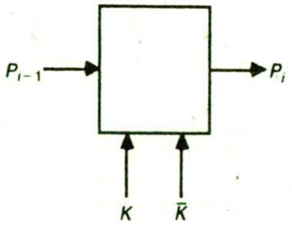


**Figure 6–47** Parity generator cell

3. (a) Construct a color-coded stick diagram to represent the design of an integrated nMOS structure to decode the three input lines $E_0$, $E_1$, and $E_2$ into eight output lines $Z_0, Z_1 \ldots, Z_7$, in accordance with the following truth table.

**Truth table**

| $E_2$ | $E_1$ | $E_0$ | $Z_0$ | $Z_1$ | $Z_2$ | $Z_3$ | $Z_4$ | $Z_5$ | $Z_6$ | $Z_7$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

(b) Discuss the expandability or otherwise of your structure and the ease with which it would translate to CMOS.

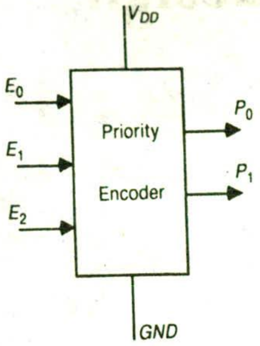
4. A priority encoder is a combinational circuit in which each input is assigned a priority with respect to the other inputs, and the output code generated at any time is that associated with the highest priority input then present.

Construct a color-coded stick diagram to implement such a structure as in the following table with Figure 6–48.

**Truth table**

| $E_2$ | $E_1$ | $E_0$ | $P_1$ | $P_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |

5. Referring to section 6.4.6 and to the development of an *Exclusive-Or* gate, design an alternative form of two I/P *Exclusive-Or* using transmission gates and inverters only. Your design should include a stick diagram and a mask layout.



Truth table

| $E_2$ | $E_1$ | $E_0$ | $P_1$ | $P_0$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 |

**Figure 6–48** Priority encoder

# 7 Subsystem design processes

*One of the pleasantest things in the world is going on a journey.*
Sir John Harrington

*The longest journey starts with a single step.*
Mao Zedong

## Objectives

This chapter and the following two carry through the design of a digital system using a top-down approach. The complete system environment is that of a 4-bit microprocessor which is readily envisaged as an interconnection of four major architectural blocks — ALU, Control Unit, I/O Unit and Memory.

The design developed in this text is that of the ALU or data path, which itself divides readily into four subsystems. This chapter concentrates our attention on the design of one of the ALU subsystems — the Shifter.

The whole design process clearly illustrates the step-by-step nature of structured design and the inherently regular nature of properly conceived subsystem architecture. A general design process is developed and set out in this chapter.

## 7.1 Some general considerations

The first question to ask about any design methodology is the time-honored 'What's in it for me? Is it going to be worthwhile investing the time to learn?'. To answer the second part first, remarkably little time is needed to learn the rudiments of VLSI design. This is largely thanks to the Mead and Conway methodology which originally brought VLSI design within the scope of the ordinary electronics engineer. In fact, the average undergraduate student of electrical or electronic engineering can acquire a basic level of competence in VLSI design for an investment of about 40 hours of lectures spread over one or more academic terms or semesters. Similarly, a 10-day full-time continuing education course can quite readily bring practicing professional engineers or computer scientists up to a similar standard. A basic level of competence is taken as the ability to apply the design methodology and make use of design tools and procedures to the point where a chip design of several hundred transistors (or higher for regular structures) can be tackled.

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.

In tackling the design of a system, we must bear in mind that topological properties are generally far more significant than the logical operations being performed. It may be said that it is better to duplicate (or triplicate, etc.) rather than communicate. This is indeed the case, and it is an approach which seems wrong to more traditional designers. In fact, even in relatively straightforward designs, as much as 40–50% of the chip may be taken up with interconnections, and it is true to say that interconnections generally pose the most acute problems in the design of large systems. Communications must therefore be given the highest priority early in the design process and a *communications strategy* should be evolved and adhered to throughout that process.

Accordingly, the architecture should be carefully chosen to allow the design objectives to be realized *and* to allow high regularity in realization.

## 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.

We will illustrate various representations during the course of the following design exercise to illustrate design processes.

### 7.2.1 The general arrangement of a 4-bit arithmetic processor

The 4-bit microprocessor has been chosen as a design example because it is particularly suitable for illustrating the design and interconnection of common architectural blocks.

Figure 7–1 sets out the basic architecture of most, if not all, microprocessors. At this stage we will consider the design of the data path only, but matters relevant to other blocks will follow in later chapters.

The data path has been separated out in Figure 7–2 and it will be seen that the structure comprises a unit which processes data applied at one port and presents its output at a second port. Alternatively, the two data ports may be combined as

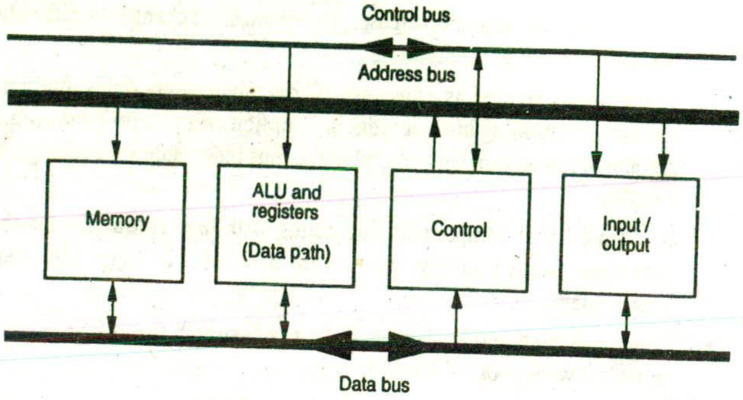


Figure 7–1 Basic digital processor structure

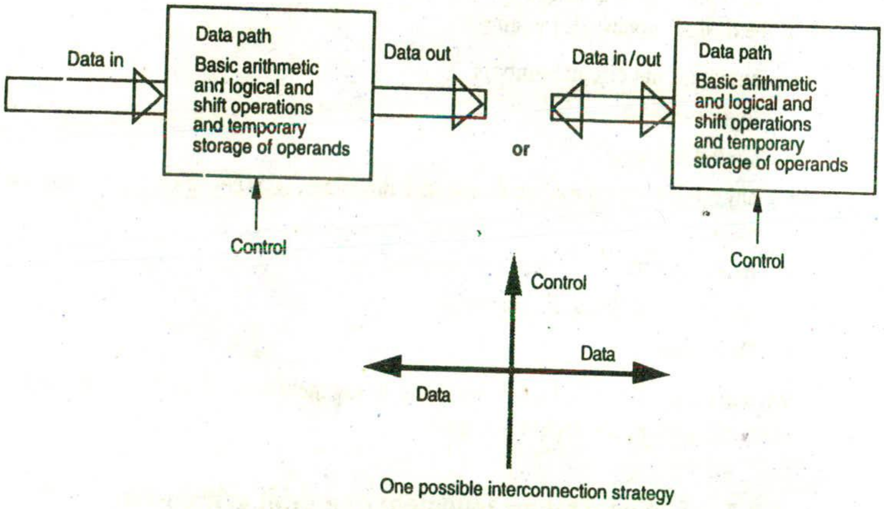

Data in
Data path
Basic arithmetic and logical and shift operations and temporary storage of operands
Data out
Control
or
Data in/out
Data path
Basic arithmetic and logical and shift operations and temporary storage of operands
Control
Control
Data
Data
One possible interconnection strategy

Figure 7–2 Communications strategy for data path

a single bidirectional port if storage facilities exist in the data path. Control over the functions to be performed is effected by control signals as indicated.

At this early stage it is essential to evolve an interconnections strategy (as shown) to which we will then adhere.

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.

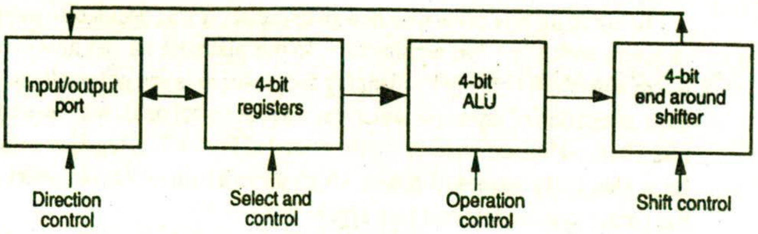
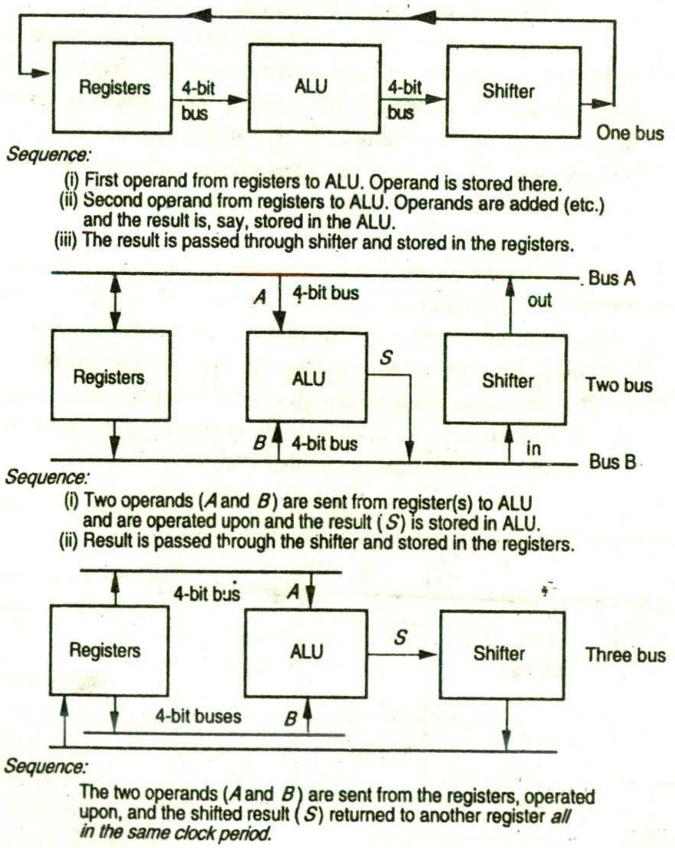


**Figure 7–3** Subunits and basic interconnections for data path

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. Some of the possibilities are shown in Figure 7–4.



*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

In pursuing this particular design exercise, it was decided to implement the structure with a two-bus architecture. In our planning we can now extend on our interconnections strategy by planning for power rails and notionally making some basic allocation of layers on which the various signal paths will be predominantly run. These additional features are illustrated in Figure 7–5, together with a tentative floor plan of the proposed design which includes some form of interface (I/O) to the parent system data bus (see Figure 7–1).

The proposed processor will be seen to comprise a register array in which 4-bit numbers can be stored, either from an input/output port or from the output of the ALU via a shifter. Numbers from the register array can be fed in pairs to the ALU to be added (or subtracted, etc.) and the result can be shifted or not, before being returned to the register array or possibly out through the I/O port. Obviously, data connections between the I/O port, ALU, and shifter must be in the form of 4-bit buses. Simultaneously, we must recognize that each of the blocks must be suitably connected to control lines so that its function may be defined for any of a range of possible operations.

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 will be very tentative since we will not as yet be able to accurately assess the area requirements, say for a 4-bit register or a 4-bit adder.

Overall interconnection strategy having been determined, stick diagrams for the circuits comprising sections of the various blocks may be developed, conforming to the required strategy. An interactive process of modification may well then take place between the various stages as the design progresses. During the design process, and in particular when defining the interconnection strategy and designing the stick diagrams, care must be taken in allocating the layers to the various data or control paths. We must remember that:

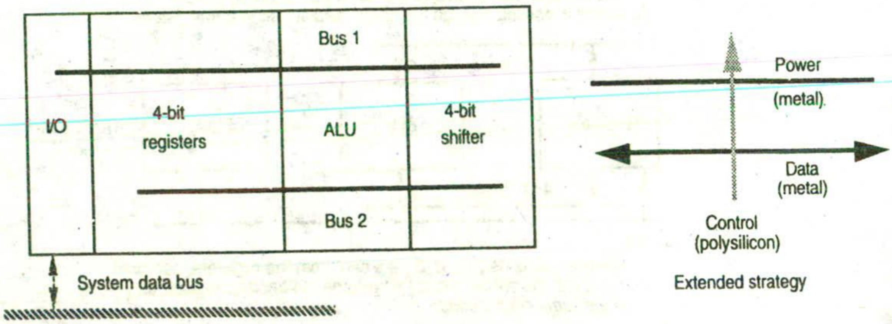


Figure 7–5 Tentative floor plan for 4-bit data path

1. Metal can cross polysilicon or diffusion without any significant effect (with some reservations to be discussed later).
2. Wherever polysilicon crosses diffusion a transistor will be formed. This includes the second polysilicon layer for processes that have two.
3. Wherever lines touch on the same level an interconnection is formed.
4. Simple contacts can be used to join diffusion or polysilicon to metal.
5. To join diffusion and polysilicon we must use either a buried contact or a butting contact (in which case all three layers are joined together at the contact) or two contacts, diffusion to metal then metal to polysilicon.
6. In some processes, a second metal layer is available. This can cross over any other layers and is conveniently employed for power rails.
7. First and second metal layers may be joined using a *via*.
8. Each layer has particular electrical properties which must be taken into account.
9. For CMOS layouts, p- and n-diffusion wires must not directly join each other, nor may they cross either a p-well or an n-well boundary.

With these factors in mind, we may now adopt suitable tactics to meet the strategic requirements when we approach the design of each subunit in turn.

### 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. In fact, a moment's consideration will reveal, for a 4-bit word, that a 1-bit shift right is equivalent to a 3-bit shift left and a 2-bit shift right is equivalent to a 2-bit shift left, etc. Thus we can achieve a capability to shift left or right by zero, one, two, or three places by designing a circuit which will shift right only (say) by zero, one, two, or three places.

The nature of the shifter having been decided on, its implementation must then be considered. Obviously, the first circuit which comes to mind is that of the shift register in Figures 6–38, 6–39, and 6–40. Data could be loaded from the output of the ALU and shifting effected; then the outputs of each stage of the shift register would provide the required parallel output to be returned to the register array (or elsewhere in the general case).

However, there is danger in accepting the obvious without question. Many designers, used to the constraints of TTL, MSI, and SSI logic, would be conditioned to think in terms 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.

In looking for a way of meeting these requirements, we should also attempt to take best advantage of the technology; for example, the availability of the switch-like MOS pass transistor and transmission gate.

We must also observe the strategy decided on earlier for the direction of data and control signal flow, and the approach adopted should make this feasible. Remember that the overall strategy in this case is for data to flow horizontally and control signals vertically.

A solution which meets these requirements emerges from the days of switch and relay contact based switching networks — the *crossbar switch.* Consider a direct MOS switch implementation of a 4 × 4 crossbar switch, as in Figure 7–6.

The arrangement is quite general and may be readily expanded to accommodate *n*-bit inputs/outputs. In fact, this arrangement is an overkill in that any input line can be connected to any or all output lines — if all switches are closed, then all inputs are connected to all outputs in one glorious short circuit. Furthermore, 16 control signals ($sw_{00} - sw_{15}$), one for each transistor switch, must be provided to drive the crossbar switch, and such complexity is highly undesirable. An adaptation of this arrangement recognizes the fact that we can couple the switch gates together

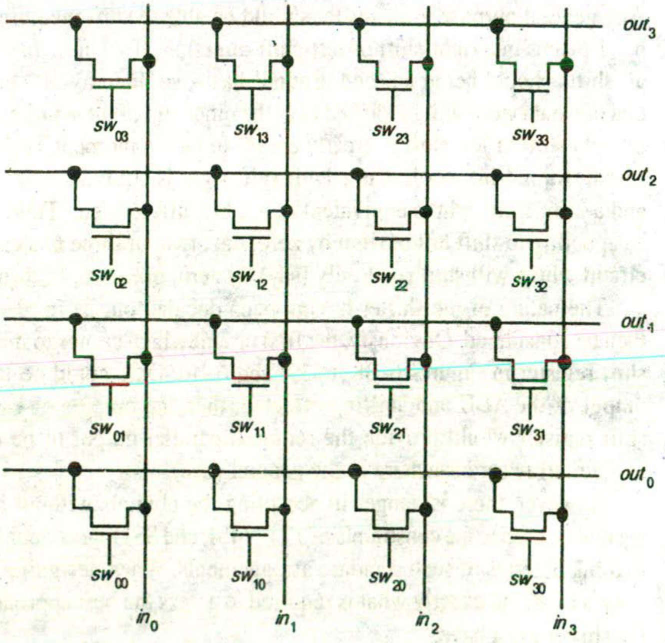


Figure 7–6 4 × 4 crossbar switch

in groups of four (in this case) and also form four separate groups corresponding to shifts of zero, one, two and three bits. The arrangement is readily adapted so that the in-lines also run horizontally (to conform to the required strategy).

The resulting arrangement is known as a *barrel shifter* and a 4 × 4-bit barrel shifter circuit diagram is given in Figure 7–7. 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.

The structure of the barrel shifter is clearly one of high regularity and generality and it may be readily represented in stick diagram form. One possible implementation, using simple n-type switches, is given in Figure 7–8.

The stick diagram clearly conveys regular topology and allows the choice of a standard cell from which complete barrel shifters of any size may be formed by replication of the standard cell. It should be noted that standard cell boundaries must be carefully chosen to allow for butting together side by side and top to bottom to retain the overall topology. The mask layout for standard cell number 2 (arbitrary choice) of Figure 7–8 may then be set out as in Figure 7–9. Once the standard cell dimensions have been determined, then any *n* × *n* barrel shifter may be configured and its outline, or bounding box, arrived at by summing up the dimensions of the replicated standard cell. The use of simple n-type switches in

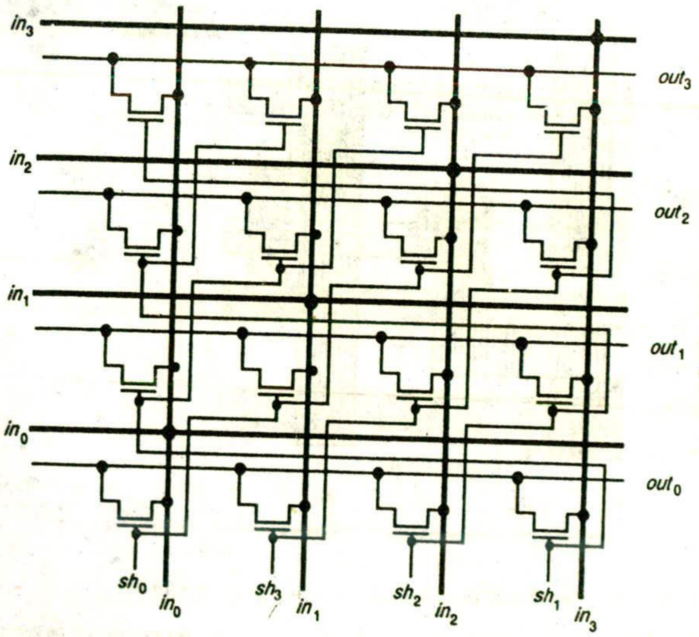


**Figure 7–7** 4 × 4 barrel shifter

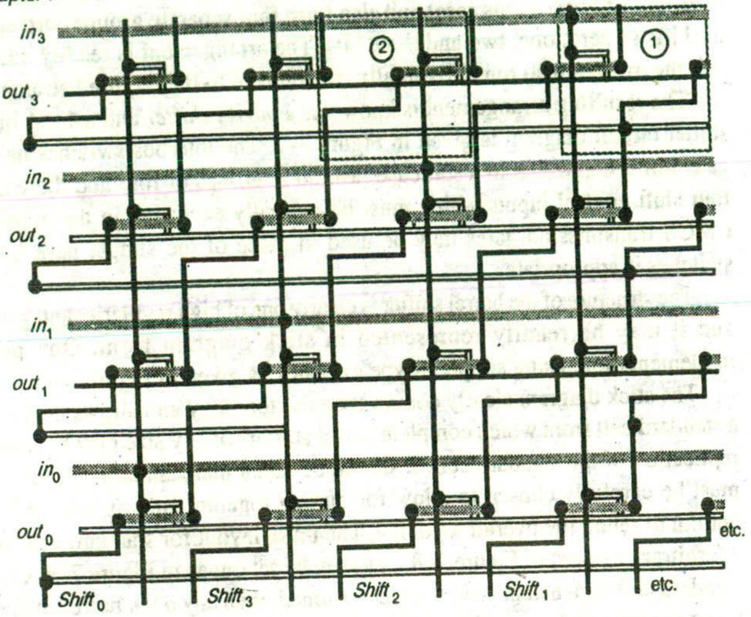


**Figure 7–8** One possible stick diagram for a 4 × 4 barrel shifter

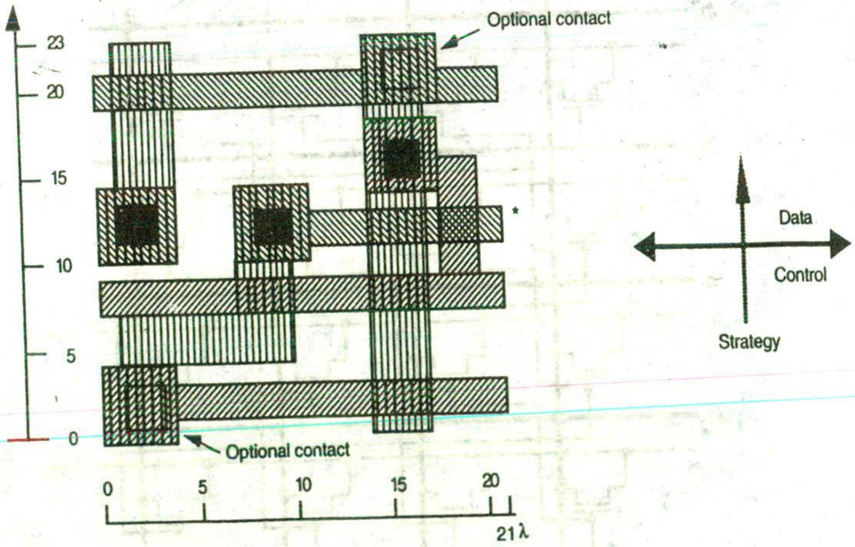


**Figure 7–9** Barrel shifter standard cell 2 — mask layout

* If this particular cell is checked for design rule errors in isolation, then an error will be generated owing to insufficient extension of polysilicon over thinox where shown. This error will *not* be present when cells are butted together. This effect is caused by the particular choice of cell boundaries and care must be taken when making such choices.

a CMOS environment might be questioned. Although there will be a degrading of logic 1 levels through n-type switches, this generally does not matter if the shifter is followed by restoring circuitry such as inverters or gate logic. Furthermore, as there will only ever be one n-type switch in series between an input and the corresponding output line, the arrangement is fast.

The minimum size *bounding box* outline for the 4 × 4-way barrel shifter is given in Figure 7–10. The figure also indicates all inlet and outlet points around the periphery together with the layer on which each is located. This allows ready placing of the shifter within the floor plan (Figure 7–5) and its interconnection with the other subsystems forming the datapath. It also emphasizes the fact that, as in this case, many subsystems need external links to complete their architecture. In this case, the links shown on the right of the bounding box must be made and must be allowed for in interconnections and overall dimensions. This form of representation also allows the subsystem geometric characterization to be that of the bounding box alone for composing higher levels of the system hierarchy.

## 7.3 Observations

At this stage it is convenient to examine the way we have approached the design of a system and of a particular subsystem in detail. The steps involved may be set out as follows:

1. Set out a specification together with an architectural block diagram.
2. Suitably partition the architecture into subsystems which are, as far as possible, self-contained and which give as simple interconnection requirements as possible.
3. Set out a tentative floor plan showing the proposed physical disposition of subsystems on the chip.
4. Determine interconnection strategy.
5. Revise 2, 3 and 4 interactively as necessary.
6. Choose layers on which to run buses and the main control signals.
7. Take each subsystem in turn and conceive a regular architecture to conform to the strategy set out in 4. Set out circuit and/or logic diagrams as appropriate. Remember that MOS switch-based logic is such that both the logic 1 and logic 0 conditions of an output must be deliberately satisfied (not as in TTL logic, where if logic 1 conditions are satisfied then logic 0 conditions follow automatically).
8. Develop stick diagrams adopting suitable tactics to observe the overall strategy (4) and choice of layers (6). Determine suitable *standard cell(s)* from which the subsystem may be formed.

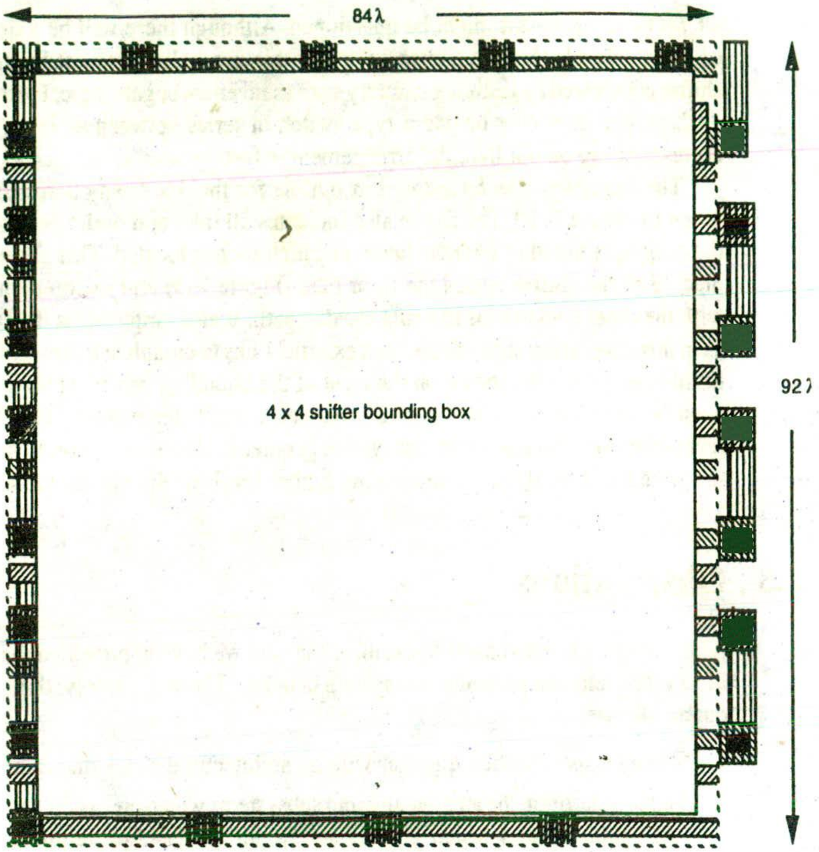


**Figure 7–10** Bounding box for 4 × 4 barrel shifter

9. Produce mask layouts for the standard cell(s), making sure that cells can be butted together, side by side and top to bottom, without design rule violation or waste of space. Determine overall dimension of the standard cell(s).
10. Cascade and replicate standard cells as necessary to complete the desired subsystem. This may now be characterized in *bounding box* form with positions and layers of inlets and outlets. External links etc. *must* be allowed for.

## 7.4 Tutorial exercises

1. (a) Set out the mask layout for a 4-way multiplexer using transmission gate switches.
   (b) Determine the overall dimensions (in terms of lambda) for your design.
   (c) Compare the attributes of this design with those of the n-type pass transistor version given in Chapter 6.
2. Using a block diagram (symbolic) form of representation for the 4-way multiplexer, draw up an interconnection diagram showing four such multiplexers configured as a 4-bit shift left/right shifter subsystem. The shifter should meet the same overall logical requirements as the barrel shifter designed in this chapter. You should carefully specify the control signals needed for this shifter design.
3. Estimate the area that will be occupied by your design of question 2, assuming the use of the multiplexer design of question 1. Compare this with the barrel shifter design developed in this chapter.

# 8 Illustration of the design process — computational elements

*Progress is a comfortable disease.*

e. e. cummings

*Beneath this slab*
*John Brown is stowed*
*He watched the ad(d)s*
*And not the road.*

Ogden Nash

## Objectives

In this chapter we progress to the arithmetic subsystem of the 4-bit data path. Once again, high regularity should be the aim of the designer. If the subsystems are regular and therefore composed of relatively few actual leaf-cell circuits, then the designer can concentrate on the main problem of VLSI design — the routing of interconnections and communication paths. It is hoped that this fact is beginning to emerge as this design progresses. Properly conceived communications — both at leaf-cell and at system levels — are the key to good design.

The arithmetic circuitry required here is relatively simple but does lead into a further consideration of adder circuitry and a fairly comprehensive survey of arrangements for multiplication.

## 8.1 Some observations on the design process

The design of the shifter, as the first subsystem of the proposed 4-bit data path, has illustrated some important features:

1. First and foremost, try to put requirements into words (an *if, then, else* approach often helps you do this) so that the most appropriate architecture or logic can be evolved.
2. If a standard cell (or cells) can be arrived at, then the actual detailed design work is confined to relatively small areas of simple circuitry (leaf-cells). Such cells can usually have their performance simulated with relative ease so that an idea of the performance of the complete subsystem may be deduced.
3. If generality as well as regularity is achieved then, for example, any size of shifter can be built up by simple replication and butting together of the standard cell(s).
4. Design is largely a matter of the topology of communications rather than detailed logic circuit design.
5. Once standard cell layouts are designed, overall area calculations can be precisely made (*not* forgetting to allow for any necessary links or other external terminations). Thus, accurate floor plan areas may be allocated.
6. VLSI design methodology for MOS circuits is not hard to learn.
7. The design rules are simple and straightforward in application.
8. A structured and orderly approach to system design is highly beneficial and becomes essential for large systems.

## 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:

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

The denominator of this expression will obviously be greatly reduced if the whole chip, or large parts of it, can be fabricated from a few standard cells, each of which is relatively simple in structure.

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

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

However, an 8 × 8-bit shifter, for example, would require no more detailed design and would have a regularity factor of 64.

Good system design can achieve regularity factors of 50 or 100 or more, and inherently regular structures, such as memories, achieve very high figures indeed.

## 8.3 Design of an ALU subsystem

Having designed the shifter, we may now turn our attention to another subsystem of the 4-bit data path (as in Figure 8–1). A convenient and appropriate choice is the ALU.

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. Obviously, a 4-bit adder must take the sum of two 4-bit numbers, and it will be seen we have assumed that all 4-bit quantities are presented in parallel form and that the shifter circuit has been designed to accept and shift a 4-bit parallel sum from the ALU.

Let us now specify that the sum is to be *stored* in parallel at the output of the adder from where it may be fed through the shifter and back to the register array. Thus, a single 4-bit data bus is needed from the adder to the shifter and another 4-bit bus is required from the shifted output back to the register array (since the shifter is merely a switch array with no storage capability). As far as the input to the adder is concerned, the two 4-bit parallel numbers to be added are to be presented in parallel on two 4-bit buses. We can also decide on some of the basic aspects of system timing at this stage and will assume clock phase $\phi_1$ as being the phase during which signals are fed along buses to the adder input and during

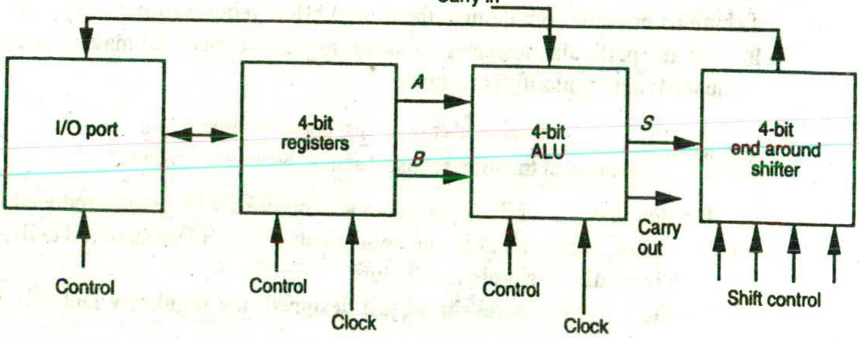
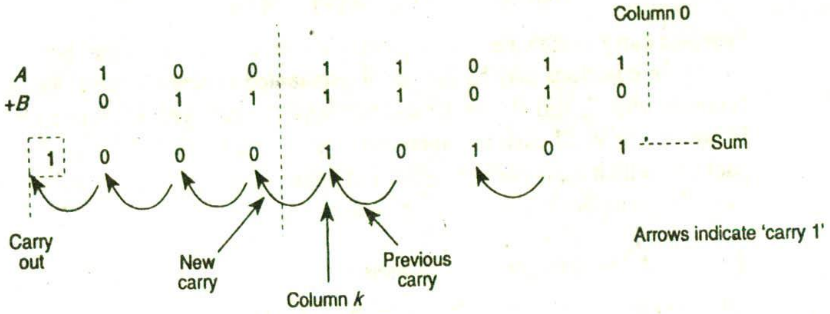


**Figure 8–1** 4-bit data path for processor (block diagram)

which their sum is stored at the adder output. Thus clock signals are required by the ALU as shown. The shifter is unclocked but must be connected to four shift control lines. It is also necessary to provide a 'carry out' signal from the adder and, in the general case, to provide for a possible 'carry in' signal, as indicated in Figure 8–1.

## 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:



It will be seen that for any column $k$ there will be *three* inputs — the corresponding bits of the input numbers, $A_k$ and $B_k$, and the 'previous carry' — *carry in* ($C_{k-1}$). It will also be seen that there are two outputs, the *sum* ($S_k$) and a *new carry* ($C_k$).

We may thus set out a truth table for the $k$ column of any adder, as in Table 8–1.

**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:

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

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

where

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

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

These equations may be directly implemented as *And-Or* functions or, most economically, $S_k$ and $H_k$ can be directly implemented with *Exclusive-Or* gates. However, for VLSI custom implementation there are none of the standard logic packages which are the delight of the TTL logic designer. It may be advantageous, then, to restate the requirements in another way.

### *8.3.1.1 Adder element requirements*

Table 8–1 reveals that the *adder requirements* may be stated thus:

$$\begin{array}{llll} \text{If} & A_k = B_k & \text{then} & S_k = C_{k-1} \\ \text{else} & S_k = \overline{C}_{k-1} & & \end{array}$$

and for the carry $C_k$

$$\begin{array}{llll} \text{If} & A_k = B_k & \text{then} & C_k = A_k = B_k\,^* \\ \text{else} & C_k = C_{k-1} & & \end{array}$$

* This relationship could also have been stated as:

$$\text{Carry} \qquad C_k = 1 \quad \text{when} \quad A_k = B_k = 1$$

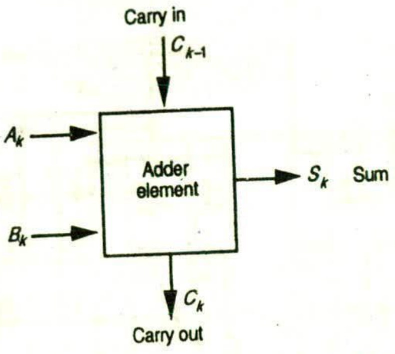
or

$$C_k = 0 \quad \text{when} \quad A_k = B_k = 0$$

### *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.

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.



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

**Figure 8–2** Adder element

Using multiplexers is an implementation of the logic circuitry for the adder element that is easy to follow, resulting directly from the way in which the requirements are stated in words (see section 8.3.1.1). This approach is illustrated in Figures 8–3 and 8–4 and it may be seen that the words used to describe the adder logic are directly implemented by the various paths through the multiplexers.

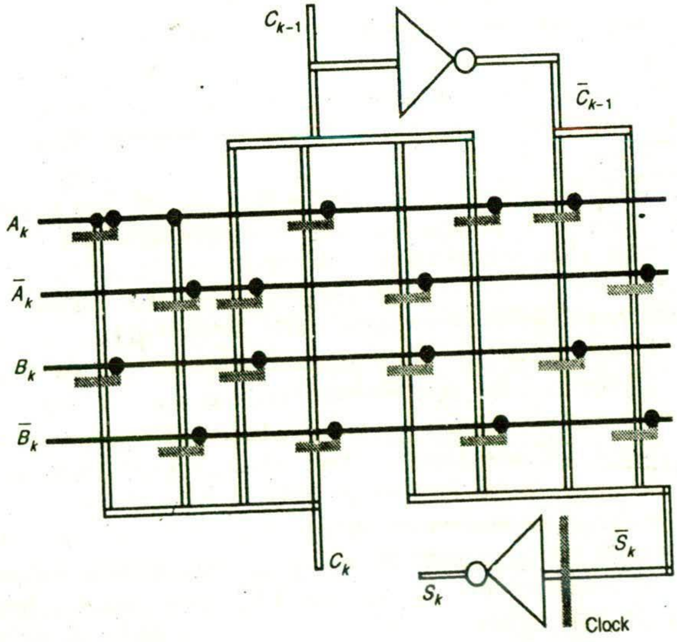


**Figure 8–3** Multiplexer (n-switches)-based adder logic with stored and buffered sum output

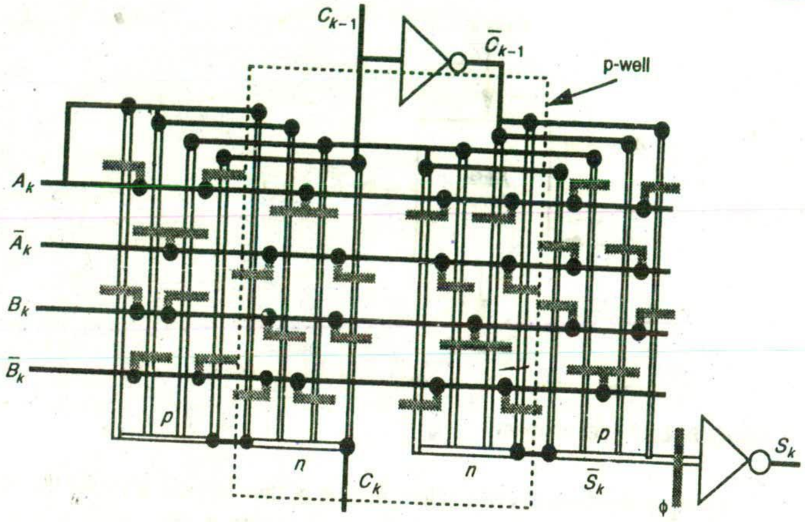


**Figure 8–4** CMOS version of adder logic

In these figures the multiplexers form $C_k$ and $\overline{S}_k$ *(not $S_k$)* to allow single inverter storage or buffering of $S_k$ if this is needed. In fact, one design actually implemented in silicon (see Figure 10–9) uses an nMOS multiplexer-based version of the adder. The basic logic requirements of this adder element, or bit-slice, are thus readily met in nMOS or CMOS technology. However, some practical factors must now be taken into account.

In order to form an *n*-bit adder, *n* adder elements must be cascaded with *carry out* of one element connecting to *carry in* of the next more significant element. Thus, the carry chain as a whole will consist of many pass transistors in series. This will give a very slow response and the carry line must therefore be suitably buffered between adder elements. (Remember, no more than four pass transistors in series — see section 4.9.) Also, we have assumed that both complements, $\overline{A}_k$ and $\overline{B}_k$, of the incoming bits are available. This may not be the case. Furthermore, signals $A_k$ and $B_k$ are to be derived from buses interconnecting the register with the ALU and may thus be taken off the bus through pass transistors. If this is the case, then these signals could not be used directly to drive the pass transistors of the multiplexers (see section 6.2.1). Finally, we must allow for storing the sum at the output of the adder, as discussed earlier in this section.

More practical general arrangements are shown in Figure 8–5. It will be seen that the adder element now contains all necessary buffering (at the expense of increased area). Seven inverter stages are required, deployed as follows (from top to bottom of Figure 8–5):

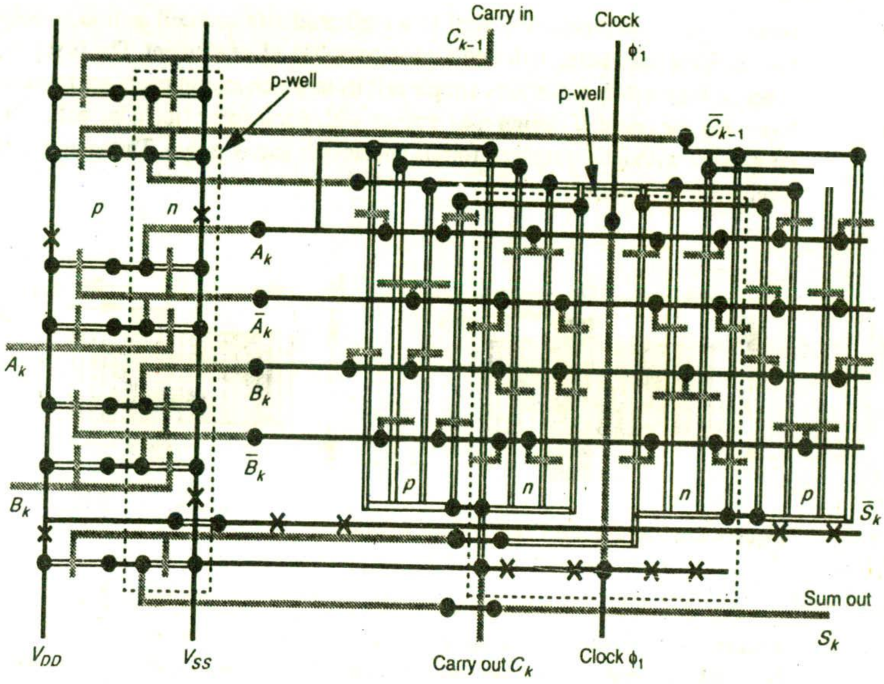


Figure 8–5 CMOS adder element

- Two inverters to form $\overline{C}_{k-1}$ and $C_{k-1}$ (buffered)
- Two inverters to form $\overline{A}_k$ and $A_k$ (buffered)
- Two inverters to form $\overline{B}_k$ and $B_k$ (buffered)
- One inverter to act as a dynamic store for $S_k$.

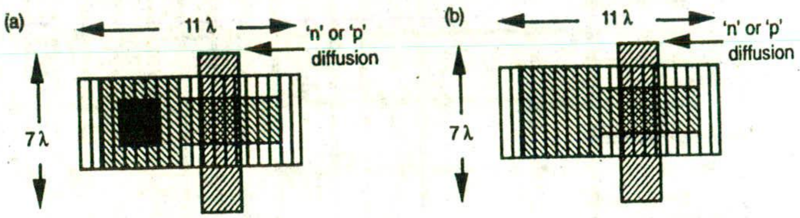
Note that only one inverter is required to store the sum digit $S_k$ *provided* that $\overline{S}_k$ rather than $\overline{S}_k$ is formed by the multiplexer. The observant reader will have already noted that the logic is configured to form the complement $\overline{S}_k$.

### 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.

The first choice is that of technology — for example, nMOS or CMOS — and this in turn decides the detailed nature of the multiplexer and inverters. Both technologies lend themselves well to a replicated standard cell approach, only two standard cells being required for the complete adder element. The first cell, given in Figure 8–6, is the very simple cell from which multiplexers are formed. Note that the one cell design may appear with a transistor (includes poly. and contact) or without a transistor (omits contact cut and/or poly.). The second cell required is an inverter.



**Figures 8–6** Multiplexer cell with or without cut

For nMOS, two versions of an inverter are needed — one for an 8:1 ratio and a second version for a 4:1 ratio. However, only one standard inverter cell design is actually needed with a choice of widths for the pull-down channel as shown in Figures 8–7 and 8–8. For completeness, a butting contact based inverter design is shown as Figure 8–7 since these contacts were used at one time by a number of nMOS fabricators.

The use of a 'standard' nMOS inverter with choice of width for the pull-down channel is a common practice. However, note that the narrow channel for the 4 :1 configuration in Figure 8–7 has been placed so that its edges are on *whole λ boundaries,* not half λ boundaries as would be the case if narrowing had been carried out symmetrically. *Always* design mask layouts having edges on whole λ boundaries. Some design rule checking software and some fabrication processes might not accept half λ edges.

More commonly, buried contacts would be used to join diffusion and poly. layers in nMOS fabrication and suitable buried contact based inverter designs are given in Figure 8–8.

In this case the vertical dimension is larger than that of Figure 8–7, but there are occasions where the lack of any metal regions in the center of the inverter is a positive advantage. For the layout shown, two metal bus lines could be run through the cell and across the inverter from side to side. This might prove a considerable advantage in saving space in certain layouts, such as register or memory arrays where data buses must run through each storage element. This could not be done when using a compact butting contact design because of the need to maintain 3λ metal to metal separation from the rails and the metal layer 'cover' on the butting contact.

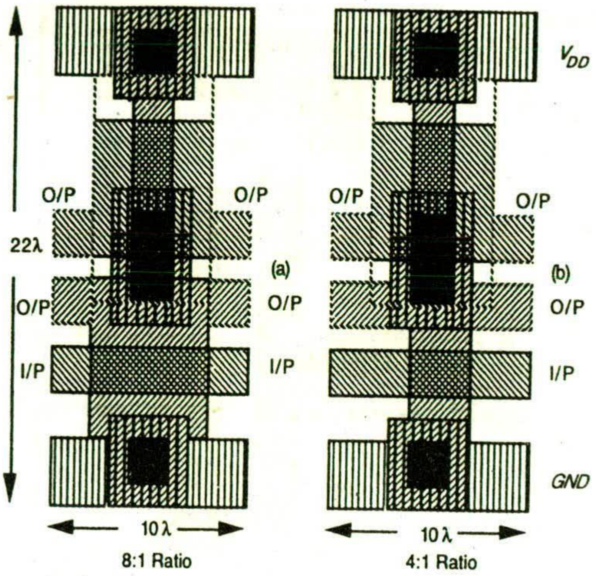


**Figure 8–7** nMOS (butting contact) inverters

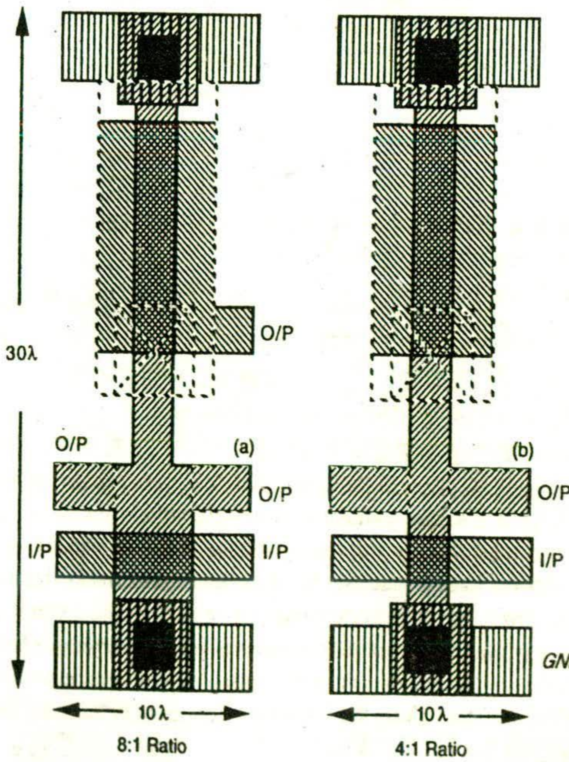


**Figure 8–8** nMOS (buried contact) inverters

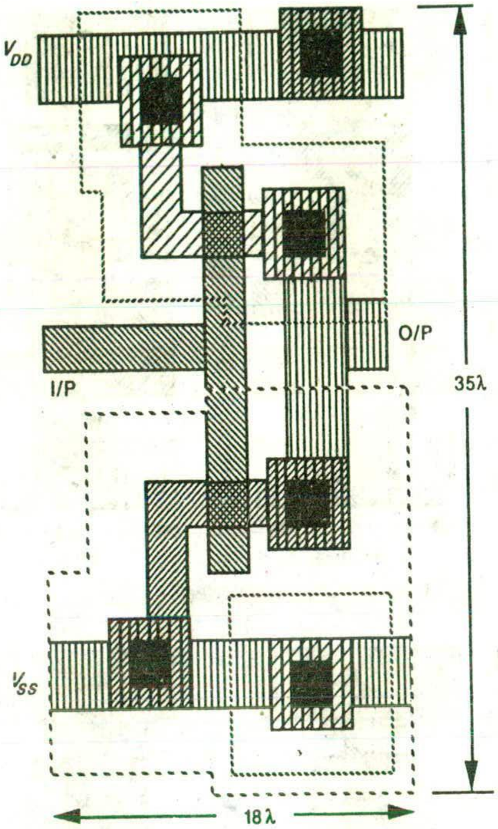


**Figure 8–9** A CMOS inverter design

For CMOS-based designs, as set out in Figure 8–5, we normally need a complementary CMOS inverter. A possible mask layout is shown in Figure 8–9.

### *8.3.1.4 Adder element bounding box*

We may now assess the area requirements for, say, the CMOS adder element as in Figure 8–5. First estimate the bounding box for the multiplexer area of the adder. Each standard multiplexer cell (Figure 8–6) is $7\lambda \times 11\lambda$ and there are 16 such elements side by side 'horizontally' and four stacked 'vertically'. We must also allow at least an additional $6\lambda$ width for the metal to metal spacings required by the clock bus passing through the center. In the vertical direction we must allow spacings for the interconnections between the tops of the multiplexers (an estimated additional $30\lambda$) and a further $10\lambda$ for the connection out from $\overline{S}_k$ and $C_k$ at the bottom. Thus, the bounding box must be at least $16 \times 7\lambda + 6\lambda = 118\lambda$ 'wide' and $4 \times 11\lambda + 30\lambda + 10\lambda = 84\lambda$ in 'height', as shown in Figure 8–10.

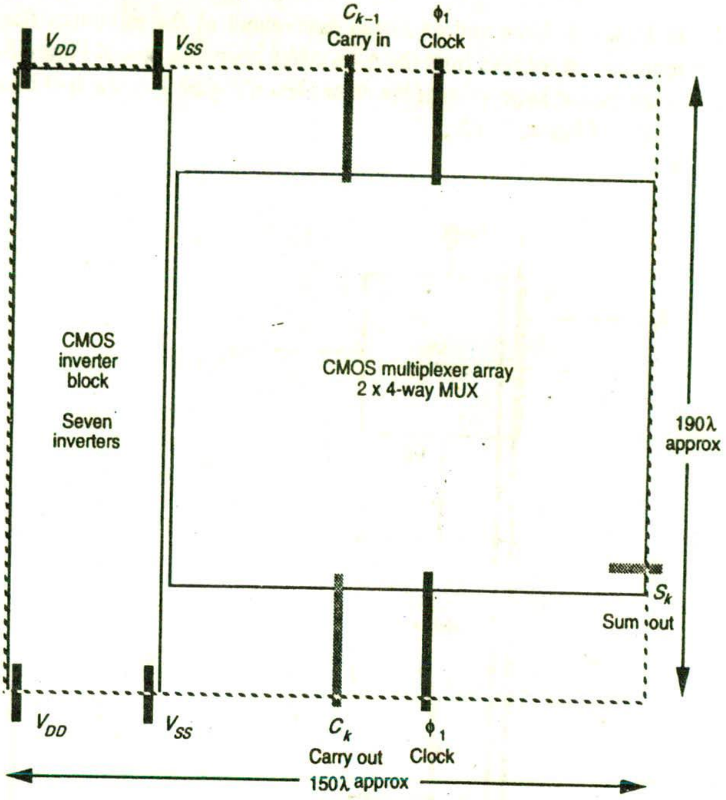


**Figure 8–10** Approximate bounding box and floor plan for CMOS adder element

To complete an assessment of the approximate area to be occupied by the CMOS adder element we need to allow for the seven inverters shown in Figure 8–5. We have already determined a bounding box outline for a suitable CMOS inverter circuit (see Figure 8–9) and it will be seen that each inverter occupies a rectangle measuring 18λ 'wide' and 35λ 'high'. Thus, seven inverters alone placed side by side will occupy an area of 126λ × 35λ and, allowing, say, an additional 50% width for space between each for connections, we have an overall area requirement of about 190λ × 35λ for the inverters. Thus, the overall bounding box (or *floor plan outline*) for a complete adder element will be approximately that given as the overall outline in Figure 8–10. Note that vertical distribution of power is required by this layout, but the direction of global power distribution may be reviewed as the design of the complete processor — floor plan as in Figure 7–5 — progresses. Details of inlet/outlet points on the inverter block and overall adder element bounding boxes will be worked out as part of the next tutorial exercise.

The 4-bit adder is then formed by cascading four adder elements as indicated in Figure 8–11(a) and an initial assessment of the minimum floor plan area requirement follows from the 4-bit adder bounding box of Figure 8–11(b). This is the second subsystem of the floor plan of Figure 7–5, the first being the barrel shifter of Figure 7–10.

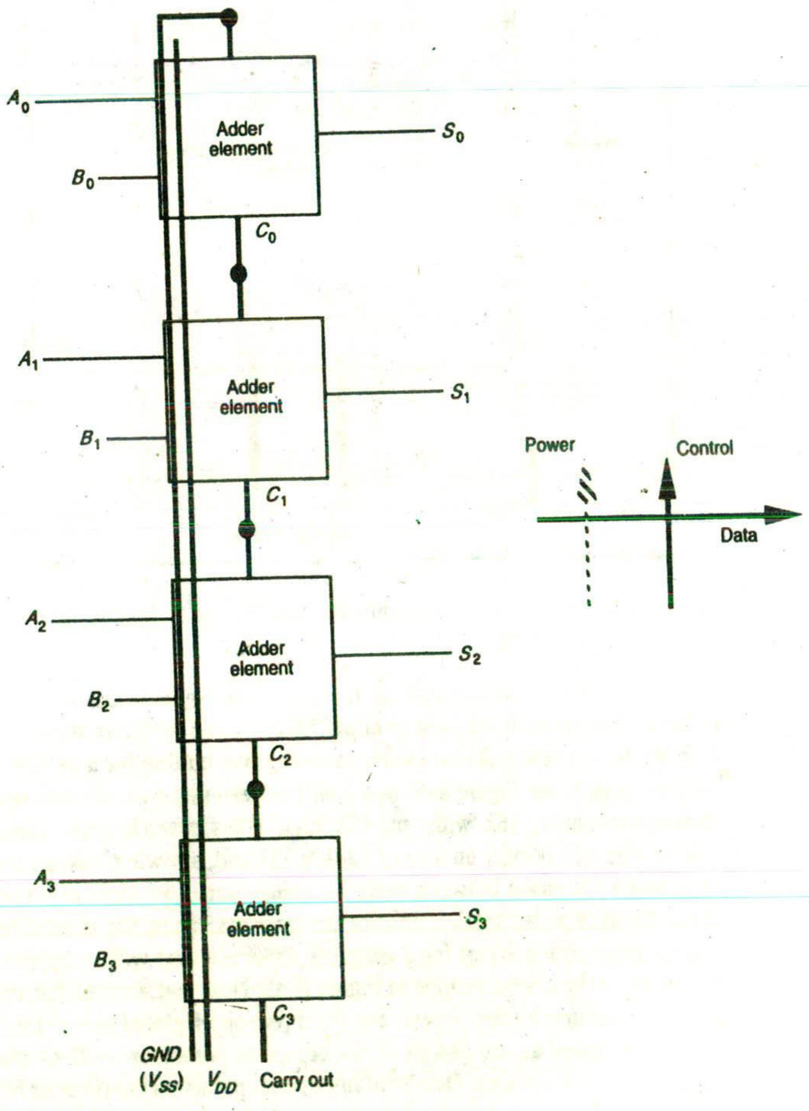


Figure 8–11(a) 4-bit adder

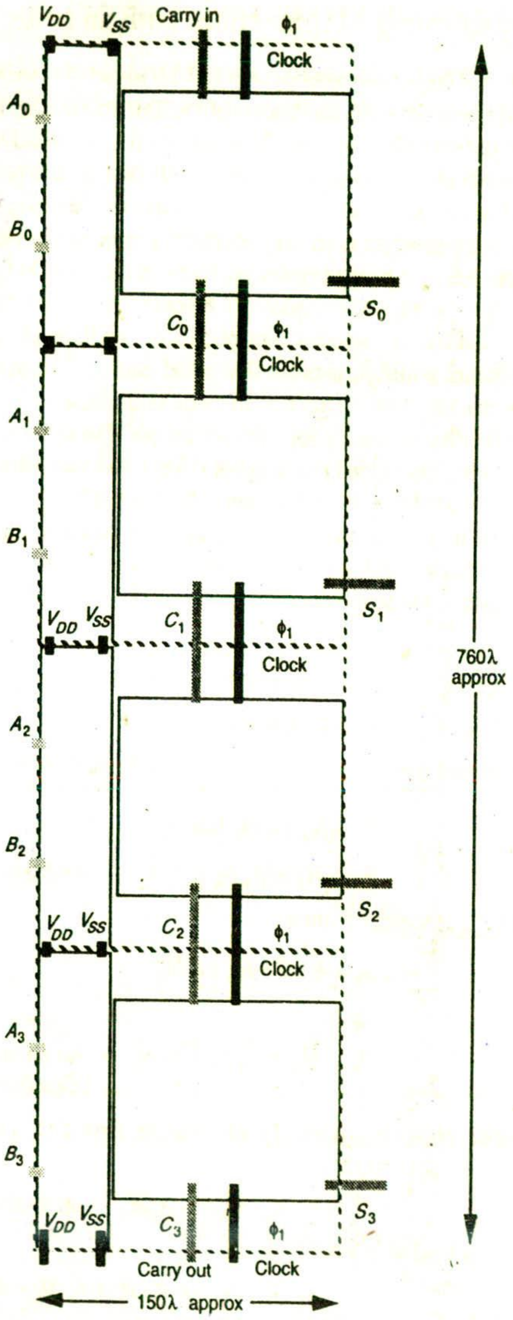


**Figure 8–11(b)** 4-bit adder outline

## 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.

Subtraction by an adder is an easy operation provided that the binary numbers $A$ and $B$ are presented in *twos complement* form. In this case, to find the difference $A - B$ it is only necessary to complement $B$ (exchange 1 for 0 and vice versa for all bits of $B$), add 1 to the number thus obtained, and then *add* this quantity to $A$ using the standard addition process discussed earlier. The output of the adder will then be the required difference in twos complement form. Note that the complement facility necessary for subtraction can also serve to form the *logical complement* (which is indeed exchanging 0 for 1 and vice versa).

It is highly desirable to keep the architecture of the ALU as simple as possible, and it would be nice if the adder could be made to perform logical operations as readily as it performs subtraction. In order to examine this possibility, consider the standard adder equation set out in section 8.3.1 and reproduced here:

2

| | | |
|---|---|---|
| | 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 $S_k = H_k = \overline{A}_k B_k + A_k \overline{B}_k$ — 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

Thus it may be seen that suitable switching of the carry line between adder elements will give the ALU logical functions. A possible arrangement of the adder elements for both arithmetic and logical functions is suggested in Figure 8–12.

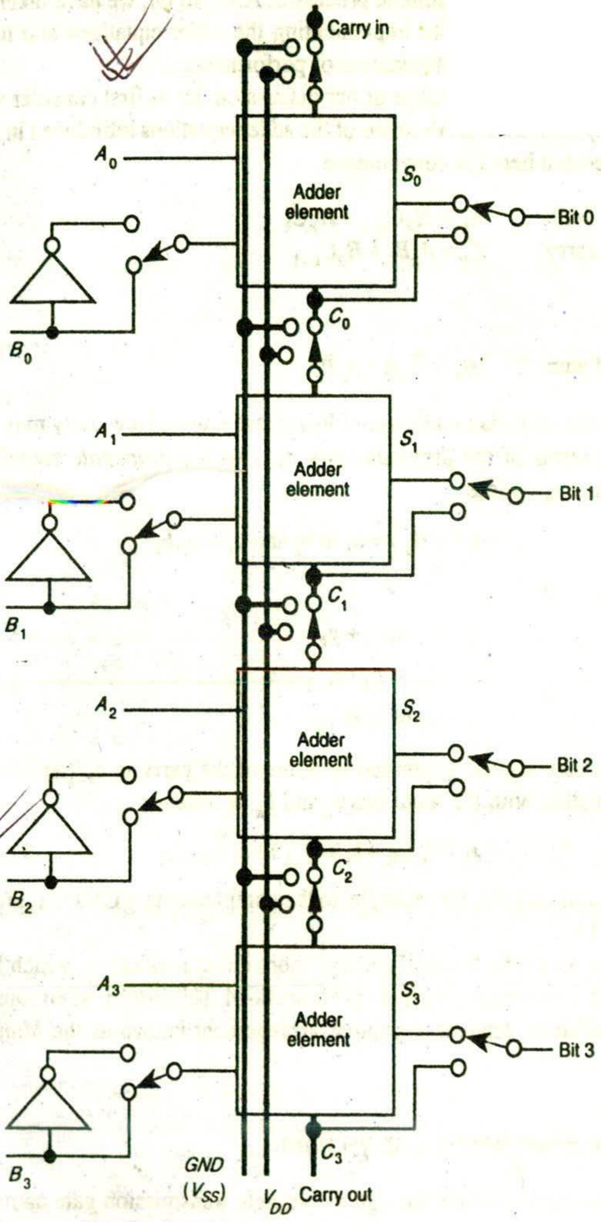


Figure 8–12 4-bit ALU

## 8.4 A further consideration of adders

A further consideration of aspects of adder circuitry is desirable since adders are the basic elements of all arithmetic processes. Also, so far, we have taken a very simple and direct approach to implementing the adder equations and have not considered refinement or optimization of performance.

In order to broaden the scope of our discussion, let us first consider some of the commonly used alternative forms of the adder equations introduced in Section 8.3.1 and 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 $H_k = \overline{A}_k B + A_k \overline{B}_k$

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 $c_k = p_k . c_{k-1} + g_k$

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

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:

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

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.*

### 8.4.1 The Manchester carry-chain

Instead of the carry passing through a complete transmission gate as in Figure 8–14, the carry path is precharged by the clock signal and the carry path may then be gated by a single n-type pass transistor as shown in Figure 8–15.

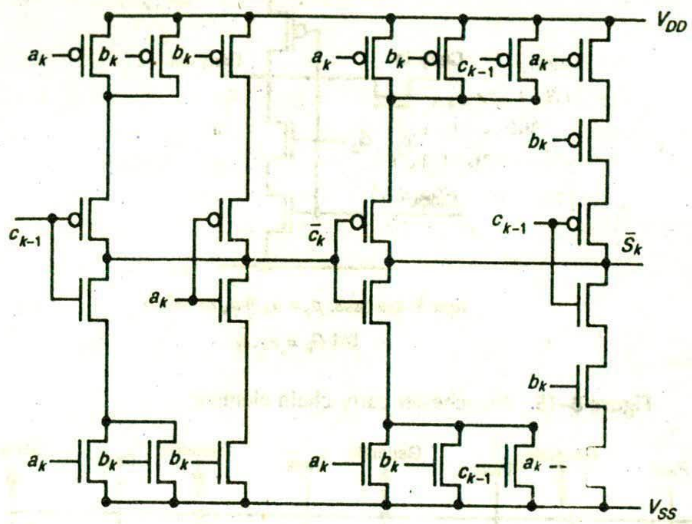


**Figure 8–13** One possible (symmetrical) adder cell arrangement

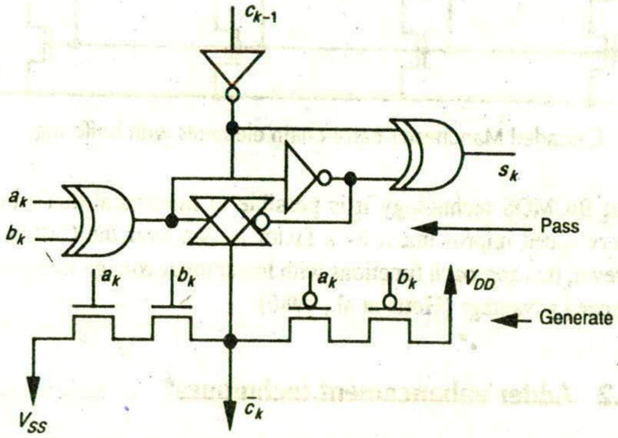
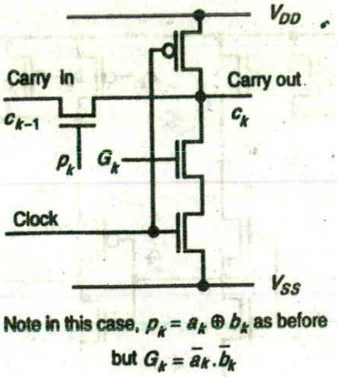


**Figure 8–14** An adder element based on the pass/generate concept

Although individual Manchester carry cells are fast, care must be taken when cascading them since this effectively connects pass transistors in series. We have already seen that the delay goes up as the square of $n$ where $n$ is the number connected in series. Obeying the rules set out earlier to cover this situation, we must buffer after every four carry chain cells as shown in Figure 8–16.



$V_{DD}$
Carry in
$c_{k-1}$
Carry out
$c_k$
$p_k$
$G_k$
Clock
$V_{SS}$
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

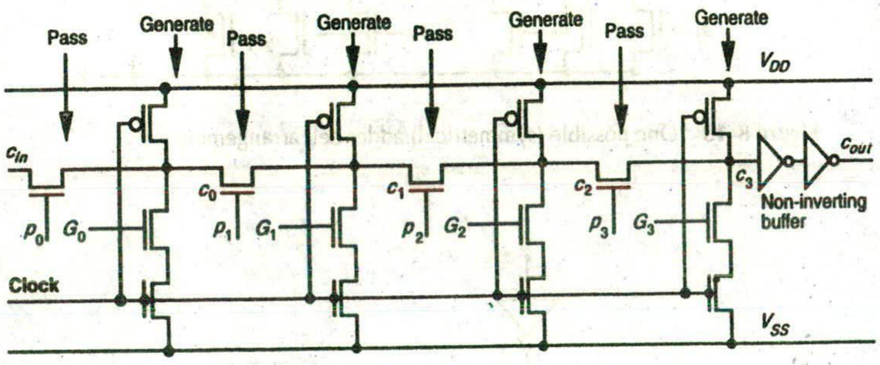


Figure 8–16 Cascaded Manchester carry-chain elements with buffering

In BiCMOS technology it is possible to implement this arrangement and achieve speed improvement by a factor of two over the CMOS arrangement. However, this approach functions with lower input voltage swings to achieve the full speed advantage (Hotta et al., 1986).

### 8.4.2 Adder enhancement techniques*

In the case of small adders ($n < 8$-bits), it is generally advantageous to adopt the relatively simple hardware of the ripple through carry. Thus, the carry completion time is clearly directly proportional to $n$. On the other hand, large adders (up to say $n = 64$ or even $n = 128$-bits) cannot afford to wait for the long completion time of a large ripple through carry line. Thus special techniques must be adopted

*This section is based on material provided by Dr B. Hochet of the Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland. The authors gratefully acknowledge this contribution.

to improve addition time. This improvement is possible only through some increase in complexity and, in consequence, at the expense of increased area in silicon. The next subsections discuss three techniques for effecting faster carry generation and each approach is characterized by a different area/performance ratio.

### 8.4.2.1 Carry select adders

For this arrangement — also referred to as a conditional sum adder — the adder is divided into blocks. Each block is composed of two adders, one with a logical 0 *carry in* and the other with a logical 1 *carry in*. The *sum* and *carry out* generated are then selected by the actual *carry in* which comes from the *carry out* output of the previous block as shown in Figure 8–17.

#### 8.4.2.1.1 Optimization of the carry select adder

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

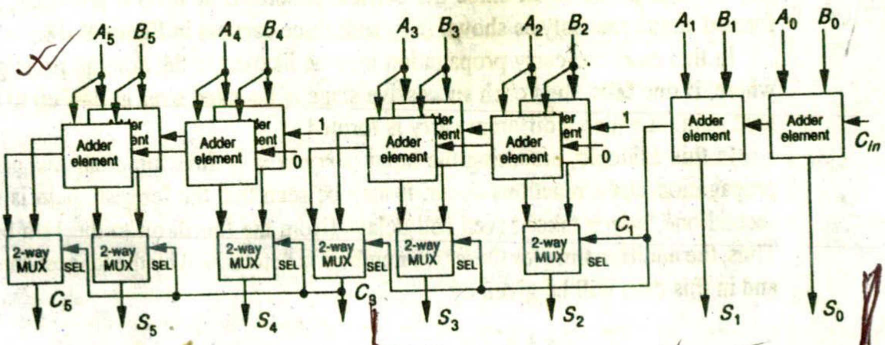
$$T = k_1 n$$

where $k_1$ is the delay through one adder cell.

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

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

where $k_2$ is the time needed by the multiplexer of the next block to select the actual output carry. A decision now has to be made on the size, in bits, of each adder block and clearly this could be 1-bit, in which case the number of multiplexers is a maximum, or two or more bits resulting in fewer multiplexers. If there are many multiplexers, then the ripple through effects occur in the multiplexer chain rather than in the carry chain through the blocks. Consequently, an optimum value must be sought for the block size.



$A_5$ $B_5$ $A_4$ $B_4$ $A_3$ $B_3$ $A_2$ $B_2$ $A_1$ $B_1$ $A_0$ $B_0$
Adder element
1
0
2-way MUX
SEL
$C_1$
$C_{in}$
$C_6$
$C_3$
$S_5$ $S_4$ $S_3$ $S_2$ $S_1$ $S_0$

Figure 8–17 Carry select adder structure (6-bit)

Suppose the *n*-bit adder is divided into *M* blocks, and that each block contains *P* adder cells in series, and considering the arrangement of Figure 8–17, we may see that the completion 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,

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

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 extended by one or more stages to account for the delay in the multiplexer. For instance, if the delay in the multiplexer is equal to the cell delay, then the size *P* of the succeeding block should be increased by one. On the other hand, if the multiplexer delay is twice that for the cell delay, then each block may have two more adder cells than the previous one; that is, *P* can be increased by two from one block to the next. The actual optimum increase in *P* from one block to its successor depends on the ratio between $k_1$ and $k_2$. However, care must be taken to properly allow for the multiplexer delay which will also depend on the number of inputs, that is, on *P*, increasing 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* 8.20

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.

In this case, the carry propagation may be likened to the domino principle, where, if one falls, then each successive stage is knocked over in turn up to the next point at which a different carry is formed.

In this example, assuming the input *carry in* = 1, three simultaneous carry propagation chain reactions occur. It may be seen that the longest chain is the second one, which takes seven cell delays (from the fourth bit to the 11th bit). Thus, the addition time for these two numbers is determined by the longest chain, and in this case will be given by

$$T = 7.k + k'$$

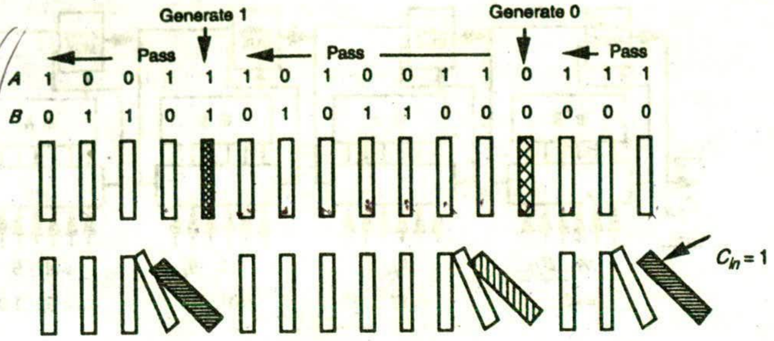

Generate 1
Generate 0
Pass
Pass
Pass
A 1 0 0 1 1 1 0 1 0 0 1 1 0 1 1 1
B 0 1 1 0 1 0 1 0 1 1 0 0 0 0 0 0
$C_{in} = 1$

**Figure 8–18** Diagrammatic representation of carry skip adder

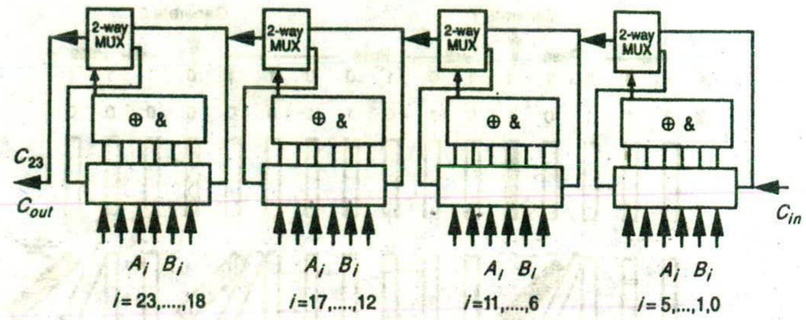
where $k$ is the cell delay and $k'$ is the time needed to compute the 11th bit sum using the carry in to the 11th bit.

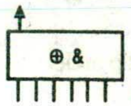
If, for a ripple carry adder, the input bits $A_i$ and $B_i$ are different for all bit positions, then the input carry is propagated at all bit positions and never generated. The addition is thus only completed after the carry has propagated along the entire adder. In this case, the computation delay must be $nk$, and although it may be less than this quite frequently, the worst case must be assumed in all cases when using the adder in, say, high speed or real-time or other time-critical applications.

Carry skip adders take advantage of both the generation and the propagation of the carry signal. They may be divided into blocks where, for each block, a special circuit is used to detect the condition when $A$ and $B$ bits differ in all bit positions in the block (that is $p_i = 1$ for all '$i$' in the block). The output signal from such a circuit is called the *block propagation signal.* If the *block propagation signal* = 1, then the carry signal entering the block can bypass it and be transmitted through a multiplexer to the next block. Figure 8–19 sets out the schematic structure of a 24-bit carry skip adder, subdivided into four blocks and based on this approach.

*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 arrangement and in this case we assume equal size blocks. Let $k_1$ denote the time needed by the carry signal to propagate through the adder cell, and $k_2$ the time needed for a carry to skip over a block. Further, let us divide the $n$-bit carry skip adder into $M$ blocks — each block containing $P$ adder cells. Since, as was the case for the ripple carry adder, the actual computing time depends on the configuration of the input numbers, the completion time may well be small but may also reach the worst case. We must thus evaluate and optimize the worst case conditions as depicted in Figure 8–20.





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

Figure 8–19 Structure of a 24-bit (for example) carry skip adder

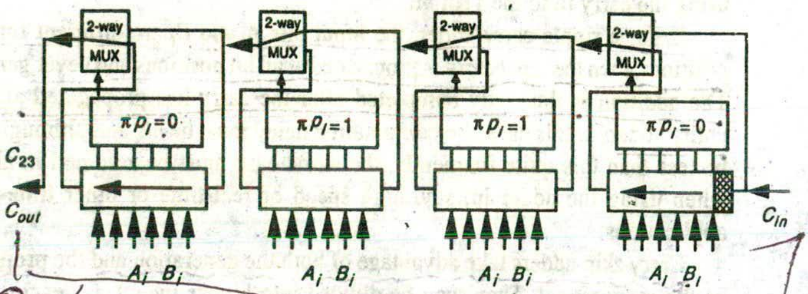


Figure 8–20 Worst case carry propagation for carry skip adder

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

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

where

$$P = n/M$$

The minimum value of $T$ is reached when

$$M = \sqrt{(2n.k_1/k_2)}$$

As for the carry select adders, a further improvement may be achieved if the adder is divided into blocks of differing sizes (Guyot et al., 1987).

Finally, Figure 8–21 shows a possible arrangement for a block, complete with its multiplexer and block propagation signal generating circuit. This particular realization leads to good regularity and thus to a high density layout in silicon.

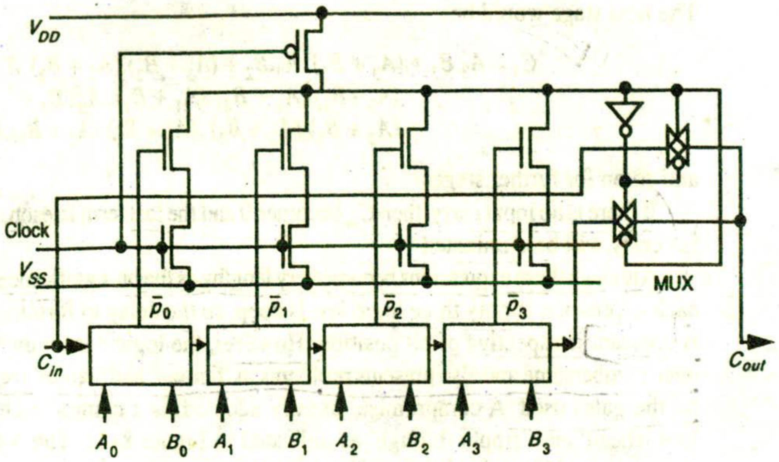


**Figure 8–21** Possible implementation of the block propagation concept

### 8.4.2.3 *Carry look-ahead (CLA) adders*

We have considered some other methods of improving adder throughput time and may now turn to algebra to seek a general solution to this problem. This is to be found in rearranging the expressions for the adder (given in section 8.3.1), in particular the expression for carry

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

noting that $H_k = A_k \bar{B}_k + \bar{A}_k B_k$ the expression can be rearranged into the form

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

Thus for $C_0$ we may write

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

which allows for an input carry; and, therefore

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

may then be written as

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

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

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

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.

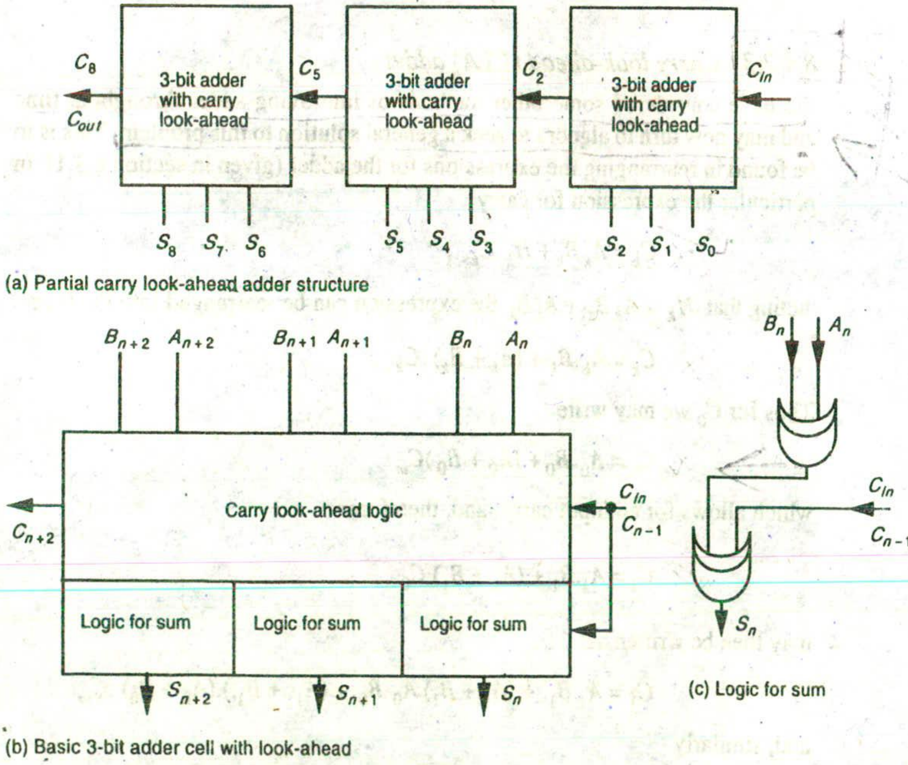
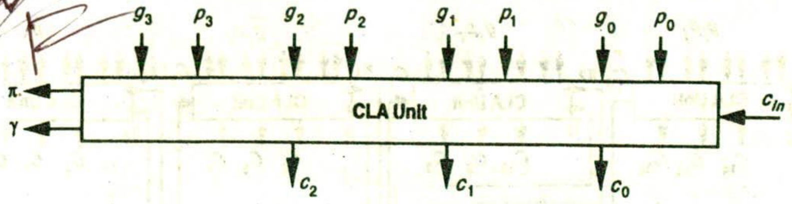

$C_8$
$C_{out}$
3-bit adder with carry look-ahead
$C_5$
3-bit adder with carry look-ahead
$C_2$
3-bit adder with carry look-ahead
$C_{in}$
$S_8$ $S_7$ $S_6$
$S_5$ $S_4$ $S_3$
$S_2$ $S_1$ $S_0$
(a) Partial carry look-ahead adder structure
$B_{n+2}$ $A_{n+2}$ $B_{n+1}$ $A_{n+1}$ $B_n$ $A_n$
$C_{n+2}$
Carry look-ahead logic
$C_{in}$
$C_{n-1}$
Logic for sum
Logic for sum
Logic for sum
$S_{n+2}$ $S_{n+1}$ $S_n$
(b) Basic 3-bit adder cell with look-ahead
$B_n$ $A_n$
$C_{in}$
$C_{n-1}$
$S_n$
(c) Logic for sum

**Figure 8–22** Carry look-ahead and ripple through compromise

[illegible]



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

**Figure 8–23** 4-bit block CLA unit

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:

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

In order to avoid a sequential propagation of carry signals between the blocks, we may generate 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 between blocks and for overall carry out as shown in Figure 8–24, which is the overall arrangement of a 16-bit 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

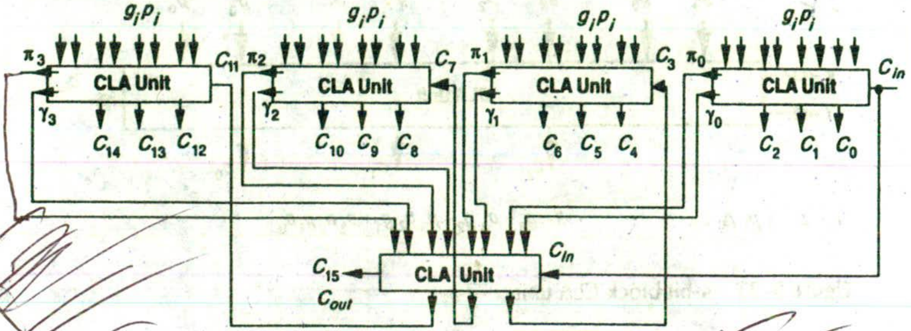


**Figure 8–24** A 16-bit, 4 × 4 block CLA adder

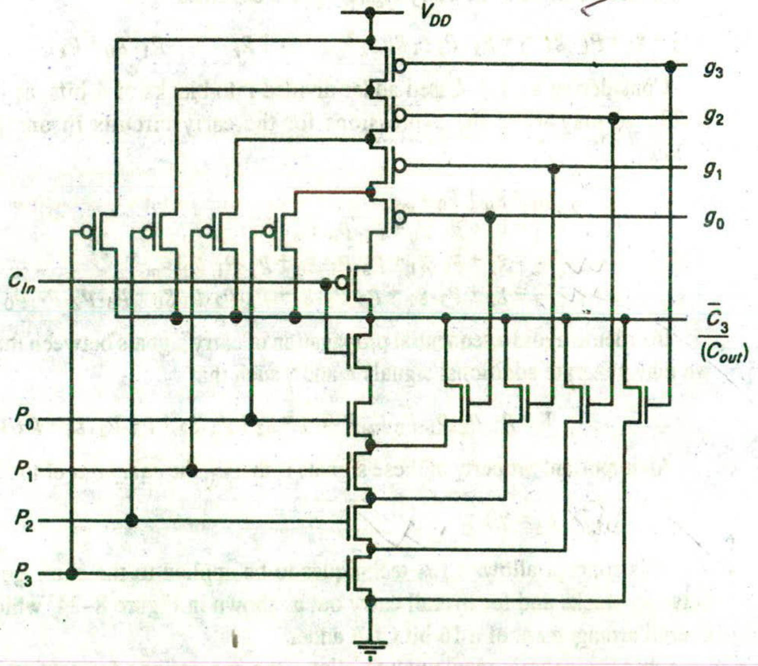


**Figure 8–25** Generation of carry out (from 4-bits and carry in)

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 $\gamma$.

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.

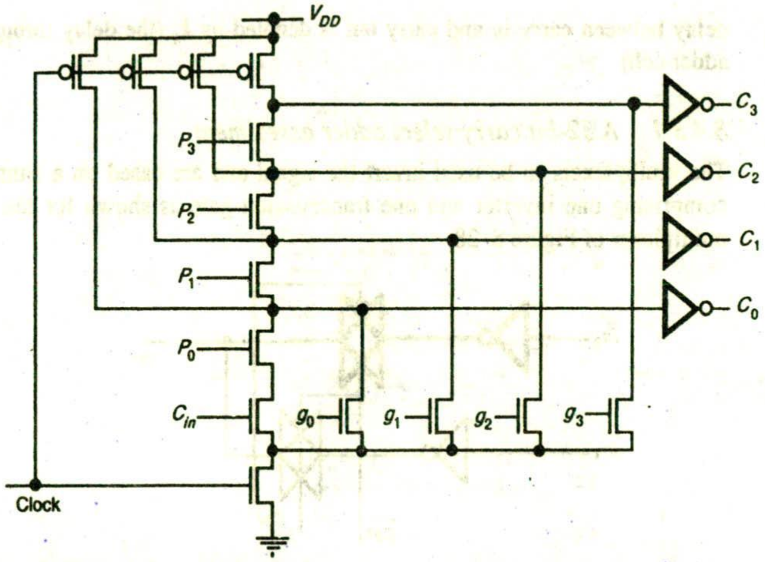


**Figure 8–26** Four-cell Manchester carry-chain

## 8.4.3 A comparison of adder enhancement techniques

This section compares the three enhancement techniques we have discussed from the point of view of area occupied combined with performance. For the purpose of our study, we will compare three 32-bit adders — one carry select, one carry skip and one carry look-ahead. For convenience the carry select and carry skip adders will be assumed to be subdivided into equal size blocks. This must be so as a graduated sizing of blocks relies on an accurate knowledge of the gate delays — information which we do not have for this comparison. The adder cell to be used is required in two versions, one as in Figure 8–14 and a second version — with inverted inputs and carry output — as in Figure 8–27. In both cases, the

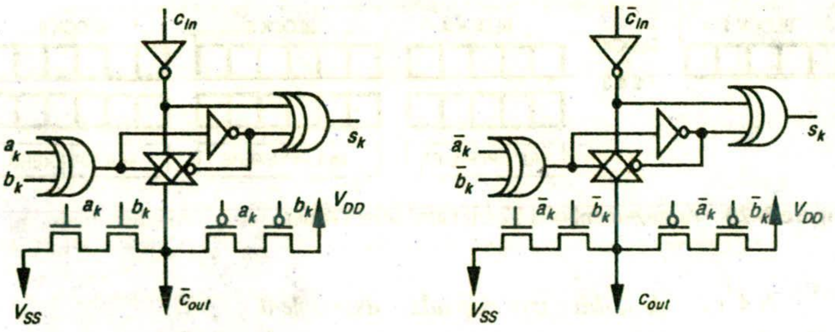


**Figure 8–27** Adder cells with alternative input/output arrangement

delay between carry in and carry out is denoted by $k_1$ (the delay through one adder cell).

### 8.4.3.1 A 32-bit carry select adder assessment

The multiplexers to be used invert the signal and are based on a simple cell comprising one inverter and one transmission gate as shown for the 2-way multiplexer of Figure 8–28.

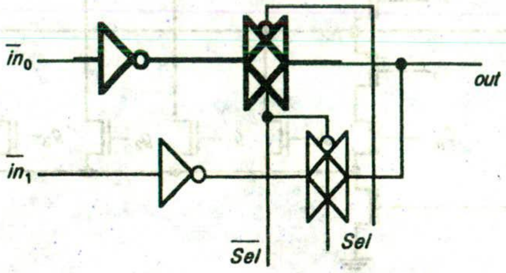


Figure 8–28 2-way multiplexer showing 'multiplexer cell' (bold lines)

Comparing this with the proposed adder cell, we may see that the multiplexer delay $k_2$ is the same as that for the adder cell so that $k_1 = k_2$ and, in consequence, the optimum block size evaluates as six. This does not divide exactly into 32, but we may choose to use four blocks of five cells and the remaining two blocks will then have six cells each as shown in Figure 8–29. The adder completion time is thus:

$$T = 5k_1 + 5k_2 = 10k$$

where $k = k_1 = k_2$.

The area of this 32-bit carry select adder is roughly twice that of a 32-bit ripple carry adder.

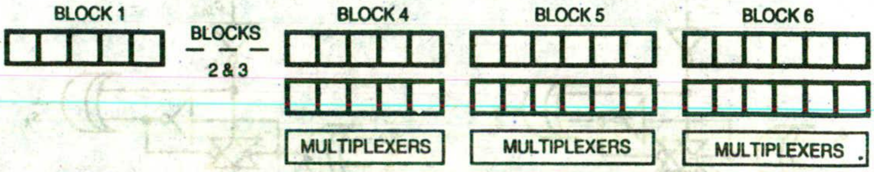


Figure 8–29 Arrangement of a 32-bit carry-select adder

### 8.4.3.2 A 32-bit carry skip adder assessment

Once again, the cell delay $k_1$ and the multiplexer (as in Figure 8–28) delay $k_2$ may be assumed to be equal. In order to simplify the propagation time assessment, we will neglect the time taken to compute all the generate and propagate, as well as the block propagation signals, since they are all computed simultaneously and may be represented as an overhead = $k \approx k_1$.

Care must be taken to allow for the inversion of the carry signal, both in the adder cell and in the multiplexer. For this reason, the block size must be an even number of bits. Again, since the ratio between the cell delay and the multiplexer delay is assumed to be 1:1, we may write

$$k_1 = k_2 = k$$

also, since the ratio $k_1 = k_2$, the optimum block size is four cells so that there will be eight blocks of equal size.

The adder completion time is thus:

$$T = 4k_1 + 4k_2 + 6k_2 + k = 15\,k$$

where $k = k_1 = k_2$.

The area of this 32-bit carry skip adder is roughly one and a half that of a 32-bit ripple carry adder.

### 8.4.3.3 *A 32-bit carry look-ahead adder assessment*

Figure 8–30 represents the structure of a 32-bit carry look-ahead adder. For reasons of simplicity in presentation, each heavy interconnect line represents the interconnection of two signals, $(g_k.p_k)$ and $(\gamma_k.\pi_k)$. The fine interconnect lines are the carry signals.

Let the delay time of a CLA unit be $k_3$, then the completion time of the adder may be assessed as follows:

At time $k_3$ : $(\gamma_k.\pi_k)$ for CLA 0–7 are set.
At time $2k_3$ : $(\gamma_k.\pi_k)$ for CLA 8 and 9 are set; $c_4$, $c_8$, and $c_{12}$ are set by CLA 8.
At time $3k_3$ : $c_{16}$ is set by CLA 10; using $c_4$, $c_8$, and $c_{12}$ CLA 1, CLA 2 and CLA 3 set their carry out.

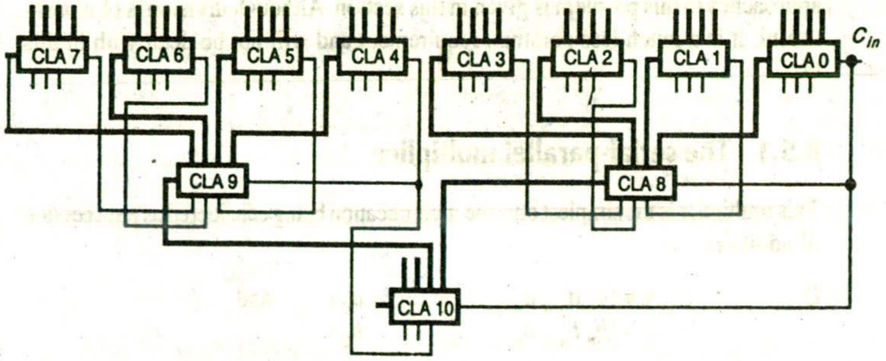


Figure 8–30 Arrangement of a 32-bit carry look-ahead adder

At time $4k_3$: $c_{20}$, $c_{24}$, and $c_{28}$, are set by CLA 9.
At time $5k_3$: Using $c_{20}$, $c_{24}$, and $c_{28}$, CLA 5, CLA 6 and CLA 7 set their carry out.

Therefore, overall time $T = 5k_3$.

The exact value of $k_3$ depends on the actual CLA adder element arrangement and on the layout used, but, allowing for three levels of logic, it could be conservatively estimated as $1.5k_1$ to $2.0\ k_1$, where $k_1$ is the delay of the simple adder cell used before in the carry select and skip adders. If this is a reasonable assumption then, in comparison with the other evaluations, overall time $T$ is given by

$$T = 7.5k \text{ to } 10k$$

However, noting the unused inputs of CLA 10 (Figure 8–30), it may be seen that a 64-bit CLA adder could be accommodated within the same overall time delay. Since the CLA cells are considerably more complex than the adder cells used in the carry select and carry skip adders, there will be a penalty in the area occupied. This is difficult to evaluate without detailed design work, but the area occupied will be several times greater than for a 32-bit ripple carry adder.

This concludes the consideration of adder circuitry. In the design of ALUs and digital processors generally, the adder is the most important circuit and is able to directly accommodate additions, subtractions and comparisons, together with a range of logical operations. Another common arithmetic requirement is for multiplication and it will be seen that the adder has an important role to play in the architecture of many multipliers.

## 8.5 Multipliers

A study of computer arithmetic processes will reveal that the most common requirements are for addition and subtraction, but that there is also a significant need for a multiplication capability. Thus, a brief overview of some common approaches to this problem is given in this section. Although division is obviously useful, it is a much less common requirement and will not be dealt with in this text.

### 8.5.1 The serial-parallel multiplier

This multiplier is the simplest one, the multiplication being considered as a succession of additions.

If $A = (a_n\, a_{n-1}\, a_{n-2} \cdots\cdots\cdots a_0)$ and
$B = (b_n\, b_{n-1}\, b_{n-2} \cdots\cdots\cdots b_0)$

then the product $A.B$ may be expressed as

$$A.B = (A.2^n . b_n + A.2^{n-1} . b_{n-1} + A . 2^{n-2} . b_{n-2} \ldots\ldots\ldots A.2^0.b_0)$$

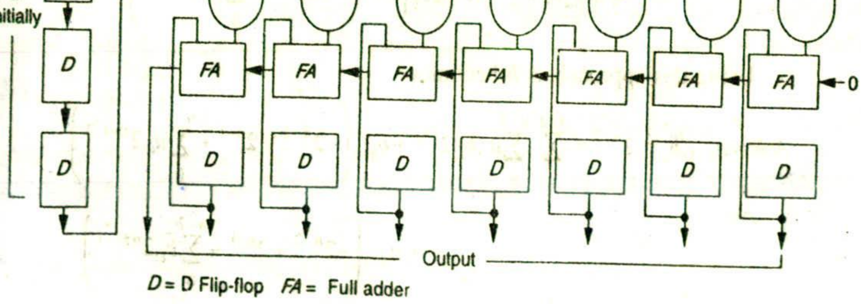
A possible form of this adder for multiplying four-bit quantities, based on this expression, is set out in Figure 8–31. Note that $D$ indicates a $D$ flip-flop and $F.A.$ indicates a full adder — or adder bit slice. Number $A$ is entered in the right-most 4-bits of the top row of $D$ flip-flops which are connected to three further $D$ flip-flops to form a 7-bit shift register to allow the multiplication of number $A$ by $2^1, 2^2 \ldots 2^n$, thus forming the *partial product* at each stage of the process.

In some cases, it may be easier to right shift the contents of the *Accumulator* — (bottom row of $D$ flip-flops) rather than left shifting $A$. This approach can be used to eliminate the least significant bits of the product if so desired.

A further reduction in hardware can result from noting that the three most significant bits of the partial product are set to zero initially, and are used only one by one as the shifting of $A$ proceeds. These three bits can therefore be used to hold three bits of number $B$ initially, thus saving three $D$ flip-flops.

The structure under discussion here is suited only to positive or unsigned operands. If the operands are negative and twos complement encoded, then:

1. The most significant bit of $B$ will have a negative weight and so a subtraction must be performed as the last step.
2. The most significant bit of $A$ must be replicated since operand $A$ must be expanded to $2N$ bits.



ure 8–31 Arrangement of a 4-bit serial-parallel multiplier

### 8.5.2 The Braun array

A relatively simple form of parallel adder is the Braun array (see Figure 8–32). All partial products $A.b_k$ are computed in parallel, then collected through a cascaded array of carry save adders. At the bottom of the array, an adder is used to convert the carry save form to the required form of output. Completion time is fixed by the depth of the array, and by the carry propagation characteristics of the adder. Notice that this multiplier is suited only to positive operands. Negative operands can be handled, for example, by the Baugh-Wooley multiplier which now follows.

### 8.5.3 Twos complement multiplication using the Baugh-Wooley method

This technique has been developed to design multipliers that are regular in structure and suited for twos complement numbers.

Let us consider two numbers $A$ and $B$:

$$A = (a_{n-1}\ldots\ldots a_0) = -a_{n-1}.2^{n-1} + \sum_{0}^{n-2} a_i.2^i$$

$$B = (b_{n-1}\ldots\ldots b_0) = -b_{n-1}.2^{n-1} + \sum_{0}^{n-2} b_i.2^i$$

The product $A.B$ is given by:

$$A.B = a_{n-1}.b_{n-1}.2^{n-2} + \sum_{0}^{n-2}\sum_{0}^{n-2} a_i.b_j.2^{i+j} - a_{n-1}\sum_{0}^{n-2} b_i.2^{n+i-1} - b_{n-1}\sum_{0}^{n-2} a_i.2^{n+i-1}$$

If we use this form, it may be seen that subtraction operations are needed as well as addition. However, the negative terms may be rewritten, for example:

$$a_{n-1}\sum_{0}^{n-2} b_i.2^{n+i-1} = a_{n-1}.\left(-2^{n-2} + 2^{n-1} + \sum_{0}^{n-2}\bar{b}_i.2^{n+i-1}\right)$$

Using this approach, $A.B$ becomes

$$A.B = a_{n-1}.b_{n-1}.2^{n-2} + \sum_{0}^{n-2}\sum_{0}^{n-2} a_i.b_i.2^{i+j} + b_{n-1}\left(-2^{n-2} + 2^{n-1} + \sum_{0}^{n-2}\bar{a}_i.2^{n+i-1}\right)$$
$$+ a_{n-1}\left(-2^{n-2} + 2^{n-1} + \sum_{0}^{n-2}\bar{b}_i.2^{n+i-1}\right)$$

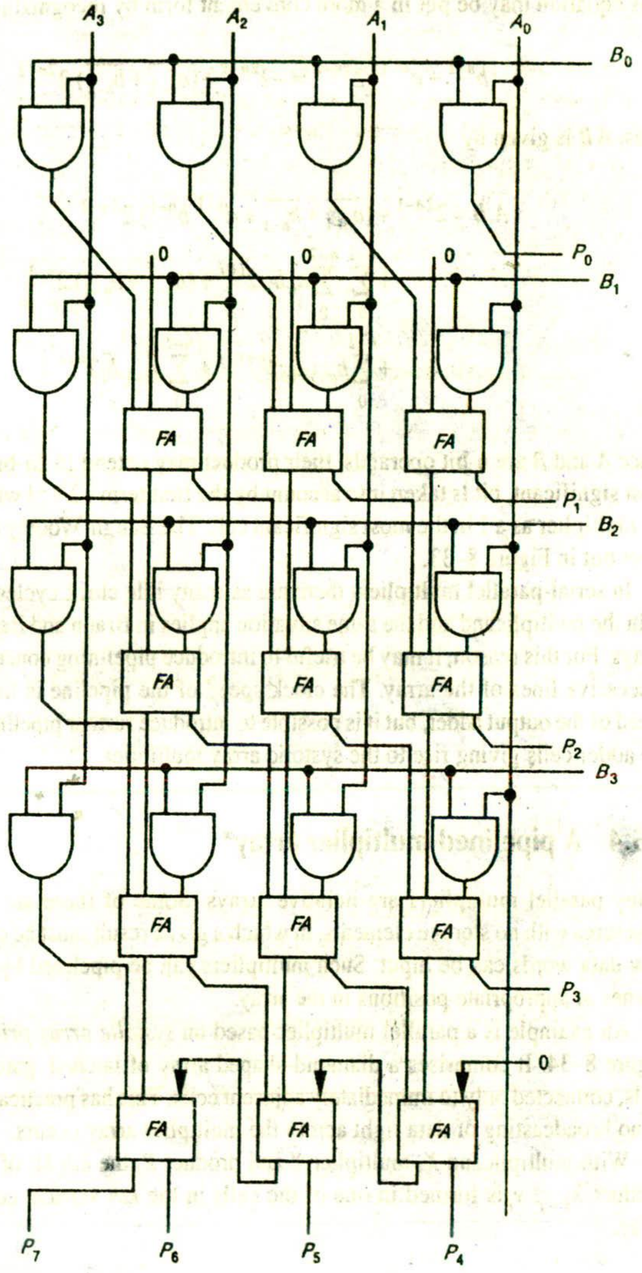


**Figure 8–32** A 4-bit Braun multiplier

This equation may be put in a more convenient form by recognizing that

$$-(b^{n-1}+a^{n-1}).2^{2n-2} = -2^{2n-1}+(\overline{a_{n-1}}+\overline{b_{n-1}}).2^{2n-2}$$

Thus, $AB$ is given by

$$A.B = 2^{2n-1}+(\overline{a_{n-1}}+\overline{b_{n-1}}+a^{n-1}.b^{n-1}).2^{2n-2}$$
$$+\sum_{0}^{n-2}\sum_{0}^{n-2} a_i.b_j.2^{i+j}+(a_{n-1}+b_{n-1}).2^{n-1}$$
$$+\sum_{0}^{n-2} b_{n-1}.\overline{a_i}.2^{n+1-j}+\sum_{0}^{n-2} a_{n-1}.\overline{b_i}2^{n+i-1}$$

Since $A$ and $B$ are $n$-bit operands, their product may extend to $2n$-bits. The first, most significant, bit is taken into account by the first term $-2^{2n-1}$ which is fed to the multiplier as a 1 in the most significant cell. The Baugh-Wooley arrangement is set out in Figure 8–33.

In serial-parallel multipliers there are as many idle clock cycles as there are 0s in the multiplicand and the same situation applies in Braun and Baugh-Wooley arrays. For this reason, it may be useful to introduce pipelining concepts between successive lines of the array. The clock speed of the pipeline is limited by the speed of the output adder, but it is possible to introduce further pipelining between the adder cellş giving rise to the systolic array multiplier.

## 8.5.4 A pipelined multiplier array*

Many parallel multipliers are iterative arrays. Some of these are carry-ripple structures with no storage elements, in which a given result must be output before new data words can be input. Such multipliers can be pipelined by introducing latches at appropriate positions in the array.

An example is a parallel multiplier based on *systolic array principles* as in Figure 8–34. It comprises a diamond-shaped array of latched, gated full adder cells, connected only to immediately adjacent cells. This has practical advantages as no broadcasting of data right across the multiplier array occurs.

With multiplicand $X$, multiplier $Y$ and product $P$, the *kth* bit of each partial product $X_{k-i}.\ y_i$ is formed in one of the cells in the *kth* vertical column of the array.

* J. V. McCanny and J. G. McWhirter, 'Completely iterative, pipelined multiplier array suitable for VLSI', *IEE Proc*, vol. 129, pt. G, no. 2, 40–46. This structure was designed by P. Evans as part of a VLSI course at the University of Adelaide, South Australia.

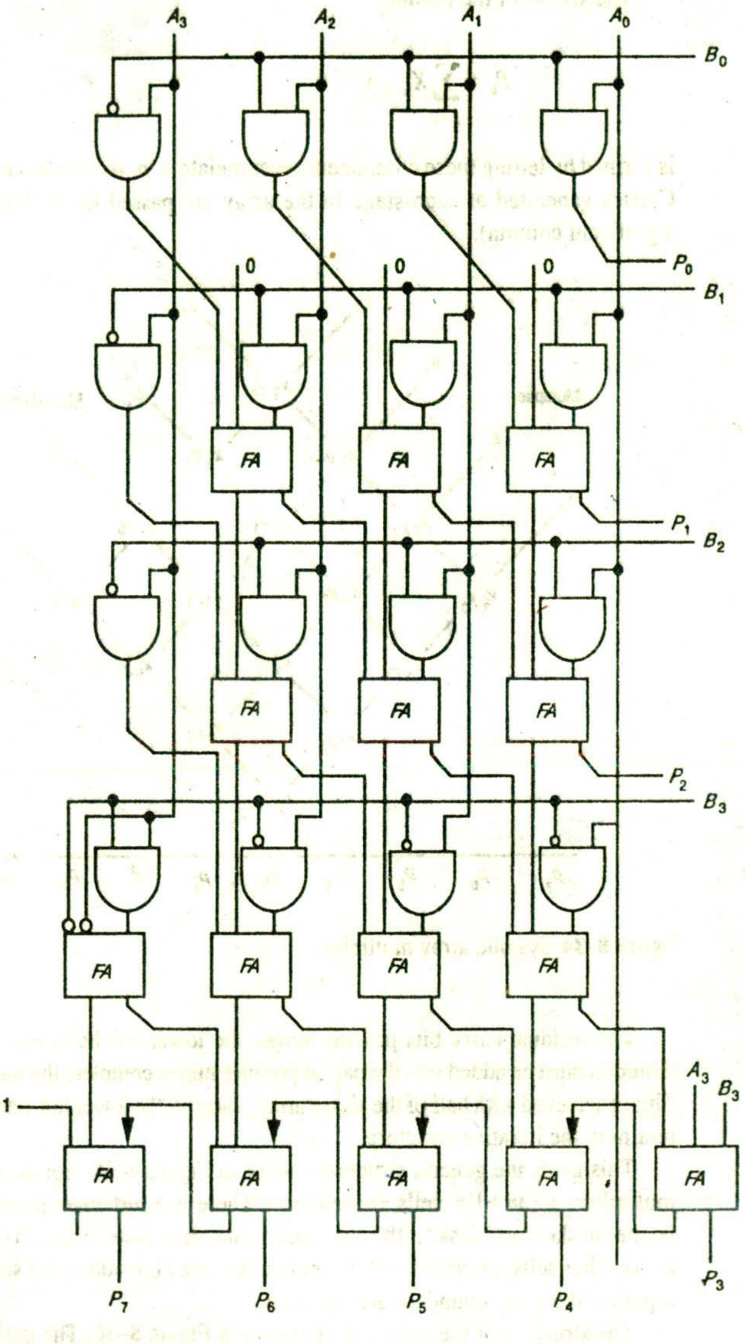


**Figure 8–33** A 4-bit Baugh-Wooley multiplier

The *kth* bit of the product

$$P_k = \sum_{i=0}^{k} X_{k-i} \cdot y_i$$

is formed by letting these components accumulate as $p_k$ passes down the column. Carries generated at each stage in the array are passed to the left (next most significant column).

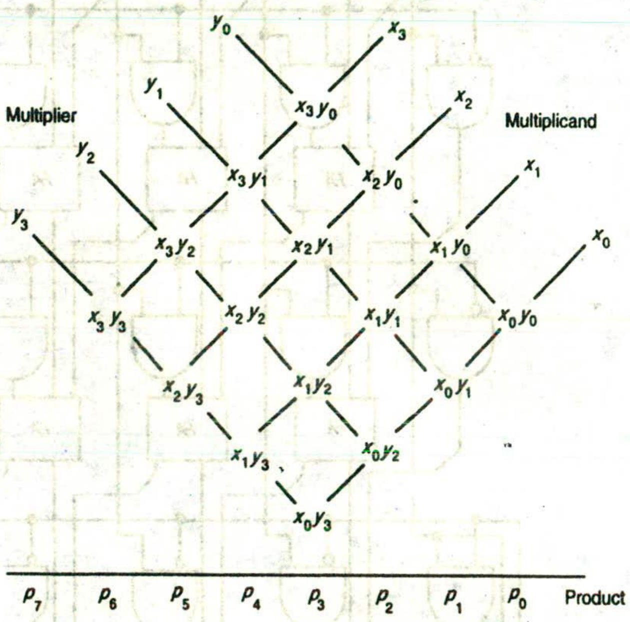


**Figure 8–34** Systolic array multiplier

The residual carry bits passing across the lower left-hand boundary of the diamond must be added into the partial product sum to complete the multiplication. This is achieved with half of the above array placed at the lower left-hand boundary, retaining the iterative structure.

This gives the general structure shown in Figure 8–35. For an *n*-bit × *n*-bit multiplier, $\frac{1}{2}(3n+1)n$ cells are required. There is a further requirement of $3n^2$ latches to skew and deskew the input and output data. Note that each cell connects to six other cells, provided that it is not on the array boundary. All sum and carry inputs at the array boundary are set to zero.

The structure of the basic cell is shown in Figure 8–36. The gating function for unsigned numbers is *x.y*.

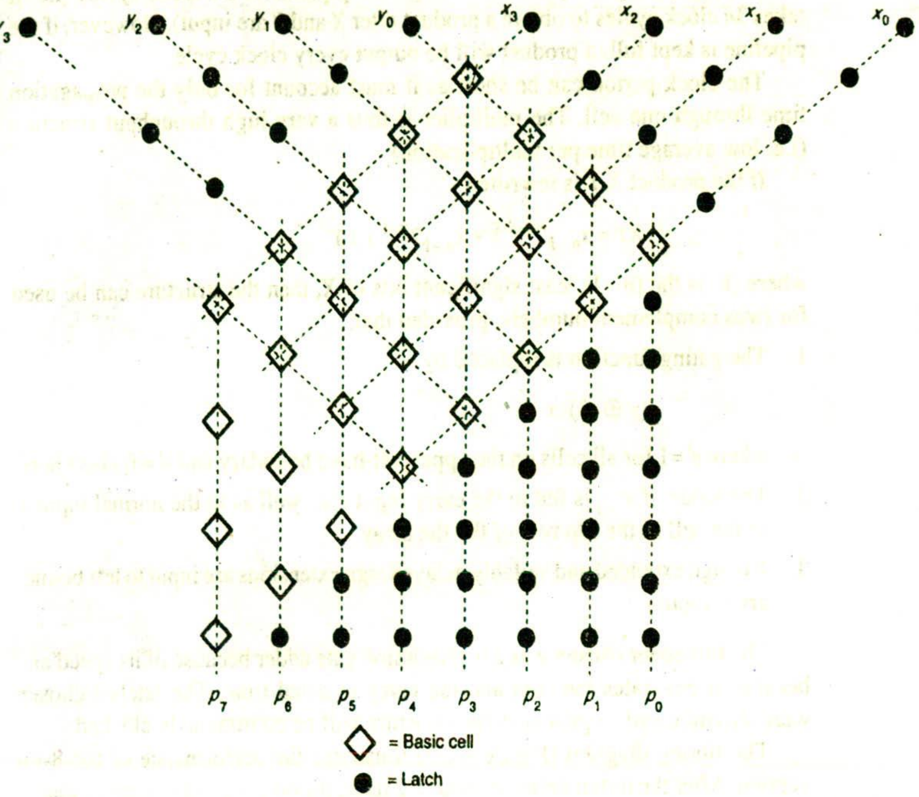
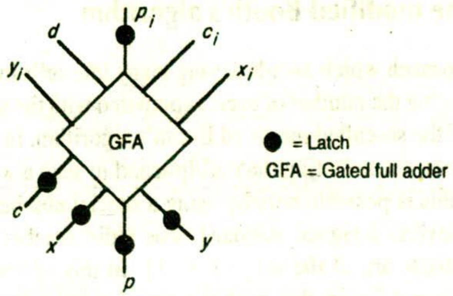


**Figure 8–35** Multiplier structure



*Note*: Where $p_i$ = partial product sum in
$p$ = partial product sum out
$c_i$ = carry ~~in~~
$c$ = carry out
$d$ = line required for twos complement operation

**Figure 8–36** Basic cell

The delay of one operation through the pipeline is $3n$ clock cycles (i.e. it takes $3n$ clock cycles to obtain a product after $X$ and $Y$ are input). However, if the pipeline is kept full, a product will be output every clock cycle.

The clock period can be short as it must account for only the propagation time through one cell. The multiplier is thus a very high throughput structure (i.e. low average time per multiplication).

If the product $X.Y$ is rewritten

$$XY = x_{n-1}2^{n-1}.\bar{Y} + x_{n-1}2^{n-1} + \tilde{x}.Y$$

where $\tilde{x}$ is the $(n-1)$ least significant bits of $X$, then the structure can be used for twos complement numbers, provided that:

1. The gating function is replaced by

   $$(y \oplus d).x$$

   where $d = 1$ for all cells on the upper left-hand boundary and $d = 0$ elsewhere.
2. The value of $x_{i-1}$ is fed to the carry input $c_i$ as well as to the normal input $x_i$ of the cell in the top row of the the array.
3. $Y$ is sign extended and suitably delayed sign extensions are input to left boundary $y_i$ inputs.

The full adder chosen was a transmission gate adder because of its speed and because it generates the sum and the carry in equal time. The latches chosen were dynamic shift registers as the structure will be continuously clocked.

The timing diagram (Figure 8–37) illustrates the performance of the 8-bit version. After the initial delay of about 1.2 µsec, the output products are available at 50 nsec intervals.

## 8.5.5 The modified Booth's algorithm

Another approach which avoids having many idle cells in a cellular multiplier as well as reducing the number of cycles compared with the serial-parallel multiplier is the use of the so-called modified Booth's algorithm. In principle, the modified algorithm requires rewriting the multiplicand in such a way that half the bits are 0. Clearly, this is possible only by using a special number system.

This converts a signed standard twos radix number into a number system where the digits are in the set {–1, 0, 1}. In this system any number may be written in several forms, that is, the system has redundancies.

Let us consider a number $B = b_{n-1}\, b_{n-2} \cdot \cdots \cdot b_1\, b_0$ written in twos complement form:

$$B = -b_{n-1}.2^{n-1} + \sum_{k=0}^{n-2} b_k.2^k$$

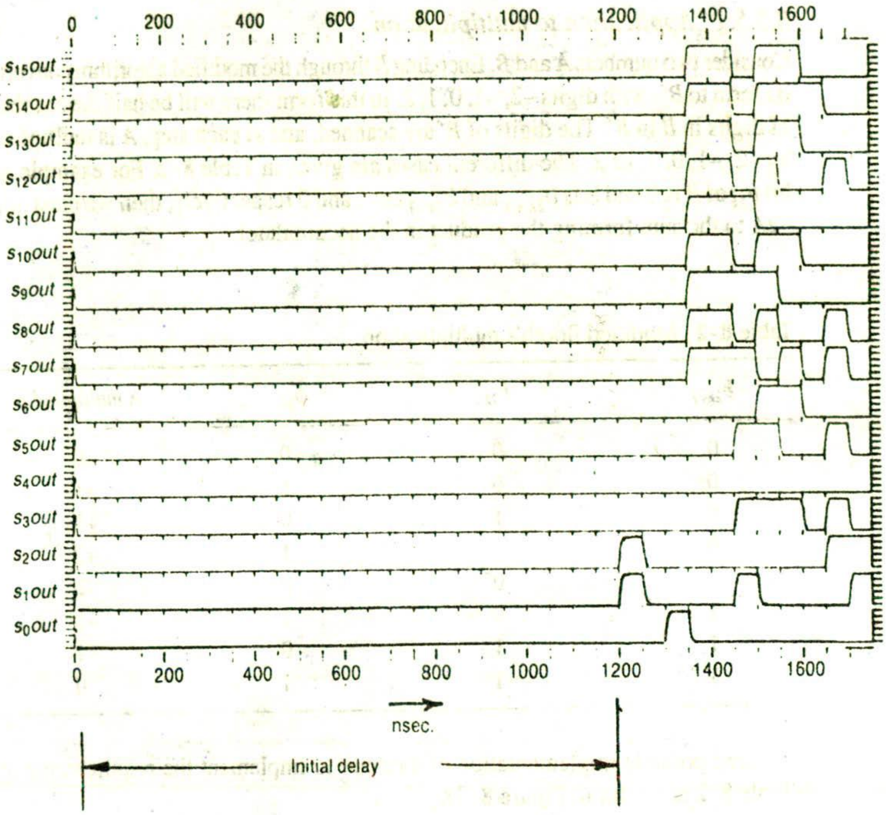


**Figure 8–37** Performance of an 8-bit multiplier

which may be rewritten as

$$B = \sum_{k=0}^{n-2-1} (b_{2k-1} + b_{2k} + 2b_{2k+1})2^k$$

with $b_1 = 0$.

In this equation, the term in the brackets is in the set {−2, −1, 0, 1, 2}, so it cannot be equal to 3 or −3. In other words, after rewriting $B$ through the modified algorithm, each pair of digits can only take the following forms: [ −1, −1], [0, −1], [0, 0], [0, 1], [1, 1], that is (−2, −1, 0, 1, 2). Another consequence of the modified Booth's algorithm is that the sign of the numbers is implicitly taken into account.

### 8.5.5.1 Application to multiplication

Consider two numbers $A$ and $B$. Encoding $B$ through the modified algorithm converts its form to $B'$ with digits –2, –1, 0, 1, 2. In this form there will be half the number of digits in $B$ in $B'$. The digits of $B'$ are scanned, and at each step, $A$ is multiplied by –2, –1, 0, 1, or 2. The different cases are given in Table 8–2. For example, if bit $b_{2k}$ of $B$ is 0 and bits $b_{2k+1}$ and $b_{2k-1}$ are 1 and 0 respectively, then we must add $-2A$ to the sum forming the product in the accumulator.

**Table 8–2** Modified Booth's multiplication

| $b_{2k+1}$ | $b_{2k}$ | $b_{2k-1}$ | *A multiplied by* |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | +1 |
| 0 | 1 | 0 | +1 |
| 0 | 1 | 1 | +2 |
| 1 | 0 | 0 | –2 |
| 1 | 0 | 1 | –1 |
| 1 | 1 | 0 | –1 |
| 1 | 1 | 1 | 0 |

One possible implementation of a circuit to implement the requirements of Table 8–2 is set out in Figure 8–38.

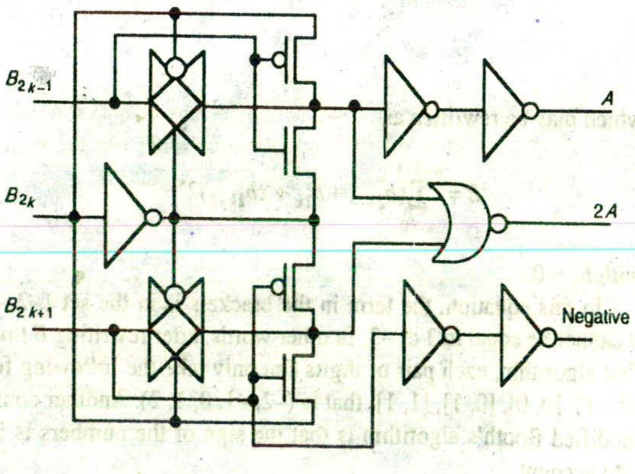


**Figure 8–38** Booth encoder

## 8.5.6 Wallace tree multipliers

Wallace trees were first introduced in 1964 (Wallace, 1964) in order to design multipliers whose completion time grows as the logarithm of the number of bits to be multiplied. The simplest Wallace tree is the full adder cell (three inputs — two outputs). More generally, an $n$ input Wallace tree, as in Figure 8–39, is an $n$-input operation with $log_2(n)$ outputs, such that the value of the output word is equal to the number of '1'*s* in the input word (consider the full adder in this context). The input bits and the least significant bit of the output word have the same weight, as shown in Figure 8–39.

An important property of Wallace trees is that they may be constructed from adder cells. Furthermore, the number of adder cells needed grows as the logarithm $log_2(n)$ of the number of input bits $n$. In a Braun or a Baugh-Wooley multiplier with a ripple carry adder, the completion time for multiplication is proportional to $2n$. If the collection of the partial products is made through Wallace trees then the completion time for getting a result, in carry save notation, should be proportional to $log_2(n)$.

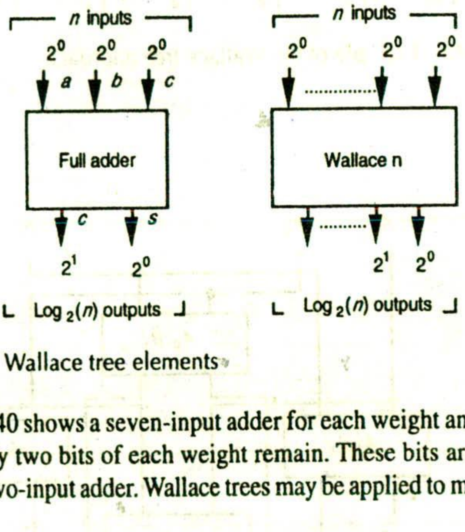


**Figure 8–39** Wallace tree elements

Figure 8–40 shows a seven-input adder for each weight and Wallace trees are used until only two bits of each weight remain. These bits are then added using the classical two-input adder. Wallace trees may be applied to multipliers in several ways.

## 8.5.7 Recursive decomposition of the multiplication

One method, based on recursive decomposition of the multiplication, consists of partitioning the operands. For instance, if $A$ and $B$ are $2p$-bit numbers, then $A$ (also $B$) may be cut into two parts $A_0$ and $A_1$ respectively, so that

$$A = 2^p . A_1 + A_0$$

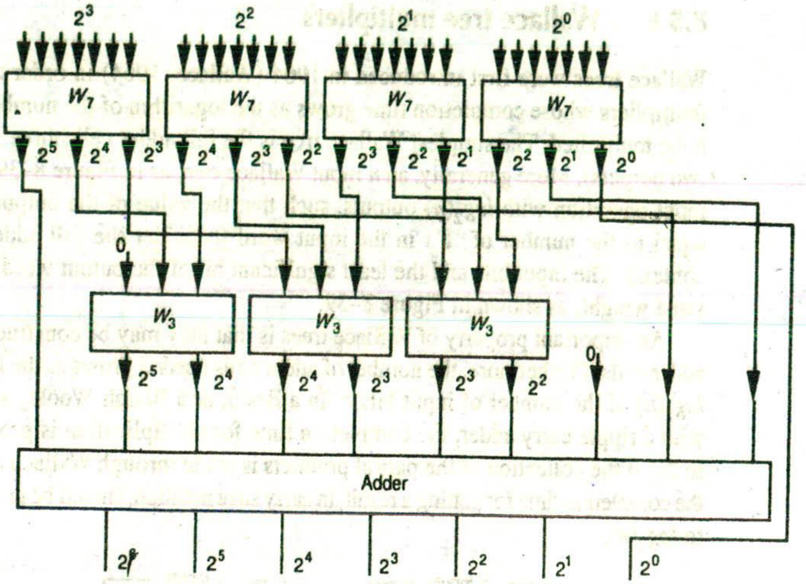


**Figure 8–40** Example of the Wallace tree approach

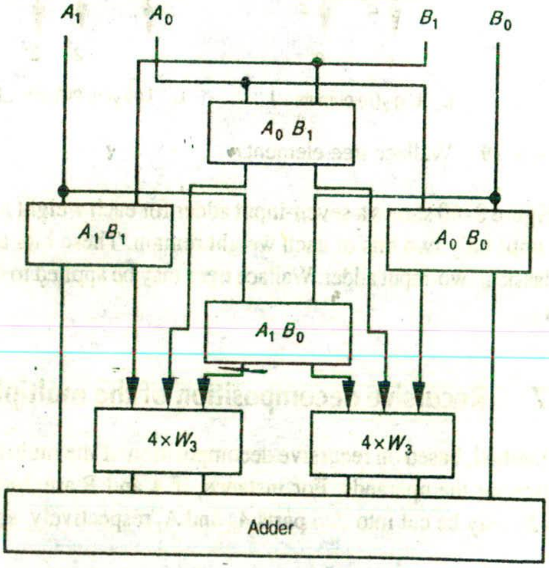


**Figure 8–41** 8-bit input word multiplier arrangement

$$B = 2^p . B_1 + B_0$$

The product $A.B$ is

$$A.B = 2^{2p}. A_1B_1 + 2^{6p}. (A_1 . B_0 + A_0 . B_1) + A_0 . B_0$$

Using this method, four $p$-bit multipliers are used to compute $A_1.B_1$, $A_0.B_1$, $A_1 . B_0$ and $A_0 . B_0$. The results are collected through Wallace trees. The arrangement of a multiplier of this type, with 8-bit input words, is shown in Figure 8–41; the interconnections have been simplified for clarity. $A_0$, $B_0$, $A_1$ and $B_1$, are in fact 4-bit numbers and the outputs of the multiplier are 8-bit products. In this figure it has been assumed that the multipliers each contain an adder so that each result is not in carry save notation and thus eight adder cells (three-input Wallace trees) are used to collect bits of the same weight. For instance, the multipliers denoted $A_0 . B_1$, $A_1 . B_0$ and $A_0 . B_0$ give bits of weight 4, 5, 6, and 7. For each of these weights, three bits (as many as there are multipliers) must be added, and thus an adder cell must be used to reduce the number of bits of the same weight to two.

### 8.5.8 Dadda's method

Another approach consists in computing all the partial products — like the Braun array — and then collecting all the bits of the same weight through Wallace trees. This is equivalent to partitioning the input operands to work with 1-bit multipliers (i.e. *And* gates). In 1965 L. Dadda developed a technique to build the Wallace layer using the minimum number of adder cells.

Consider $k$ bits of the same weight $i$ coming from $k$ partial products. When adding these $k$ bits by a $k$-input Wallace tree, bits of weights $i + 1$, $i + 2$, . . . etc. appear which must in turn be added to the bits of weights $i + 1$, $i + 2$, . . . coming from other partial products. Dadda's method consists in handling all bits in the collecting Wallace layer so as to minimize the number of adder cells as well as the critical path between the partial product generation and the final addition. All the developments of this technique may be found in the reference (Dadda, 1965). In conclusion, Wallace tree multipliers should be used only for large operands and where the performance is critical since the arrangement results in poor regularity due to the routing area needed to collect the partial products.

## 8.6 Observations

This chapter has provided possible designs for the arithmetic subsystem forming part of the complete data path we are designing. Both the subsystems so far designed have comprised only combinational logic with the exception of possible storage requirements at the 'Sum' output of the adder. The third subsystem, to be designed next, will introduce a need for memory or storage and this leads to a review of some possible memory elements and relevant characteristics.

## 8.7 Tutorial exercises

1. Referring to Figure 8–12, design switches and other logic as necessary to implement the functions performed by the mechanical switches drawn in Figure 8–12. Work out the control lines needed to enable the ALU to perform add, subtract, logical *And*, logical *Or*, logical *Exclusive-Or*, and logical *Equality* operations.
2. Draw a bounding box representation with all inlet and output points shown (as in Figure 8–10) for the *logic circuitry* of an adder, using CMOS multiplexers (Figure 8–4) and CMOS inverters as suggested in Figure 8–9. You may wish to proceed as follows.

   Continue the design of a standard CMOS adder element (as represented in stick diagram form in Figure 8–5) by working out a layout for the complete inverter block and then representing it as a bounding box with inlet and outlet points indicated by layer and position. *Hint:* Design a suitable mask layout for the CMOS inverters and then represent each inverter circuit in bounding box form — with inlet and outlet points — so that only one inverter needs to be drawn in detail in setting out your layout.

   Interconnect the inverter block bounding box with CMOS multiplexer-based adder logic (as in Figure 8–4). Work out an accurate bounding box representation for the complete adder element showing inlet and outlet points, etc., by position and layer.
3. What are the overall dimensions of a 4-bit CMOS adder? Using the bounding box representations draw an accurate floor plan of the whole 4-bit adder (as in Figure 8–11) showing position and layer of inlet and outlet points.
4. Carry out the design of a 4-bit CMOS carry look-ahead adder up to stick diagram form. Then determine what standard cells are needed and design a mask layout for each.

## 8.8 References

Dadda, L. (1965, March) 'Some schemes for parallel multipliers', *Alta Frequenza*, Vol. 19.

Guyot, A., Hochet, B., and Muller, J. M. (1987, October) 'A way to build efficient carry-skip adders', *IEE Trans. on Computers*, Vol. C–36.

Hotta, T. et al. (1986, October) 'CMOS/Bipolar circuit for 60 MHz digital processing', *IEEE Journal of Solid State Circuits*, 803–13, Vol. 21, No. 5.

Muller, J. M. (1989) *Arithmétique des Ordinateurs*, Editions Masson, Collection Etudes et Recherches en Informatique, Paris.

Wallace, C. S. (1964, February) 'A suggestion for a fast multiplier', *IEEE Trans. on Electronic Computers*, 14–17.

Wang, I. S. and Fisher, A. L. (1989, April) 'Ultrafast compact 32-bit CMOS adders in multiple-output domino logic', *IEEE Journal of Solid State Circuits*, Vol. 24.

# 9 Memory, registers, and aspects of system timing

*Ay, now the plot thickens very much upon us.*
George Villiers, 2nd Duke of Buckingham

## Objectives

The 4-bit data path design continues with the 4 × 4-bit register array. This raises the subject of memory/storage elements and techniques. Some of the possible dynamic and static memory cells are presented and key properties compared. The concept of an array of memory cells is extended to include RAM arrays and some of the needs for selection and control are explored.

Two of the subsystems of the 4-bit data path (as in Figures 7–3 and 8–1) having already been designed, it is now appropriate to consider the register arrangements in which the 4-bit quantities to be presented to the adder and shifter will be stored. The question of data storage is an important one which has already been mentioned a number of times. It raises the question of the choice of storage elements or memory cells as well as the questions of configuring arrays of such cells and the selection of a given cell or group of cells in an array.

Before looking at register arrangements, we should set out some ground rules for the design of the 4-bit processor. It is essential that such rules should be established early in the piece so that a uniform approach to 'reading, writing and refresh' is adhered to throughout. In practice, such rules would have been set out much earlier than this, but our progress through this text is such that in this case they are most effectively established here and would not have meant much earlier on.

## 9.1 System timing considerations

1. A two-phase non-overlapping clock signal is assumed to be available, and this clock alone will be used throughout the system.
2. Clock phases are to be identified as $\phi_1$ and $\phi_2$ where $\phi_1$ is assumed to lead $\phi_2$.
3. Bits (or data) to be stored are *written* to registers, storage elements, and subsystems on $\phi_1$ of the clock; that is, write signals *WR* are *Anded* with $\phi_1$.
4. Bits or data written into storage elements may be assumed to have settled before the immediately following $\phi_2$ signal, and $\phi_2$ signals may be used to *refresh* stored data where appropriate.
5. In general, delays through data paths, combinational logic, etc. are assumed to be less than the interval between the leading edge of $\phi_1$ of the clock and the leading edge of the following $\phi_2$ signal.
6. Bits or data may be *read* from storage elements on the next $\phi_1$ of the clock; that is, read signals *RD* are *Anded* with $\phi_1$. Obviously, *RD* and *WR* are generally mutually exclusive to any one storage element.
7. A general requirement for system stability is that there must be at least one clocked storage element in series with every closed loop signal path.

Strict adherence to a set of rules such as this will greatly simplify the ta k of the system designer and also help to avoid some of the disasters which will almost certainly occur if a haphazard approach is taken.

## 9.2 Some commonly used storage/memory elements

*Everyone complains of his memory, but no one complains of his judgment.*
Duc de la Rochefoucauld

In order to make a comparative assessment of some possible storage elements, we will consider the following factors:

- area requirement;
- estimated dissipation per bit stored;
- volatility.

### 9.2.1 The dynamic shift register stage

One method of storing a single bit is to use the shift register approach previously introduced in section 6.5.4 (and also Figures 3–14, 3–17, 6–36, 6–37, 6–38, 6–39 and 6–40).

#### 9.2.1.1 Area

This calculation applies to an nMOS design, as in Figure 6–40(a), with buried contacts. Allowing for the sharing of $V_{DD}$ and $GND$ rails between adjacent rows of register cells, each bit stored will require

$$(22\lambda \times 28\lambda) \times 2 \div 1200\lambda^2$$

For example, for $\lambda = 2.5\ \mu m$

$$\text{Area/bit} = 7500\ \mu m^2$$

To give an idea of what this implies, such area requirements would result in a maximum number of bits stored on a 4 mm × 4 mm chip area ≈ 2.1 kbits.

For a CMOS design, as in Figure 6–40(b), and allowing for the sharing of $V_{DD}$ and $V_{SS}$ rails between adjacent rows of register cells, each bit stored will require

$$(38\lambda \times 28\lambda) \times 2 \div 2100\lambda^2$$

For example, for $\lambda = 2.5\ \mu m$

$$\text{Area/bit} \approx 13{,}000\ \mu m^2$$

Such area requirements would result in a maximum number of bits stored on a 4 mm × 4 mm chip area ≈ 1.2 kbits.

#### 9.2.1.2 Dissipation

In the case of CMOS designs, the static dissipation is very small and calculation at this stage will not be meaningful since only the switching dissipation will be significant (particularly at high speeds). This dynamic power consumption $P_d$ can be written as

$$P_d = m.(C_L \cdot V_{DD}^2 \cdot f)$$

where $m$ is the duty cycle, $C_L$ is effective load capacitance and $f$ is the clock frequency.

In the nMOS case we can readily calculate the static dissipation, noting that in practice the switching dissipation would add to this. Each inverter stage has a ratio of 8:1 and if the layout of Figure 6–40(a) (buried contacts) is used, then, noting that one inverter of the pair must always be 'on',

$$Z_{p.u.} = 4R_s$$

and

$$Z_{p.d.} = \tfrac{1}{2}R_s$$

Therefore

$$\text{Current} = \frac{V_{DD}}{Z_{p.u.} + Z_{p.d.}} = \frac{5\text{ V} \times 10^6}{4.5 \times 10^4} = \frac{500}{4.5}\,\mu\text{A} \approx 110\,\mu\text{A}$$

Therefore

$$\frac{\text{Static dissipation}}{\text{Bit stored}} = V_{DD} \times \text{current} = 5\text{ V} \times 110\,\mu\text{A} = 550\,\mu\text{W}$$

Thus, 2.1 kbits on a single chip would dissipate

$$2.1 \times 10^3 \times 550 \times 10^{-6} = 1.15 \text{ watts}$$

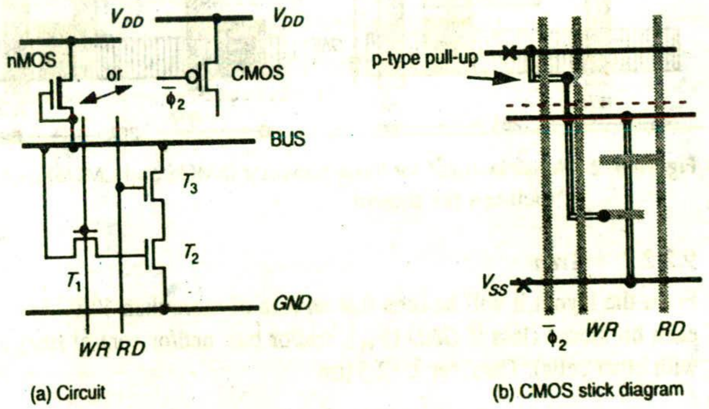
Dissipation can be reduced by using alternative geometry, but this is at the expense of increased area.

### *9.2.1.3 Volatility*

Data is stored by the charge on the gate capacitance of each inverter stage, so that data storage time (without refresh) is limited to 1 msec or less.

## 9.2.2 A three-transistor dynamic RAM cell

An arrangement which has been used in RAM (random access memory) and other storage arrangements is set out in Figure 9–1.



*Note*: *WR* and *RD* are coincident with $\phi_1$.

**Figure 9–1** Three-transistor dynamic memory cell

With regard to Figure 9–1(a), the action is as follows:

1. With the *RD* control line in the Lo state, then a bit may be read from the bus through $T_1$ by taking *WR* to the Hi state so that the logic level on the bus is communicated to the gate capacitance of $T_2$. Then *WR* is taken Lo again.

2. The bit value is then stored for some time by $Cg$ of $T_2$ while both *RD* and *WR* are Lo.
3. To read the stored bit it is only necessary to make *RD* Hi and the bus will be pulled down to ground through $T_3$ and $T_2$ if a 1 was stored. Otherwise, $T_2$ will be non-conducting and the bus will remain Hi due to its pull-up arrangements.

   Note that the complement of the stored bit is read onto the bus, but this presents few problems and can be taken care of at some common point in the memory array.

A stick diagram for the cell identified in Figure 9–1(a) is presented as Figure 9–1(b), and possible mask layouts follow in Figure 9–2. Note that this figure gives both nMOS and CMOS designs.

To return to our main theme, it is now appropriate to assess the three-transistor cells in the same manner as the previous one.

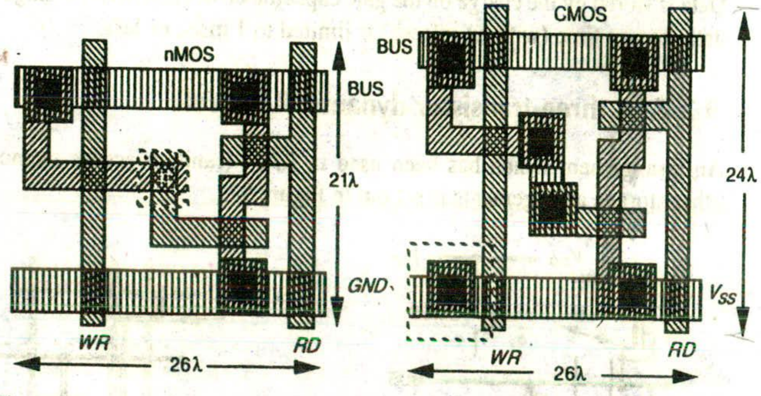


**Figure 9–2** Mask layouts* for three-transistor (nMOS and CMOS) memory cell *(pull-ups not shown)

### 9.2.2.1 *Area*

From the layout it will be seen that an area of more than $500\lambda^2$ is required for each bit stored (less if *GND* ($V_{SS}$), and/or bus, and/or control lines are shared with other cells). Thus, for $\lambda = 2.5\ \mu m$

$$\text{Area/bit} \doteqdot 3000\ \mu m^2$$

Thus, to continue the previous example, the maximum number of bits which could be accommodated on a 4 mm × 4 mm silicon chip is > 4.8 kbits.

### 9.2.2.2 *Dissipation*

Static dissipation is nil since current flows only when *RD* is Hi and a 1 is stored. Thus, the actual dissipation associated with each bit stored will depend on the bus pull-up and on the duration of the *RD* signal and on the switching frequency.

#### *9.2.2.3 Volatility*

The cell is 'dynamic' and will hold data only for as long as sufficient charge remains on $C_g$ (of $T_2$).

### 9.2.3 A one-transistor dynamic memory cell

The area occupied by each bit stored in each of the previous cases is quite considerable, which clearly limits, say, the number of bits which could be stored on a single chip of reasonable size.

Various approaches have been taken to reduce the area per bit requirements and one such approach is the one-transistor cell as shown in Figure 9–3. The concept of the single transistor cell is quite simple, as may be seen from Figure 9–3(a). It basically consists of a capacitor $C_m$ which can be charged during *'write'* from the *read/write* line, provided that the *row select* line is Hi. The state of the charge $C_m$ can be read subsequently by detecting the state of the charge via the same *read/write* line with the *row select* line Hi again, and a sense amplifier of a suitable nature can be designed to differentiate between a stored 0 and a stored 1.

However, in practice the cell is slightly more complex than first considerations might suggest, since special steps must be taken to ensure that $C_m$ has sufficient capacitance to allow ready detection of the stored content.

The most obvious and readily fabricated $C_m$ in the structure under consideration would be to extend and enlarge the diffusion area comprising the source ($S$) of the pass transistor in Figure 9–3(b). We would then rely on the junction capacitance between the n-diffusion region and the p-substrate to form $C_m$. However, if we consult Table 4–2 (which gives capacitance values for a typical MOS process), we will see that the capacitance per unit area of diffusion is much less than the capacitance per unit area between gate and channel (i.e. between the channel under the thin gate oxide and the polysilicon gate area).

If we use the diffusion to substrate capacitance alone, a comparatively large area will be required to give any significant value of capacitance; for example, at least $16\lambda^2$ will be needed to give a capacitance equal to $1\square C_g$ (i.e. 0.01 pF in the 5 µm MOS process being considered). A solution is to create a much more significant capacitor by using a polysilicon plate (which is connected to $V_{DD}$) over the diffusion area. Thus, $C_m$ is formed as a three-plate structure as indicated in Figure 9–3(c). For example, for the area given in Figure 9–3(d), $C_{Diff.\text{-}Poly.} = 10\square C_g$ (= 0.1 pF), while the contribution from the diffusion region to the substrate will be much smaller but will add some 25% to this figure, giving a total $C_m$ of 0.125 pF for the layout considered. Even so, careful design is necessary to achieve consistent readability.

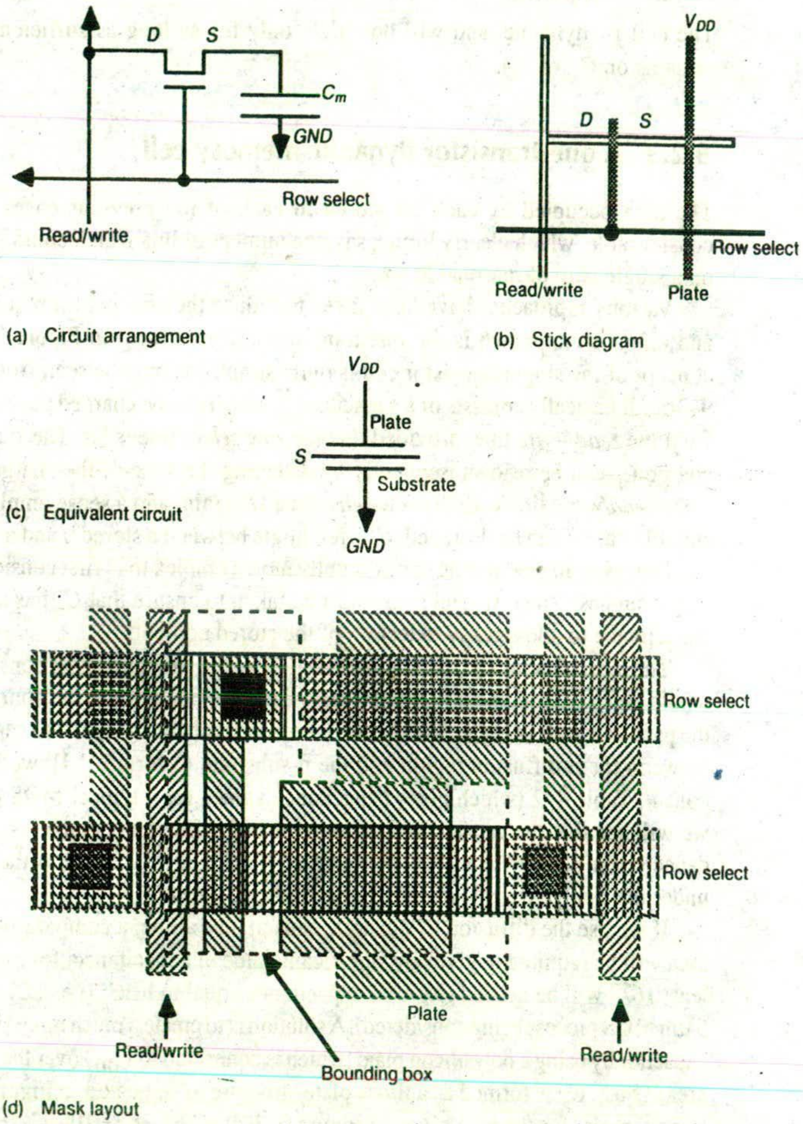


**Figure 9–3** One-transistor memory cell

#### 9.2.3.1 Area

The area enclosed to indicate the standard cell in Figure 9–3(d) is $200\lambda^2$. Thus for $\lambda = 2.5\ \mu m$, area/bit stored $= 200\lambda^2 = 1250\ \mu m^2$.

Therefore, the number of bits per 4 mm × 4 mm chip area is approximately 12 kbits (allowing some 'overheads' for sensing, etc.).

#### 9.2.3.2 Dissipation

There is no static power associated with the cell itself, but there must be an allowance for switching energy while writing to and reading from the storage elements.

#### 9.2.3.3 Volatility

Quite obviously, leakage current mechanisms will deplete the charge stored in $C_m$ and thus the data will be held for only up to 1 msec or less. Therefore, periodic refresh operations must be provided. It will also be realized that reading the cell is a destructive operation and that the stored bit must be rewritten every time it is read.

### 9.2.4 A pseudo-static RAM/register cell

So far, all the storage elements considered have been volatile and thus have an implied need to be periodically refreshed. This is not always convenient and it is necessary to consider the design of a static storage cell which will hold data indefinitely. A common way of meeting this need is to store a bit in two inverter stages in series with feedback, say, on $\phi_2$ to refresh the data every clock cycle. Circuit arrangements are shown in Figures 9–4(a) and 9–5(a) and it will be seen that a bit may be written to the cell from the bus by energizing the *WR* line. From our system timing consideration of section 9.1, we will assume *WR* to occur in coincidence with $\phi_1$ of the clock. Thus, the bit is stored on $C_g$ of inverter 1 and will be reproduced complemented at the output of inverter 1 and true at the output of inverter 2. It will be seen that during every $\phi_2$ of the clock the stored bit is refreshed through the gated feedback path from the output of inverter 2 to the input of inverter 1. Thus the bit will be held as long as $\phi_2$ of the clock recurs at intervals less than the decay time of the stored bit. To read the state of the cell it is only necessary to energize the *RD* line, which is also assumed coincident with $\phi_1$ of the clock, and the bit will be read onto the bus.

Note that:

1. *WR* and *RD* must be mutually exclusive (but are both coincident with $\phi_1$).
2. If $\phi_2$ is used for refresh, then the cell must not be read during $\phi_2$ of the clock unless the feedback path is inhibited during *RD*. If an attempt is made to read

the cell onto the bus during refresh, then charge sharing effects between the bus and input ($C_g$) capacitances may cause the destruction of the stored bit.

3. Cells must be stackable, both side by side and top to bottom. This must be carefully considered together with the overall strategy to be observed when the layout is drawn.
4. Allow for other bus lines to run through the cell so that register and memory arrays are readily configured.

With these factors in mind, it is possible to draw up stick diagrams as in Figures 9–4(b) and 9–5(b), which show the nMOS and CMOS basic cells.

Mask level layouts follow from this; a possible layout for an nMOS cell

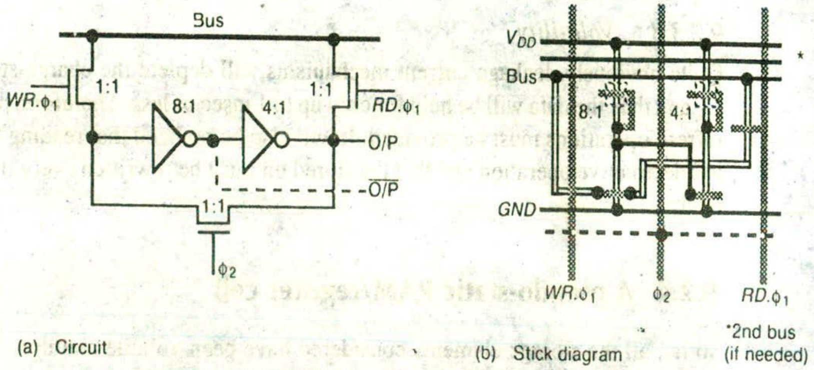


**Figure 9–4** nMOS pseudo-static memory cell

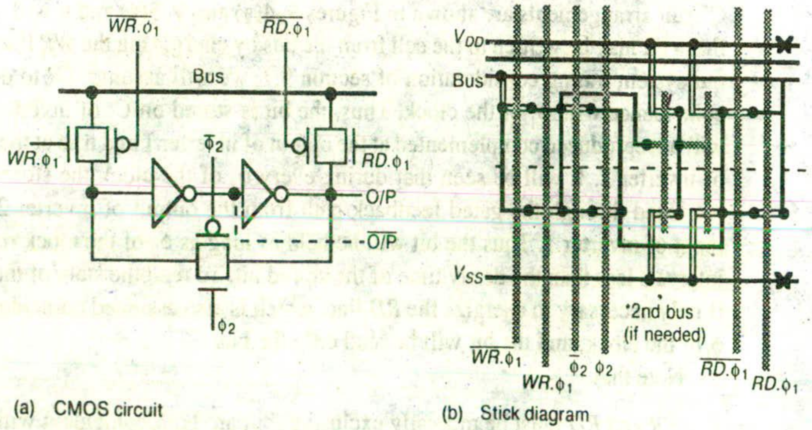


**Figure 9–5** CMOS pseudo-static memory cell

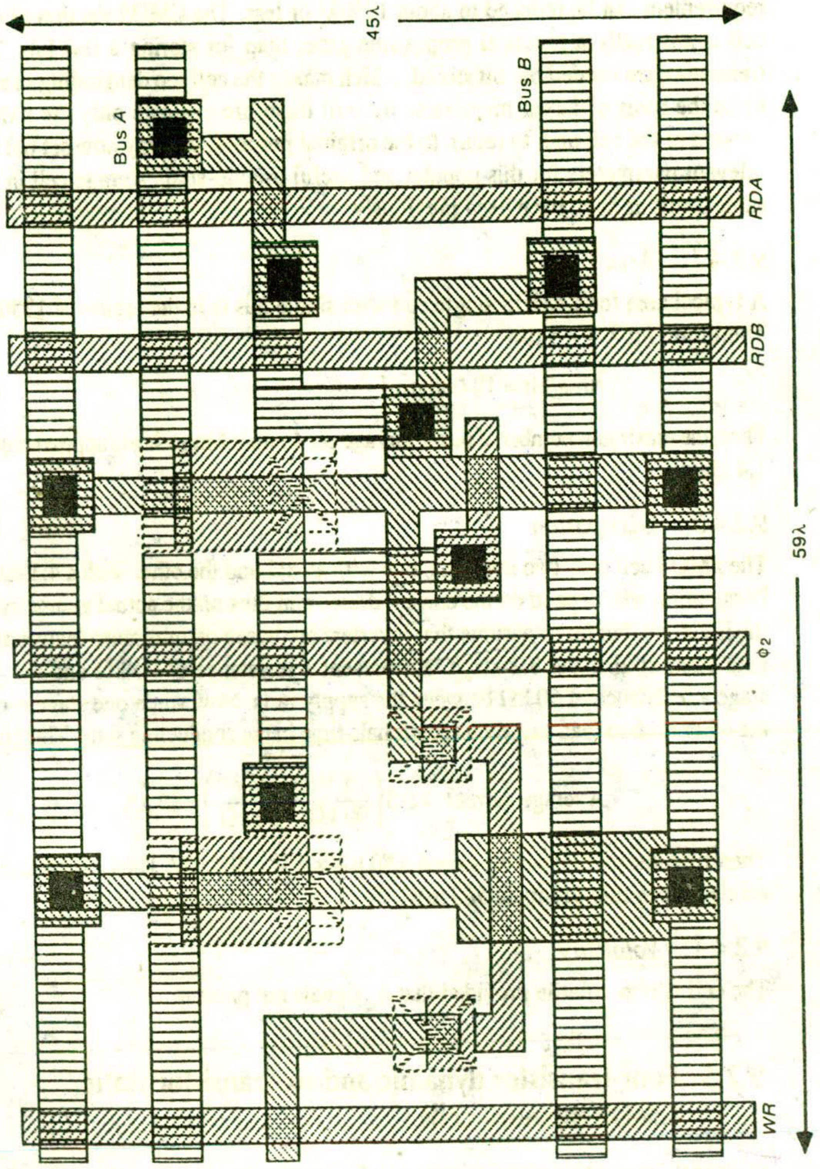


**Figure 9–6** nMOS pseudo-static memory cell with read to either of two buses

which can be written to from bus *A* and can be read onto bus *A* or bus *B* is given in Figure 9–6.

The mask layout shown in Figure 9–6 occupies an area of $59\lambda \times 45\lambda = 2655\lambda^2$, but if we are considering a single bus and a more compact layout then the area requirement can be reduced to about $1750\lambda^2$ or less. The CMOS version of this cell is *not* really a practical proposition other than for storing a few bits. Ten transistors are needed per bit stored, which makes the cell too demanding in area to be the basis of larger memories. We will therefore evaluate only the nMOS version of the cell and, to return to the original purpose, we may now set out the relevant parameters for this popular and useful pseudo-static storage cell in the same terms as have been used previously.

#### 9.2.4.1 *Area*

A typical area for a nMOS single cell with single bus is in the region of $1750\lambda^2$. Therefore, for $\lambda = 2.5\ \mu m$

$$\text{Area/bit} \approx 10\,000\ \mu m^2$$

Thus, the maximum number of bits of storage per 4 mm × 4 mm chip is approximately 1.4 kbits.

#### 9.2.4.2 *Dissipation*

The nMOS cell uses two inverters, one with an 8:1 and the other with a 4:1 ratio. Dissipation will depend on the current drawn and thus on the actual geometry of the inverters, but let us assume that inverters are based on minimum feature size gate areas so that the 8:1 stage will present a resistance of 90 kΩ and the 4:1 stage a resistance of 50 kΩ between the supply rails. Now when one stage is off, the other is on so that, say, each spends half-time in the conducting state. Therefore

$$\text{Average current } = 0.5\left(\frac{5\text{ V}}{90\text{ k}\Omega} + \frac{5\text{ V}}{50\text{ k}\Omega}\right) \approx 80\ \mu\text{A}$$

Therefore dissipation per bit stored = 80 μA × 5 V = 400 μW. Thus 1.4 kbits on a single chip would dissipate 560 mW.

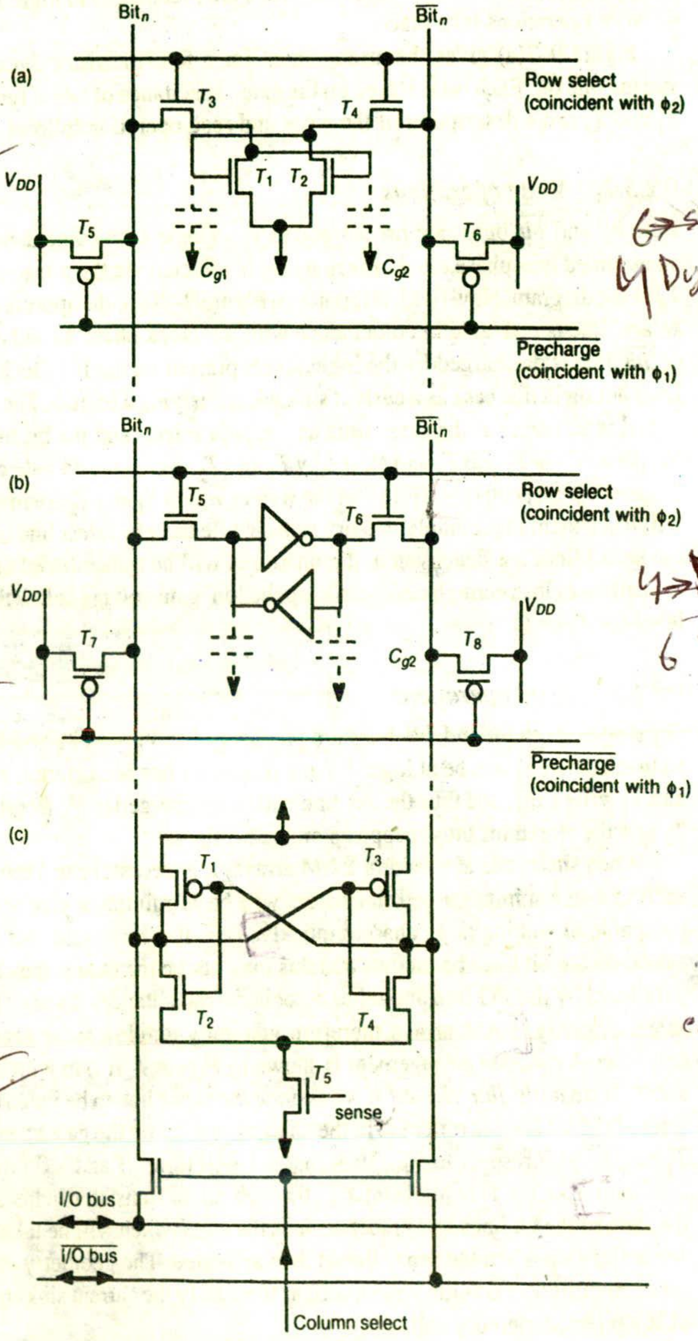
#### 9.2.4.3 *Volatility*

The cell is non-volatile provided that $\phi_2$ signals are present.

### 9.2.5 Four-transistor dynamic and six-transistor static CMOS memory cells

Most of the preceding memory cells involved n-type transistors and can therefore be implemented in either nMOS or CMOS designs. The cells about to be described utilize both n-type and p-type transistors and are therefore intended for CMOS systems only (although the dynamic element can be readily adapted to nMOS-only implementation).

Both the dynamic and static elements, set out in Figure 9–7, use a two bus per bit arrangement so that associated with every bit stored there will be a *bit* and


(a)
Bit$_n$
$\overline{\text{Bit}}_n$
Row select (coincident with $\phi_2$)
$T_3$
$T_4$
$T_1$
$T_2$
$V_{DD}$
$T_5$
$T_6$
$C_{g1}$
$C_{g2}$
$\overline{\text{Precharge}}$ (coincident with $\phi_1$)
(b)
Bit$_n$
$\overline{\text{Bit}}_n$
Row select (coincident with $\phi_2$)
$T_5$
$T_6$
$V_{DD}$
$T_7$
$T_8$
$C_{g2}$
$\overline{\text{Precharge}}$ (coincident with $\phi_1$)
(c)
$T_1$
$T_3$
$T_2$
$T_4$
$T_5$
sense
I/O bus
$\overline{\text{I/O bus}}$
Column select

F

6 ~~Static~~
4 Dynamic

4 ~~Dynamic~~
6 Static

F

F

commond part.

**Figure 9–7** Dynamic and static memory cells

a $\overline{bit}$ bus as shown. In both cases the buses are precharged to logic 1 before read or write operations take place.

Figure 9–7(a) gives the arrangement for a four-transistor dynamic cell for storing one bit. Each bit is stored on the gate capacitance of two n-type transistors $T_1$ and $T_2$ and a description of the write and read operation follows.

### 9.2.5.1 *Write operations*

Both *bit* and $\overline{bit}$ buses are precharged to $V_{DD}$ (logic 1) in coincidence with $\phi_1$ of an assumed two-phase clock. Precharging is effected via the p-transistors $T_5$ and $T_6$ in the diagram. Now (with reference to Figure 9–7(c)), the appropriate *'column select'* line is activated in coincidence with the clock phase $\phi_2$ and either the *bit* or $\overline{bit}$ line is discharged by the logic levels present on the I/O *bus* lines, the I/O lines acting in this case as a current sink when carrying a logic 0. The *'row select'* signal is activated at the same time as *'column select'* and the bit line states are 'written in' via $T_3$ and $T_4$ and stored by $T_1$ and $T_2$ as charges on gate capacitances $C_{g2}$ and $C_{g1}$ respectively. Note that the way in which $T_1$ and $T_2$ are interconnected will force them into complementary states while the *row select* line is high. Once the select lines are deactivated, the bit stored will be remembered until the gate capacitances lose enough charge to drop the 'on' gate voltage below the threshold level for $T_1$ or $T_2$.

### 9.2.5.2 *Read operations*

Once again both *bit* and $\overline{bit}$ lines are precharged to $V_{DD}$ via $T_5$ and $T_6$ during $\phi_1$ so that both lines will be at logic 1. Now if, say, a 1 has been stored, $T_2$ will be on and $T_1$ will be off, and thus the $\overline{bit}$ line will be discharged to $V_{ss}$ (logic 0) through $T_2$ and the stored bit thus reappears on the bit lines.

When such cells are used in RAM arrays, it is necessary to keep the area of each cell to a minimum and transistors will be of minimum size and therefore incapable of sinking large charges quickly. Thus it is important that the charges stored on the bit lines be modest and this may not be the case if they are directly paralleled by the I/O line and other associated capacitances through the column select circuitry. RAM arrays therefore generally employ some form of sense amplifier. A possible arrangement is shown in Figure 9–7(c) in which $T_1$, $T_2$, $T_3$, and $T_4$ form *a flip-flop circuit.* If we assume the sense line to be inactive, then the state of the bit lines is reflected in the charges present on the gate capacitances of $T_1$ and $T_3$ with respect to $V_{DD}$ such that a 1 will turn off and a 0 turn on either transistor. Current flowing from $V_{DD}$ through an on transistor helps to maintain the state of the bit lines and predetermines the state which will be taken up by the sense flip-flop when the sense line is then activated. The geometry of the single sense amplifier per column will be such as to amplify the current sinking capability of the selected memory cell.

Figure 9–7(b) indicates an adaption of the basic dynamic cell, just considered, to form a static memory cell. At the expense of two additional transistors per bit stored, the transistors $T_1$ and $T_2$ of Figure 9–7(a) can each be replaced by an inverter as shown in Figure 9–7(b). This arrangement will clearly render the cell static in its data-storing capabilities.

The general arrangement of a RAM utilizing the circuits considered here appears later in this chapter (Figure 9–18).

### 9.2.6 JK flip-flop circuit

No consideration of memory elements would be complete without the JK flip-flop. The JK flip-flop is a particularly widely used arrangement and is an example of a static memory element. It is also most useful in that other common arrangements such as the D flip-flop and the T flip-flop are readily formed from the JK arrangement. Edge-triggered circuits are conveniently designed with an ASM (algorithmic state machine) approach — see C. Clare, *Designing Logic Systems Using State Machines*, McGraw-Hill, 1983) — and the design equations for a JK flip-flop, as in Figure 9–8, follow from an ASM chart setting out the requirements as in Figure 9–9. It should be noted that the flip-flop is assumed to have an asynchronous clear (*Clr*) input as well as the clocked *J* and *K* inputs, and that *J* and *K* are read in during the Hi level of the clock ϕ, and the data thus read is transferred to the output on the falling edge of ϕ.

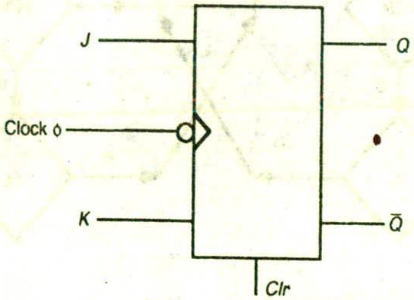


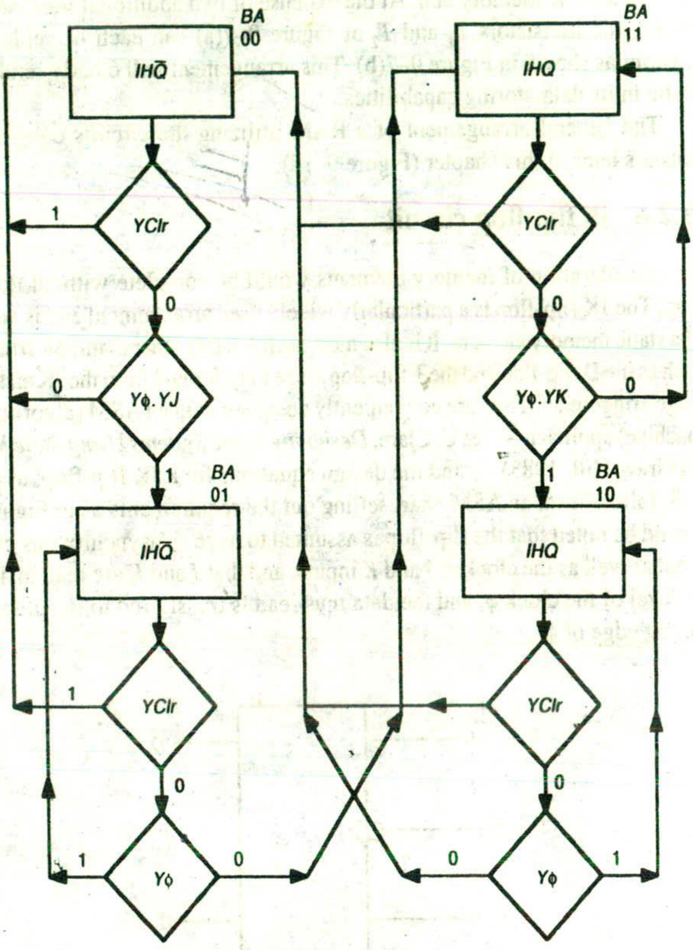
**Figure 9–8** JK flip-flop

Design equations are readily derived from the ASM chart of Figure 9–9 and, making the secondary variable assignments (*AB* in the figure), we may express the requirements as follows:

$$A = a.(\overline{Clr}).(\bar{b}+\bar{\phi}+\bar{K})+\bar{b}.(\overline{Clr}).J.\phi$$

$$B = (\overline{Clr}).(a.\bar{\phi}+b.\phi)$$

where output $Q = B$, and $a$ and $b$ are the fed back state of the secondary variables $A$ and $B$ respectively.



*Note:* *IHQ* = *Q* output 'Hi' (Immediate)
*YClr* = Yes clear
*Yϕ.YJ* = Yes clock and *J*, . . ., etc.

**Figure 9–9** ASM chart for JK flip-flop

### 9.2.6.1 *Logic gate implementations*

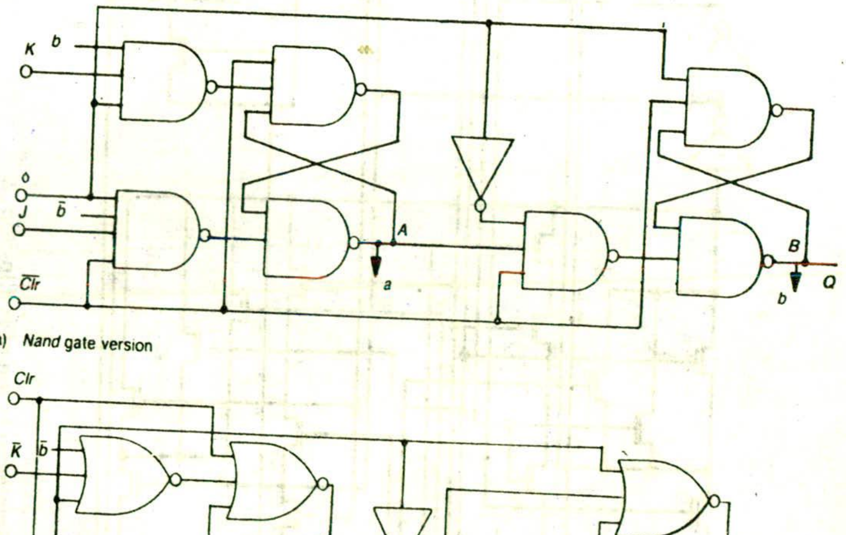
We are now faced with a choice of implementations based on *Nand* or *Nor* or switch logic. The expressions for *A* and *B* are readily realized in *Nand* or *Nor* logic, as shown in Figure 9–10, and it will be seen that a master/slave arrangement applies in each case.

However, an initial consideration of each arrangement will reveal that, for nMOS, the *Nand arrangement is impractical,* owing to the relatively large number of gates requiring three or more inputs which will therefore be inherently large in area and slow in performance. The obvious nMOS altemative is a *Nor* gate arrangement which is a practical proposition and can be readily implemented.

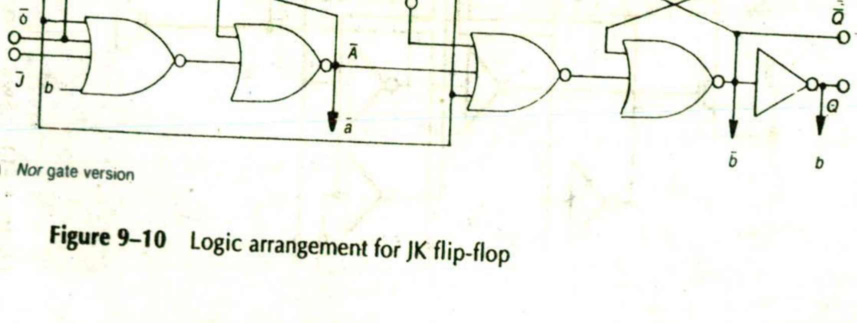
For CMOS, both *Nand* and *Nor* gates are suitable although the *Nor* gate is generally slower.

### *9.2.6.2 Switch logic and inverter implementation*

In setting out an arrangement of *n* pass transistors to realize the logical requirements, we must bear in mind earlier considerations on the nature of switch logic networks: that is, there should be no more than four pass transistors in series (section 4.9); pass transistors are not to be used to drive the gates of other pass transistors; the logic 0 as well as the logic 1 transmission conditions are to be deliberately satisfied. Thus, we need to implement the expressions for $\bar{A}$ and $\bar{B}$ *as well as* the expressions for *A* and *B* given earlier in this section. The resulting arrangement is given at Figure 9–11 and is a realization of the JK flip-flop based on n-pass transistor logic and inverters only.



(a) *Nand* gate version



(b) *Nor* gate version

**Figure 9–10** Logic arrangement for JK flip-flop

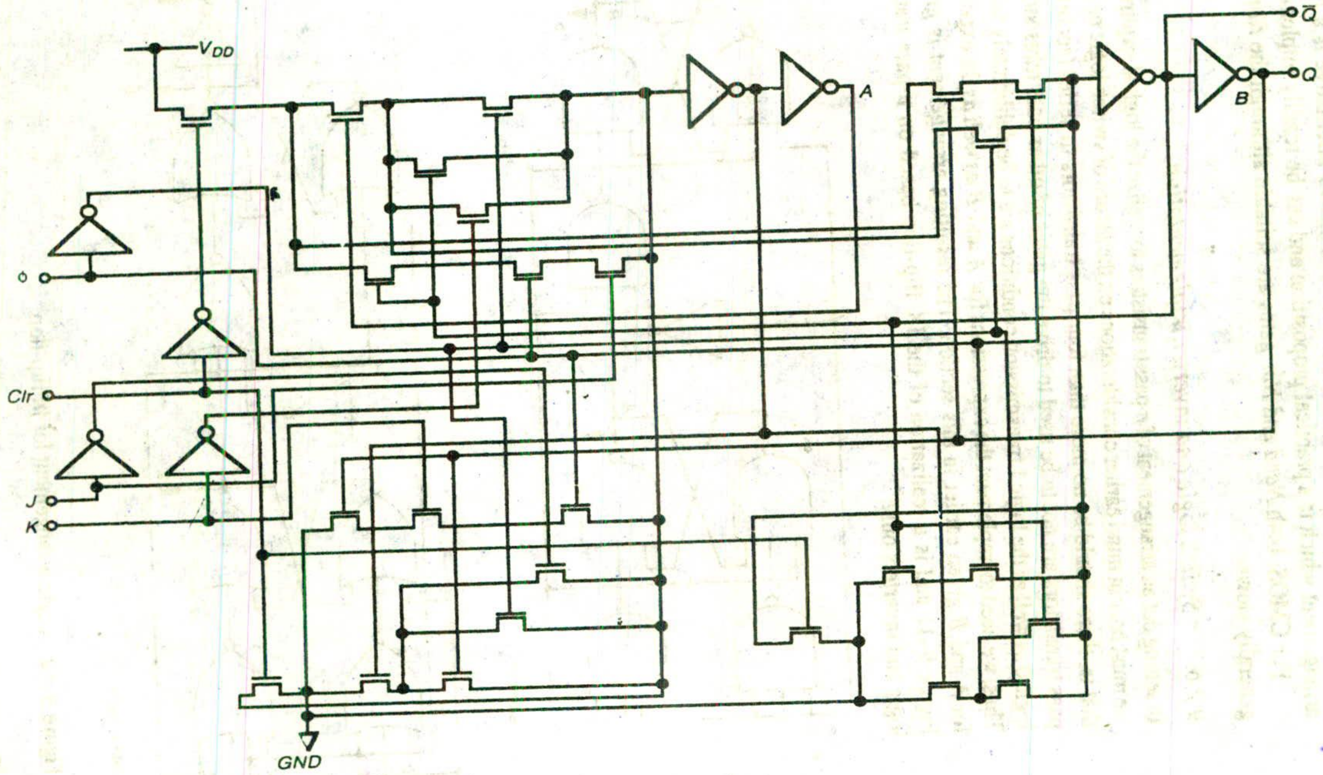


**Figure 9–11** Switch logic implementation of JK flip-flop

### 9.2.7 D flip-flop circuit

A D flip-flop is readily formed from a JK flip-flop by renaming the *J* input *D* and then replacing connections to *K* by $\overline{D}$ (see Figure 9–10). Similarly, a T (Toggle) flip-flop is formed from the JK by making $J = K = E$, where *E* is the toggle enabling input.

It should also be noted that the arrangements given may be simplified by the omission of the *Clr* input, or that a *Preset* input can be substituted for or added to the *Clr* input if required. Furthermore, the way in which clock activation takes place may be modified by a reshaping of requirements in the ASM chart of Figure 9–9 and a consequent reformulation of the JK flip-flop design equations given at the beginning of section 9.2.6 of this text.

However, a much simpler version of the D flip-flop is obtained from a pseudo-static approach, as in Figure 9–12 for CMOS. Clearly, an nMOS version is also readily configured.

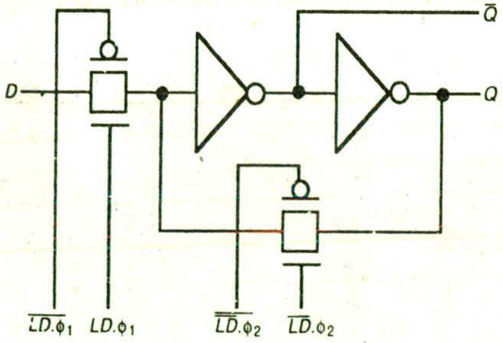


**Figure 9–12** A CMOS pseudo-static D flip-flop

## 9.3 Forming arrays of memory cells

The memory cells discussed in section 9.2 and others will most often be used in arrays of some form or other. Typical arrays are registers and random access memories (RAM) and these arrays will be used as examples in this section. We must not forget, however, that another common application is to use memory elements individually as 'flags' or 'status bits' in system design. In any event, there must be some means of *selecting* a particular cell or group of cells and some means of effecting *read* or *write* operations.

-19

### 9.3.1 Building up the floor plan for a 4 × 4-bit register array

This will be the third subsystem to be considered for the 4-bit data path, the floor plan of which appeared as Figure 7–5; the first two subsystem minimum bounding box outline dimensions have been given as Figure 7–10 (4 × 4 barrel shifter) and Figure 8–11(b) (4-bit adder). The fourth and final subsystem — the I/O port facilities — will be left for the reader to consider as an exercise in completing a system design (prior to adding inlet and outlet pads through which a chip is bonded to the outside world).

Starting with a bounding box representation of the chosen memory cell — in this case we have presented typical dimensions and connections of a pseudo-static cell with two bus lines, as in Figure 9–13 — we can arrive at a bounding box for a single 4-bit register and hence the floor plan for a 4 × 4-bit register array.

The bounding box representation of the cell is 'stacked' to form a 4-bit register as in Figure 9–14, the overall vertical dimension being about 180λ. Note how the cells stack 'vertically' to form a 4-bit word and note that although a 'vertical' distribution of power has been assumed at the input of the register, power distributes

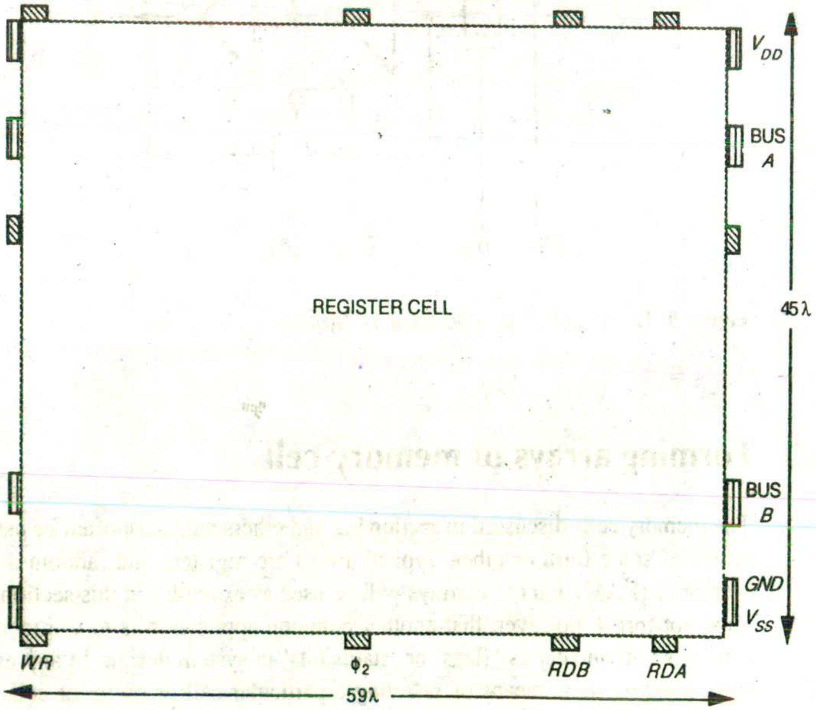


**Figure 9–13** Bounding box for register cell

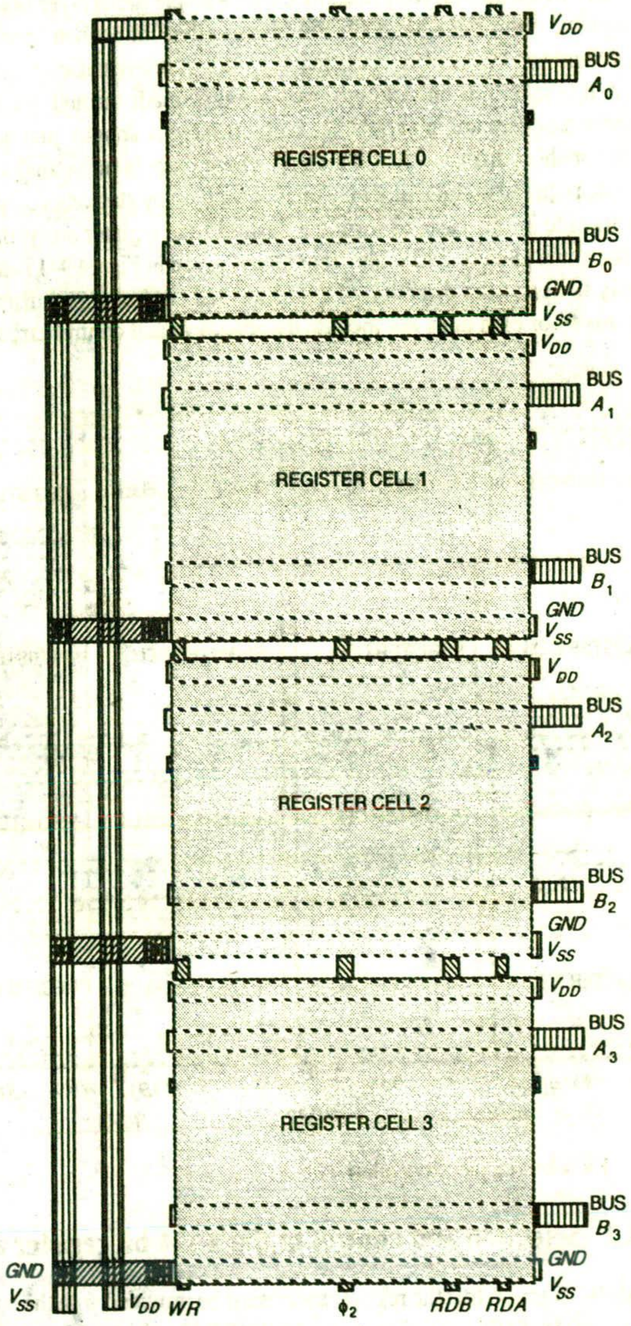


**Figure 9–14** 4-bit register floor plan

horizontally thereafter. Note that since only a single metal layer has been assumed throughout, short but wide diffusion 'duck unders' have then been used to allow the ground (or $V_{SS}$) rail to cross the $V_{DD}$ rail. However, it must be stressed that $V_{DD}$ and *GND* ($V_{SS}$) connections must *always* be made through metal rails, except where crossovers are unavoidable; such crossovers should then normally be by means of short diffusion 'duck unders' where there is no second metal layer.

The required architecture may then be built up by stacking complete registers, side by side in this case, to form the desired four-register array, the dimensions being around 180λ × 240λ. The floor plan is given in Figure 9–15 and the diagram clearly indicates the direction of data flow and control signal distribution. Note that this floor plan does *not* include the selection and control circuitry.

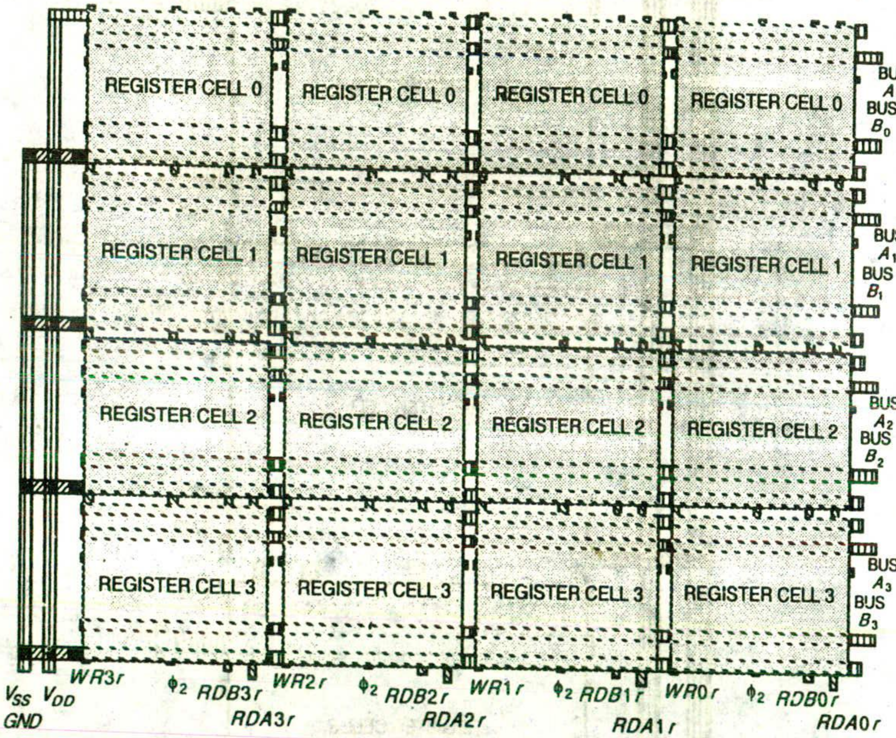
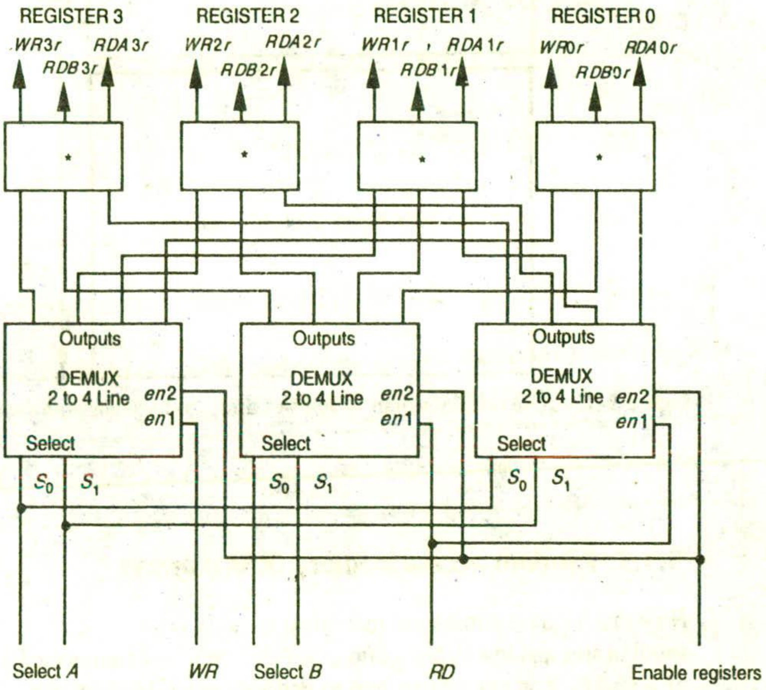


**Figure 9–15** 4 × 4-bit register floor plan

## 9.3.2 Selection and control of the 4 × 4-bit register array

Figure 9–15 shows that the register array must be provided with the control signals *WR0r–WR3r, RDB0r–RDB3r,* and *RDA0r–RDA3r,* derived from register select signals so that each register may be selected for read or write. We must also note *that we* need the capability to select two registers simultaneously for connection to the adder and that, in some cases, we may wish to read the contents of a single register to both A and B buses. One approach is to make use of decoder (or demultiplexer) circuits to route the control signals, as suggested in Figure 9–16. (For the reader unfamiliar with demultiplexer circuits, the select lines allow routing of a single input to any one of the output lines, that is, like a multiposition switch, and are the converse of the multiplexer which selects any one of a number of input lines to be routed to a single output.)



*Note*: The effective input to each decoder is en1.en2.
* Indicates buffers/drivers as required.

**Figure 9–16** Decoder-based selection and control

The whole register array and selection and control circuitry may then be represented in floor plan form, as in Figure 9–17; note that the details of the selection and control circuitry have not been given here.

Note also that this register subsystem is the third of the four main functional blocks of the data path (Figure 7–1), for which we have already designed the shifter (Figures 7–8 to 7–10) and the adder subsystems (Figure 8–11(b)). The completion of the floor planning is discussed in the next chapter.

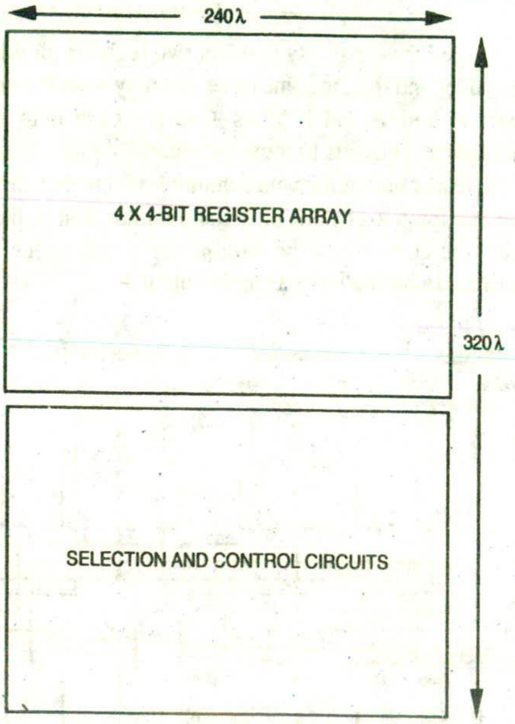


**Figure 9–17** Overall floor-plan — register array and select/control circuits

## 9.3.3 Random access memory (RAM) arrays

Now that we have considered individual memory cells, some of which are quite small in area and low in dissipation, and also the use of memory cells in an array of registers, it is not a large step to consider much larger arrays of which the RAM (random access memory) is the most commonplace.

It is relatively easy to form arrays of memory cells. For example, the CMOS memory cell of Figure 9–7(b) and associated sense circuitry (as in Figure 9–7(c)) will form a RAM array as shown in Figure 9–18. A suitable mask layout for the memory cell used here is suggested in Figure 9–19 to give an idea of overall area for a particular size of array.

Finally, the architecture of a typical RAM array storing 'words' is illustrated by the floor plan and main interconnections for a 16-location × 4-bit word array in Figure 9–20. It will be seen that, in this case, the incoming address lines are decoded into row and column (with *RD* or *WR*) select lines, which are then used to select individual words in the memory. It will be noted that $V_{DD}$ and *GND*

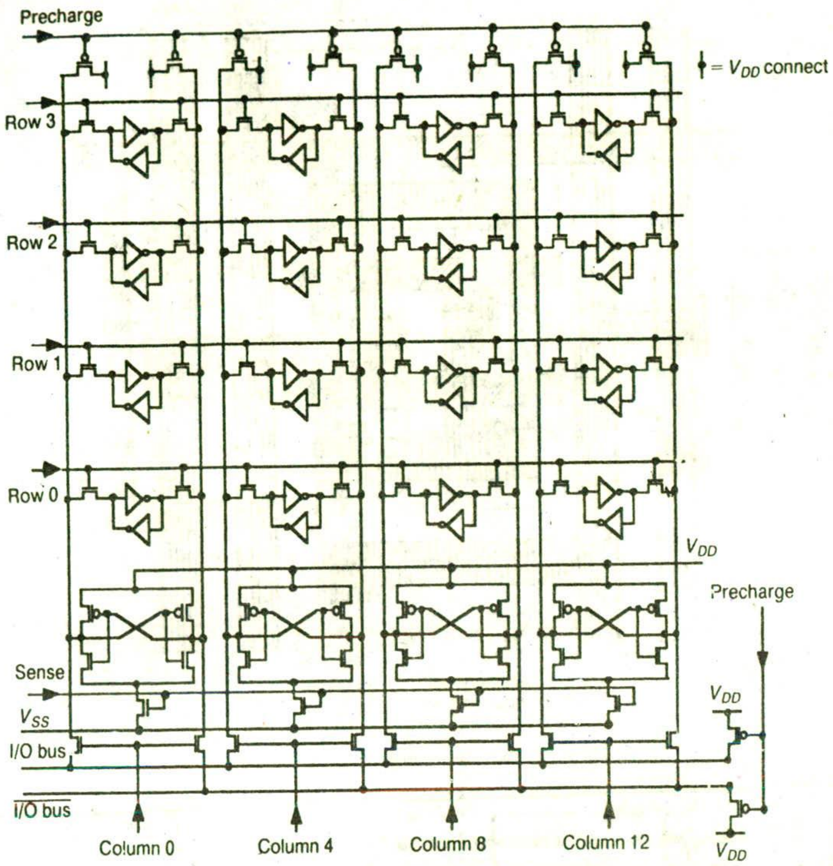
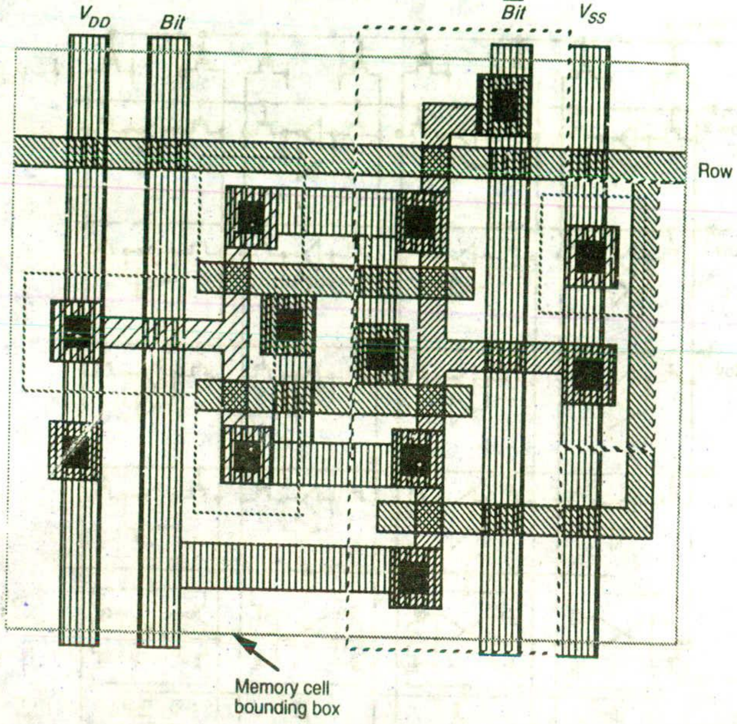
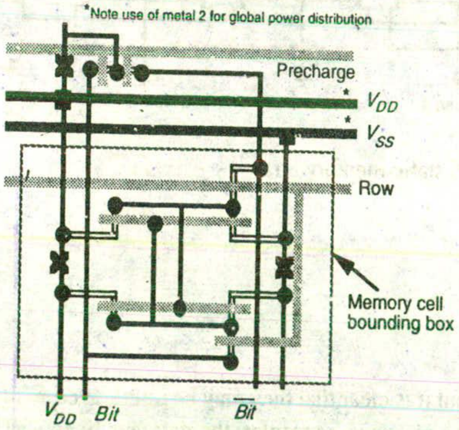


**Figure 9–18** 16-bit CMOS static memory array

($V_{ss}$) rails are not shown, but it is clear that they may be interleaved with the data bus lines. Note that in large arrays in particular, the data bus lines will become relatively long and must therefore be run on the metal layer to avoid excessive capacitance or series resistance. As discussed earlier, the bus capacitance *and* the control line capacitances *must* be allowed for in the design.

To complete this section, Figure 9–21 is a plot of the *metal layer* only for a 16-location × 4-bit memory. The regularity of the memory array, the way in which the buses are run, and the various subsections of the floor plan are clearly evident.





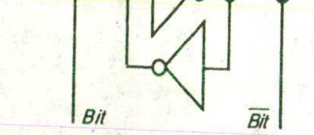


**Figure 9–19** CMOS static memory cell-mask and stick layout

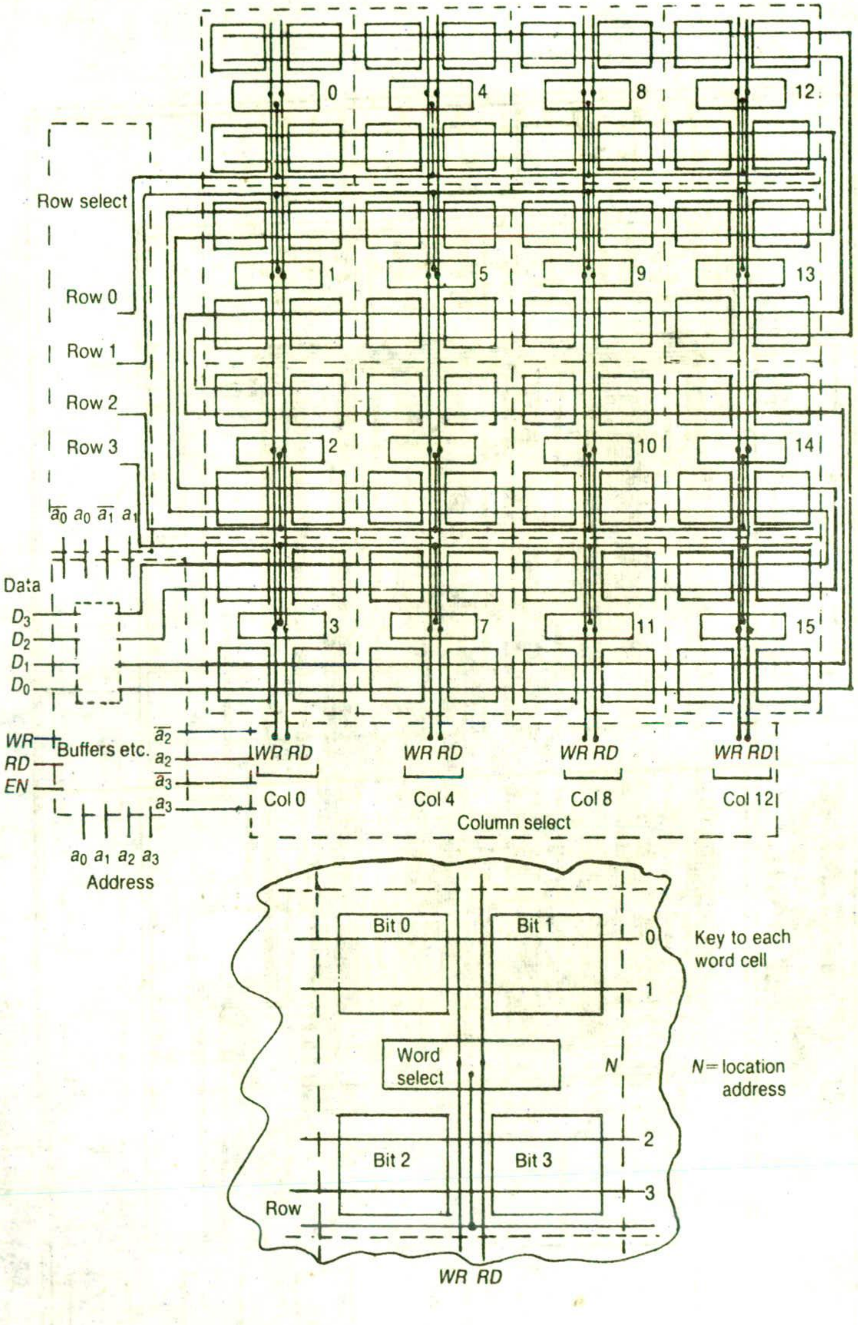


**Figure 9–20** Floor plan of 16 × 4-bit RAM

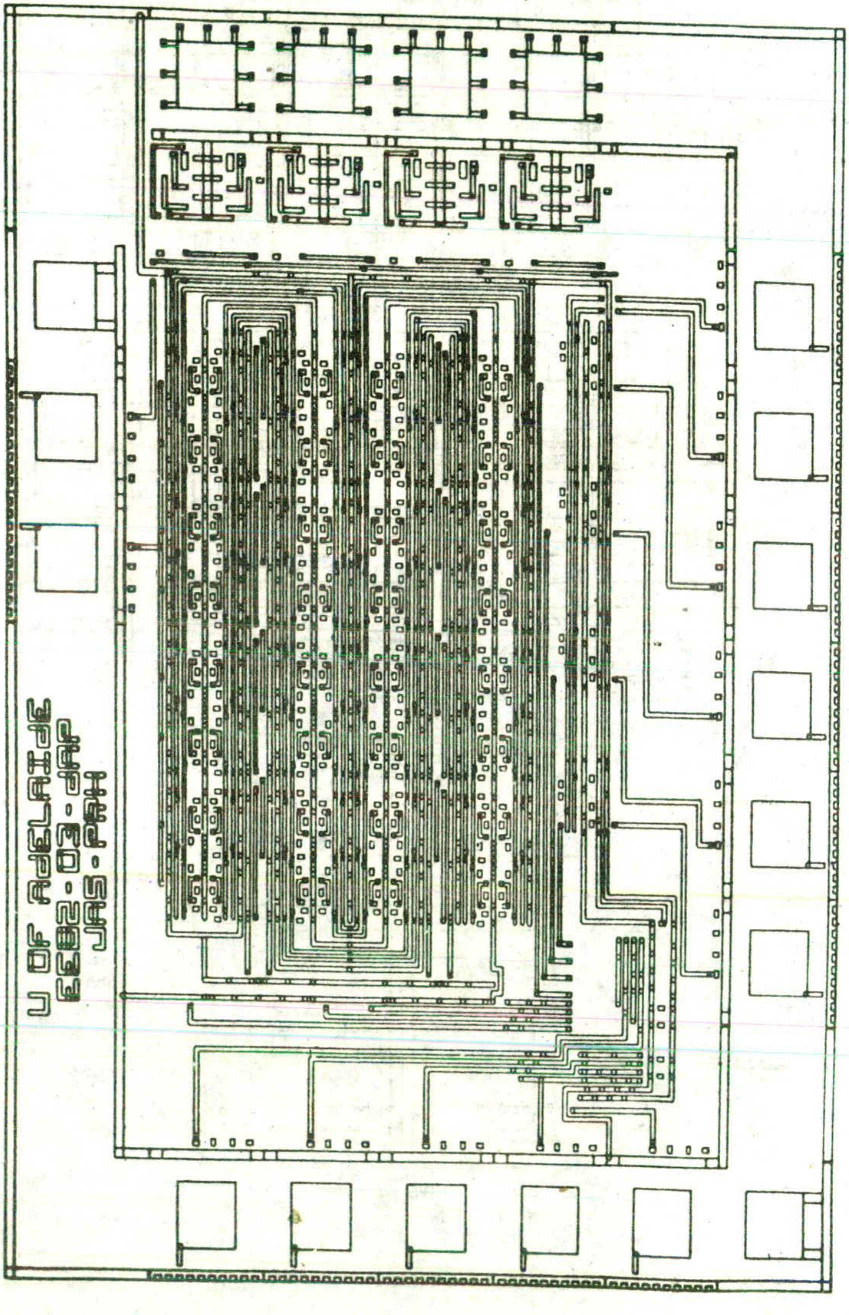


**Figure 9–21** Metal layer only for 16 × 4-bit RAM

## 9.4 Observations

This chapter has completed our introduction to most of the techniques and many of the commonly used circuit arrangements for VLSI design in Silicon.

We have now completed the design of three of the four subsystems comprising the 4-bit data path which has been the 'vehicle' we have used to explore design processes.

We have also begun to see that communications are a very important aspect of design and this will be further emphasized in the next chapter.

## 9.5 Tutorial exercises

1. Design a two-line to four-line decoder (demultiplexer) circuit to the mask layout level and determine its bounding box. Then work out the arrangement of, and area occupied by, the 4 × 4-bit register select and control circuit of Figure 9–16.
2. Taking a 16-location × 2-bit RAM arrangement as an example, suggest (with sketches only) how such an arrangement could be configured for two-port operation (i.e. data can be read or written to any location from either of two 2-bit data buses).
3. For the 4 × 4-bit register designed in this chapter, and using the select and control circuit suggested in question 1 above, work out all communication paths and interconnections between the three subsystems of the 4-bit data path we have so far designed. Use a bounding box representation for each subsystem and clearly indicate the layers on which interconnections are made. What is a suitable overall area for the processor as so far designed?
4. *nMOS*: Using a 4-bit word arrangement of Figure 9–22, and consulting the RAM arrangement of Figure 9–20, draw a mask level layout for the word select circuit associated with each 4-bit word and thus determine the overall area needed for each word stored. Next, design stick diagram level arrangements for the remaining blocks on the floor plan — that is, the row and column selection circuits and the input buffers and drivers, etc. Then, *estimate* the area needed for the 16 × 4-bit RAM as a whole (without I/O pads).
5. *CMOS*: Starting with the 16-bit RAM array of Figure 9–18, design suitable decoding and control circuitry to allow row and column selection and read and write operation of the array. (You may find it useful to refer to section 9.2.5.)

   Design *one* memory cell as far as the mask layout and determine a suitable area per bit stored.

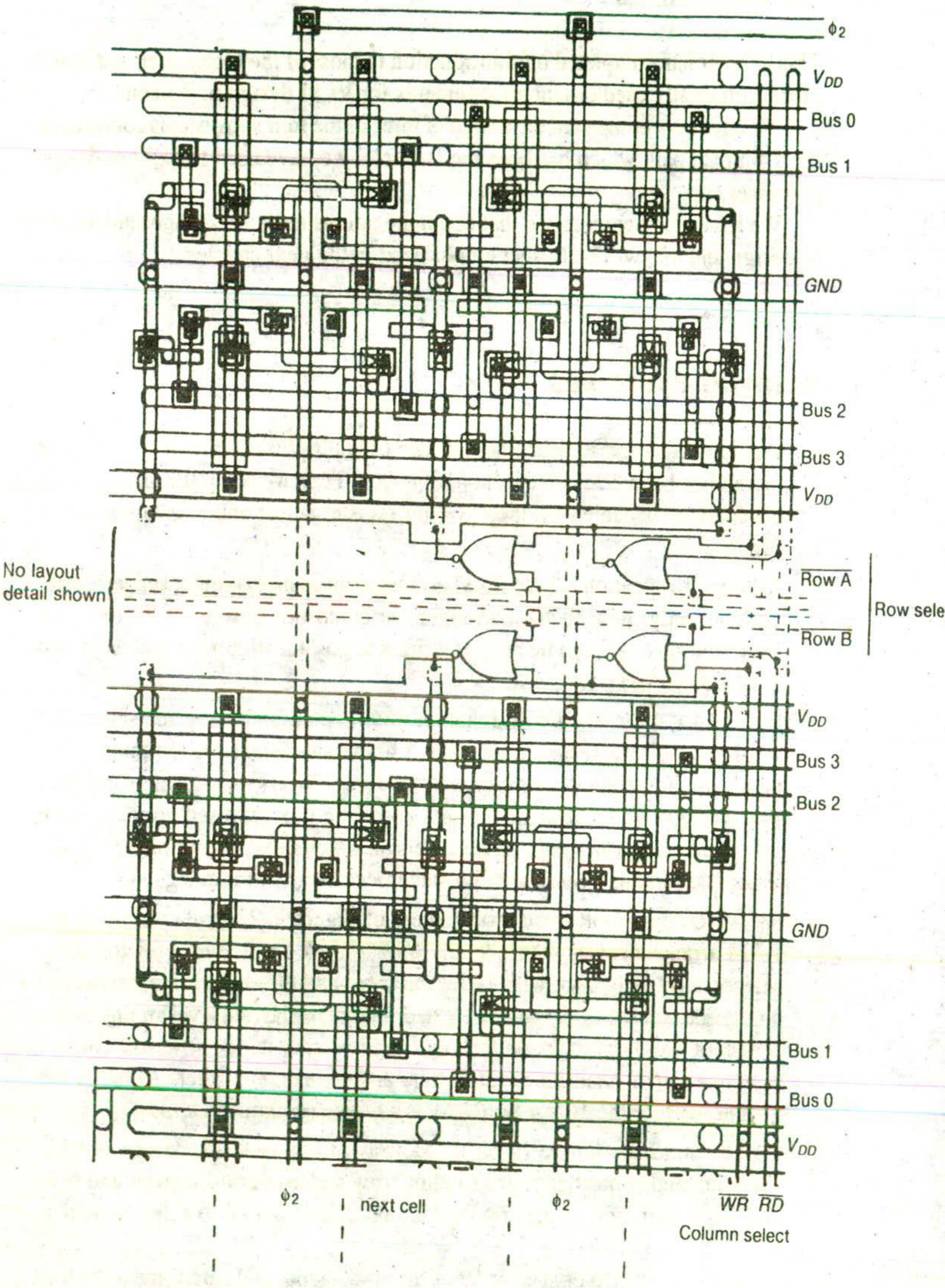


**Figure 9–22** Two 4-bit words of nMOS RAM array