VLSI Design
Course Notes

Dinesh Sharma
EE Department, IIT Bombay

October 13, 2022

Contents

List of Figures

Chapter 1

Basics of Semiconductor Devices

In this booklet, we review the fundamentals of Semiconductor Physics and basics of device operation. We shall concentrate largely on elemental semiconductors such as silicon or germanium, and most numerical values used for examples are specific to silicon.

1.1 Semiconductor fundamentals

A semiconductor has two types of mobile charge carriers: negatively charged *electrons* and positively charged *holes*. We shall denote the concentrations of these charge carriers by n and p respectively. The discussions in this booklet apply to elemental semiconductors (like silicon) which belong to group IV of the periodic table. We can intentionally add impurities from groups III and V to the semiconductor. These impurities are called dopants. Impurities from group III are called *acceptors* while those from group V are called *donors*. Each donor atom has an extra electron, which is very loosely bound to it. At room temperature, there is sufficient thermal energy present, so that the loosely bound electron breaks free from the donor, leaving the donor positively charged. This contributes an additional electron to the free charge carriers in the semiconductor, and a positive ionic charge at a *fixed* location in the semiconductor. Similarly, an acceptor atom captures an electron, thus producing a mobile hole and becoming negatively charged itself. A semiconductor without any dopants is called *intrinsic*. An unperturbed semiconductor must be charge neutral as a whole. If we denote the concentration of ionised donors by N_d^+ and the concentration of ionised acceptors by N_a^-, we can write for the net charge density at any point in the semiconductor as:

$$\rho = q(N_d^+ - N_a^- + p - n) \tag{1.1}$$

where q is the absolute value of the electronic charge. In an unperturbed semiconductor, ρ will be zero everywhere. Electrons and holes are generated thermally - the availability of energy equal to the band gap of the semiconductor results in the generation of an *electron - hole pair*. Simultaneously, electrons and holes can recombine to annihilate each other, giving out energy which is equal to the band gap of the semiconductor. Thus we have the reversible reaction:

$$\mathrm{e}^- + \mathrm{h}^+ \rightleftharpoons E_g$$

Where E_g is the band gap energy of the semiconductor.
Applying the law of mass action to the above reaction, we can write for the equilibrium concentration of holes and electrons:

$$n \cdot p = \text{constant}$$

The above relation applies to doped as well as intrinsic semiconductors. But for an intrinsic semiconductor,

$$n = p \equiv n_i$$

Therefore, the constant in the equation connecting n and p must be n_i^2. Thus, for a semiconductor *in equilibrium*,

$$n \cdot p = n_i^2 \tag{1.2}$$

Since n and p are not independent, but are constrained by the above relation, we can define a single independent variable, the Fermi potential by

$$\Phi_{\mathrm{F}} \equiv \frac{K_B T}{q} ln \frac{p}{n_i} = \frac{K_B T}{q} ln \frac{n_i}{n} \tag{1.3}$$

Where $\mathrm{K_B}$ is the Boltzmann constant, T is the absolute temperature and q is the absolute value of the electronic charge. At room temperature, K_BT/q is approximately 26 mV and n_i is of the order of $10^{10}/\mathrm{cm}^3$ for silicon. Now electron and hole concentrations are given by:

$$\begin{aligned} n &= n_i e^{-\frac{q\Phi_{\mathrm{F}}}{K_BT}} \\ p &= n_i e^{\frac{q\Phi_{\mathrm{F}}}{K_BT}} \end{aligned} \tag{1.4}$$

To simplify these relations, we define a dimensionless Fermi potential by:

$$u_F \equiv \frac{q\Phi_{\mathrm{F}}}{K_BT} = \ln(p/n_i) = \ln(n_i/n)$$

then:

$$\begin{aligned} n &= n_i e^{-u_F} \\ p &= n_i e^{u_F} \end{aligned} \tag{1.5}$$

Generally, a semiconductor will be doped with only one kind of impurity. A semiconductor doped with donors will have many more electrons than holes. This type of semiconductor is called N type, and electrons are the *majority* carriers in this type of semiconductor. Similarly, holes are the majority carriers in a semiconductor doped with acceptors and it is termed P type. If both types of dopants are present, the one present in higher concentration determines the 'type' of the semiconductor. The *net* doping is defined as the difference in the concentrations of the more abundant and the less abundant dopants.

In most practical cases, the ratio of majority to minority carriers is very high. The concentration of majority carriers is very nearly equal to the net dopant concentration. To take a typical example, consider P type silicon with boron concentration of 10^{16} atoms/cm^3. This gives:

$$\begin{aligned} p &= N_a = 10^{16}/\text{cm}^3 \\ n &= n_i^2/p \approx 10^{20}/10^{16}/\text{cm}^3 = 10^4/\text{cm}^3 \\ p/n &\approx 10^{12}\ ! \end{aligned}$$

1.1.1 Band Diagrams

The above concepts are often visualised with the help of band diagrams. The arrangement of atoms in a semiconductor results in certain electron energies which are not permitted. Thus, the energy range is divided into bands of permitted energy values alternating with forbidden gaps.

The highest such band which is nearly filled with electrons is called the valance band. Unoccupied levels in this band correspond to holes. For stability, electrons seek the lowest energy level available. If a vacancy is available at a lower energy - an electron at a higher energy will drop to this level. The vacancy thus bubbles up to a higher level. Therefore, holes seek the highest *electron* energy available.

The band just above the valance band is called the conduction band. In a semiconductor, this is partially filled. Conduction in a semiconductor is caused by electrons in the conduction band (which are normally to be found at the lowest energy in the conduction band) and holes in the valance band - (found at the highest electron energy in the valance band).

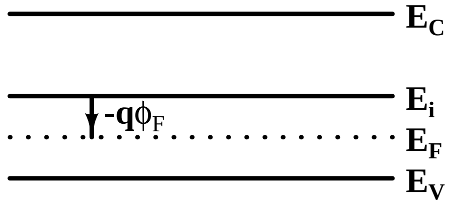


Figure 1.1: Semiconductor Bands

Band diagrams are plots of electron energies as a function of position in the semiconductor. Typically, the top of the valance band (corresponding to minimum hole energy) and the bottom of the conduction band are plotted. We can show the Fermi potential and the corresponding Fermi energy(= -qΦ_F) in the band diagram of silicon as a level in the band gap. We use the halfway point between the conduction and the valence band as the reference for energy and potential. When n = p = n_i, the Fermi potential is 0 (from eq. 1.3) and correspondingly, the Fermi energy lies at the intrinsic Fermi level halfway in the band gap. (Actually, this level can be slightly away from the middle of the band gap depending on the density of allowed states in the conduction and valance bands - but for now, we'll ignore this). When holes are the majority carriers, $\Phi_F$is positive and the Fermi energy (= -q Φ_F) lies below the mid gap level, as shown in figure 1.1. When electrons are the majority carriers, $\Phi_F$is negative, and the Fermi energy lies above the mid gap level.

1.1.2 A semiconductor in the presence of an electric field

In the presence of an electric field, the electrostatic potential is different at different positions. The energy of an electron has an extra component $= -q\phi$ where ϕ is the electrostatic potential.

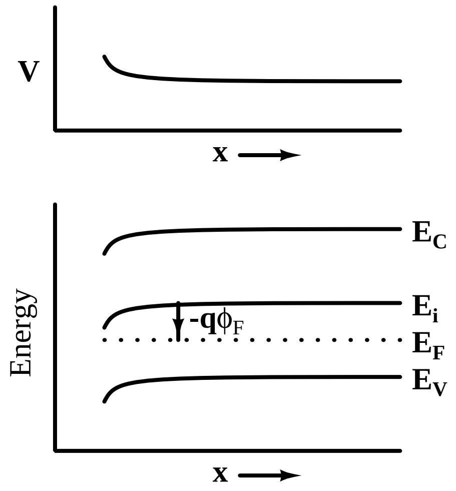


Figure 1.2: Potential distribution and Band Diagram in the presence of a field

Consequently in the band diagram the conduction, valance and intrinsic levels are bent. In equilibrium, the Fermi level is still straight. (We shall see later that in the absence of a current, the slope of the Fermi level must vanish). Relations for n and p must now take the electrostatic potential as well as the Fermi potential into account and the electron and hole concentrations are not uniform over the semiconductor. If we represent the concentrations of electrons and holes without any applied field by n_0 and p_0 respectively, then in the presence of a field (but in equilibrium),

$$n = n_0 e^{\frac{q\phi}{K_B T}}$$

$$p = p_0 e^{-\frac{q\phi}{K_B T}} \tag{1.6}$$

where ϕ is the electrostatic potential.

If we define a dimensionless electrostatic potential by:

$$u \equiv \frac{q\phi}{K_B T} \tag{1.7}$$

we can write the above relations as:

$$\begin{aligned} n = n_0 e^{u} &= n_i e^{(u-u_F)} \\ p = p_0 e^{-u} &= n_i e^{-(u-u_F)} \end{aligned} \tag{1.8}$$

Since there is equilibrium, even though electron and hole concentration is not uniform, the product of n and p is still constant and equal to n_i^2 everywhere.

1.1.3 Non-equilibrium case

The above relations assume a semiconductor in equilibrium. It is possible to create excess carriers in the semiconductor over those dictated by equilibrium considerations. For example, if we shine light on a semiconductor, electron-hole pairs will be created. Since the value of n *as well as* that of p goes up, the np product will exceed n_i^2, till the equilibrium is restored after the light is turned off (by enhanced recombination). If the number of excess carriers is small compared to the majority carriers, we may assume that the carrier concentrations are still described by relations like those given above. However, the concentrations of electrons and holes are not constrained by relation(1.2) any more. Therefore, we cannot use the same value of u_F for describing electron as well as hole concentrations. We now have *separate* values of Φ_F for electrons and holes. These are called *quasi* Fermi levels (or imrefs) for electrons and holes, $\Phi_{F_n}$and Φ_{F_p}, defined by the relations

$$\begin{aligned} n &= n_i e^{(u-u_{F_n})} \\ p &= n_i e^{-(u-u_{F_p})} \end{aligned} \tag{1.9}$$

Where u_{F_n} and u_{F_p} are the dimensionless versions of quasi Fermi levels $\Phi_{F_n}$and Φ_{F_p}defined as in equation(1.7)). The np product is now given by

$$np = n_i^2 e^{(u_{F_p}-u_{F_n})} \tag{1.10}$$

and is no longer constant. Because the number of additional carriers is assumed to be small compared to the majority carriers, the concentration of majority carriers and hence its quasi Fermi level is very close to the equilibrium value. The relative change in the concentration of minority carriers could, however, be large and consequently the minority carrier quasi Fermi level could be substantially different from the equilibrium Fermi level.

1.2 The p-n diode

We shall analyse the abrupt pn junction, in reverse and forward bias. We assume that the

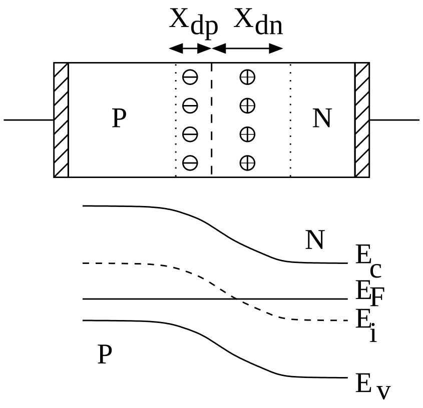


Figure 1.3: The abrupt p-n junction

doping density is constant and its value = N_a on the P side and N_d on the N side, changing abruptly at the metallurgical junction as shown. Because there is a strong concentration gradient for electrons and holes at the junction, there will be a diffusion current of holes towards the N side and of electrons towards the P side. As these carriers leave behind ionised dopants, small regions on either side of the junction acquire a charge. The P side, from where positively charged holes have left, (leaving behind negatively charge acceptor ions), acquires a negative potential. Similarly, the N side becomes positively charged. The regions from where mobile charges have left, are called depletion regions. The potential difference resulting from this charge redistribution (called the built-in voltage) opposes further diffusion of carriers. A dynamic equilibrium is reached when the drift current due to this potential difference and the diffusion current due to the concentration gradient become equal and opposite. In equilibrium, The electron as well as hole currents must be zero individually (principle of detailed balance). Writing the electron and hole current densities as sums of their respective drift and diffusion current densities:

$$\begin{aligned} J_n &= nq\mu_n(-\frac{\partial\phi}{\partial x}) + qD_n\frac{\partial n}{\partial x} \\ J_p &= pq\mu_p(-\frac{\partial\phi}{\partial x}) - qD_p\frac{\partial p}{\partial x} \end{aligned} \tag{1.11}$$

From equation(1.9)

$$\begin{aligned} \frac{\partial n}{\partial x} &= n_i e^{(u-u_{F_n})}\frac{\partial}{\partial x}(u-u_{F_n}) \\ \frac{\partial p}{\partial x} &= n_i e^{(u_{F_p}-u)}\frac{\partial}{\partial x}(u_{F_p}-u) \end{aligned}$$

or

$$\begin{aligned}\frac{\partial n}{\partial x} &= n\frac{q}{K_BT}\frac{\partial}{\partial x}(\phi - \Phi_{F_n})\\ \frac{\partial p}{\partial x} &= p\frac{q}{K_BT}\frac{\partial}{\partial x}(\Phi_{F_p} - \phi)\end{aligned}$$

Using Einstein relations ($\frac{q}{K_BT}D = \mu$), and Substituting in the relations for J_n and J_p,

$$\begin{aligned}J_n &= -nq\mu_n(\frac{\partial\phi}{\partial x}) + nq\mu_n\frac{\partial}{\partial x}(\phi - \Phi_{F_n})\\ J_p &= -pq\mu_p(\frac{\partial\phi}{\partial x}) - pq\mu_p\frac{\partial}{\partial x}(\Phi_{F_p} - \phi)\end{aligned}$$

Which leads to

$$\begin{aligned}J_n &= -nq\mu_n\frac{\partial\Phi_{F_n}}{\partial x};\\ J_p &= -pq\mu_p\frac{\partial\Phi_{F_p}}{\partial x};\end{aligned} \tag{1.12}$$

When there is no flow of current, $\Phi_{F_n} = \Phi_{F_p} = \Phi_F$. according to the relations derived above, the derivative of $\Phi_F$must vanish everywhere for zero current. Thus, the Fermi level is constant and the same at the two sides of the junction. The Fermi potentials before being put in contact were:

$$\begin{aligned}\Phi_F = &\ \tfrac{K_BT}{q}\ln(N_a/ni) && \text{P side} : x < 0\\ \Phi_F = &\ -\tfrac{K_BT}{q}\ln(N_d/ni) && \text{N side} : x > 0\end{aligned}$$

The Fermi potential difference was, therefore, $\frac{K_BT}{q}\ln\left(\frac{N_dN_a}{n_i^2}\right)$. Since after being put in contact, the Fermi levels have equalised on the two sides, the built in voltage must be equal and opposite to this potential, taking the P side to a negative potential and the N side to a positive potential. We can write for the magnitude of the built in voltage:

$$V_{bi} = \frac{K_BT}{q}\ln\left(\frac{N_aN_d}{n_i^2}\right) \tag{1.13}$$

1.2.1 pn Diode in Reverse Bias

The diode is reverse biased when we apply a voltage such that the n side is more positive as compared to the p side. In this case, the applied voltage is in the same direction as the built-in field, which opposes the movement of majority carriers and widens the depletion regions on either side of the junction. We analyse the reverse biased diode by making the depletion approximation. We assume that in reverse bias, the depletion regions have zero carrier density, and the field is completely confined to depletion regions. Solving Poisson's equation in P region ($x < 0$) and the N region ($x > 0$)

$$\begin{aligned} \frac{\partial^2\phi}{\partial x^2} &= \tfrac{qN_a}{\epsilon_{si}} && \text{(for x < 0)} \\ \frac{\partial^2\phi}{\partial x^2} &= -\tfrac{qN_d}{\epsilon_{si}} && \text{(for x > 0)} \end{aligned}$$

Integrating with respect to x

$$\begin{aligned} \frac{\partial\phi}{\partial x} &= \tfrac{qN_a}{\epsilon_{si}}x + c1 && \text{(for x < 0)} \\ \frac{\partial\phi}{\partial x} &= -\tfrac{qN_d}{\epsilon_{si}}x + c2 && \text{(for x > 0)} \end{aligned}$$

where c_1 and c_2 are constants of integration, which can be evaluated from the condition that the field vanishes at the edge of the depletion regions at $-X_{dp}$ and at X_{dn}. This leads to

$$\begin{aligned} \frac{\partial\phi}{\partial x} &= \tfrac{qN_a}{\epsilon_{si}}(x + X_{dp}) && \text{(for x < 0)} \\ \frac{\partial\phi}{\partial x} &= -\tfrac{qN_d}{\epsilon_{si}}(x - X_{dn}) && \text{(for x > 0)} \end{aligned} \tag{1.14}$$

Since the value of the field must match at x = 0;

$$N_a X_{dp} = N_d X_{dn} \tag{1.15}$$

Integrating equation (1.14) once again with respect to x, we get

$$\begin{aligned} \phi &= \tfrac{qN_a}{\epsilon_{si}}\left(\tfrac{x^2}{2} + X_{dp}x\right) + c_3 && \text{(for x < 0)} \\ \phi &= -\tfrac{qN_d}{\epsilon_{si}}\left(\tfrac{x^2}{2} - X_{dn}x\right) + c_4 && \text{(for x > 0)} \end{aligned}$$

Where the constants of integration c_3 and c_4 can again be evaluated from the boundary conditions at $-X_{dp}$ and X_{dn}. If we require that the potential is 0 at $-X_{dp}$ and V at X_{dn},

$$\begin{aligned} c_3 &= \frac{qN_a}{2\epsilon_{si}}X_{dp}^2 \\ c_4 &= V - \frac{qN_d}{2\epsilon_{si}}X_{dn}^2 \end{aligned}$$

Substituting these values, we get:

$$\begin{aligned} \phi &= \tfrac{qN_a}{\epsilon_{si}}\left(\tfrac{x^2+X_{dp}^2}{2} + X_{dp}x\right) && \text{(for x < 0)} \\ \phi &= V - \tfrac{qN_d}{\epsilon_{si}}\left(\tfrac{x^2+X_{dn}^2}{2} - X_{dn}x\right) && \text{(for x > 0)} \end{aligned} \tag{1.16}$$

Since the potential at x = 0 should be continuous,

$$\frac{qN_a}{2\epsilon_{si}}X_{dp}^2 = V - \frac{qN_d}{2\epsilon_{si}}X_{dn}^2$$

$$\text{so, } V = \frac{q}{2\epsilon_{si}}(N_a X_{dp}^2 + N_d X_{dn}^2) \tag{1.17}$$

making use of equation (1.15), we can write

$$\begin{aligned} V &= \frac{qN_a X_{dp}^2}{2\epsilon_{si}N_d}(N_d + N_a) \\ &= \frac{qN_d X_{dn}^2}{2\epsilon_{si}N_a}(N_d + N_a) \end{aligned}$$

which leads to

$$\begin{aligned} X_{dp} &= \sqrt{\frac{2\epsilon_{si}V}{q(N_d + N_a)}\frac{N_d}{N_a}} \\ X_{dn} &= \sqrt{\frac{2\epsilon_{si}V}{q(N_d + N_a)}\frac{N_a}{N_d}} \end{aligned} \tag{1.18}$$

From which the total depletion width can be calculated as:

$$X_d \equiv X_{dp} + X_{dn} = \sqrt{\frac{2\epsilon_{si}V}{q(N_d + N_a)}}\left(\sqrt{\frac{N_d}{N_a}} + \sqrt{\frac{N_a}{N_d}}\right)$$

which gives

$$X_d = \sqrt{\frac{2\epsilon_{si}V}{q}\left(\frac{1}{N_a} + \frac{1}{N_d}\right)} \tag{1.19}$$

The voltage V in the above expressions is the total voltage across the junction. Since there is a reverse bias of V_{bi} for a zero applied voltage, that will add (in magnitude) to the applied reverse voltage. Using equation(1.13) we can write:

$$V = V_{bi} + V_{appl} = V_{appl} + \frac{K_B T}{q}\ln\left(\frac{N_a N_d}{n_i^2}\right) \tag{1.20}$$

1.2.2 The pn diode in forward bias

If we apply an external voltage, such that the P side is made positive with respect to the N side, the applied voltage will reduce the built in voltage across the junction. The magnitude of the built-in voltage is such that it balances the drift and diffusion currents, resulting in zero net current. But if the voltage across the junction is reduced, a net current will flow through the diode. This is the forward mode of operation. Because of this flow of current, electrons are injected into the P side and holes into the N side. Consequently, the concentration of carriers is no longer at the equilibrium value. We denote the *equilibrium* value of electron and hole concentrations on P and N side by $n_{p_0}, n_{n_0}, p_{p_0}, p_{n_0}$ respectively. Since the majority carrier concentration in equilibrium is equal to the doping density, we have:

$$n_{n_0} \approx N_d, \qquad p_{p_0} \approx N_a \qquad \text{and} \qquad n_{p_0} = n_i^2/N_a, \qquad p_{n_0} = n_i^2/N_d$$

According to equation(1.10)

$$np = n_i^2 e^{(u_{F_p} - u_{F_n})}$$

As we make the potential of P type more positive compared to N type, the np product in forward bias is greater than n_i^2. From relations(1.12), we see that the change in quasi Fermi levels is small wherever the carrier concentration is high. Thus, we can assume that the quasi Fermi levels of the *majority* carriers at either side of the junction remain at their equilibrium values. Hence the voltage across the junction is given by

$$V = \phi_{F_p} - \phi_{F_n}$$

and therefore the *non-equilibrium* np product is given by

$$np = n_i^2 e^{\left(\frac{qV}{K_B T}\right)}$$

therefore,

$$\begin{aligned} n_p &= \frac{n_i^2}{p_p} e^{\left(\frac{qV}{K_B T}\right)} = n_{p_0} e^{\left(\frac{qV}{K_B T}\right)} \\ p_n &= \frac{n_i^2}{n_n} e^{\left(\frac{qV}{K_B T}\right)} = p_{n_0} e^{\left(\frac{qV}{K_B T}\right)} \qquad (1.21) \\ & \qquad (1.22) \end{aligned}$$

The continuity equation for any particle flow can be written as

$$\nabla.(\text{particle current density}) = -\frac{\partial}{\partial t}(\text{particle concentration})$$

Applying it to electron and hole currents in 1 dimension on the n side,

$$\frac{\partial}{\partial x}\left(\frac{J_n}{-q}\right) = U$$
$$\frac{\partial}{\partial x}\left(\frac{J_p}{q}\right) = U$$

where U is the net recombination rate. Using relation(1.11), we have

$$\begin{aligned}\frac{\partial}{\partial x}\left(n_n\mu_n\frac{\partial\phi}{\partial x}-D_n\frac{\partial n_n}{\partial x}\right) &= U\\ \frac{\partial}{\partial x}\left(p_n\mu_p\frac{\partial\phi}{\partial x}+D_p\frac{\partial p_n}{\partial x}\right) &= U\end{aligned}$$

or

$$\begin{aligned}\mu_n\frac{\partial n_n}{\partial x}\frac{\partial\phi}{\partial x}+\mu_n n_n\frac{\partial^2\phi}{\partial x^2}-D_n\frac{\partial^2 n_n}{\partial x^2} &= U\\ \mu_p\frac{\partial p_n}{\partial x}\frac{\partial\phi}{\partial x}+\mu_p p_n\frac{\partial^2\phi}{\partial x^2}+D_p\frac{\partial^2 p_n}{\partial x^2} &= U\end{aligned}$$

Assuming the regions outside the small depletion regions to be charge neutral,

$$(n_n - n_{n_0}) \approx (p_n - p_{n_0})$$

We define ambipolar diffusion and lifetime by the relations

$$D_a \equiv \frac{n_n+p_n}{n_n/D_p+p_n/Dp} \tag{1.23}$$

$$\tau_a \equiv \frac{p_n-p_{n_0}}{U}=\frac{n_n-n_{n_0}}{U} \tag{1.24}$$

multiplying the electron continuity equation with $\mu_p p_n$ and the hole continuity equation with $\mu_n n_n$ and combining, we get

$$-\frac{p_n-p_{n_0}}{\tau_a}+D_a\frac{\partial^2 p_n}{\partial x^2}+\frac{n_n-p_n}{n_n/\mu_p+p_n/\mu_n}\frac{\partial p_n}{\partial x}\frac{\partial\phi}{\partial x}=0 \tag{1.25}$$

If we make the low injection assumption ($p_n << n_n \approx n_{n_0}$), this reduces to

$$-\frac{p_n-p_{n_0}}{\tau_p}+D_p\frac{\partial^2 p_n}{\partial x^2}+\mu_p\frac{\partial p_n}{\partial x}\frac{\partial\phi}{\partial x}=0 \tag{1.26}$$

In the neutral region, $\frac{\partial\phi}{\partial x}$ is zero, so the above simplifies further to

$$\frac{\partial^2 p_n}{\partial x^2}-\frac{p_n-p_{n_0}}{D_p\tau_p}=0 \tag{1.27}$$

This can be solved with the boundary condition given by relation(1.21) and noting that $p_n = p_{n_0}$ at $x=\infty$ to give:

$$p_n-p_{n_0}=p_{n_0}\left(e^{\frac{qV}{K_BT}}-1\right)e^{\frac{x-x_n}{L_p}} \tag{1.28}$$

where

$$L_p \equiv \sqrt{D_p \tau_p} \tag{1.29}$$

Evaluating the hole current at X_{dn}, we get

$$J_p = -qD_p \frac{\partial p_n}{\partial x} = \frac{qD_p p_{n_0}}{L_p}\left(e^{\frac{qV}{K_B T}} - 1\right) \tag{1.30}$$

Similarly, we can evaluate the electron current on the p side as

$$J_n = qD_n \frac{\partial n_p}{\partial x} = \frac{qD_n n_{p_0}}{L_n}\left(e^{\frac{qV}{K_B T}} - 1\right) \tag{1.31}$$

which gives the total current density as

$$J = J_p + J_n \;=\; J_s\left(e^{\frac{qV}{K_B T}} - 1\right) \tag{1.32}$$

$$\text{Where } J_s \;\equiv\; \frac{qD_p p_{n_0}}{L_p} + \frac{qD_n n_{p_0}}{L_n} \tag{1.33}$$

1.3 The MOS Capacitor

It is important to understand the MOS capacitor in order to understand the behaviour of the the MOS transistor. Before we describe the MOS structure, it is useful to review the basic electrostatics as applied to parallel plate capacitors. We shall then go on to analyse the MOS structure.

1.3.1 The Parallel Plate Capacitor

The parallel plate capacitor consists of two parallel metallic plates of area A, separated by an insulator of thickness t_i and dielectric constant ϵ. If we place a charge Q on the upper plate, it attracts charges of opposite sign in the bottom plate, while repelling charges of the same sign. If the bottom plate is connected to ground, the repelled charge flows to ground. Now

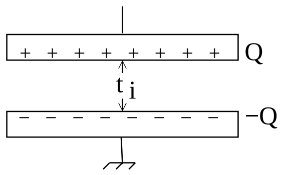


Figure 1.4: The parallel Plate capacitor

the two capacitor plates hold equal and opposite charge. This charge resides just next to the insulator on either side of it. This is true, *whatever the quantity or sign of charge* placed on the upper plate. The inducing and induced charge are always separated by the thickness of the insulator, t_i. Therefore this structure has a *constant* capacitance given by:

$$C_{\text{total}} = \frac{A\epsilon}{t_i}$$

Since there are no charges inside the dielectric, the electric field in the insulator is constant and the electrostatic potential changes linearly from one plate to the other.

1.3.2 The MOS capacitor

In a MOS capacitor, we replace the lower plate by a semiconductor. Unlike a metal, a semiconductor can have charges distributed in its bulk. For the sake of an example, let us

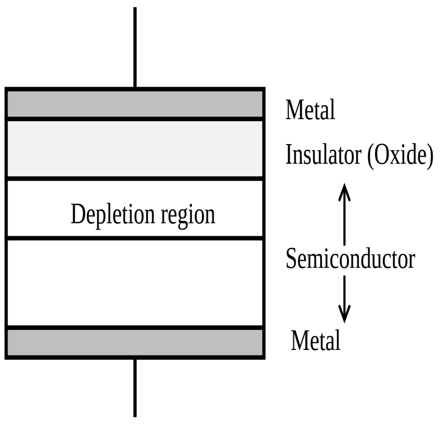


Figure 1.5: The MOS structure

consider a P type semiconductor (Si) doped to 10^{16}atoms /cm^3. As we know, holes outnumber electrons in this semiconductor by an extremely large factor. If we place a negative charge on the upper plate, holes will be attracted by this charge, and will accumulate near the silicon-insulator interface. This situation is analogous to the parallel plate capacitor and thus, the capacitance will be the same as that for a parallel plate capacitor. If, however, we place a positive charge on the upper plate, negative charges will be attracted by it and positive charges will be repelled. In a P type semiconductor, there are very few electrons. The negative charge is provided by the ionised acceptors after the holes have been pushed away from them. But the acceptors are fixed in their locations and cannot be driven to the edge of the insulator. Therefore, the distance between the induced and inducing charges increases - so the capacitance is lower as compared to the parallel plate capacitor. As more and more positive charge is placed on the upper plate, holes from a thicker slice of the semiconductor are driven away, and the incremental induced charge is farther from the inducing charge. Thus

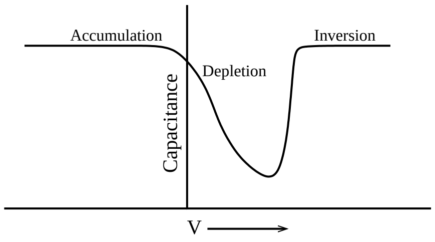


Figure 1.6: Low frequency capacitance for a MOS capacitor

the capacitance continues to decrease. This does not, however, continue indefinitely. We know from the law of mass action that as hole density reduces, the electron density increases. At some point, the hole density is reduced and electron density increased to such an extent that electrons now become the "majority" carriers near the interface. This is called *inversion.* Beyond this point, more positive charge on the upper plate is answered by more electrons in the semiconductor. But the electrons are mobile, and will be attracted to the silicon insulator interface. Therefore, the capacitance quickly increases to the parallel plate value.

1.3.3 Quantitative Analysis

Consider a one dimensional representation of the MOS structures as shown in the figure below. The origin is assumed to be at the silicon-oxide interface and the positive x direction is into

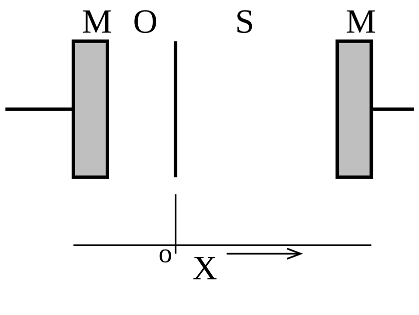
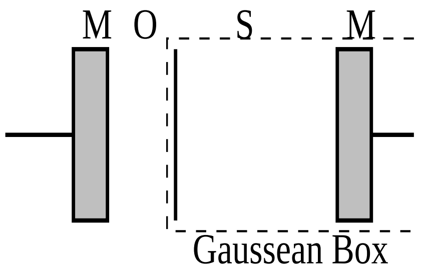


Figure 1.7: co-ordinate system used for analysis

the bulk of silicon. Using a one dimensional analysis, we want to relate the semiconductor charge to the applied gate voltage. In a practical case, there is a potential difference between two dissimilar materials in contact. Also, the silicon - oxide interface will have some fixed charge sitting there. However, we consider the ideal case first - where there is no built in contact potential between the semiconductor and the metal, and there is no interface charge.

Ideal Case

Let the back surface of Si be at zero potential and the voltage applied to the gate terminal be V_g. Let the electrostatic potential at any point x be denoted by $\phi(x)$ and let the potential at the silicon-oxide interface be ϕ_s.



We construct a Gaussian box passing through the interface and extending to $+\infty$. According to Gauss law, the integral of the outward pointing D vector around the box should be equal to the charge contained inside. The only boundary where D is non zero is the one passing through the interface. Therefore,

$$Area \times \epsilon_{ox} \frac{\phi_s - V_g}{tox} = \text{ Total Charge in silicon}$$

If we define Q_{si} to be the semiconductor charge *per unit area*, and C_{ox} to be the parallel plate capacitance *per unit area*, we get

$$V_g = \phi_s - \frac{Q_{si}}{C_{ox}}$$

Thus, the surface potential and the applied gate voltage can be related to each other. If the surface potential is known, we can evaluate the semiconductor charge by integrating the Poisson's equation in the semiconductor, once.

We can write the Poisson's equation in the semiconductor as

$$\nabla \cdot \mathbf{D} = \rho$$

or

$$-\epsilon_{si} \frac{\partial^2 \phi}{\partial x^2} = q(N_d^+ - N_a^- + p - n)$$

Since the electrostatic potential is dependent only on x, we can change partial derivatives to total derivatives.

$$-\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = \frac{\mathrm{d}}{\mathrm{d}x}\left(-\frac{\mathrm{d}\phi}{\mathrm{d}x}\right) = \frac{\mathrm{d}}{\mathrm{d}x}\left(\mathcal{E}\right)$$

where $\mathcal{E}$ is the electrostatic field. Changing the variable from x to ϕ.

$$-\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = \frac{\mathrm{d}\mathcal{E}}{\mathrm{d}x} = \left(\frac{\mathrm{d}\phi}{\mathrm{d}x}\right)\frac{\mathrm{d}}{\mathrm{d}\phi}(\mathcal{E}) = -\mathcal{E}\frac{\mathrm{d}}{\mathrm{d}\phi}(\mathcal{E}) = -\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}\phi}\left(\mathcal{E}^2\right)$$

If we define

$$u \equiv \beta\phi \qquad \text{where } \beta \equiv \frac{q}{K_B T}$$

We get

$$-\frac{\mathrm{d}^2\phi}{\mathrm{d}x^2} = -\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}\phi}\left(\mathcal{E}^2\right) = -\frac{\beta}{2}\frac{\mathrm{d}}{\mathrm{d}u}\left(\mathcal{E}^2\right) \tag{1.34}$$

The right hand side of the Poisson's equation represents the charge density. In the absence of an applied voltage, this must be zero everywhere. Therefore,

$$q(N_d^+ - N_a^- + p_0 - n_0) = 0$$

where p_0 and n_0 represent the hole and electron density in the absence of an applied field. therefore,

$$N_d^+ - N_a^- = -(p_0 - n_0)$$

Substituting equation(1.34) and the above in the Poisson's equation,

$$-\frac{\beta\epsilon_{si}}{2}\frac{\mathrm{d}}{\mathrm{d}u}\left(\mathcal{E}^2\right) = q\left[p - p_0 - (n - n_0)\right]$$

so

$$\frac{\mathrm{d}}{\mathrm{d}u}\left(\mathcal{E}^2\right) = -\frac{2qp_0}{\beta\epsilon_{si}}\left[\frac{p}{p_0} - 1 - \frac{n_0}{p_0}\left(\frac{n}{n_0} - 1\right)\right]$$

From equation(1.8)

$$n = n_0 e^u \qquad \text{and} \qquad p = p_0 e^{-u}$$

So,

$$\frac{\mathrm{d}}{\mathrm{d}u}\left(\mathcal{E}^2\right) = -\frac{2qp_0}{\beta\epsilon_{si}}\left[e^{-u} - 1 - \frac{n_0}{p_0}(e^u - 1)\right]$$

This can be integrated from x = ∞ (where $\mathcal{E}$ = 0 and u = 0) to x to give

$$\mathcal{E}^2 = \frac{2qp_0}{\beta\epsilon_{si}}\int_0^u \left[-e^{-u} + 1 + \frac{n_0}{p_0}(e^u - 1)\right]\mathrm{d}u$$

Integrating and putting limits at 0 and u, we get

$$\mathcal{E}^2 = \frac{2qp_0}{\beta\epsilon_{si}}\left[e^{-u} - 1 + u + \frac{n_0}{p_0}(e^u - 1 - u)\right]$$

Therefore

$$\mathcal{E} = \pm\sqrt{\frac{2qp_0}{\beta\epsilon_{si}}}\left[e^{-u} - 1 + u + \frac{n_0}{p_0}(e^u - 1 - u)\right]^{\frac{1}{2}}$$

And thus, the displacement vector D can be evaluated as:

$$D = \epsilon_{si}\mathcal{E} = \pm\sqrt{\frac{2qp_0\epsilon_{si}}{\beta}}\left[e^{-u} - 1 + u + \frac{n_0}{p_0}(e^u - 1 - u)\right]^{\frac{1}{2}} \tag{1.35}$$

This equation permits us to calculate D, given the value of u. In particular, the value of D at the surface (which is required for integration over the Gaussean box), can be evaluated from u_s.

In fact if u is very small, the exponentials in u can be expanded to second order. The first two terms cancel with 1 and u, leaving

$$D = \epsilon_{si}\mathcal{E} = \pm\sqrt{\frac{2qp_0\epsilon_{si}}{\beta}}\left[u^2/2 + \frac{n_0}{p_0}(u^2/2)\right]^{\frac{1}{2}}$$

$$\frac{\partial u}{\partial x} \simeq \mp\sqrt{\frac{q\beta p_0}{\epsilon_{si}}}\left(1 + \frac{n_0}{p_0}\right)u$$

This leads to exponential solutions for u with a characteristic length $L_D = \sqrt{\frac{\epsilon_{si}}{q\beta p_0}}$. This implies that small local perturbations in potential tend to decrease exponentially, with this characteristic length. This length is known as the extrinsic Debye Length.

For the p doped semiconductor under consideration,
$\frac{n_0}{p_0} << 1$, so $(1+\frac{n_0}{p_0}) \simeq 1$. In this case, the characteristic length (known as the extrinsic Debye Length) is $L_D = \sqrt{\frac{\epsilon_{si}}{q\beta p_0}}$.
(In the intrinsic case, $n_0 = p_0$, so $1 + \frac{n_0}{p_0} = 2$).
Thus, in the intrinsic case, we get an additional factor of 2 in the denominator under the square root.)

By putting $u = u_s$ in eq. 1.35, we get the D vector at the surface. We construct a Gaussean box passing through the interface and enclosing the semiconductor (as described in section 1.3.3) The charge contained in the box is then the integral of the outward pointing D vector over the surface of the box. D is non zero only at the interface. The outward pointing D is along the negative x axis. Therefore by application of Gauss theorem,

$$\text{Sem. Charge} = \text{Area} \times (-D)$$

Hence the charge in the semiconductor *per unit area* is:

$$Q_{si} \quad = \quad \mp\frac{\sqrt{2}\epsilon_{si}}{\beta L_D}\left[e^{-u_s} - 1 + u_s + \frac{n_0}{p_0}\left(e^{u_s} - 1 - u_s\right)\right]^{\frac{1}{2}} \tag{1.36}$$

$$\begin{aligned}\text{where}\quad u_s &\equiv \beta\phi_s\\ \beta &\equiv \frac{q}{K_BT}\\ \text{and}\quad L_D &\equiv \sqrt{\frac{\epsilon_{si}}{q\beta p_0}} = \text{The Extrinsic Debye Length}\end{aligned}$$

Notice that Q_{si} is the charge in the semiconductor *per unit area.* In this treatment, we shall use symbols of the type Q and C with various subscripts to denote the corresponding charges and capacitance values *per unit area.* Q_{si} consists of mobile as well as fixed charge. The mobile charge is contributed by holes when $u_s < 0$ and by electrons when $u_s > 0$ (for a P type semiconductor). As we shall see later, the mobile electron charge is substantial only when the positive surface potential exceeds a threshold value.

The fixed charge is contributed by the depletion charge when the surface potential is positive. The depletion charge per unit area can be calculated by the depletion formula.

$$Q_{depl} = -qN_aX_d = \sqrt{2qN_a\epsilon_{si}\phi_s} \qquad (\phi_s > 0)$$

A somewhat more accurate expression for depletion charge accounts for slightly lower charge density at the edge of the depletion region by subtracting K_BT/q from ϕ_s.

$$Q_{depl} = -qN_aX_d = \sqrt{2qN_a\epsilon_{si}(\phi_s - K_BT/q)} \qquad (\phi_s > K_BT/q) \tag{1.37}$$

Calculated values for the total semiconductor charge per unit area (ie. inclusive of depletion and mobile charge) and just the depletion charge per unit area have been plotted in figure 1.8 for a P type semiconductor doped to $10^{16}/\text{cm}^3$. For small positive surface potential, the total semiconductor charge contains only depletion charge. However, beyond a surface potential near $2\Phi_F$, the total charge exceeds the depletion charge very rapidly. This additional charge is due to mobile minority carriers (in this case, electrons).

Practical case

A practical MOS structure will differ from the ideal case assumed above in a few respects. There is a built-in potential difference between the metal used and Si, due to the difference between their work functions. This shifts the relationship between $V_g$and ϕ_s. Also, there is a fixed oxide charge which resides essentially at the silicon-oxide interface. Thus, the total charge in the Gaussian box includes this fixed charge and the semiconductor charge. These two non-idealities can be accounted for by modifying the relationship between $V_g$and $\phi_s$to be

$$V_g = \Phi_{ms} + \phi_s - \frac{Q_{si} + Q_{ox}}{C_{ox}} \tag{1.38}$$

Where Φ_{ms} is the metal to semiconductor work function difference.

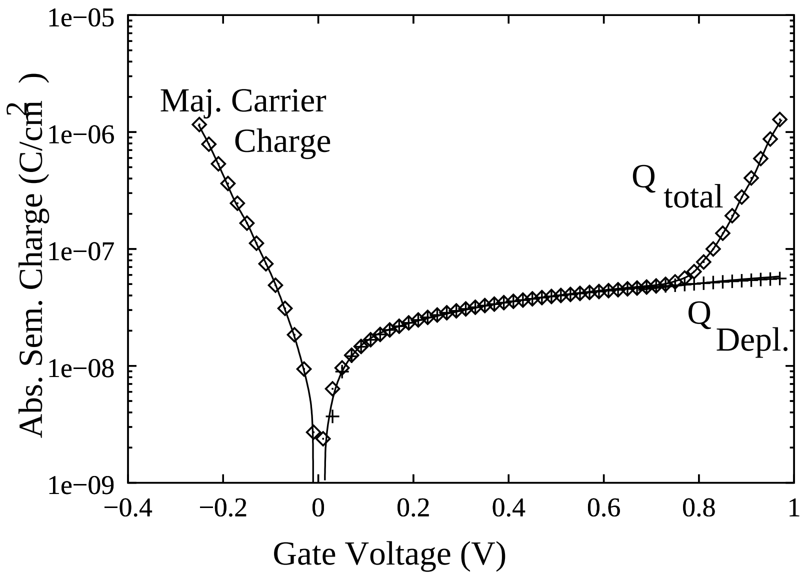


Figure 1.8: semiconductor charge as a function of surface potential

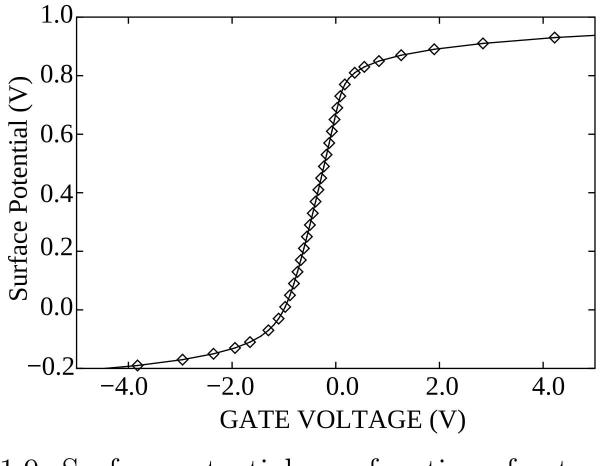


Figure 1.9: Surface potential as a function of gate voltage

Figure 1.9 shows the surface potential as a function of applied voltage for a MOS capacitor with oxide thickness of 22.5 nm, substrate doping of 10^{16}/cc, oxide charge of 4×10^{10}q and aluminium as the gate metal. The surface potential changes quite slowly as a function of gate voltage in the accumulation and inversion regions.

The absolute value of semiconductor charge has been plotted as a function of applied gate voltage in figure 1.10. (The charge is actually negative for positive gate voltages). As one can see, for small positive gate voltages, the entire semiconductor charge is depletion charge. As the voltage exceeds a threshold voltage, the total charge becomes much larger than the

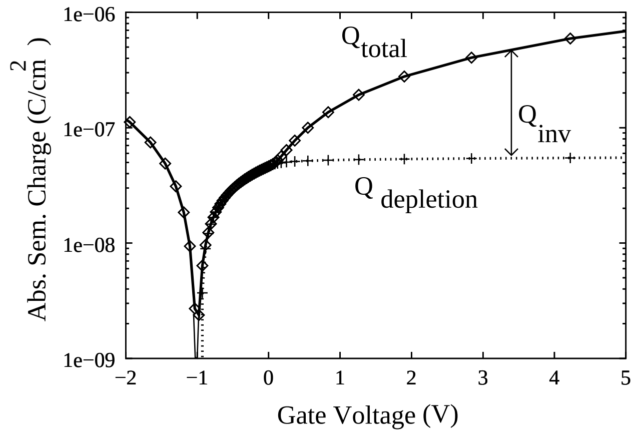


Figure 1.10: Semiconductor charge as a function of gate voltage

depletion charge. The excess charge is provided by mobile electron charges. This is the *inversion* region of operation, where electrons become the majority carriers near the surface in a p type semiconductor. Notice that the depletion charge is practically constant in this region. This region begins when the surface potential exceeds $2\Phi_F$.

1.4 The MOS Transistor

Inversion converts a p type semiconductor to n type at the surface. We can use this fact to construct a transistor. We place semiconductor regions strongly doped to N type on either side of a MOS capacitor made using P type silicon. Now if we try to pass a current between

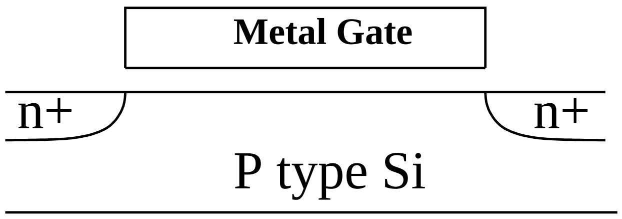


Figure 1.11: A MOS Transistor

these two N regions when inversion has not occurred, we encounter series connected NP and PN diodes on the way. Whatever the polarity of the voltage applied to pass current, one of these will be reverse biased and practically no current will flow.

However, after inversion, the intervening P region would have been converted to N type. Now there are no junctions as the whole surface region is n type. Current can now be easily passed between the two n regions. This structure is an n channel MOS transistor. pMOS transistors can be similarly made using P regions on either side of a MOS capacitor made on n type silicon. When current flows in an n channel transistor, electrons are supplied by the more negative of the two n+ contacts. This is called the source electrode. The more positive n+ contact collects the electrons and is called the drain. The current in the transistor is controlled by the metal electrode on top of the oxide. This is called the gate electrode.

1.4.1 I-V characteristics of a MOS transistor

A quantitative derivation of the current-voltage characteristics of the MOS device is complicated by the fact that it is inherently a two dimensional device. The *vertical* field due to the gate voltage sets up a mobile charge density in the channel region as seen in figure 1.10. The *horizontal* field due to source-drain voltage causes these charges to move, and this constitutes the drain current. Therefore, a two dimensional analysis is required to calculate the transistor current, which can be quite complex. However, reasonably simple models can be derived by making several simplifying assumptions.

1.4.2 A simple MOS model

We make the following simplifying assumptions:

- The vertical field is much larger than the horizontal field. Then, the resultant field is nearly vertical, and the results derived for the 1 dimensional analysis for the MOS capacitor can be used to calculate the point-wise charge density in the channel. This is known as the *gradual channel* approximation. Accurate numerical simulations have shown that this approximation is valid in most cases.
- The source is shorted to the bulk.
- The gate and drain voltages are such that a continuous inversion region exists all the way from the source to the drain.
- The depletion charge is constant along the channel.
- The total current is dominated by drift current.
- The mobility of carriers is constant along the channel.

Figure 1.12 shows the co-ordinate system used for evaluating the drain current. The x axis points into the semiconductor, the y axis is from source to the drain and the z axis is along the width of the transistor. The origin is at the source end of the channel. We represent the channel voltage as V(y), which is 0 at the source end and V_d at the drain end. We assume the current to be made up of just the drift current. Since we are carrying out a quasi 2 dimensional analysis, all variables are assumed to be constant along the z axis. Let n(x,y)

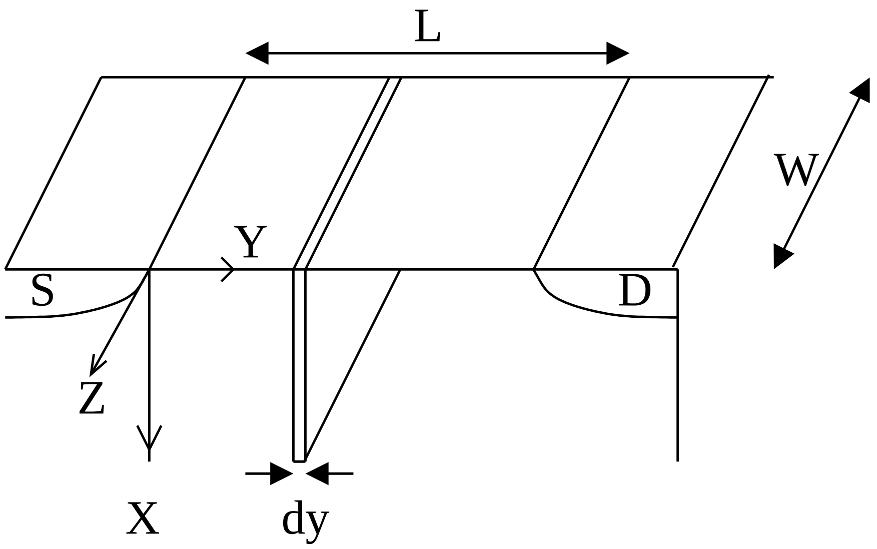


Figure 1.12: Coordinate system used for analysing the MOS transistor

be the concentration of mobile carriers (electrons for an n channel device) at the position x,y (for any z). The drift current density at a point is

$$\begin{aligned} J &= \text{no. of carriers} \times \text{charge per carrier} \times \text{velocity} \\ &= n(x,y) \times (-q) \times \mu \times \left(-\frac{\partial V(y)}{\partial y}\right) \\ &= \mu n(x,y) q \frac{\partial V(y)}{\partial y} \end{aligned}$$

Integrating the current density over a semi-infinite plane at the channel position y (as shown in the figure 1.12) will then give the drain current.

$$I_d = \int_{x=0}^{\infty} \int_{z=0}^{W} \mu n(x,y) q \frac{\partial V(y)}{\partial y} dz dx$$

Since there is no dependence on z, the z integral just gives a multiplication by W. Therefore,

$$I_d = \mu W q \int_{x=0}^{\infty} n(x,y) \frac{\partial V(y)}{\partial y} dx$$

the value of n(x,y) is non zero in a very narrow channel near the surface. We can assume that $\frac{\partial V(y)}{\partial y}$ is constant over this depth. Then,

$$I_d = \mu W q \frac{\partial V(y)}{\partial y} \int_{x=0}^{\infty} n(x,y) dx$$

but $q\int_{x=0}^{\infty} n(x,y)dx = -\mathrm{Q_n}(y)$ where $\mathrm{Q_n}$(y) is the electron charge per unit area in the semiconductor at point y in the channel. ($\mathrm{Q_n}$(y) is negative, of course). therefore

$$I_d = -\mu W \frac{\partial V(y)}{\partial y} \mathrm{Q_n}(y) \tag{1.39}$$

Integrating the drain current along the channel gives

$$\begin{aligned} \int_0^L I_d dy &= -\mu W \int_0^L \mathrm{Q_n}(y) \frac{\partial V(y)}{\partial y} dy \\ I_d \times L &= -\mu W \int_0^{V_d} \mathrm{Q_n}(y) \mathrm{d}V(y) \\ \text{So, } I_d &= -\mu \frac{W}{L} \int_0^{V_d} \mathrm{Q_n}(y) \mathrm{d}V(y) \end{aligned}$$

We now use the assumption that the surface potential due to the vertical field saturates around $2\Phi_\mathrm{F}$if we are in the inversion region. Therefore, the total surface potential at point y is V(y) + 2 Φ_F. Now, by Gauss law and continuity of normal component of D at the interface,

$$C_{ox}\left(V_g - \Phi_{\mathrm{MS}} - \phi_s\right) = -\left(Q_{si} + Q_{ox}\right)$$

therefore,

$$-Q_{si} = C_{ox}\left(V_g - \Phi_{\mathrm{MS}} - V(y) - 2\Phi_\mathrm{F} + Q_{ox}/C_{ox}\right)$$

However,

$$Q_{si} = \mathrm{Q_n} + \mathrm{Q_{depl}}$$

So

$$\begin{aligned} -\mathrm{Q_n}(y) &= -Q_{si}(y) + \mathrm{Q_{depl}} \\ &= C_{ox}\left(V_g - \Phi_{\mathrm{MS}} - V(y) - 2\Phi_\mathrm{F} + (Q_{ox} + \mathrm{Q_{depl}})/C_{ox}\right) \end{aligned}$$

We have assumed the depletion charge to be constant along the channel. Let us define

$$\mathrm{V_T} \equiv \Phi_{\mathrm{MS}} + 2\Phi_\mathrm{F} - \frac{(Q_{ox} + \mathrm{Q_{depl}})}{C_{ox}}$$

then

$$-\mathrm{Q_n}(y) = C_{ox}(V_g - \mathrm{V_T} - V(y))$$

and therefore,

$$\begin{aligned} I_d &= \mu C_{ox} \frac{W}{L} \int_0^{V_d} (V_g - \mathrm{V_T} - V(y)) \mathrm{d}V(y) \\ &= \mu C_{ox} \frac{W}{L} [(V_g - \mathrm{V_T})V_d - \frac{1}{2} V_d^2] \end{aligned} \tag{1.40}$$

This derivation gives a very simple expression for the drain current. However, it requires a lot of simplifying assumptions, which limit the accuracy of this model.
If we do not assume a constant depletion charge along the channel, we can apply the depletion formula to get its dependence on V(y).

$$\mathrm{Q}_{\mathrm{depl}} = -\sqrt{2\epsilon_{si} q N_a (V(y) + 2\Phi_{\mathrm{F}})}$$

then,

$$-\mathrm{Q}_{\mathrm{n}} = C_{ox}\left(V_g - \Phi_{\mathrm{MS}} - V(y) - 2\Phi_{\mathrm{F}}\right) + Q_{ox} - \sqrt{2\epsilon_{si} q N_a (V(y) + 2\Phi_{\mathrm{F}})}$$

which leads to

$$\begin{aligned} I_d = \mu C_{ox}\frac{W}{L}\Bigg[&\left(V_g - \Phi_{\mathrm{MS}} - 2\Phi_{\mathrm{F}} + \frac{Q_{ox}}{C_{ox}}\right)V_d - \frac{1}{2}V_d^2 \\ &- \frac{2}{3}\frac{\sqrt{2\epsilon_{si} q N_a}}{C_{ox}}\left((V_d + 2\Phi_{\mathrm{F}})^{3/2} - (2\Phi_{\mathrm{F}})^{3/2}\right)\Bigg] \end{aligned}$$

This is a more complex expression, but gives better accuracy.

1.4.3 Modeling the saturation region

The treatment in the previous section is valid only if there is an inversion layer all the way from the source to the drain. For high drain voltage, the local vertical field near the drain is not adequate to take the semiconductor into inversion. Several models have been used to describe the transistor behaviour in this regime. The simplest of these defines a saturation voltage at which the channel just pinches off at the drain end. The current calculated for this voltage by the above models is then supposed to remain constant at this value for all higher drain voltages. The pinch-off voltage is the drain voltage at which the channel just vanishes near the drain end. Therefore, at this point the gate voltage $\mathrm{V_g}$ is just less than a threshold voltage above the drain voltage $\mathrm{V_d}$. Thus, at this point,

$$V_{dsat} = V_g - V_T$$

The current calculated at $\mathrm{V_{dsat}}$ will be denoted as $\mathrm{I_{dss}}$. Thus,

$$I_{dss} = \mu C_{ox}\frac{W}{L}[(V_g - \mathrm{V_T})^2 - \frac{1}{2}(V_g - \mathrm{V_T})^2]$$

for the simple transistor model. Thus

$$I_{dss} = \frac{1}{2}\mu C_{ox}\frac{W}{L}(V_g - \mathrm{V_T})^2 \qquad (1.41)$$

The drain current is supposed to remain constant at this $\mathrm{V_d}$ independent value for all drain voltages $> V_g - V_T$.

Early Voltage approach

Assuming a constant current in the saturation region leads to an infinite output resistance. This can lead to exaggerated estimates of gain from an amplifier. Therefore, we need a more realistic model for the transistor current in the saturation region. One of these is a generalisation of the model proposed by James Early for bipolar transistors. This model is not strictly applicable to MOS transistors. However, due to its numerical simplicity, it is often used in compact models for circuit simulation.

A geometrical interpretation of the Early model states that the drain current increases linearly in the saturation region with drain voltage, and if saturation characteristics for different gate voltages are produced backwards, they will all cut the drain voltage axis at the same (negative) drain voltage point. The absolute value of this voltage is called the Early Voltage V_E.

The current equations in saturation mode now become:

$$\begin{aligned} I_{dss} &\equiv I_d(V_g, V_{dss}) \\ I_d &= I_{dss}\frac{V_d + V_E}{V_{dss} + V_E} \qquad \text{For } V_d > V_{dss} \end{aligned} \tag{1.42}$$

Any model can be used for calculating the drain current for $V_d < V_{dss}$. The value of V_{dss} will be determined by considerations of continuity of the drain current and its derivative at the changeover point from linear to saturation regime. For example, if we use the simple model described in eq. 1.40,

$$\begin{aligned} & \frac{\partial I_d}{\partial V_d} = \mu C_{ox}\frac{W}{L}\left(V_g - V_T - V_d\right) && \text{For } V_d \le V_{dss} \\ \text{And} \quad & \frac{\partial I_d}{\partial V_d} = \frac{I_{dss}}{V_{dss} + V_E} && \text{For } V_d \ge V_{dss} \\ \text{Where} \quad & I_{dss} \equiv \mu C_{ox}\frac{W}{L}\left[\left(V_g - V_T\right)V_{dss} - \frac{1}{2}V_{dss}^2\right] && \end{aligned}$$

On matching the value of $\frac{\partial I_d}{\partial V_d}$ on both sides of V_{dss}, we get

$$V_{dss} = V_E\left(\sqrt{1 + \frac{2\left(V_g - V_T\right)}{V_E}} - 1\right)$$

In practice, V_E is much larger than $V_g - V_T$. If we expand the above expression, we find that to first order the value of V_{dss} remains the same as the one used in the simple model - that is, $V_g - V_T$. Expansion to second order gives

$$V_{dss} \simeq \left(V_g - V_T\right)\left(1 - \frac{V_g - V_T}{2V_E}\right) \tag{1.43}$$

Simulation Model

Since the value of V_{dss} does not change substantially from the ideal saturation case, a simpler approach can be tried. The drain current is calculated using the ideal saturation model and its value is multiplied by a correction factor $= (1 + \lambda V_d)$ in saturation *as well as* in linear regime. This automatically assures continuity of I_d and its derivative. λ is a fit parameter, whose value is $\approx 1/V_E$. This approach is used in SPICE, a popular circuit simulation program.

1.5 MOS Device Scaling

Since the transistor current depends on W/L, it is interesting to see what happens if we reduce both W and L, keeping their ratio constant. We have to adjust other parameters as well, in order to ensure that the transistor works without problems.

Due to technological constraints, we cannot reduce lateral geometries without reducing layer thicknesses. To define finer lateral dimensions through etching etc., we need the layers to be thinner. Thus all dimensions, vertical or lateral, need to be scaled by the same factor. To ensure that higher fields in the device do not cause breakdown, we have to scale down all the voltages by the same factor as L. (This is known as constant field scaling).

We also need to scale depletion widths in the same ratio as W and L. This is essential in order to scale down the separation between transistors and to control channel length modulation due to drain voltage. This requires doping densities to be scaled up by the same factor as the one used to scale down W and L.

So we define a scaling factor S, and reduce W, L, junction depths and oxide thicknesses etc. by this factor. Doping densities need to be increased by factor S. All working voltages and the Threshold voltage V_T need to be scaled down by S. Once this scaling is done, we are interested in evaluating the impact on the circuit performance.

1.5.1 Consequences of Scaling

We assume classical or Constant Field scaling in the following.

Device Area: Since W reduces by $\downarrow S$ and L reduces by $\downarrow S$, the area reduces by $\downarrow S^2$.

Packing Density: For a given chip area, the number of devices which can be packed in this chip will go up by $\uparrow S^2$.

C_{ox}**:** The gate capacitance per unit area is given by ϵ/t_{ox}. Since t_{ox} scales down by $\downarrow S$, C_{ox} increases by $\uparrow S$. C_{ox} determines the transconductance, so this increase is good.

Load capacitance: All dimensions, including depletion widths have been scaled down by $\downarrow S$. Total capacitance $= \epsilon A/t$. Now A reduces by $\downarrow S^2$, while the dielectric thickness (be it oxide or depletion width) reduces by $\downarrow S$. The net effect is that total capacitance $= \epsilon\text{Area} \downarrow S^2/t \downarrow S$ reduces by $\downarrow S$.

Voltages: All voltages such as V_{DS}, V_{GS}, V_T etc. are scaled down by $\downarrow S$ to keep the field constant.

Drain current: I_{DS} is given by $\mu C_{ox}(W/L) f(V_{DS}, V_{GS}, V_T)$. Since all voltages are scaled down by $\downarrow S$ and f is a square function of voltages both in linear mode and saturation, f will scale down as $\downarrow S^2$. Thus,

$$I_{DS} = \mu C_{ox}(\uparrow S)(W \downarrow S/L \downarrow S) f(V_{DS}, V_{GS}, V_T)(\downarrow S^2)$$

So combining all dependences, $I_{DS} \downarrow S$.

Slew Rate: Slew rate is the rate of change of voltage at any node. Since $I = C\frac{dV}{dt}$, the slew rate goes as $I(\downarrow S)/C(\downarrow S)$. Thus slew rate remains unchanged with scaling.

Delay: Delay is given by the total voltage change divided by $\frac{dV}{dt}$. Since all voltages are scaled down by $\downarrow S$, while $\frac{dV}{dt}$ remains unchanged, the delay reduces as $\downarrow S$.

Static Power: It is given by $V \times I$. So it scales as $(\downarrow S)(\downarrow S)$, that is $\downarrow S^2$.

Dynamic Power: Dynamic power is given by $C_{total} V^2 f$. This scales as $(\downarrow S)(\downarrow S^2)(\uparrow S)$. Thus dynamic power reduces as $\downarrow S^2$ even when the frequency of operation is increased by $\uparrow S$ to take advantage of shorter delays, which scale down by $(\downarrow S)$.

With all dimensions and voltages divided by the factor S(> 1), We can summarise the impact of using constant field Scaling as follows:

Device area	$\propto W \times L : (\downarrow S)(\downarrow S)$	$\downarrow S^2$
C_{ox}	ϵ_{ox}/t_{ox} : const$/(\downarrow S)$	$\uparrow S$
C_{total}	$\epsilon A/t : (\downarrow S^2)/(\downarrow S)$	$\downarrow S$
V_{DS}, V_{GS}, V_T	Voltages : $(\downarrow S)$	$\downarrow S$
I_d	$\mu C_{ox}(W/L)(\propto V^2)$: $(\uparrow S)$(const)$(\downarrow S^2)$	$\downarrow S$
Slew Rate $\frac{dV}{dt}$	$I/C_{total} : (\downarrow S)/(\downarrow S)$	*const.*
Delay	$V/\frac{dV}{dt} : (\downarrow S)/(const)$	$\downarrow S$
Static Power	$V \times I : (\downarrow S)(\downarrow S)$	$\downarrow S^2$
dynamic power	$C_{total} V^2 f : (\downarrow S)(\downarrow S^2)(\uparrow S)$	$\downarrow S^2$
Power delay product	delay $\times$ power$(\downarrow S)(\downarrow S^2)$	$\downarrow S^3$
Power density	power/area : $(\downarrow S^2)/(\downarrow S^2)$	*const.*

Thus, scaling leads to:

- Improved packing density: $\uparrow S^2$
- Improved speed: delay $\downarrow S$
- Improved power consumption: $\downarrow S^2$

So, circuit performance improves dramatically with transistor scaling. This provides the motivation for making transistors as small as possible.

What are the limits on scaling?

These come from processing technology limitations, device limitations and circuit considerations such as reduced signal to noise ratio due to reduced supply voltages.

1.5.2 Moore's "Law"

In 1965, Gordon Moore, the co-founder of Fairchild Semiconductor as well as Intel, described a doubling every year in the number of components per integrated circuit. It is an observation of a trend and an empirical relationship – not a physical or natural law! However, given the prominence of Gordon Moore, it is widely referred to as Moore's Law.

In 1975, Moore modified his observation for the rate of device scaling and predicted a doubling of device density every two years. It is remarkable that this trend has continued over several decades. It is only in the last decade that the rate of doubling has slowed down remarkably, as we hit several physical limits.

Device scaling started initially as an empirical observation. The theoretical basis of constant field device scaling was laid down in a landmark paper in 1974 from a group of scientists from IBM.

R.H. Dennard, F. H. Gaensslen, Hw A-Nien Yu, V. L. Rideout, E. Bassous, and A. R, LeBlanc, "Design of Ion-Implanted MOSFETs with Very Small Physical Dimensions", IEEE Journal of Solid-State Circuits, Vol. SC-9, No. 5, pp. 256-268, 1974.

This is one of the most quoted papers in this field. I strongly recommend that you read it – for its contents, but also to learn from its style of technical writing.

1.5.3 The technology road map

The incredible rate of increase in circuit performance has been possible through careful planning. The semiconductor industry used Moore's prediction for setting specific targets for development in process technology, processing equipment and for research and development in critical areas of device Physics. The result of this planning was the creation of an International Technology Roadmap for Semiconductor Scaling – or ITRS. ITRS has been revised every year till recently. A new ITRS has not been issued after 2016.

It is hard to track and scale the optimum size of numerous structures on an Integrated circuit. As discussed earlier, it is common to describe feature sizes in units of a parameter called λ. Now sizes of various structures can be described in units of λ. As we scale technologies, we just scale the value of λ. Feature sizes remain the same in units of λ, which is convenient.

The smallest feature on a chip is the contact window. The value of λ is so defined that the smallest feature size is 2λ. The smallest registration rule – for example the extent to which a contact window must be inside a diffused region – is λ.

As a result of careful planning and the considerable financial rewards of improved MOS technology, feature sizes have been continually scaled. The table below gives the commonly used channel lengths by the year in various decades.

1971	10 μm	1974	6 μm	1977	3 μm				
1981	1.5 μm	1984	1 μm	1987	800 nm				
1990	600 nm	1993	350 nm	1996	250 nm	1999	180 nm		
2001	130 nm	2003	90 nm	2005	65 nm	2007	45 nm	2009	32 nm
2012	22 nm	2014	14 nm	2017	11 nm				

The scaling rate has slowed down after 2010. This is because feature sizes had already reached about 20 nm – about 3% of the wavelength of sodium light!

1.5.4 Demand from Processing Technology

Circuit performance improves dramatically with transistor scaling. This provides the motivation for making transistors as small as possible.
What demands does it place on processing technology?

- Scaling requires much higher resolution in defining geometries. Size of the finest patterns in the state of the art technologies is about 10nm. This is about a fiftieth of the wavelength of sodium light!

- Advanced photo-lithographic techniques need to be used to define such fine geometries. We need deep UV lithography and even XRay lithography to define such fine structures.
- Etching techniques have to be improved to define such fine structures. Dry etching using plasma or reactive ion etching is used rather than wet chemical etching to define such fine structures

1.5.5 Limits of scaling

Scaling is being limited now due to several reasons.

- We are reaching limits of resolution possible with photo-selective processes and etching etc.
- Traditional Device Physics is not valid any more for such small structures.

Remember, the lattice constant of Silicon is $\approx$ 0.5 nm. So there are as few as 20 atoms between source and drain of a 10 nm channel MOSFET. Clearly, conduction models based on statistics will not hold here.

Indefinite voltage scaling is not possible. If the voltage is scaled down drastically,

- signal to noise ratio will become poor,
- leakage currents will become dominant as KT/q has not been scaled and current equations of junctions involve qV/KT.
- System considerations such as interconnect delay will limit performance gain.
- At low voltages, supplying power requires higher currents. Feeding such high currents through IC pins becomes impractical.

1.5.6 Short Channel Effects

Several effects become prominent once transistor dimensions are made very small. The classical model for MOS transistors assumed a uniform doping in the channel region. However, because of heavy doping in the source/drain region, there is a sideways diffusion of impurities into the channel. If the channel length is quite short, the region of non-uniform doping becomes a large fraction of the channel length. This results in considerable deviations from the transistor model.

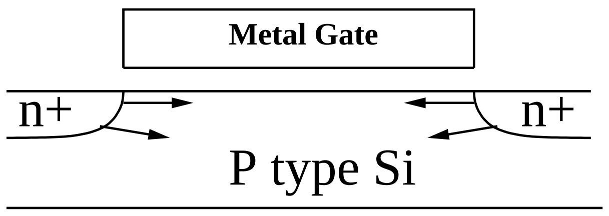


Figure 1.13: Sideways diffusion from source drain regions

Threshold voltage of a long channel transistor is independent of channel length. As we scale down channel length, the threshold voltage becomes dependent on channel length. (Short channel effect on V_T).

Also, as the drain comes closer to the source, the field due to the drain channel junction reaches the source channel junction. This reduces the barrier to carrier injection from the source into the channel. This is known as Drain Induced Barrier Lowering or DIBL.

1.5.7 Narrow Channel Effects

As we scale down devices, channel widths as well as channel lengths are reduced. The threshold voltage becomes dependent on channel width as well as channel length for scaled down devices. This is because the depletion charge on the sides of the channel is no more negligible compared to the charge directly under the gate. For uniformly doped devices, V_T increases as the channel is made narrower. However, the dependence is more complex when doping is non-uniform.

1.6 Breakdown Phenomena

1.6.1 Avalanche Breakdown

The drain channel junction is reverse biased. In saturation region, there is high field region next to the gate. If the field exceeds some critical value, carrier multiplication will occur, leading to avalanche breakdown. Multiplication produces excess electron-hole pairs. Electrons are attracted towards the positively biased drain. Holes drift towards the source and constitute a “base current” for the parasitic lateral npn transistor.
Carrier multiplication near drain can result in a sharp increase in drain current. This is the avalanche breakdown of the transistor. It is also possible that the parasitic bipolar turns on, due to the base current provided by the drifting holes from the drain junction, adding its

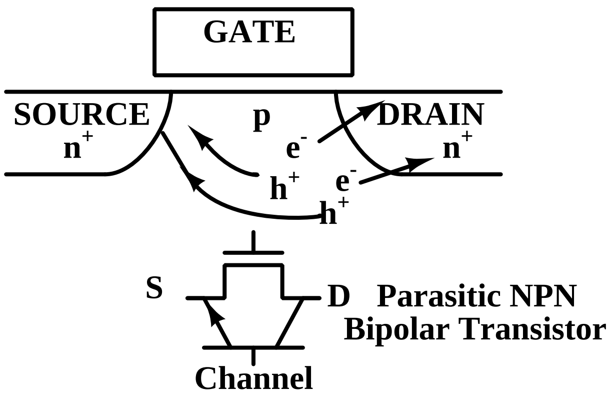


Figure 1.14: Avalanche breakdown at high fields

current to the drain current. The additional current due to the bipolar action, combined with carrier multiplication near the drain can result in early breakdown of the transistor.

1.6.2 Punch Through

If the channel is very short, at high drain voltages, the depletion region due to the drain-substrate junction can reach the source. Due to the drain field, the source/substrate junction will get forward biased and will inject current into the channel, even if the gate voltage is below V_T. This is an extreme case of drain induced barrier lowering and results in heavy current even though the transistor is supposed to be 'OFF'. This is known as "Punch Through".

1.7 Parasitic Devices

1.7.1 Field transistors

As we make the devices needed for the desired circuit, several other devices get formed. The most common of these is the field transistor. A MOS like structure exists between unrelated diffusion areas due to metal lines crossing over unrelated diffusion areas.

1.7.2 Latch up due to parasitic pnpn structures

Fig.1.16 shows the lay out of an inverter. (As we shall learn later, this is a bad layout!). While the lay out does form an inverter as desired, it also forms a parasitic latch-up structure which can turn on, shorting V_{DD} to ground and destroying the IC due to the resulting heavy

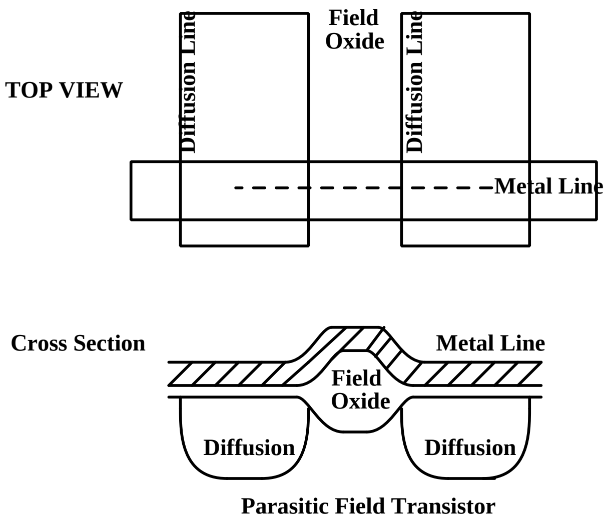


Figure 1.15: Parasitic Field transistor in MOS technology

current.

A vertical pnp transistor is formed by

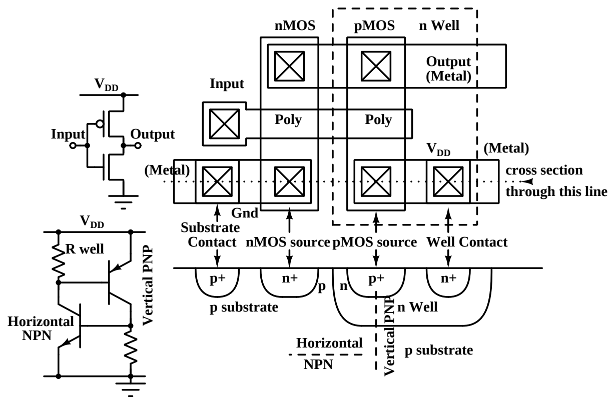


Figure 1.16: Formation of a parasitic latch up structure in an inverter

1. the p+ source of a pMOS transistor connected to V_{DD} (which becomes the emitter),
2. the n well (which becomes the base), and
3. the p substrate (which becomes the collector of this transistor).

The n well is connected to V_{DD} through a resistive path, which represents the resistance of the n well to the well contact.
This can also be seen in Fig.1.17. A horizontal npn transistor is formed by

1. the n+ source of an nMOS transistor connected to ground (which becomes the emitter),
2. the p substrate, (which becomes the base), and
3. the n well, (which becomes the collector).

Since the collector of the npn and the base of the pnp are both formed by the n well, these two are connected. Similarly, the collector of the pnp and the base of the npn are formed by the p substrate, so these are connected too.

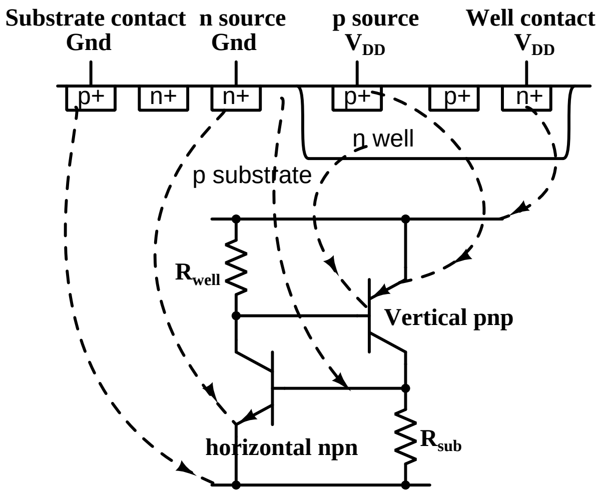


Figure 1.17: Circuit for the parasitic latch up circuit

From the equivalent circuit given in Fig.1.18, we can see that it forms a positive feedback circuit.

- An increase in the base current of the pnp will be amplified by its β_p and a large part of it will flow through the base emitter junction of the npn transistor.
- This part will be amplified by the β_n of the npn and a substantial part of it will go through the base emitter junction of the pnp transistor.

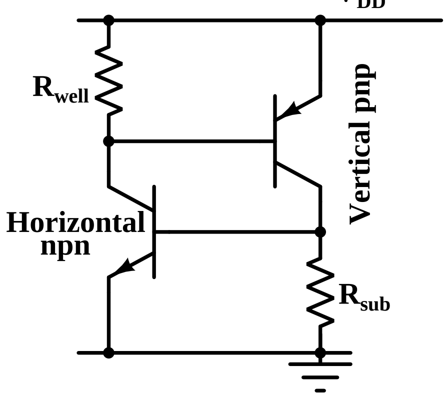


Figure 1.18: Positive feed back in the latch-up structure

If the product of the two amplification factors β_p and β_n and the current division ratios between the resistors and the base emitter junctions exceeds 1, the currents will keep increasing due to this feedback, till there is a dead short between V_{DD} and ground. This is called latch up.

1.7.3 Preventing Latch up

To prevent latch up, we must reduce the β of the parasitic bipolar transistors, and make sure that most of the collector current of either transistor is directed to the resistor and not to the base-emitter junction of the other transistor.

This can be done through process steps as well as through layout design rules.

The doping gradient of the n well should be made retrograde. (Doping should increase as we go deeper). This kills the current gain β_p of the pnp transistor.
The n well should have a guard ring connected to V_{DD}, which will collect any current which could form the base current of the pnp.

In layout, substrate and well contacts should be placed frequently, to reduce the value of R_{well} and $R_{substrate}$. n channel transistors should be placed far from the edge of the n well. This increases the base width of the npn transistor and kills its current gain. p channel transistors should also be placed far from the well edge and the n well should be deep to kill the gain of the pnp transistor.

Chapter 2

Static CMOS Logic Design

Static logic circuits are those which can hold their output logic levels for indefinite periods as long as the inputs are unchanged. Circuits which depend on charge storage on capacitors are called dynamic circuits and will be discussed in a later chapter.

2.1 Static CMOS Design style

The most common design style in modern VLSI design is the Static CMOS logic style. In this, each logic stage contains pull up and pull down networks which are controlled by input signals. The pull up network contains p channel transistors, whereas the pull down network is made of n channel transistors. The networks are so designed that the pull up and pull down networks are never 'on' simultaneously. This ensures that there is no static power consumption.

2.2 CMOS Inverter

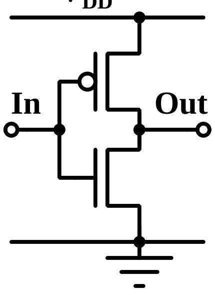


Figure 2.1: The basic CMOS inverter

The simplest of such logic structures is the CMOS inverter. In fact, for any CMOS logic design, the CMOS inverter is the basic gate which is first analyzed and designed in detail. Thumb rules are then used to convert this design to other more complex logic. The basic CMOS inverter is shown in fig. 2.1. We shall develop the characteristics of CMOS logic through the inverter structure, and later discuss ways of converting this basic structure more complex logic gates.

2.2.1 Static Characteristics

The range of input voltages can be divided into several regions.

- nMOS 'off', pMOS 'on'
- nMOS saturated, pMOS linear
- nMOS saturated, pMOS saturated
- nMOS linear, pMOS saturated
- nMOS 'on', pMOS 'off'

Let us compute the output voltage for a series of input voltages from 0V to V_{DD}. We have plotted the individual drain currents of the n and p channel transistors as functions of the output voltage for different input voltages V_{in}.

The gate voltage for the nMOS transistor = V_{in} and the drain voltage = V_{out} while the

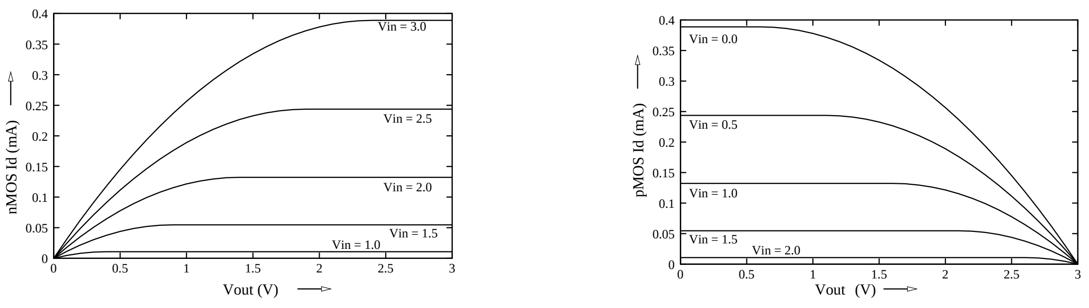


Figure 2.2: nMOS and pMOS drain currents for different input voltages

absolute gate voltage for pMOS is $V_{DD} - V_{in}$ and the absolute drain voltage is $V_{DD} - V_{out}$.

The Output voltage of the inverter will be the value where the two currents are equal for any given V_{in}.

nMOS ‘off’, pMOS ‘on’

For $0 < V_i < V_{Tn}$ the n channel transistor is ‘off’, the p channel transistor is ‘on’ and the output voltage $= V_{DD}$. This is the normal digital operation range with input = ‘0’ and output = ‘1’.

nMOS saturated, pMOS linear

In this regime, both transistors are ‘on’. The input voltage V_i is $> V_{Tn}$, but is small enough so that the n channel transistor is in saturation, and the p channel transistor is in the linear regime. In static condition, the output voltage will adjust itself such that the currents through the n and p channel transistors are equal.

The absolute value of gate-source voltage on the p channel transistor is V_{DD}- V_i, and therefore

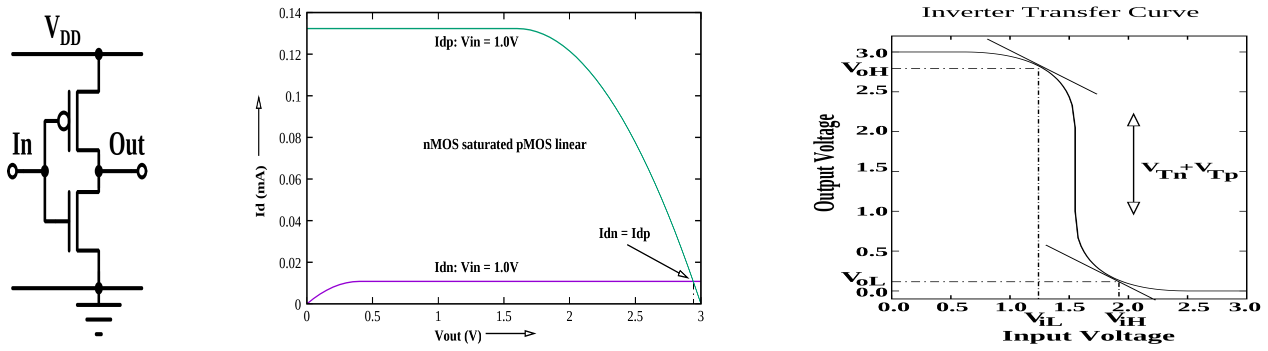


Figure 2.3: Output voltage for nMOS in saturation, pMOS in linear regime

the “over voltage” on its gate is V_{DD}- V_i- V_{Tp}. The drain source voltage of the pMOS has an absolute value V_{DD}-V_o.

Therefore,

$$I_d = K_p\left[(V_{DD} - V_i - V_{Tp})(V_{DD} - V_o) - \frac{1}{2}(V_{DD} - V_o)^2\right] = \frac{K_n}{2}(V_i - V_{Tn})^2 \qquad (2.1)$$

Where symbols have their usual meanings.

We define $\beta \equiv \mathrm{K_n}/\mathrm{K_p}$. We make the substitution $V_{dp} \equiv V_{DD} - V_o$, where $V_{dp}$is the absolute value of the drain-source voltage for the p channel transistor. Then,

$$(V_{DD} - V_i - V_{Tp})V_{dp} - \frac{1}{2}V_{dp}^2 = \frac{\beta}{2}(V_i - V_{Tn})^2 \qquad (2.2)$$

Which gives the quadratic

$$\frac{1}{2}V_{dp}^2 - V_{dp}(V_{DD} - V_i - V_{Tp}) + \frac{\beta}{2}(V_i - V_{Tn})^2 = 0 \qquad (2.3)$$

Solutions to the quadratic are:

$$V_{dp} = (V_{DD} - V_i - V_{Tp}) \pm \sqrt{(V_{DD} - V_i - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{2.4}$$

These equations are valid only when the pMOS is in its linear regime. This requires that

$$V_{dp} \equiv V_{DD} - V_o \leq V_{DD} - V_i - V_{Tp}$$

Therefore, we must choose the negative sign. Thus

$$V_{DD} - V_o = (V_{DD} - V_i - V_{Tp}) - \sqrt{V_{DD} - V_i - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{2.5}$$

Therefore,

$$V_o = V_i + V_{Tp} + \sqrt{(V_{DD} - V_i - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{2.6}$$

Since V_o must be $\geq V_i + V_{Tp}$, the limit of applicability of the above result is given by

$$(V_{DD} - V_i - V_{Tp})^2 = \beta(V_i - V_{Tn})^2$$

That is, the solution for V_o is valid for

$$V_i \leq \frac{V_{DD} + \sqrt{\beta} V_{Tn} - V_{Tp}}{1 + \sqrt{\beta}} \tag{2.7}$$

In the case where we size the n and p channel transistors such that

$$K_n = K_p; \text{ so } \beta = 1$$

we have

$$V_o = (V_i + V_{Tp}) + \sqrt{(V_{DD} - V_{Tn} - V_{Tp})(V_{DD} - 2V_i + V_{Tn} - V_{Tp})} \tag{2.8}$$

with

$$V_i \leq \frac{V_{DD} + V_{Tn} - V_{Tp}}{2}$$

nMOS saturated, pMOS saturated

At the limit of applicability of eq. 2.7, when the input voltage is exactly at

$$V_i = \frac{V_{DD} + \sqrt{\beta} V_{Tn} - V_{Tp}}{1 + \sqrt{\beta}} \tag{2.9}$$

both transistors are saturated. Since the currents of both transistors are independent of their drain voltages in this condition, we do not get a unique solution for V_o by equating drain currents.

The currents will be equal for all values of V_o in the range

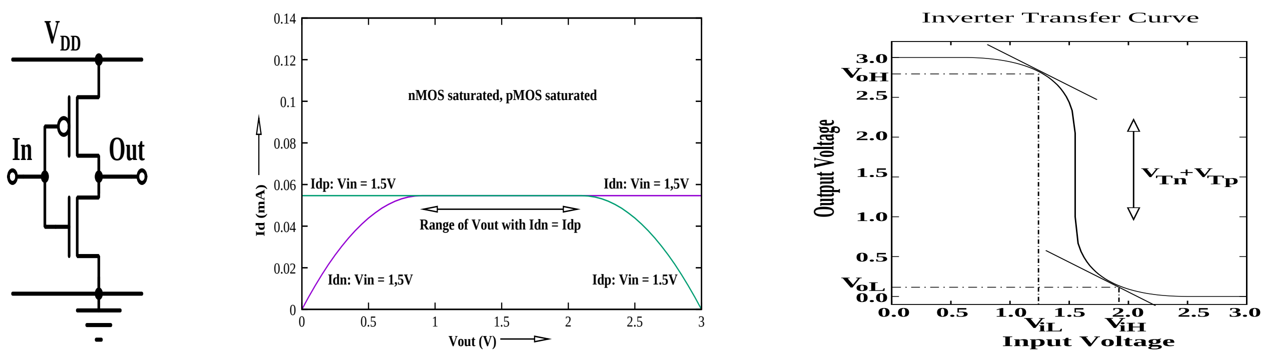


Figure 2.4: Output voltage when both transistors are saturated

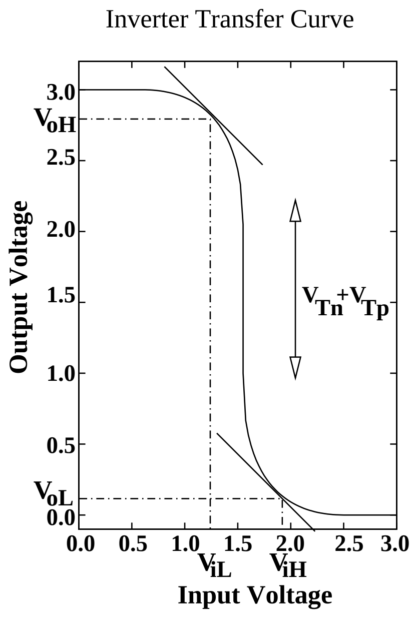


Figure 2.5: Transfer Curve of a CMOS inverter

$$V_i - V_{Tn} \leq V_o \leq V_i + V_{Tp}$$

Thus the transfer curve of an inverter shows a drop of V_{Tn}+ V_{Tp} at a voltage near $V_{DD}/2$. This is actually an artifact of the simple transistor model chosen for this analysis, which assumes perfect saturation of drain current. In a real case, the drain current does depend on the drain voltage (albeit weakly) in the saturation region. If the model incorporates an Early Voltage like effect, the drop near the middle of the characteristic is more gradual.

nMOS linear, pMOS saturated

At the gate voltage given by eq. 2.9, both transistors are saturated. As we increase V_i beyond this value, such that

$$\frac{V_{DD} + \sqrt{\beta}V_{Tn} - V_{Tp}}{1 + \sqrt{\beta}} < V_i < V_{DD} - V_{Tp}$$

both transistors are still 'on', but nMOS enters the linear regime while pMOS gets saturated. Equating currents in this condition,

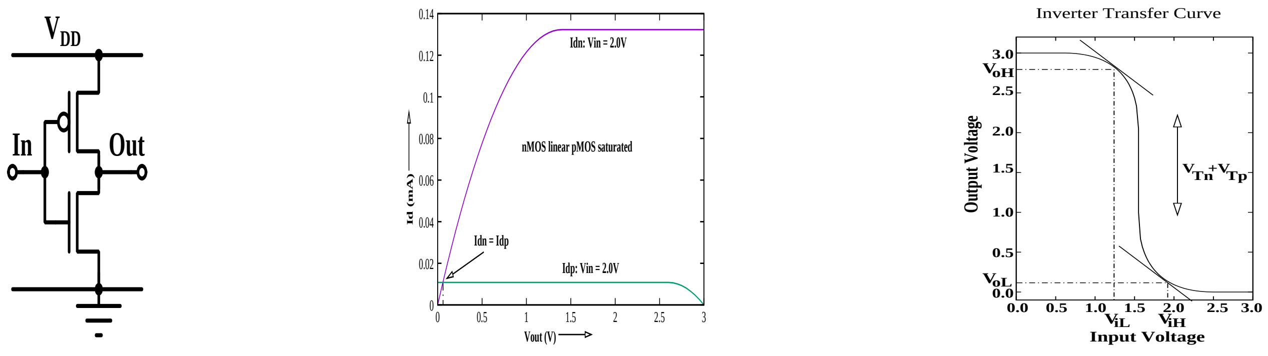


Figure 2.6: Output voltage when nMOS is in linear regime while pMOS is saturated.

$$I_d = \frac{K_p}{2}(V_{DD} - V_i - V_{Tp})^2 = K_n\left[(V_i - V_{Tn})V_o - \frac{1}{2}V_o^2\right] \tag{2.10}$$

From this, we get the quadratic equation

$$\frac{1}{2}V_o^2 - (V_i - V_{Tn})V_o + \frac{(V_{DD} - V_i - V_{Tp})^2}{2\beta} = 0 \tag{2.11}$$

This has solutions

$$V_o = (V_i - V_{Tn}) \pm \sqrt{(V_i - V_{Tn})^2 - \frac{(V_{DD} - V_i - V_{Tp})^2}{\beta}} \tag{2.12}$$

Since the equations are valid only when the n channel transistor is in the linear regime ($V_o < V_i - V_{Tn}$), we choose the negative sign. This gives,

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_i - V_{Tn})^2 - \frac{(V_{DD} - V_i - V_{Tp})^2}{\beta}} \tag{2.13}$$

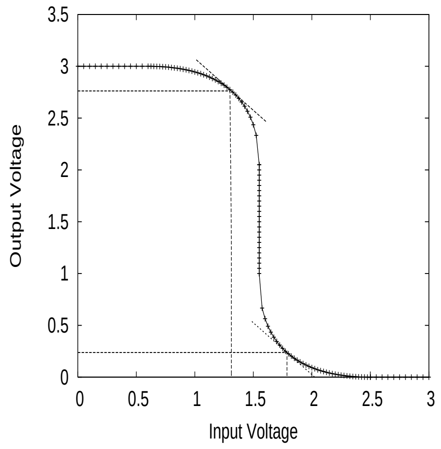
Again, in the special case where $\beta = 1$, we have

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_{DD} - V_{Tn} - V_{Tp})(2V_i - V_{DD} - V_{Tn} + V_{Tp})} \tag{2.14}$$

nMOS 'on', pMOS 'off'

As we increase the input voltage beyond V_{DD}- V_{Tp}, the p channel transistor turns 'off', while the n channel conducts strongly. As a result, the output voltage falls to zero. This is the normal digital operation range with input = '1' and output = '0'.

The figure below shows the transfer curve of an inverter with V_{DD}= 3V, V_{Tn}= 0.6V and V_{Tp}= 0.5V, and $\beta = 1$.



The plot produced by SPICE for this circuit with realistic models is quite similar.

2.2.2 Noise margins

The requirement from a digital circuit is that it should distinguish logic levels, but be insensitive to the exact analog voltage at the input. This implies that the flat portions of the transfer curve (where $\frac{\partial V_o}{\partial V_i}$ is small) are suitable for digital logic. We select two points on the transfer curve where the slope ($\frac{\partial V_o}{\partial V_i}$) is -1.0. The coordinates of these two points define the values of (V_{iL},V_{oH}) and (V_{iH},V_{oL}). Robust digital design requires that the output high level be higher than what is acceptable as a high level at the input ($V_{oH} > V_{iH}$). The difference between these two levels is the 'high' noise margin. This is the amount of noise that can ride on the worst case 'high' output and still be accepted as a 'high' at the input of the next gate. Similarly, we require $V_{oL} < V_{iL}$. The difference, $V_{iL} - V_{oL}$ is the 'low' noise margin. Obviously, it is of interest to evaluate the values of these noise margins. For the discussion which follows, we shall use the expressions derived earlier for $\beta = 1$ to keep the algebra simple.

Calculation of V_{iL} and V_{oH}

from eq. (2.8)

$$V_o = (V_i + V_{Tp}) + \sqrt{(V_{DD} - V_{Tn} - V_{Tp})(V_{DD} + V_{Tn} - V_{Tp} - 2V_i)}$$

From this, we can evaluate $\frac{\partial V_o}{\partial V_i}$ and set it = -1.

$$\frac{\partial V_o}{\partial V_i} = -1 = 1 - \sqrt{\frac{V_{DD} - V_{Tn} - V_{Tp}}{V_{DD} + V_{Tn} - V_{Tp} - 2V_i}} \tag{2.15}$$

This gives

$$V_{iL} = \frac{3V_{DD} + 5V_{Tn} - 3V_{Tp}}{8} \tag{2.16}$$

Substituting this in eq.(2.8), we get

$$\begin{aligned} V_{oH} &= \frac{3V_{DD} + 5V_{Tn} - 3V_{Tp}}{8} + V_{Tp} + \\ &\quad \sqrt{(V_{DD} - V_{Tn} - V_{Tp})\left(V_{DD} + V_{Tn} - V_{Tp} - \frac{3V_{DD} + 5V_{Tn} - 3V_{Tp}}{4}\right)} \\ &= \frac{3V_{DD} + 5V_{Tn} + 5V_{Tp}}{8} + \sqrt{(V_{DD} - V_{Tn} - V_{Tp})\left(\frac{V_{DD} - V_{Tn} - V_{Tp}}{4}\right)} \\ &= \frac{3V_{DD} + 5V_{Tn} + 5V_{Tp}}{8} + \frac{V_{DD} - V_{Tn} - V_{Tp}}{2} \\ &= \frac{7V_{DD} + V_{Tn} + V_{Tp}}{8} \end{aligned}$$

So

$$V_{oH} = \frac{7V_{DD} + V_{Tn} + V_{Tp}}{8} = V_{DD} - \frac{V_{DD} - V_{Tn} - V_{Tp}}{8} \tag{2.17}$$

Calculation of V_{iH} and V_{oL}

When the input is 'high', we should use eq.(2.14).

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_{DD} - V_{Tn} - V_{Tp})(2V_i - V_{DD} - V_{Tn} + V_{Tp})}$$

Differentiating with respect to V_i gives

$$\frac{\partial V_o}{\partial V_i} = -1 = 1 - \sqrt{\frac{V_{DD} - V_{Tn} - V_{Tp}}{2V_i - V_{DD} - V_{Tn} + V_{Tp}}} \tag{2.18}$$

$$\text{So} \qquad \frac{V_{DD} - V_{Tn} - V_{Tp}}{2V_{in} - V_{DD} - V_{Tn} + V_{Tp}} = 4$$

$$\text{Therefore} \qquad V_{DD} - V_{Tn} - V_{Tp} = 8V_{in} - 4V_{DD} - 4V_{Tn} + 4V_{Tp}$$

From where, we get

$$V_{iH} = \frac{5V_{DD} + 3V_{Tn} - 5V_{Tp}}{8} \tag{2.19}$$

Substituting the value of V_{iH} for V_{in} in eq.2.14), we get

$$\begin{aligned}
V_{oL} &= \frac{5V_{DD} + 3V_{Tn} - 5V_{Tp}}{8} - V_{Tn} - \\
&\quad \sqrt{(V_{DD} - V_{Tn} - V_{Tp})\left(\frac{5V_{DD} + 3V_{Tn} - 5V_{Tp}}{4} - V_{DD} - V_{Tn} + V_{Tp}\right)} \\
&= \frac{5V_{DD} - 5V_{Tn} - 5V_{Tp}}{8} - \sqrt{(V_{DD} - V_{Tn} - V_{Tp})\left(\frac{V_{DD} - V_{Tn} - V_{Tp}}{4}\right)} \\
&= \frac{5V_{DD} - 5V_{Tn} - 5V_{Tp}}{8} - \frac{V_{DD} - V_{Tn} - V_{Tp}}{2} = \frac{V_{DD} - V_{Tn} - V_{Tp}}{8}
\end{aligned}$$

So

$$V_{oL} = \frac{V_{DD} - V_{Tn} - V_{Tp}}{8} \tag{2.20}$$

Calculation of Noise Margins

The high noise margin is given by

$$V_{oH} - V_{iH} = \frac{V_{DD} - V_{Tn} + 3V_{Tp}}{4} \tag{2.21}$$

Similarly, the Low noise margin is

$$V_{iL} - V_{oL} = \frac{V_{DD} + 3V_{Tn} - V_{Tp}}{4} \tag{2.22}$$

The two noise margins can be made equal by choosing equal values for V_{Tn} and V_{Tp}.

2.2.3 Dynamic Considerations

In this section, we analyze the dynamic behaviour of the inverter. For the calculation of rise and fall times, we shall assume that only one of the two transistors in the inverter is 'on'. (Notice that this is more conservative than the input high and low conditions determined by slope considerations in eq.2.19 and 2.16). We shall continue to use the simple model described at the beginning of this booklet.

Rise time

When the input is low, the n channel transistor is 'off', while the p channel transistor is 'on'. The equivalent circuit in this condition is shown in fig. 2.7. From Kirchoff's current law at the output node,

$$I_{dp} = C\frac{dV_o}{dt}$$

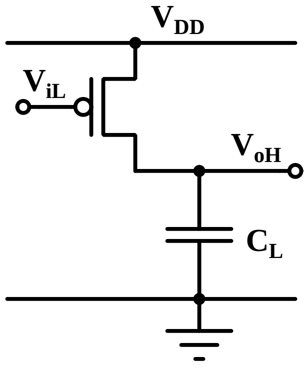


Figure 2.7: CMOS inverter with the nMOS 'off'

so,

$$\frac{dt}{C} = \frac{dV_o}{I_{dp}}$$

This separates the variables, with the LHS independent of operating voltages and the RHS independent of time. Integrating both sides, we get

$$\frac{\tau_{rise}}{C} = \int_0^{V_{oH}} \frac{dV_o}{I_{dp}}$$

Till the output rises to V_{iL}+ V_{Tp}, the p channel transistor is in saturation. Since the current is constant, the integration is trivial. If $V_{oH} > V_{iL} + V_{Tp}$ (which is normally the case), the integration range can be broken into saturation and linear regimes. Thus

$$\begin{aligned}\frac{\tau_{rise}}{C} &= \int_0^{V_{iL}+V_{Tp}} \frac{dV_o}{\frac{K_p}{2}(V_{DD}-V_{iL}-V_{Tp})^2} \\ &\quad + \int_{V_{iL}+V_{Tp}}^{V_{oH}} \frac{dV_o}{K_p\left[(V_{DD}-V_{iL}-V_{Tp})(V_{DD}-V_o)-\frac{1}{2}(V_{DD}-V_o)^2\right]}\end{aligned}$$

We define $V_1 \equiv V_{DD} - V_o$ and $V_2 \equiv V_{DD} - V_{iL} - V_{Tp}$, so $dV_o = -dV_1$.
We get

$$\frac{K_p\tau_{rise}}{2C} = \frac{V_{iL}+V_{Tp}}{V_2^2} - \int_{V_2}^{V_{DD}-V_{oH}} \frac{dV_1}{2V_1V_2 - V_1^2}$$

The integral can be evaluated as

$$\begin{aligned} I &\equiv -\int_{V_2}^{V_{DD}-V_{oH}} \frac{dV_1}{2V_1V_2 - V_1^2} \\ &= \frac{1}{2V_2}\int_{V_{DD}-V_{oH}}^{V_2} \left(\frac{1}{V_1} + \frac{1}{2V_2 - V_1}\right) dV_1\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2V_2}\left[\ln\frac{V_1}{2V_2 - V_1}\right]_{V_{DD}-V_{oH}}^{V_2} \\
&= \frac{1}{2V_2}\ln\frac{2V_2 - V_{DD} + V_{oH}}{V_{DD} - V_{oH}}
\end{aligned}
$$

Therefore,

$$\frac{K_p\tau_{rise}}{2C} = \frac{V_{iL} + V_{Tp}}{V_2^2} + \frac{1}{2V_2}\ln\frac{2V_2 - V_{DD} + V_{oH}}{V_{DD} - V_{oH}}$$

or

$$\frac{K_p\tau_{rise}}{2C} = \frac{V_{iL} + V_{Tp}}{(V_{DD} - V_{iL} - V_{Tp})^2} + \frac{1}{2(V_{DD} - V_{iL} - V_{Tp})}\ln\frac{2V_2 - V_{DD} + V_{oH}}{V_{DD} - V_{oH}}$$

Thus,

$$
\begin{aligned}
\tau_{rise} &= \frac{C(V_{iL} + V_{Tp})}{\frac{K_p}{2}(V_{DD} - V_{iL} - V_{Tp})^2} \\
&+ \frac{C}{K_p(V_{DD} - V_{iL} - V_{Tp})}\ln\frac{V_{DD} + V_{oH} - 2V_{iL} - 2V_{Tp}}{V_{DD} - V_{oH}} \qquad (2.23)
\end{aligned}
$$

The first term is just the constant current charging of the load capacitor. The second term represents the charging by the pMOS in its linear range. This can be compared with resistive charging, which would have taken a charge time of

$$\tau = RC\ln\frac{V_{DD} - V_{iL} - V_{Tp}}{V_{DD} - V_{oH}}$$

to charge from V_{iL}+ V_{Tp} to V_{oH}.

Fall time

When the input is high, the n channel transistor is 'on' and the p channel transistor is 'off'. If the output was initially 'high', it will be discharged to ground through the nMOS. To analysis the fall time, we apply Kirchoff's current law to the output node. This gives

$$I_{dn} = -C\frac{dV_o}{dt}$$

Again, separating variables and integrating from the initial voltage ($= V_{DD}$) to some terminal voltage V_{oL} gives

$$\frac{\tau_{fall}}{C} = -\int_{V_{DD}}^{voL}\frac{dV_o}{I_{dn}}$$

The n channel transistor will be in saturation till the output voltage falls to V_i- V_{Tn}. Below this voltage, the transistor will be in its linear regime. Thus, we can divide the integration

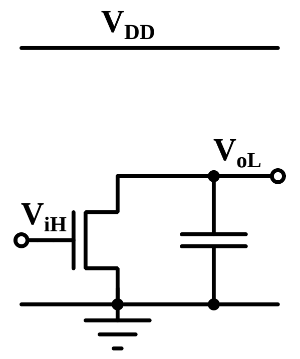


Figure 2.8: CMOS inverter with the pMOS 'off'

range in two parts.

$$\begin{aligned}
\frac{\tau_{fall}}{C} &= -\int_{V_{DD}}^{V_i - V_{Tn}} \frac{dV_o}{I_{dn}} - \int_{V_i - V_{Tn}}^{V_{oL}} \frac{dV_o}{I_{dn}} \\
&= \int_{V_i - V_{Tn}}^{V_{DD}} \frac{dV_o}{\frac{K_n}{2}(V_i - V_{Tn})^2} \\
&\quad + \int_{V_{oL}}^{V_i - V_{Tn}} \frac{dV_o}{K_n\left[(V_i - V_{Tn})V_o - \frac{1}{2}V_o^2\right]}
\end{aligned}$$

Therefore

$$\begin{aligned}
\frac{K_n \tau_{fall}}{2C} &= \frac{V_{DD} - V_i + V_{Tn}}{(V_i - V_{Tn})^2} + \int_{V_{oL}}^{V_i - V_{Tn}} \frac{dV_o}{2V_o(V_i - V_{Tn}) - V_o^2} \\
&= \frac{V_{DD} - V_i + V_{Tn}}{(V_i - V_{Tn})^2} + \frac{1}{2(V_i - V_{Tn})} \int_{V_{oL}}^{V_i - V_{Tn}} dV_o \left(\frac{1}{V_o} + \frac{1}{2(V_i - V_{Tn}) - V_o} \right)
\end{aligned}$$

Which gives

$$\begin{aligned}
\frac{K_n \tau_{fall}}{2C} &= \frac{V_{DD} - V_i + V_{Tn}}{(V_i - V_{Tn})^2} + \frac{1}{2(V_i - V_{Tn})} \left[\ln \frac{V_o}{2(V_i - V_{Tn}) - V_o} \right]_{V_{oL}}^{V_i - V_{Tn}} \\
&= \frac{V_{DD} - V_i + V_{Tn}}{(V_i - V_{Tn})^2} + \frac{1}{2(V_i - V_{Tn})} \ln \frac{2(V_i - V_{Tn}) - V_{oL}}{V_{oL}}
\end{aligned}$$

and therefore

$$\tau_{fall} = \frac{C(V_{DD} - V_i + V_{Tn})}{\frac{K_n}{2}(V_i - V_{Tn})^2} + \frac{C}{K_n(V_i - V_{Tn})} \ln \frac{2(V_i - V_{Tn}) - V_{oL}}{V_{oL}} \tag{2.24}$$

Again, the first term represents the time taken to discharge at constant current in the saturation regime, whereas the second term is the quasi-resistive discharge in the linear regime.

2.2.4 Trade off between power, speed and robustness

As we scale technologies, we improve speed and power consumption. However, as we can see from the expression for noise margins, (eq 2.21 and eq 2.22) the noise margin becomes worse. We can improve noise margins by choosing relatively higher threshold voltages. However, this will reduce speeds. We could also increase V_{DD}- but that would increase power dissipation. Thus we have a trade off between power, speed and noise margins.

This choice is made much more complicated by process variations, because we have to design for the worst case.

2.2.5 CMOS Inverter Design Flow

The CMOS inverter forms the basis of most static CMOS logic design. More complex logic can be designed from it by simple thumb rules. A common (though not universal) design requirement is symmetric charge and discharge behaviour and equal noise margins for high and low logic values. This requires matched values of K_n and K_p and equal values of $V_{Tn}$and V_{Tp}. For a constant load capacitance, rise and fall times depend linearly on K_n and K_p. Thus it is a straightforward calculation to determine transistor geometries if speed requirements and technological parameters are given. However, as transistor geometries are made larger, self loading can become significant. We now have to model the load capacitance as

$$C_{Load} = C_{ext} + \alpha K_n$$

where we have assumed that $\beta = K_n/K_p$ is kept constant. α is a technological constant. We use the expressions for $K\tau/C$ which depend only on voltages. Once these values are calculated, the geometry can be determined.

In the extreme case, when self capacitance dominates the load capacitance, K/C becomes constant and τ becomes geometry independent. There is no advantage in using wider transistors in this regime to increase the speed. It is better to use multi-stage logic with tapered buffers in this regime. This will be discussed in the module on Logical Effort.

2.3 Conversion of CMOS Inverters to other logic

Once the basic CMOS inverter is designed, other logic gates can be derived from it. All logic gates will have a pull up circuit consisting of pMOS transistors and pull down circuits consisting of nMOS transistors. The more complex logic gates derived from the inverter design should continue to provide the following essential characteristics of CMOS sytle of logic:

1. The logic gate should not consume current when the input is not changing. This implies that at no time should the pull up and pull down circuits be simulataneously 'ON', providing a path from V$_{DD}$ to ground.

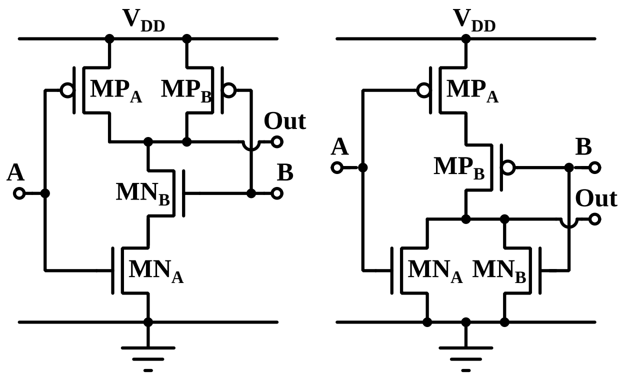


Figure 2.9: CMOS implementation of 2 input NAND and NOR gates

2. The logic gate output should be continuously driven to a '1' or a '0'. This means that one of the pull up and pull down networks must be 'ON' for any input combination. This ensures that if due to noise or leakage, the output voltage level changes, it will restored to V$_{DD}$ or ground by the pull up or pull down circuit which is 'ON'.

Apart from these essential properties, we would like to have the desirable property that the rise and fall times of the output are equal.

These conditions require that any input combination which produces a '0' at the output should turn the pull down network 'ON', and simultaneously ensure that the pull up network is 'OFF'. Similarly, for all input combinations for which the output is '1', the pull up circuit should be 'ON' and the pull down circuit should be OFF. Since the logic gate is derived from an inverter, the output should be expressed in a canonical form which has a bar (inversion) on top. Also, each input should drive an nMOS transistor in the pull down network and a pMOS transistor in the pull up network, to ensure that the essential requirements of CMOS logic are met.

Now consider a NAND gate. The output should be '0' only when both inputs are at '1'. Since a '1' at the input turns the nMOS transistors 'ON' and the pMOS transistors 'OFF', the nMOS transistors must be in series. Then the pull down circuit as a whole will be 'ON' only when both inputs are at '1'. Both inputs being at '1' ensures that both pMOS transistors are 'OFF', so the pull up circuit is off as required.

If either or both inputs are '0', the output should be '1'. To ensure this, the pull up circuit should be 'ON' while the pull down circuit should be 'OFF'. Either input being '0' will turn the corresponding nMOS transistor 'OFF'. The series connection of nMOS transistors is therefore suitable for this. To make sure that even if one input is at '0', the pull up circuit is

'ON', the pMOS transistors should be in parallel.

Similarly, for NOR logic, we can see that the nMOS transistors should be in parallel, while the pMOS transistors must be connected in series. Fig.2.9 shows the implementation of two input NAND and NOR gates.

For more complex logic, the logic expression shouldh be put in a cannonical form with a bar on top to indicate inversion and the expression should be a sum of products.

For every '.' in the expression, we put the corresponding n channel transistors in series and the corresponding p channel transistors in parallel. For every '+', we put the n channel transistors in parallel and the p channel transistors in series. This rule is known as the series-parallel rule.

Fig.2.10 shows the implementation of $\overline{A.B + C.(D + E)}$ in CMOS logic design style.

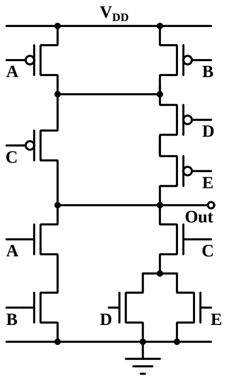


Figure 2.10: CMOS implementation of $\overline{A.B + C.(D + E)}$

Apart from the series-parallel connection, we have to decide how to determine the geometry of the transistors which are so connected. This is done by using what is known as the series-parallel rule. We scale the transistor widths up by the number of devices (n or p) put in series. The geometries are left untouched for devices put in parallel. One can justify this rule by analogy with on resistance of transistors. Two transistors in series will have the same on resistance as the single transistor in the inverter if each of the series transistors has half the on resistance of the inverter transistor. This can be done by making each of the series transistors twice as wide.

While at first sight it might appear that in case of parallel transistors, we can make do with half the width of the corresponding transistor in the inverter, it is not so. This is because parallel transistors must be ‘on’ whenever *either* of the transistors is ‘on’. Thus in the worst case, only one of the parallel transistors may be ‘on’ and this transistor must provide the same on resistance as the corresponding transistor in the inverter. Hence, for parallel transistors, we leave the geometry unchanged.

Chapter 3

Pseudo-nMOS Logic Design

CMOS design style ensures that the logic consumes no static power. This is because the pull down and pull up networks are never ‘on’ simultaneously. However, this requires that signals have to be routed to the n pull down network *as well as* to the p pull up network. This means that the load presented to every driver is high. This fact is exacerbated by the fact that n and p channel transistors cannot be placed close together as these are in different wells which have to be kept well separated in order to avoid latch-up.

Pseudo nMOS design style reduces dynamic power (by reducing capacitive loading) at the cost of having non-zero static power by replacing the pull up network by a single pMOS transistor with its gate terminal grounded. The pseudo nMOS inverter is shown below. Notice

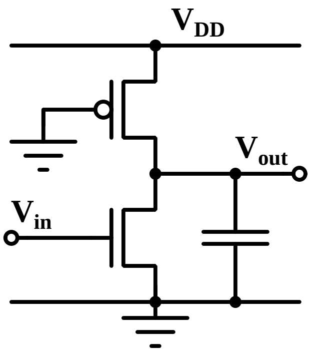


Figure 3.1: Pseudo nMOS inverter

that since the pMOS is not driven by signals, it is always ‘on’. The effective gate voltage seen by the pMOS transistor is V_{DD}. Thus the over-voltage on the p channel gate is always V_{DD}- V_{Tp}. This transistor will be saturated for output voltage $\leq V_{Tp}$ and will be in the linear regime for output voltage $\geq V_{Tp}$.

When the nMOS is turned 'on', a direct current path exists between supply and ground and so, static power will be dissipated.

3.1 Static Characteristics

As we sweep the input voltage from ground to V_{DD}, we encounter the following regimes of operation:

- For $V_i \leq V_{Tn}$, nMOS is 'off'.
- For low input voltage ($> V_{Tn}$), nMOS is saturated, pMOS is in linear regime.
- As V_i is raised further, V_o continues to fall. Eventually we enter the regime where nMOS is linear and pMOS is also linear.
- Finally, as we keep raising V_i, the output falls below V_{Tp}, provided the nMOS transistor is sufficiently wide. Now the nMOS is in linear regime, while pMOS is saturated.

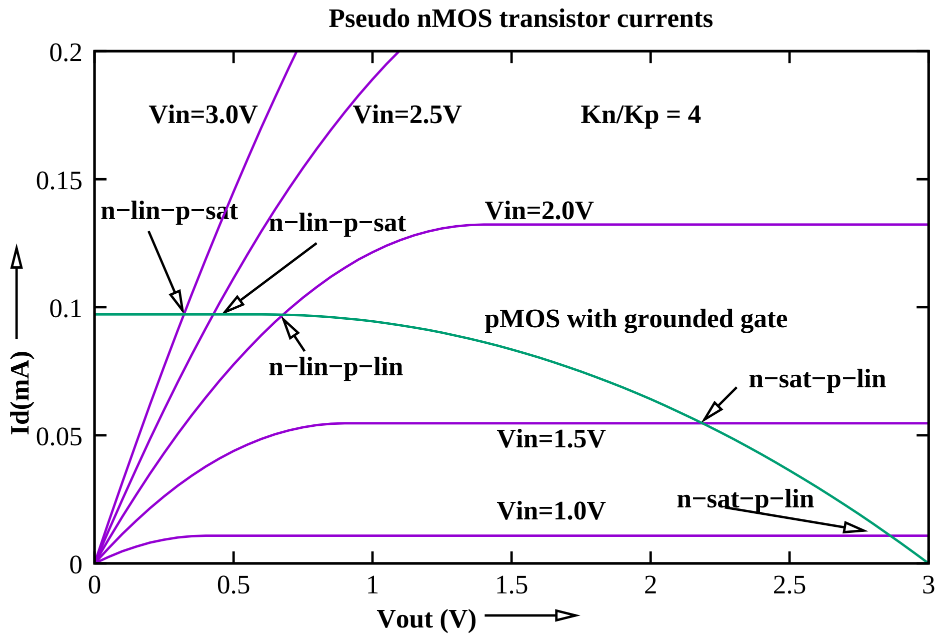


Figure 3.2: Transistor currents in a Pseudo nMOS inverter

3.1.1 For $0 \leq V_i \leq V_{Tn}$: nMOS 'off'

When the input voltage is less than V_{Tn}, the n channel transistor is off. Since no current flows through the p channel transistor, its drain-source voltage must be zero. Thus the output is 'high' (and $= V_{DD}$). No static power is dissipated in this condition.

3.1.2 nMOS saturated, pMOS linear

As the input voltage is raised above V_{Tn}, we enter this region. At this stage, the output voltage is close to V_{DD} (so more than $V_i - V_{Tn}$). Therefore the n channel transistor will be in saturation. The output voltage is also much higher than V_{Tp}, so the p channel transistor is in linear mode of operation. Equating currents through the n and p channel transistors, we get

$$\frac{K_n}{2}(V_i - V_{Tn})^2 = K_p\left[(V_{DD} - V_{Tp})(V_{DD} - V_o) - \frac{1}{2}(V_{DD} - V_o)^2\right] \tag{3.1}$$

defining $\beta \equiv K_n/K_p, V_1 \equiv V_{DD} - V_o$ and $V_2 \equiv V_{DD} - V_{Tp}$, we get

$$\frac{1}{2}V_1^2 - V_2V_1 + \frac{\beta}{2}(V_i - V_{Tn})^2 = 0 \tag{3.2}$$

with solutions

$$V_1 = V_2 \pm \sqrt{V_2^2 - \beta(V_i - V_{Tn})^2}$$

substituting the values of V_1 and V_2 and choosing the sign which puts V_o in the correct range, we get

$$V_o = V_{Tp} + \sqrt{(V_{DD} - V_{Tp})^2 - \beta(V_i - V_{Tn})^2} \tag{3.3}$$

This expression is valid as long as the n channel transistor is in saturation and the p channel transistor is in linear regime. As we keep increasing the input voltage V_i, the output will keep falling. Eq.3.3 will no more be valid when either the output falls below V_{Tp}, so that the p channel transistor enters saturation, or it falls below V_i- V_{Tn}, so that the n channel transistor enters the linear regime.
The output will fall below V_{Tp} when

$$V_{DD} - V_{Tp} = \sqrt{\beta}(V_i - V_{Tn}) \qquad \text{or} \qquad V_i = V_{Tn} + \frac{V_{DD} - V_{Tp}}{\sqrt{\beta}}$$

On the other hand it will fall to $V_i - V_{Tn}$ when

$$V_i - V_{Tn} = V_{Tp} + \sqrt{(V_{DD} - V_{Tp})^2 - \beta(V_i - V_{Tn})^2}$$

This can be solved to show that nMOS will enter its linear regime when

$$V_i > V_{Tn} + \frac{V_{Tp} + \sqrt{V_{Tp}^2 + (\beta + 1)V_{DD}(V_{DD} - 2V_{Tp})}}{\beta + 1}. \tag{3.4}$$

For common values of parameters, this happens before pMOS enters saturation.

3.1.3 nMOS linear, pMOS linear

$$\text{for } V_i > V_{Tn} + \frac{V_{Tp} + \sqrt{V_{Tp}^2 + (\beta+1)V_{DD}(V_{DD} - 2V_{Tp})}}{\beta+1}.$$

the nMOS enters its linear mode of operation. At this point the output voltage is $> V_{Tp}$, so the pMOS transistor is also in its linear regime. This combination of nMOS as well as pMOS being in linear regime will continue as we increase V_i, till V_o falls to V_{Tp}.

To determine the output voltage when both transistors are in their linear regimes, We can again equate the nMOS and pMOS currents using the transistor current equations for linear regime. The solution is straightforward, though algebraically tedious.
Equating n and p channel transistor currents,

$$K_n\left[(V_i - V_{Tn})V_o - \frac{1}{2}V_o^2\right] = K_p\left[(V_{DD} - V_{Tp})(V_{DD} - V_o) - \frac{1}{2}(V_{DD} - V_o)^2\right] \tag{3.5}$$

Defining $\beta \equiv K_n/K_p$, we can write

$$\beta\left[(V_i - V_{Tn})V_o - \frac{1}{2}V_o^2\right] = V_{DD}^2 - V_{Tp}V_{DD} - V_oV_{DD} + V_oV_{Tp} - \frac{1}{2}(V_{DD}^2 + V_o^2 - 2V_{DD}V_o)$$

$$\text{or } \beta(V_i - V_{Tn})V_o - \frac{\beta}{2}V_o^2 = \frac{1}{2}V_{DD}^2 - V_{Tp}V_{DD} + V_{Tp}V_o - \frac{1}{2}V_o^2$$

This gives

$$\frac{\beta-1}{2}V_o^2 - (\beta(V_i - V_{Tn}) - V_{Tp})V_o + \frac{V_{DD}}{2}(V_{DD} - 2V_{Tp}) = 0 \tag{3.6}$$

Solving this quadratic will give the value of V_o.

$$V_o = \frac{\beta(V_i - V_{Tn}) - V_{Tp} \pm \sqrt{(\beta(V_i - V_{Tn}) - V_{Tp})^2 - (\beta-1)V_{DD}(V_{DD} - 2V_{Tp})}}{\beta-1}$$

To keep V_o in the correct operating range, and because V_o is a decreasing function of V_i , we choose the negative sign. Thus,

$$V_o = \frac{\beta(V_i - V_{Tn}) - V_{Tp} - \sqrt{(\beta(V_i - V_{Tn}) - V_{Tp})^2 - (\beta-1)V_{DD}(V_{DD} - 2V_{Tp})}}{\beta-1} \tag{3.7}$$

3.1.4 nMOS linear, pMOS saturated

As the input voltage is raised still further, the output voltage will reach and then fall below V_{Tp}. The nMOS continues in its linear regime, and now the pMOS transistor enters its saturation regime.

Equating currents, we get

$$K_n \left[(V_i - V_{Tn})V_o - \frac{1}{2}V_o^2 \right] = \frac{K_p}{2}(V_{DD} - V_{Tp})^2$$

which gives

$$\frac{1}{2}V_o^2 - (V_i - V_{Tn})V_o + \frac{(V_{DD} - V_{Tp})^2}{2\beta}$$

This can be solved to get

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_i - V_{Tn})^2 - (V_{DD} - V_{Tp})^2/\beta} \tag{3.8}$$

3.2 Noise margins

As in the case of CMOS inverter, we find points on the transfer curve where the slope is -1.

When the input is low and output high, we should use Eq(3.3). Differentiating this equation with respect to V_i and setting the slope to -1, we get

$$V_{iL} = V_{Tn} + \frac{V_{DD} - V_{Tp}}{\sqrt{\beta(\beta + 1)}} \tag{3.9}$$

and

$$V_{oH} = V_{Tp} + \sqrt{\frac{\beta}{\beta + 1}}\,(V_{DD} - V_{Tp}) \tag{3.10}$$

When the input is high and the output low, we use Eq(3.8). Again, differentiating with respect to V_i and setting the slope to -1, we get

$$V_{iH} = V_{Tn} + \frac{2}{\sqrt{3\beta}}\,(V_{DD} - V_{Tp}) \tag{3.11}$$

and

$$V_{oL} = \frac{(V_{DD} - V_{Tp})}{\sqrt{3\beta}} \tag{3.12}$$

To make the output 'low' value lower than V_{Tn}, we get the condition

$$\beta > \frac{1}{3}\left(\frac{V_{DD} - V_{Tp}}{V_{Tn}}\right)^2$$

This condition on values of β places a requirement on the ratios of widths of n and p channel transistors. The logic gates work properly only when this equation is satisfied. Therefore this kind of logic is also called 'ratioed logic'. In contrast, CMOS logic is called ratioless logic because it does not place any restriction on the ratios of widths of n and p channel transistors for static operation. The noise margin for pseudo nMOS can be determined easily from the expressions for V_{iL}, V_{oL}, V_{iH}, V_{oH}.

3.3 Dynamic characteristics

In the sections above, we have derived the behaviour of a pseudo nMOS inverter in static conditions. In the sections below, we discuss the dynamic behaviour of this inverter.

3.3.1 Rise Time

When the input is low and the output rises from 'low' to 'high', the nMOS is off. The situation is identical to the charge up condition of a CMOS gate with the pMOS being biased with its gate at 0V. This gives

$$\tau_{rise} = \frac{C}{K_p(V_{DD} - V_{Tp})}\left[\frac{2V_{Tp}}{V_{DD} - V_{Tp}} + \ln\frac{V_{DD} + V_{oH} - 2V_{Tp}}{V_{DD} - V_{oH}}\right] \tag{3.13}$$

3.3.2 Fall Time

Analytical calculation of fall time is complicated by the fact that the pMOS load continues to dump current in the output node, even as the nMOS tries to discharge the output capacitor.

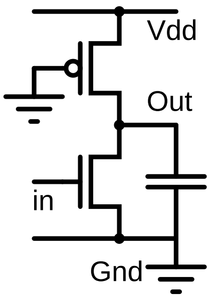


Figure 3.3: 'high' to 'low' transition on the output

Thus the nMOS should sink the discharge current as well as the drain current of the pMOS transistor. We make the simplifying assumption that the pMOS current remains constant at its saturation value through the entire discharge process. (This will result in a slightly pessimistic value of discharge time). Then,

$$I_p = \frac{K_p}{2}(V_{DD} - V_{Tp})^2$$

. We can write the KCL equation at the output node as:

$$I_n - I_p + C\frac{dV_o}{dt} = 0$$

which gives

$$\frac{\tau_{fall}}{C} = -\int_{V_{DD}}^{V_{oL}} \frac{dV_o}{I_n - I_p}$$

We define $V_1 \equiv V_i - V_{Tn}$ and $V_2 \equiv V_{DD} - V_{Tp}$. The integration range can be divided into two regimes. nMOS is saturated when $V_1 \le V_o < V_{DD}$ and is in linear regime when $V_{oL} < V_o < V_1$. Therefore,

$$\frac{\tau_{fall}}{C} = -\int_{V_{DD}}^{V_1} \frac{dV_o}{\frac{1}{2}K_n V_1^2 - I_p} - \int_{V_1}^{V_{oL}} \frac{dV_o}{K_n(V_1 V_o - \frac{1}{2}V_o^2) - I_p}$$

$$\text{So } \frac{K_n \tau_{fall}}{2C} = \frac{V_{DD} - V_1}{V_1^2 - 2I_p/K_n} + \int_{V_{oL}}^{V_1} \frac{dV_o}{2V_1 V_o - V_o^2 - 2I_p/K_n}$$

We define $V_2^2 \equiv 2I_p/K_n$. The denominator of the integral term can then be factorized by adding and subtracting V_1^2

$$\begin{aligned} 2V_1V_o - V_o^2 - V_1^2 + V_1^2 - V_2^2 &= -(V_1 - V_o)^2 + \left(\sqrt{V_1^2 - V_2^2}\right)^2 \\ &= \left(\sqrt{V_1^2 - V_2^2} + V_1 - V_o\right)\left(\sqrt{V_1^2 - V_2^2} - V_1 + V_o\right) \end{aligned}$$

The integral term can then be written as:

$$\frac{1}{2\sqrt{V_1^2 - V_2^2}}\left(\int_{V_{oL}}^{V_1} \frac{dV_o}{\sqrt{V_1^2 - V_2^2} + V_1 - V_o} + \int_{V_{oL}}^{V_1} \frac{dV_o}{\sqrt{V_1^2 - V_2^2} - (V_1 - V_o)}\right)$$

The integration over V_o can now be carried out to get

$$\frac{1}{2\sqrt{V_1^2 - V_2^2}} \ln \frac{\sqrt{V_1^2 - V_2^2} - (V_1 - V_o)}{\sqrt{V_1^2 - V_2^2} + V_1 - V_o}\Bigg|_{V_{oL}}^{V_1}$$

On putting the limits for the integral, the upper limit gives 0. So the integral can be written as

$$\frac{1}{2\sqrt{V_1^2 - V_2^2}} \ln \frac{\sqrt{V_1^2 - V_2^2} + V_1 - V_{oL}}{\sqrt{V_1^2 - V_2^2} - (V_1 - V_{oL})}$$

Thus we can write down the expression for fall time as:

$$\frac{K_n \tau_{fall}}{2C} = \frac{V_{DD} - V_1}{V_1^2 - 2I_p/K_n} + \frac{1}{2\sqrt{V_1^2 - V_2^2}} \ln \frac{\sqrt{V_1^2 - V_2^2} + V_1 - V_{oL}}{\sqrt{V_1^2 - V_2^2} - (V_1 - V_{oL})}$$

Substituting back for V_2^2, we get

$$\frac{K_n\tau_{fall}}{2C} = \frac{V_{DD}-V_1}{V_1^2-2I_p/K_n} + \frac{1}{2\sqrt{V_1^2-2I_p/K_n}}\ln\frac{\sqrt{V_1^2-2I_p/K_n}+V_1-V_{oL}}{\sqrt{V_1^2-2I_p/K_n}-(V_1-V_{oL})}$$

Since $I_p = K_p(V_{DD}-V_{Tp})^2/2$, $2I_p/K_n$ is just $(V_{DD}-V_{Tp})^2/\beta$.

$$\frac{K_n\tau_{fall}}{2C} = \frac{V_{DD}-V_1}{V_1^2-2I_p/K_n} + \frac{1}{2\sqrt{V_1^2-2I_p/K_n}}\ln\frac{\sqrt{V_1^2-2I_p/K_n}+V_1-V_{oL}}{\sqrt{V_1^2-2I_p/K_n}-(V_1-V_{oL})} \qquad (3.14)$$

$$\text{Where}\quad 2I_p/K_n = \frac{(V_{DD}-V_{Tp})^2}{\beta} \quad\text{and}\quad V_1 = V_i - V_{Tn}$$

This relation was derived using the pessimistic assumption that the p channel transistor dumps its saturation current over the entire discharge range.

In any case, this relation is not used for designing the inverter because the limit on the size of the n channel transistor is put by static considerations and not by the fall time.

3.4 Pseudo nMOS design Flow

We design the basic inverter first and then map the inverter design to other logic circuits. The load device size is calculated from the rise time. From Eq. 3.13 we have

$$\tau_{rise} = \frac{C}{K_p(V_{DD}-V_{Tp})}\left[\frac{2V_{Tp}}{V_{DD}-V_{Tp}} + \ln\frac{V_{DD}+V_{oH}-2V_{Tp}}{V_{DD}-V_{oH}}\right]$$

Given a value of τ_{rise}, operating voltages and technological constants, K_p and hence, the geometry of the p channel transistor can be determined.

Geometry of the n channel transistor in the reference inverter design can be determined from static considerations. Using Eq. 3.8, the output 'low' level is given by:

$$V_o = (V_i - V_{Tn}) - \sqrt{(V_i-V_{Tn})^2 - (V_{DD}-V_{Tp})^2/\beta}$$

If the desired value of the output 'low' level is given, we can calculate β. But $\beta \equiv K_n/K_p$ and K_p is already known. This evaluates K_n and hence, the geometry of the n channel transistor.

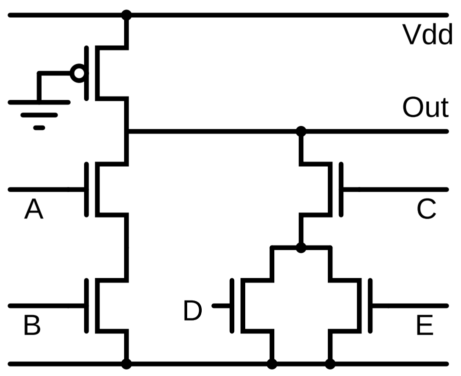


Figure 3.4: Pseudo NMOS implementation of $\overline{A.B + C.(D + E)}$

3.5 Conversion of pseudo nMOS Inverter to other logic

Once the basic pseudo nMOS inverter is designed, other logic gates can be derived from it. The procedure is the same as that for CMOS, except that it is applied only to nMOS transistors. The p channel transistor is kept at the same size as that for an inverter.

The logic is expressed as a sum of products with a bar (inversion) on top. For every '.' in the expression, we put the corresponding n channel transistors in series and for every '+', we put the n channel transistors in parallel. We scale the transistor widths up by the number of devices put in series. The geometries are left untouched for devices put in parallel. Fig.3.4 shows the implementation of $\overline{A.B + C.(D + E)}$ in pseudo NMOS logic design style.

Chapter 4

Dual Rail Logic Design

4.1 Complementary Pass gate Logic

This logic family is based on multiplexer logic and makes use of Shannon's Boolean expansion theorem.

Given a Boolean function F(x1, x2, ..., xn), we can express it as:

$$F(x1, x2, \ldots, xn) = xi \cdot f1 + \overline{xi} \cdot f2$$

where f1 and f2 are reduced expressions for F with xi forced to '1' and '0' respectively. Thus, F can be implemented with a multiplexer controlled by xi which selects f1 or f2 depending on xi. f1 and f2 can themselves be decomposed into simpler expressions by the same technique.

To implement a multiplexer, we need both xi and $\overline{xi}$. Therefore, this logic family needs all inputs in true as well as in complement form. In order to drive other gates of the same type, it must produce the outputs also in true and complement forms. Thus each signal is carried by two wires. This logic style is called "Complementary Pass-gate Logic" or CPL for short.

4.1.1 Basic Multiplexer Structure

Pure pass-gate logic contains no 'amplifying' elements. Therefore, it has zero or negative noise margin. (Each logic stage degrades the logic level). Therefore, multiple logic stages cannot be cascaded. We shall assume that each stage includes conventional CMOS inverters to restore the logic level. Ideally, the multiplexer should be composed of complementary pass gate transistors. However, we shall use just n channel transistors as switches for simplicity. This gives us the multiplexer structure shown in fig.4.1.

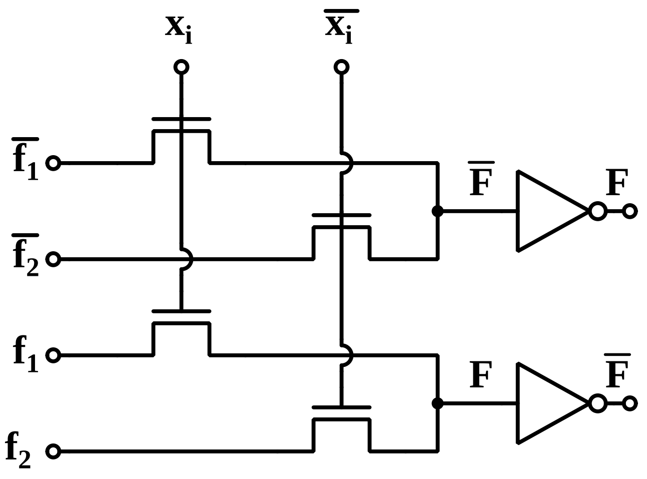


Figure 4.1: Basic Multiplexer with logic restoring inverters

4.1.2 Logic Design using CPL

Since both true and complement outputs are generated by CPL, we do not need separate gates for AND and NAND functions. The same applies to OR-NOR, and XOR-XNOR functions.

To take an example, let us consider the XOR-XNOR functions. Because of the inverter,

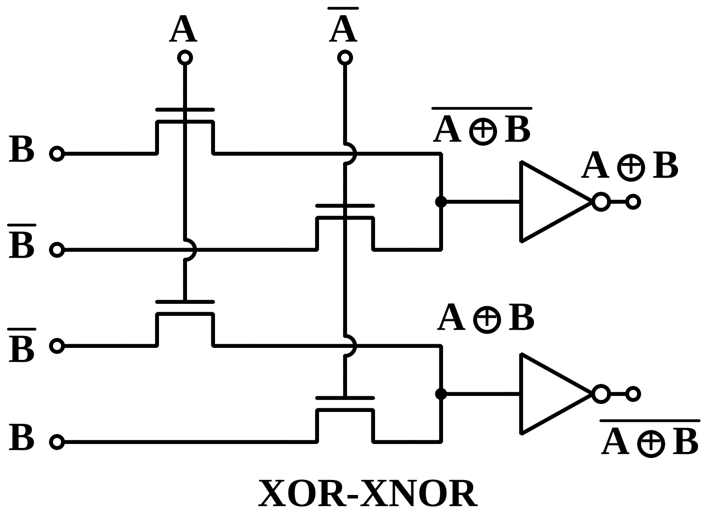


Figure 4.2: Implementation of XOR and XNOR by CPL logic.

the multiplexer for the XOR output first calculates the XNOR function given by $A.B + \overline{A}.\overline{B}$.

If we put $A = `1'$, this reduces to $B$ and for $A = `0'$, it reduces to $\overline{B}$. Similarly, for the XNOR output, we generate the XOR expression $= A.\overline{B} + \overline{A}.B$ which will be inverted by the logic level restoring inverter. The expression reduces to $\overline{B}$ for $A = `1'$ and to $B$ for $A = `0'$. This leads to an implementation of XOR-XNOR as shown in fig.4.2

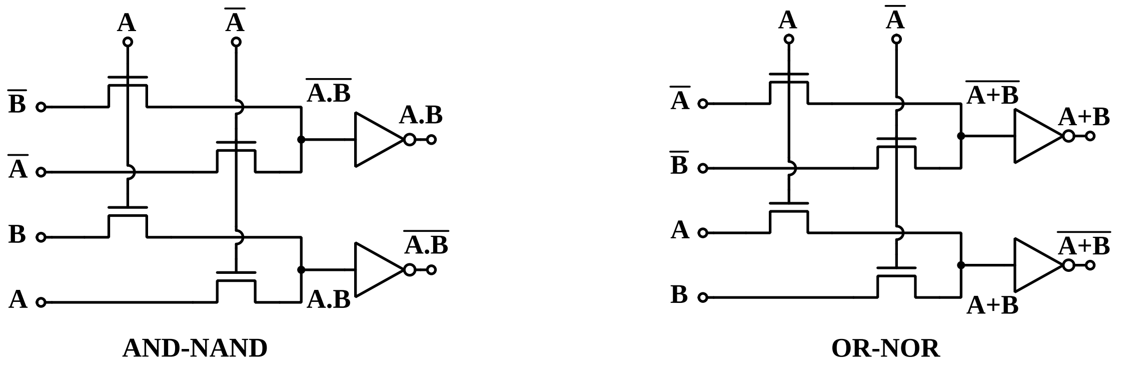


Figure 4.3: Implementation of (a) AND-NAND and (b) OR-NOR functions using complementary pass-gate logic.

Implementation of AND and OR functions is similar. In case of AND, the multiplexer should output $\overline{A.B}$ to be inverted by the buffer. This reduces to $\overline{B}$ when A ='1'. When A = '0', it evaluates to $1 = \overline{A}$. For NAND output, the multiplexer should output A.B, which evaluates to B for A = '1' and to '0' (or A) when A = '0'.

4.1.3 Buffer Leakage Current

The circuit configuration described above uses nMOS multiplexers. This limits the 'high'

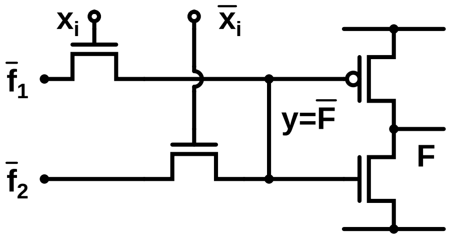


Figure 4.4: High leakage current in inverter

output of the multiplexer (node y - which is the input for the inverter) to V_{DD} - V_{Tn}. Consequently, the pMOS transistor in the buffer inverter never quite turns off. This results in static power consumption in the inverter. This can be avoided by adding a pull up pMOS as shown

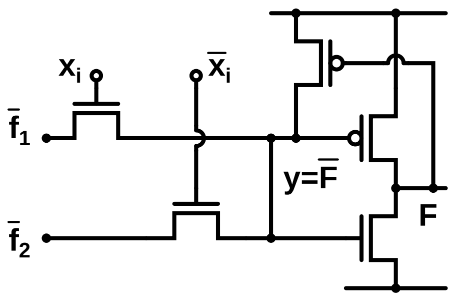


Figure 4.5: Pull up pMOS to avoid leakage in the inverter

in fig. 4.5. When the multiplexer output (y) is ‘low’, the inverter output is high. The pMOS is therefore off and has no effect. When the multiplexer output goes ‘high’, the inverter input charges up, the output starts falling and turns the pMOS on. Now, as the multiplexer output (y) approaches V_{DD} - V_{Tn}, the nMOS switch in the multiplexer turn off. However, the pMOS pull up remains ‘on’ and takes the inverter input all the way to V_{DD}. This avoids leakage in the inverter.

However, this solution brings up another problem. Consider the equivalent circuit when the inverter output is ‘low’ and the pMOS is ‘on’. Now if the multiplexer output wants to go

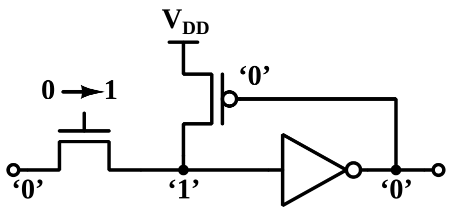


Figure 4.6: Problem with a low to high transition on the output

‘low’, it has to fight the pMOS pull-up - which is trying to keep this node ‘high’.

In fact, the multiplexer n transistor and the pull up p transistor constitute a pseudo nMOS inverter. Therefore, the multiplexer output cannot be pulled low unless the transistor geometries are appropriately ratioed.

4.2 Cascade Voltage Switch Logic

We can understand this logic configuration as an attempt to improve pseudo-nMOS logic circuits. Consider the NOR gate shown below: Static power is consumed by this NOR circuit

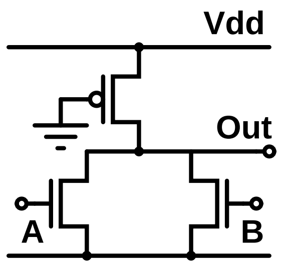


Figure 4.7: Pseudo-nMOS NOR

whenever the output is 'LOW'. This happens when A OR B is TRUE. We wish that the pMOS could be turned off for just this combination of inputs.

To turn the pMOS transistor off, we need to apply a 'HIGH' voltage level to its gate whenever A OR B is true. This obviously requires an OR gate. Non-inverting gates cannot be made in a single stage. However, We can create the OR function by using a NAND of $\overline{A}$ and $\overline{B}$ as shown in figure 4.8. But then what about the pMOS drive of this circuit?

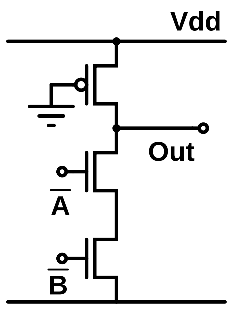


Figure 4.8: Pseudo-nMOS OR from complemented inputs

We want to turn the pMOS of this OR circuit off when both $\overline{A}$ and $\overline{B}$ are 'HIGH'; i.e. when A = B = 0. This means we would like to turn the pMOS of this circuit off when the NOR of A and B is 'TRUE'.

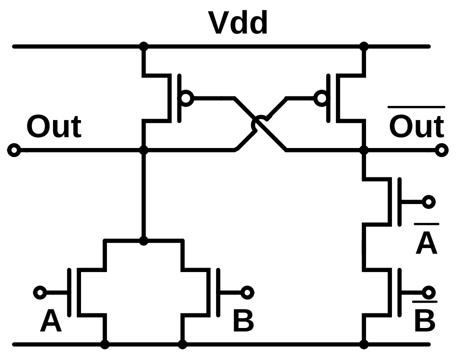


Figure 4.9: OR-NOR implementation in Cascade Voltage Switch Logic

But we already have this signal as the output of the first (NOR) circuit! So the two circuits can drive each other's pMOS transistors and avoid static power consumption. This kind of logic is called Cascade Voltage Switch Logic (CVSL). It can use any network f and its complementary network $\overline{f}$ in the two cross-coupled branches. The complementary network is constructed by changing all series connections in f to parallel and all parallel connections to series, and complementing all input signals.

CVSL shares many characteristics with static CMOS, CPL and pseudo-nMOS.

- Like CMOS static logic, there is no static power consumption.
- Like CPL, this logic requires both True and Complement signals. It also provides both True and complement outputs. (Dual Rail Logic).
- Like pseudo nMOS, the inputs present a single transistor load to the driving stage.
- The circuit is self latching. This reduces ratioing requirements.

Chapter 5

Dynamic Logic

5.1 Dynamic Logic

In this style of logic, some nodes are required to hold their logic value as a charge stored on a capacitor. These nodes are not connected to their 'drivers' permanently. The 'driver' places the logic value on them, and is then disconnected from the node. Due to leakage etc., the logic value cannot be held indefinitely. Dynamic circuits therefore require a *minimum* clock frequency to operate correctly. Use of dynamic circuits can reduce circuit complexity and power consumption substantially, compared to pseudo NMOS.

5.2 Basic CMOS Dynamic Gate

When the clock is low, pMOS is on and the bottom nMOS is off. The output is 'pre-charged' to '1' unconditionally. When the clock goes high, the pMOS turns off and the bottom nMOS comes on. The circuit then conditionally discharges the output node, if (A+B).C is TRUE. This implements the function $\overline{(A+B).C}$.

5.2.1 Problem with Cascading CMOS dynamic logic

The dynamic circuit described above can run into problems when several dynamic gates are cascaded. Consider the case when the circuit described in Fig. 5.1 is followed by an inverter. Let us take two cases – when (A+B).C is FALSE and when it is TRUE.

When (A+B).C is FALSE, There is no problem X pre-charges to '1' and remains at '1'. Therefore the inverter sees the correct logic value at its input all the time and discharges the output to '0', which is the expected output.

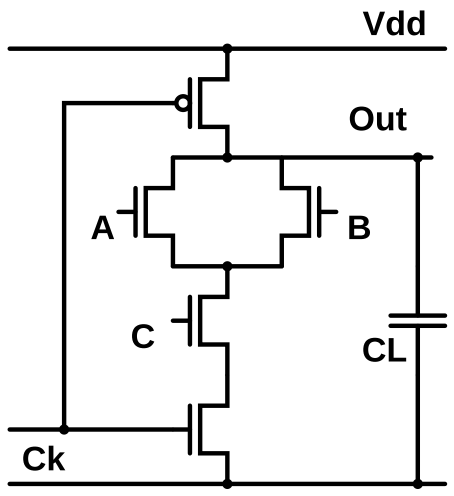


Figure 5.1: CMOS dynamic gate to implement $\overline{(A+B).C}$.

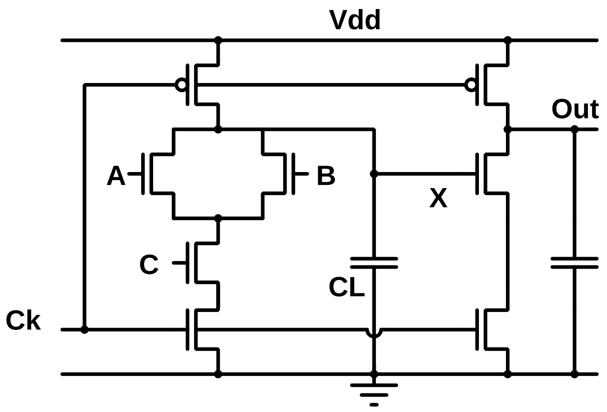
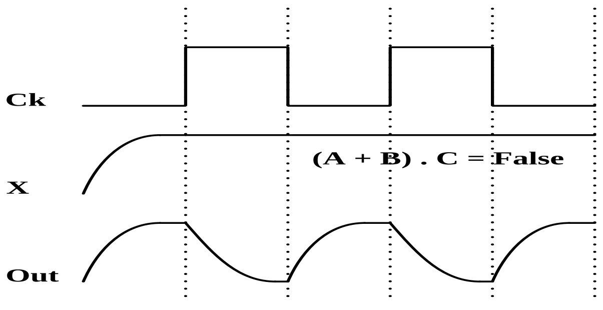
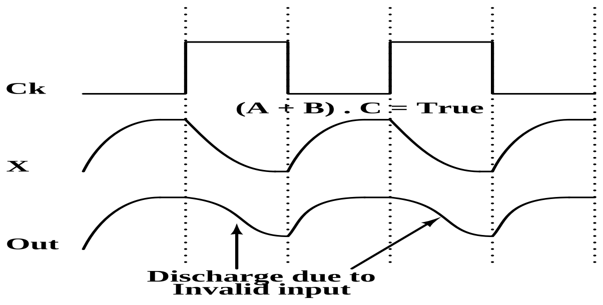


Figure 5.2: Dynamic gate for $\overline{(A+B).C}$ cascaded with an inverter.

However, When (A+B).C is TRUE, there is a problem.
The correct value for X is now '0'. After the pre-charge cycle, X takes some time to discharge to '0'. So for some time after pre-charge, its output is held at the wrong value of '1'. During this time, the charge placed on the output capacitor of inverter is leaked away to ground as the input to nMOS of the inverter is not '0' and the clock input is high. This can lead to a wrong evaluation of the logic function!

In a dynamic logic stage, the output is pre-charged to '1'. If the final output is sup-





posed to be ‘0’, there will be some time during which the output will still be at the wrong value of ‘1’. This transiently wrong value can discharge the pre-charged output capacitor of the next stage and thus lead to malfunction. So we need to isolate the output of a dynamic logic stage from the input of the next dynamic logic stage till it has acquired the correct value.

This means the operation of dynamic logic should proceed in distinct time slots or ‘phases’. The output of the stage should be connected to the input of the next stage only in the phases in which its output is valid and stable. So the sequence of operations should be:

- Pre-charge the output in the first phase.
- Disable pre-charging and carry out logic evaluation in the next stage. The Output should be disconnected from the next stage during pre-charge as well as evaluation phases.
- In the final phase(s), the output holds its correct value. In this phase, connect the output to the next stage and disconnected it from pre-charge and evaluate circuits of the current stage.
- Thus, a minimum of 3 phases are required – pre-charge, evaluate and output-valid.
- In a 4-phase implementation, the valid state holds for a duration of two phases instead of just one.

- To implement this kind of dynamic logic, we need a multi-phase clock.

An example scheme for generating a 4 phase clock is shown below.
The nor gate inserts a 1 in the shift register whenever it sees three 0's at Q0, Q1 and Q2.

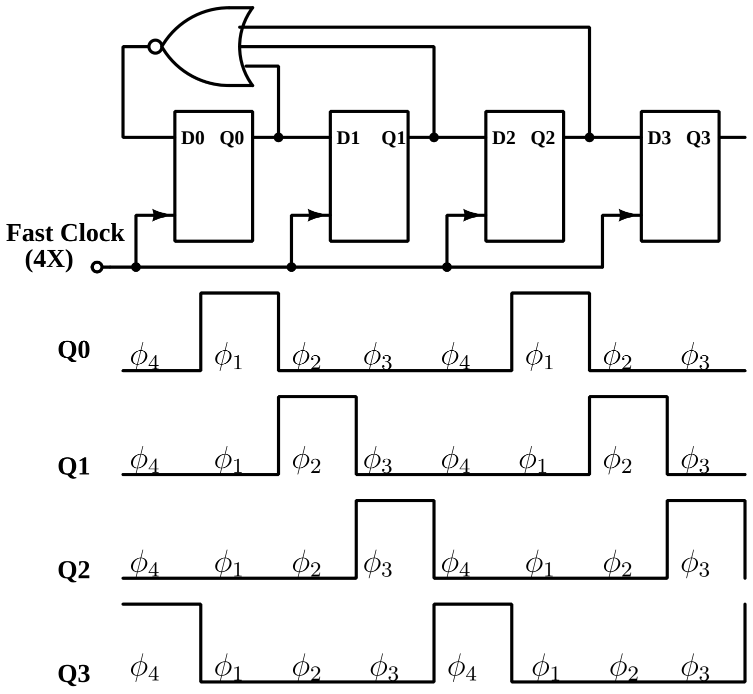


Figure 5.3: Generation of phase clocks for dynamic logic

If any of these is a 1, it inserts 0's. Thus, exactly one D flipflop among the four holds a 1 at any time, all the rest hold 0. Thus, each Q output is high during a single phase of a 4 phase cycle.

5.3 Four Phase Dynamic Logic

As discussed above, We use different phases for pre-charge, evaluation and for holding a valid output.

For the circuit shown in Fig. 5.4, we have a 4 phase clock. Ck_{mn} is a clock which is high during the m and n phases of the clock. Similarly, $\overline{Ck_{mn}}$ is a signal which is low during m and n phases of the clock. Combined clock signals of the type Ck_{mn} can be generated easily using simple logic.

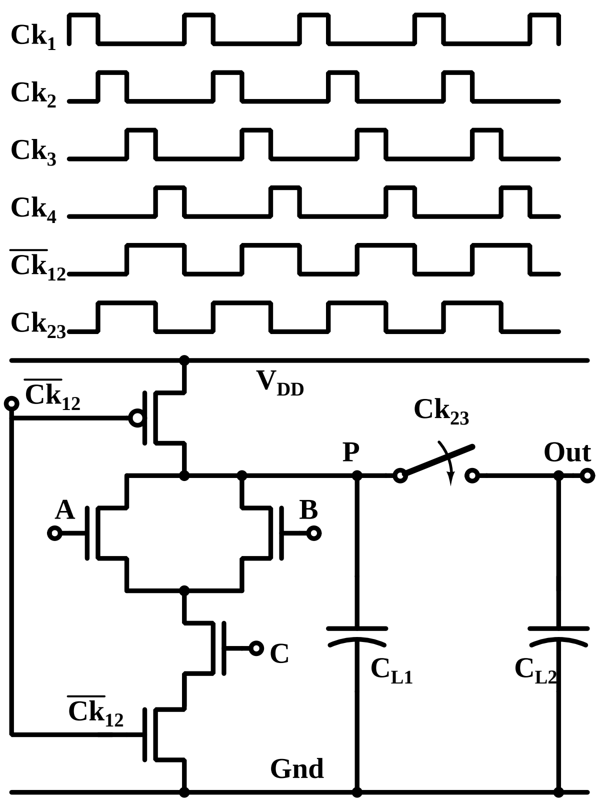


Figure 5.4: CMOS 4 phase dynamic logic

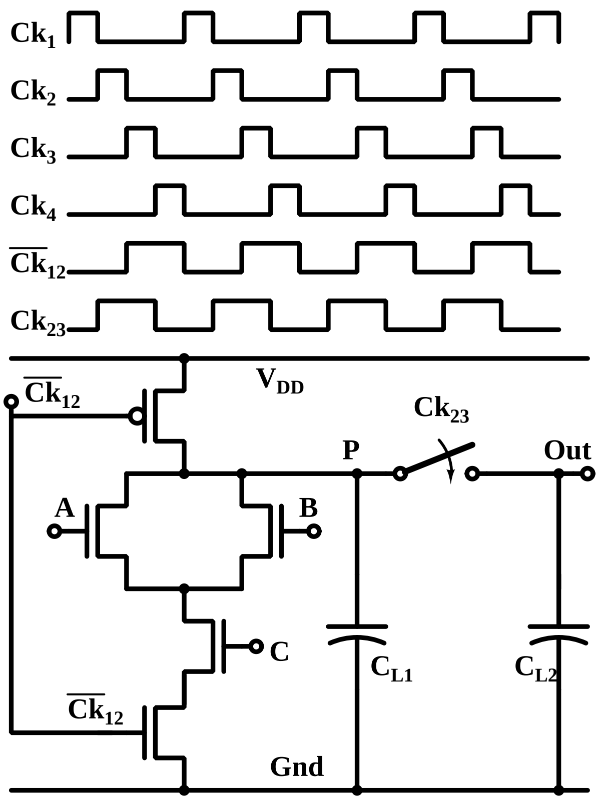
Now, for the gate shown in the figure, In phase 1, $\overline{Ck_{12}} = 0, Ck_{23} = 0$. So the pMOS is on and the bottom (clocked) nMOS is off. The output is disconnected from transistors. Node P pre-charges to '1' while the output holds its old value.

In phase 2, $\overline{Ck_{12}} = 0, Ck_{23} = 1$. So the pMOS is on, bottom (clocked) nMOS is off and the output capacitor is in parallel with the capacitor on node P. As a result, node P, as well as the output node pre-charge to 1.

In phase 3, $\overline{Ck_{12}} = 1, Ck_{23} = 1$. So the pre-charge pMOS is off and the bottom (clocked) nMOS is on. Capacitors at node P and at the output are still in parallel.

If $(A + B) \cdot C = 1$, both capacitors will discharge to ground through the signal transistors and the bottom (clocked) nMOS. Otherwise the output will remain at 1. Thus the gate evaluates in phase 3 and acquires the correct output value at the end of this phase.

In phase 4, $\overline{Ck_{12}} = 1, Ck_{23} = 0$. The output is isolated from transistors and will hold its



- Ck_{mn} is defined as a clock signal which is 'high' during phase m and phase n of the clock.
- Similarly, $\overline{\mathrm{Ck}_{mn}}$ is a clock signal which is 'low' during phase m and phase n of the clock.

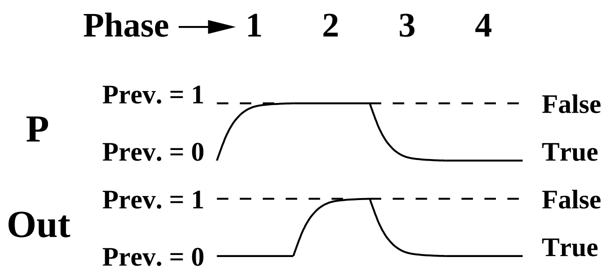


Figure 5.5: Operation of a type 3 gate

valid value. In phase 4 as well as in phase 1, the output is isolated from the driver and retains its valid value. However, Notice that node P no more holds a valid value in phase 1, since it pre-charges to '1' in this phase.

This is called a type 3 gate, and needs the inputs to be valid and stable in phase 3. By changing the clocks to $\overline{Ck_{23}}$ and Ck_{34}, we shall get a type 4 gate which evaluates in phase 4 and whose output remains valid in phases 1 and 2. Similarly, by cyclic permutation of clock signals, we can get type 1 and type 2 gates. A type 3 gate evaluates in phase 3 and is valid in phases 4 and 1. Similarly, we can have type 4, type 1 and type 2 gates.
The output of a type 3 gate is correct and stable in phases 4 and 1. Consequently, its output can be used by type 4 and type 1 gates without any malfunction.

Each logic gate type in 4-phase dynamic logic design, needs its inputs to be valid and stable during one phase and provides an output which is valid and stable for two clock phases. A type 3 gate can drive a type 4 or a type 1 gate. Similarly, type 4 will drive types 1 and 2; type 1 will drive types 2 and 3; and type 2 will drive types 3 and 4. We can use a 2 phase clock if we stick to type 1 and type 3 gates (or type 2 and type 4 gates) as these can drive

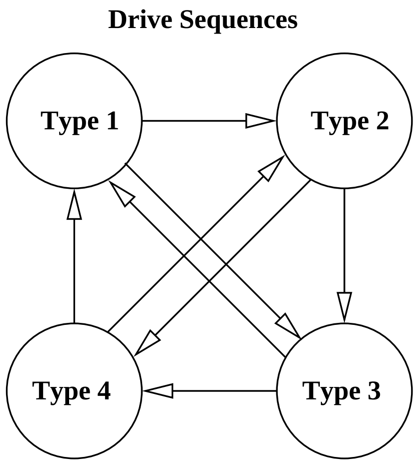


Figure 5.6: CMOS 4 phase dynamic logic drive constraints

each other.

5.4 Domino Logic

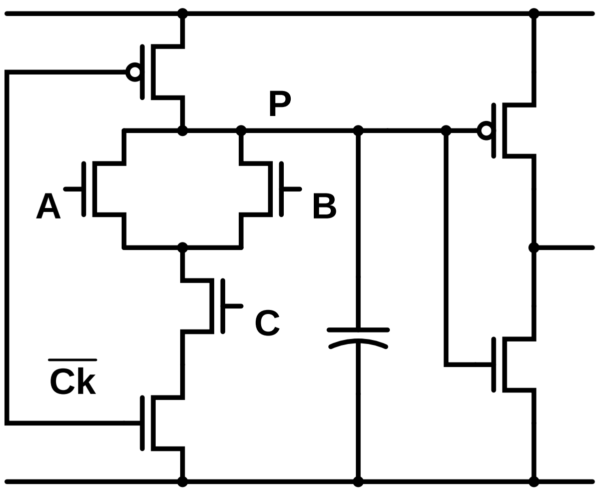


Figure 5.7: CMOS domino logic

Another way to eliminate the problem with cascading logic stages is to use a static inverter after the CMOS dynamic gate. Recall that cascading of dynamic CMOS stage causes problems because the output is pre-charged to V_{DD}. If the final value of a stage is meant to be zero, the next stage nMOS to which this output is connected erroneously sees a one till the pre-charged output is brought down to zero. During this time, it ends up discharging its own pre-charged output, which it was not supposed to do.

If an inverter is added, the output is held ‘low’ before logic evaluation. Now, if the final output of this gate is ‘0’, there is no problem anyway. If the final output was supposed be ’1’, the next stage is erroneously held at zero for some time. However, this does not result in a false evaluation by the next stage. The only effect it can have is that the next stage starts its evaluation a little later.

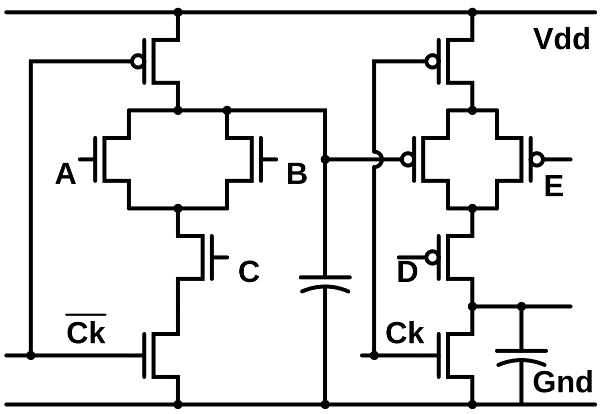
Domino logic is fast, because the pre-charge cycle is common to all stages. Once pre-charge is done, each stage evaluates one after the other (hence the name - domino logic). Thus for n stage logic, there is only one cycle of pre-charge and n cycles of evaluation, rather than n cycles of pre-charge and n of evaluation.

However, the addition of an inverter means that the logic is non-inverting. Therefore, it cannot be used to implement any arbitrary logic function. In synchronous digital circuits, combinational logic alternates with clocked latches. If inversion is required, we insert a latch at that place and take the $\overline{Q}$ output of the latch to the next group of domino logic.

5.5 Zipper logic

Instead of using an inverter, we can alternate n and p evaluation stages. Now each stage is inverting and any logic can be implemented using such stages. The constraint now is that an n stage can only drive a p stage, and a p stage can only drive an n stage. This kind of logic is called *zipper* logic, in analogy with a zipper where links from alternate sides are joined one after the other. The n stage is pre-charged ‘high’. If the output was supposed to be ‘high’, it is anyway correct and cannot cause a malfunction. If the eventual output was supposed to be ‘low’, this transiently wrong ‘high’ output will not harm the next stage whose evaluation transistors are p type. A ‘high’ output will keep the transistors off for some time and no charge will be leaked away from capacitors. When the stage reaches its correct output, the next stage will evaluate its output correctly.

Similarly, the p stage will be pre-discharged to ‘low’. If the output was supposed to be ‘low’, it is anyway correct and cannot cause a malfunction. If the eventual output was supposed to be ‘high’, this transiently wrong ‘low’ output will not harm the next stage whose evaluation transistors are n type. A ‘low’ output will keep the nMOS evaluation transistors of the next stage off for some time and no charge will be leaked away from capacitors. When the stage reaches its correct output, the next stage will evaluate its output correctly.



A, B, C must be from p stages.
D and E must be from n stages.

Figure 5.8: Zipper logic

Chapter 6

Circuits with Mixed Design Styles

Some commonly used circuits use a mixture of styles. The availability of transistor switches which can pass signals in either direction provides a unique flexibility to CMOS technology based designs. Mixing traditional CMOS logic gates with MOS transistor switches in this way leads to efficient implementation of many useful circuits. Examples of this class of circuits are transmission gates, tri-stateable inverters, multiplexers, tiny-xor circuits and D latches and flipflops.

6.1 Transmission gates

A single transistor used as a switch has limitations in passing rail to rail signals. An nMOS transistor is inefficient for passing a '1', while a pMOS transistor acting as a switch in inefficient for passing a '0'. However, if we use an nMOS and a pMOS in parallel, they provide a good transmission gate with a reasonable on resistance over the entire voltage range. Both the n

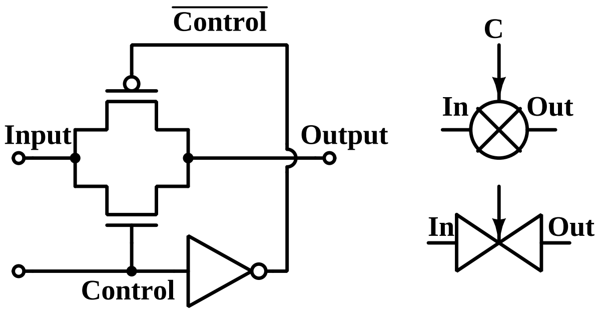


Figure 6.1: a transmission gate

and p channel transistor are on when the control input is high and both are off when it is low. The inverter is a CMOS inverter.

Notice that the input and output are interchangeable.

This circuit is represented by either of the symbols shown on the right.

6.2 Tri-stateable inverter

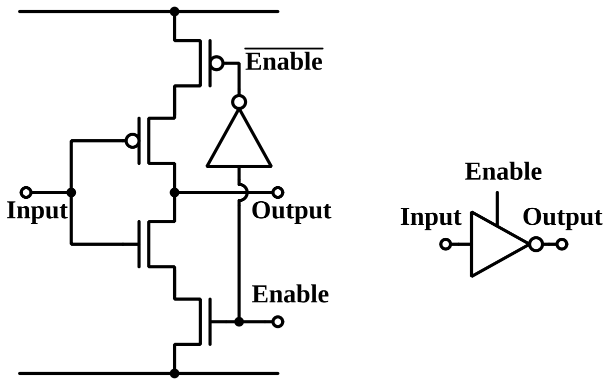


Figure 6.2: a tri-stateable inverter

We often need an inverter which can be put into a high impedance state using a control input. The high impedance state is the third state of the inverter, apart from producing a '0' or a '1' at the output. Such inverters are called tri-stateable inverters. When Enable = 0, both pull up and pull down are off. The inverter then presents a high impedance output.

When Enable = 1, power is supplied to the inner pMOS and nMOS transistors and the circuit acts like a CMOS inverter.

The symbol shown on the right is used to represent a tri-stateable inverter.

6.3 Multiplexers

By shorting the outputs of two tri-stateable inverters of which only one is active, we can implement an inverting mux. The enable signals of the two tri-stateable inverters are complementary. So only one of these is on at any given time. Depending on the state of the enable signal, the input to the enabled inverter appears at the output in an inverted form.

The two way multiplexer can be generalized to an n way multiplexer by adding a decoder, which generates Sel and $\overline{Sel}$ for each input, which in turn control a tri-stateable inverter each. The figure below shows a 4 way multiplexer. The Decoder provides outputs such that only

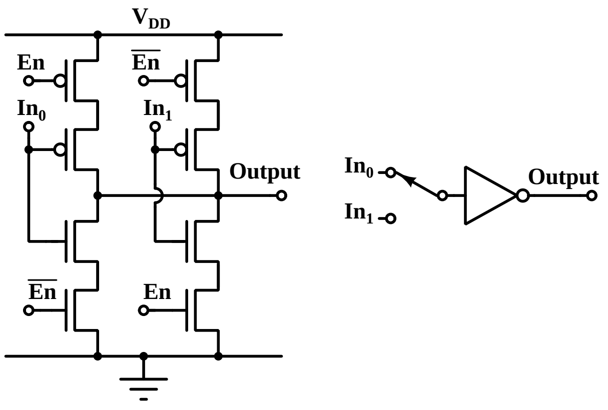


Figure 6.3: a two way mux

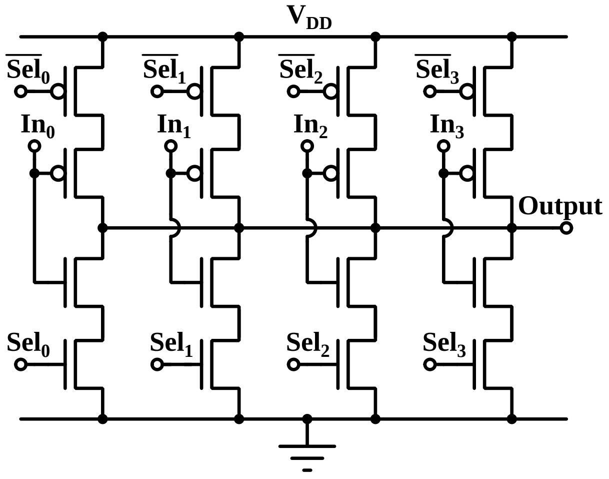


Figure 6.4: 4 input multiplexer

one *Sel* signal is high and the corresponding $\overline{Sel}$ signal is low. So the selected data appears inverted at the output and all other tri-stateable inverters are disabled.

6.4 Tiny XOR and XNOR

A compact XOR circuit can be made using pass gates.

When A = 0, the pass gate on the right is on and passes B to the output. Also, the inverter like circuit in the middle has 0 applied on top and a 1 applied at the bottom. Both transistors act as source followers and thus, also drive the output to B.

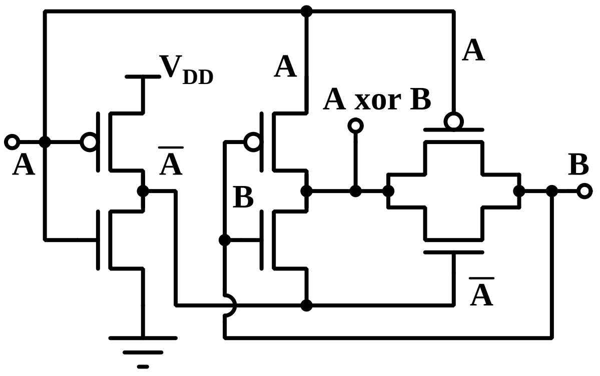


Figure 6.5: “Tiny” XOR circuit using a transmission gate

When A = 1, the pass gate is open and the middle circuit has 1 at the top and 0 at the bottom. So it acts like a regular inverter and provides $\overline{B}$ at the output.

Then, by Shannon’s Boolean expansion theorem, this constitutes the XOR function: $A \cdot \overline{B} + \overline{A} \cdot B$.

A compact XNOR circuit can be made similarly.

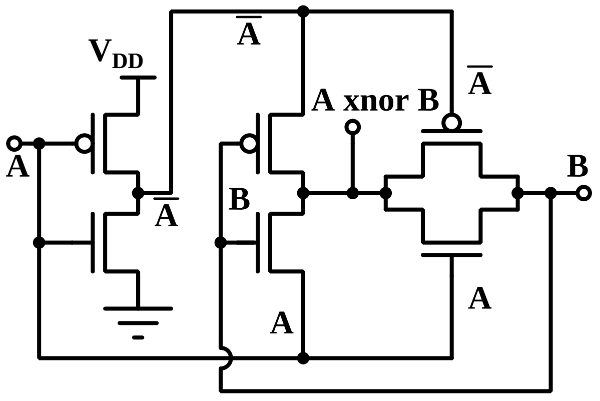


Figure 6.6: “Tiny” XNOR circuit using a transmission gate

When A = 1, the pass gate on the right is on and passes B to the output. Also, the inverter like circuit in the middle has 0 applied on top and a 1 applied at the bottom. Both transistors act as source followers and thus, also drive the output to B.

When A = 0, the pass gate is open and the middle inverter like circuit has 1 at the top and 0 at the bottom. So it acts like a regular inverter and provides $\overline{B}$ at the output.

By Shannon's Boolean expansion theorem, this constitutes the XNOR function: $A{\cdot}B{+}\overline{A}{\cdot}\overline{B}$

6.5 D latch and flipflop

Two inverters and two pass gates can be combined to form a transparent D latch. This latch is level sensitive.

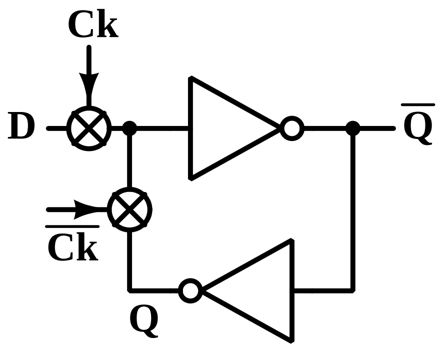


Figure 6.7: A transparent D latch using transmission gates

When the clock signal is high, the input is connected to the two inverters, and the feedback switch is open. Thus, D is buffered through to Q. This latch is called transparent in this stage, because a change in the value of D will result in a corresponding change in the Q outputs.

When the clock goes low, the inverters are disconnected from the D input and the feedback switch is closed. this results in the value of D (which existed when the clock went from high to low) being capture by the latch. In this state, changes in D do not affect the Q outputs.

The two pass gates and the upper inverter in the transparent D latch form a two way multiplexer between D and Q.

Two transparent latches with complementary clocks can be connected in a master slave fashion to form an edge sensitive D flip flop. When the clock is low, the first latch is transparent but the second latch hold its previous value. When the clock transitions to high, the first latch captures the value of D at this instant. The second latch becomes transparent.

Outputs don't follow D in either state of the clock. Q reflects the value of D at the most recent positive transition on Ck.

We some times need an edge sensitive D flipflop which can be reset asynchronously. Q

It is possible to replace the two pass gates and the connected inverter as described above by an inverting two way mux. The feedback inverter can continue to be a traditional CMOS inverter.

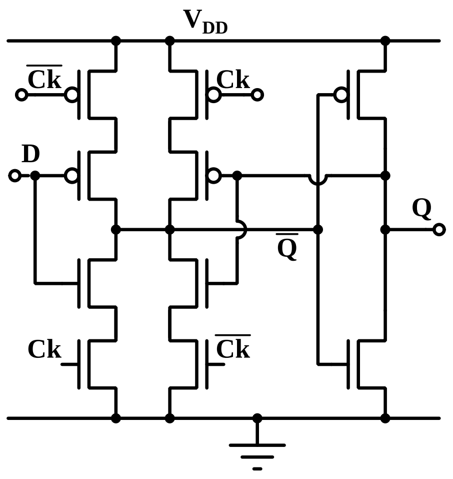


Figure 6.8: A transparent D latch using a two way multiplexer

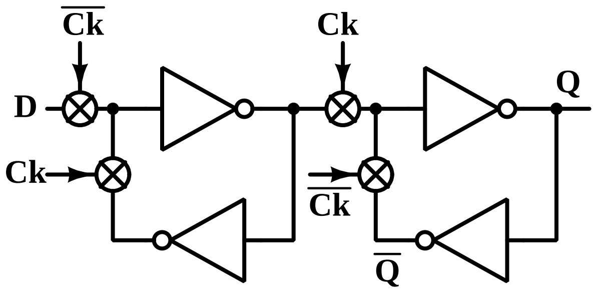


Figure 6.9: Edge sensitive master slave D flipflop

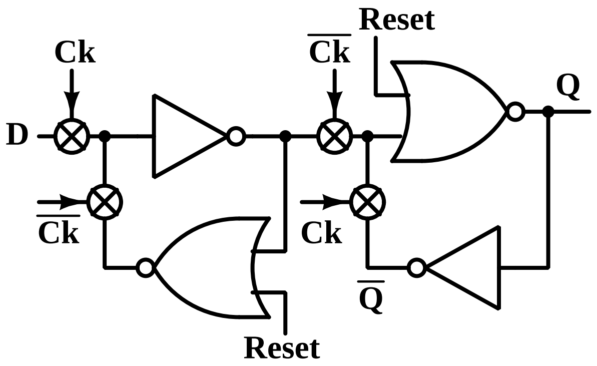


Figure 6.10: Edge sensitive D flipflop with asynchronous reset

should go to 0 as soon as reset is applied. To do this, the inverter supplying Q is changed to a NOR, with Reset as its other input. However, this is not sufficient. The flipflop should remain reset even when the reset signal is removed and Ck is at 0. For this, the lower inverter in the first latch should also be changed to a NOR gate, with reset as its other input.

Use of asynchronous reset is not recommended due to testability concerns.

If we need both set and reset for an edge sensitive D flipflop, we should change the other two inverters also to NOR. Q should go to 1 as soon as Set is applied. As is common in

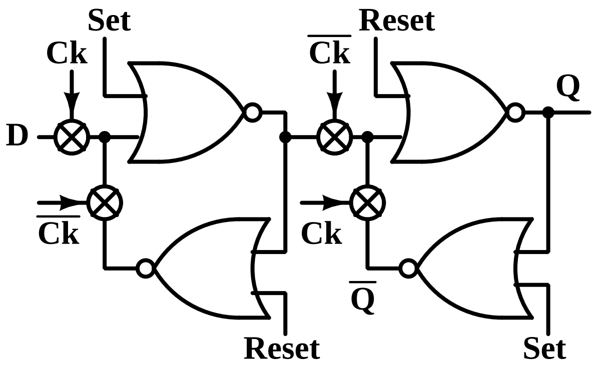


Figure 6.11: Edge sensitive D flipflop with asynchronous set and reset

all asynchronous set and reset controls, the two of these should never be applied simultaneously.so Reset should be 0 when Set goes to 1.
When Ck = 1, a 1 on set (with 0 on reset) will force Q to 1.
When Ck = 0, a 1 on set will force the first latch output to 0. This will be inverted by the NOR in the transparent second latch to produce a 1 at the Q output.

However, the use of asynchronous set or reset is not recommended due to testability concerns.

Chapter 7

Semi-Custom Design

7.1 Semi-custom Design

In the previous chapters, we have looked at design techniques used when we have to design all parts of a VLSI circuit from scratch. This includes the choice of logic styles, adjustment of transistor geometries, their layout etc. to meet a given set of specifications. This is called full custom design. While this permits an optimal tradeoff between power, speed and complexity, the whole process is long and laborious. The cost of developing a custom designed VLSI circuit is very high and can be justified only by those designs which sell in very large quantities.
Some applications can afford to compromise on speed, power and complexity in order to achive a quick design and fabrication time and lower costs. These use a design style known as semi-custom design. In semi-custom design, part of the design and fabrication is already done for us. We take this pre-fabricated template and customize it to perform the functions that our VLSI needs to perform. This approach has several advantages. The prefabricated part can be processed and kept ready for customization. Then the time to take an application to the market is only the processing which is necessary after customization.

Also, different applications can use the same pre-fabricated template. While each application may have a relatively small market, the template will be used in very large numbers, making it economically feasible. Obviously, the pre-fabricated template is not customized for final requirements of a particular final circuit. So it will not be an optimal design for a given application. However, most applications do not require absolutely the best performance. In such cases, the time to market and cost can be drastically reduced by using semi-custom design.

Clearly, if the customization step occurs later in the design and fabrication cycle, the time to market will be shorter. However, the overall implementation may have a lower performance, since a higher fraction of the design and fabrication cycle is non-specific to the actual design.

An example of semi-custom design is Field Programmable Gate Array (FPGA) based design. In this case, customization is done *after* the product has actually been delivered to the end user. (That is what the term Field Programmable refers to). The actual VLSI template which can cater to this late customization is quite complex. Therefore the area is much higher and the speed much lower compared to what could have been achieved by a custom designed circuit. For example, system clock speeds possible with FPGAs are of the order of hundreds of MHz, whereas custom circuits can go to clock speeds of about 4 GHz. To implement a given function, the silicon area consumed by an FPGA may be an order of magnitude higher than the area consumed by custom designed circuit performing the same function. However, since many applications can use the same FPGA, the FPGA chip is produced in very large quantities, which provides economies of scale. Thus, for designs which are not produced in very large quantities, FPGA based design may be the most cost effective solution inspite of the overhead in complexity, power consumption and delay.

7.2 Procedure for Semi-Custom Design

There are two components of Semi-custom design technique.

1. Customization techniques which will be used for adapting the “fabric” to implement the final application on the given template.

2. The design of the basic template or the “fabric” which can be customized as late as possible and which can be used by a large variety of applications.

In the technologies used for fabricating VLSI circuits, transistors are defined much earlier than the interconnects. Interconnects are defined towards the end of the fabrication procedure. Therefore customization of a pre-fabricated template is commonly done by changing the interconnects.

7.3 Customization of interconnects

Customization of interconnects is done by placing programmable interconnect devices at appropriate points in the pre-fabricated template. These include fuses, anti-fuses and transistor based interconnect.

Fuses Customization of interconnnects can be done through programmable removal of an existing connection. These work like fuses in electrical circuits. The connection to be customized is connected in the template through a narrow neck like structure. The

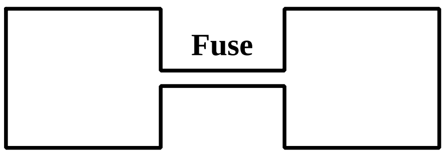


Figure 7.1: fuse structure

narrow part of the interconnect is capable of passing normal operating currents of the circuit. However, during customization, one may pass a pulse of heavy current through this, which ‘blows’ the fuse and disconnects the wires leading up to the fuse.

Anti-fuses In these structures, there is no connection between the programming nodes in the un-programmed state of the interconnect. The current path is interrupted through a thin layer of insulator. By applying a high voltage pulse, this insulator can be broken

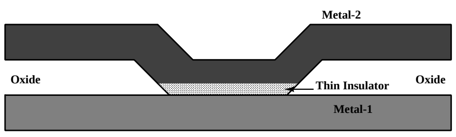


Figure 7.2: An anti-fuse. The insulator could be amorphous silicon.

down and a short created across it. This action is opposite to that of a fuse, so such structures are called anti-fuses.

Transistor based connection In this approach, a transistor is placed in the current path as a switch. When this transistor is on, there is a connection. When it is off, there is no connection. The state of the transistor is decided by a memory, which can be pre-loaded at power on.

7.4 Implementation of Configurable Template

7.4.1 Programmable Logic Arrays

Logic functions can be expressed in a generic sum of products form.

We would like a regular circuit which can be re-configured to implement any sum of products. One way is to use a customisable array which produces different product terms and another customisable array which sums the products so produced. CMOS logic is not convenient for implementing this architecture. This is because simultaneous configuration of the p

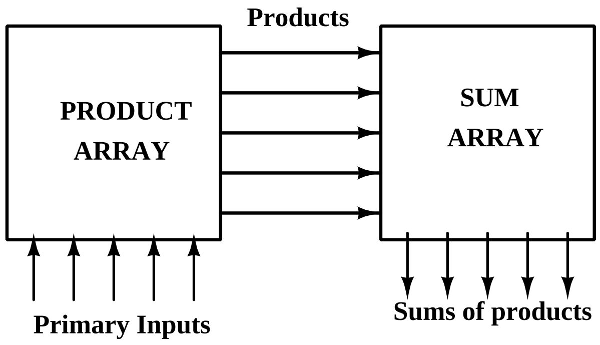


Figure 7.3: Architetcture of a Programmable Logic Array

channel pull up network as well as that of the n channel pull down network is inconvenient. Pseudo NMOS gates are more suitable for re-configuration in this case, as the pull up is just a single grounded gate PMOS whose geometry and interconnection remains the same for all logic.

But Pseudo NMOS circuits are ratioed. Since the geometry of transistors is defined early during fabrication, we would like an architecture where transistor geometry does not depend on the logic being implemented.

Implementing sums is not a problem, since this will be done through a NOR like configura-

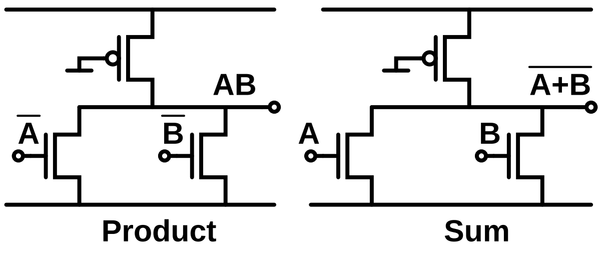


Figure 7.4: Product and sum implementation in a PLA

tion. In NOR configuration, each term contributes a transistor in parallel, whose geometry is the same as that of the reference inverter on which the design is based. Thus, sum of logic terms can be implemented with transistors whose geometry is fixed. However, implementing products presents a problem, since this requires NAND configuration, where the geometry of series connected pull down devices depends on the number of devices connected in series and hence, on the specific logic being implemented.

How do we implement the product function then?

We can use the expression : $A \cdot B = \overline{\overline{A} + \overline{B}}$.
Now the product of A and B can be implemented as the NOR of $\overline{A}$ and $\overline{B}$, which does not use series connected transistors. By adding inverters at the input and output as required, we can implement products or sums using only NOR type of circuits, which use constant geometry NMOS transistors in parallel.

Product Array

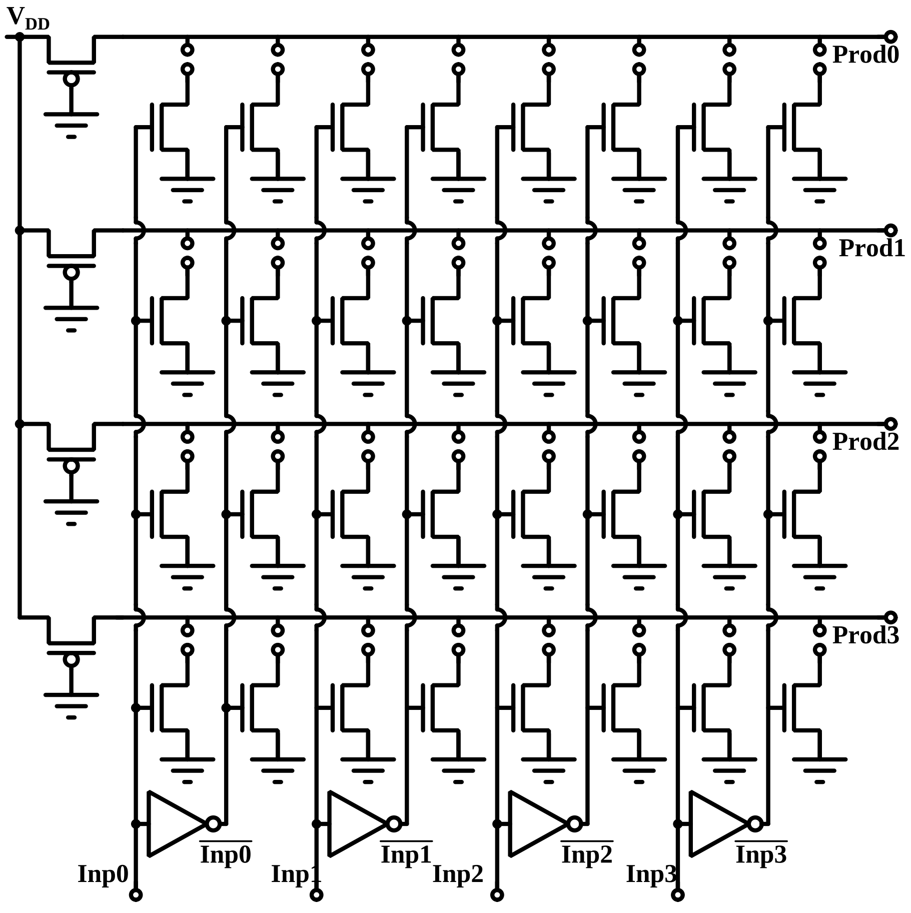


Figure 7.5: A programmable product array

Suppose we want to generate p distinct products of n inputs. Since for product implementation, we need complemented inputs, we connect a static CMOS inverter to each input. Now we have access to complemented as well as uncomplemented inputs. To design the product array, we place nMOS transistors in a matrix of $2n$ columns and p rows where n is the number of inputs and p is the number of distinct products that we want to generate. Since the complement of each of the inputs is produced using inverters, n signals and n complements drive the gates of nMOS transistors in $2n$ columns. unconnected. Each row generates

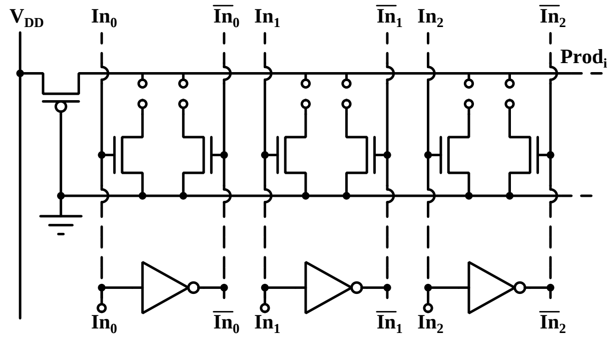


Figure 7.6: One row of a PLA

a distinct product. It is pulled up by a grounded gate pMOS transistor. So this matrix can produce p distinct products. By connecting links selectively at drains of nMOS transistors in any row, chosen transistors can be included in parallel in the NOR configuration. Connecting a transistor includes the complement of the input driving its gate in the product term.

Each column is driven by an input or its complement. Thus each of these nMOS transistors has its source grounded and its gate connected to an input or its complement. Programming involves either connecting its drain to the row which forms the output or leaving it unconnected. If the drain is connected to the output, the transistor comes in parallel with other connected transistors, as required by the pseudo NMOS configuration.
If we want to include a signal (or its complement) in the product term, we connect the drain of the transistor driven by the complemented input (or uncomplemented input) to the row. If we do not want either the signal or its complement in the product, we simply leave both nMOS transistors

In the example shown in Fig.7.7, we have generated products $\overline{A} \cdot B \cdot C$, $\overline{B} \cdot C$, $A \cdot \overline{C}$ and $\overline{A} \cdot B \cdot D$. To generate $\overline{A} \cdot B \cdot C$, links from drains of transistors driven by $A, \overline{B}$ and $\overline{C}$ are connected to the top product row. For $\overline{B} \cdot C$, drains of transistors driven by B and $\overline{C}$ are connected to the output line in the second row. Other products are generated similarly.

Sum Array

We can form the sum matrix similarly. Each of the products generated by the product array drives gates of a nMOS transistors in a row, arranged in a $p \times s$ matrix, where s is the number of different sums of products that we need to generate. The circuit in Fig.7.8 can produce sums of selected product terms. By connecting links at drains of nMOS transistors

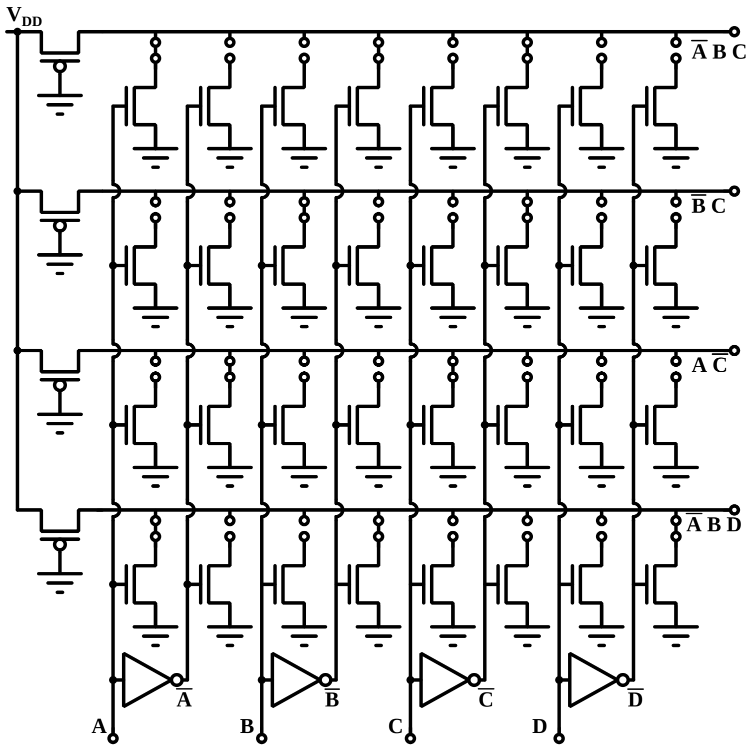


Figure 7.7: Example for generation of different products from inputs

in a particular column, chosen products can be included in a sum output. Several columns can be used to generate different sums of products. Outputs are NORs of products. Inverters convert these to ORs of products.

Implementation of Finite State Machines

A finite state machine has the following components:

1. Storage elements to store the current state,
2. Random logic to compute the next state as a function of current state and current inputs, and
3. Random logic to compute the current output as a function of current state and optionally, the current inputs.

By adding latches at the output of the PLA, we can provide for all of these. Output from latches is fed back to the product array. The PLA now computes both the next state and current outputs with primary inputs and current state as its inputs.

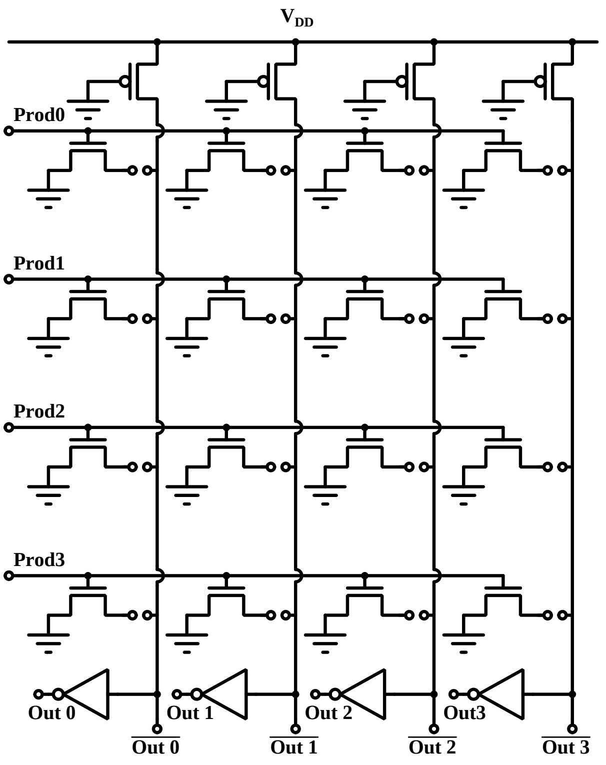

V_{DD}
Prod0
Prod1
Prod2
Prod3
Out 0
Out 1
Out 2
Out3
$\overline{\text{Out 0}}$
$\overline{\text{Out 1}}$
$\overline{\text{Out 2}}$
$\overline{\text{Out 3}}$

Figure 7.8: Sum array for generating sums of products

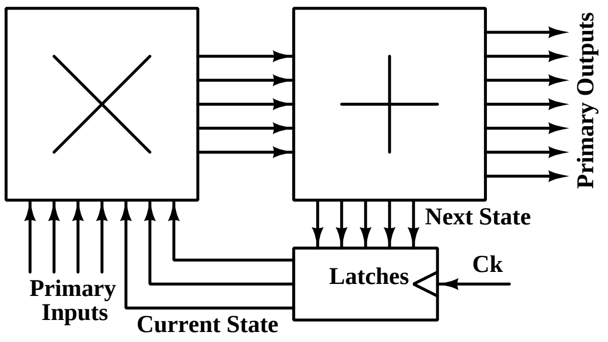


Figure 7.9: A finite state machine implemented with a PLA

7.4.2 Sea of Gates

The programmable logic array uses pseudo nMOS design style as its base. This involves static power consumption. The stray capacitance of a programmable plane is high – which makes PLAs rather slow.

How can we use CMOS style gates in a semi-custom design? The "Sea of Gates" template provides the capability to use CMOS style gates in a semi-custom design. In CMOS logic, each input goes to an nMOS as well as a pMOS. In this style of design, all transistors are pre-placed in a pattern which is optimum for implementing CMOS gates because there are pairs of nMOS and PMOS transistors driven by the same input at their gates. Once an array

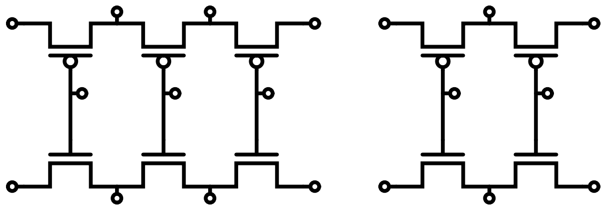


Figure 7.10: Pattern of transistors in a sea of gates template

of such patterns is placed in the chip, interconnects will determine what kind of logic will be implemented. Both n and p channel transistors appear to be in series here! How do we construct regular CMOS logic gates using these?

Actually, by wiring the apparently series connected devices, we can convert them to series or parallel as required. The example in Fig.7.11 shows a NAND gate, whose output is fed to

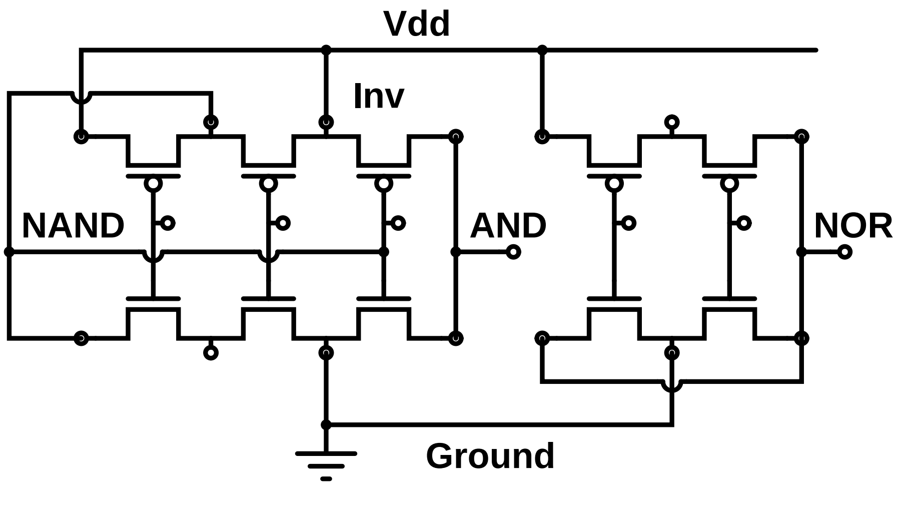


Figure 7.11: Implementing a NAND, Inverter and NOR gates in sea of gates

an inverter to form the AND function. The structure on the right forms a NOR gate.

The sea of gates template makes use of the fact that all inputs go to an nMOS as well as to a pMOS in CMOS style gates. What about structures which don't follow this rule? For example, in a transmission gate, the nMOS and pMOS transistors are driven by complementary signals.

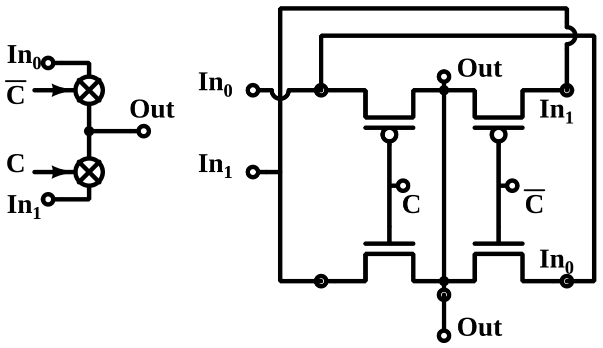


Figure 7.12: A pair of transmission gates with complementary control inputs

Transmission gates are often used in pairs with one or the other being on. (This is so because Inputs should not be left floating in static CMOS design). A pair of transmission gates can be implemented as shown in Fig.7.12, coupling diagonally opposite transistors in a 2-pair. As an application of this, a transparent D latch uses transmission gates with complementary control inputs. It can be implemented as shown in Fig.7.13. Two such latches can

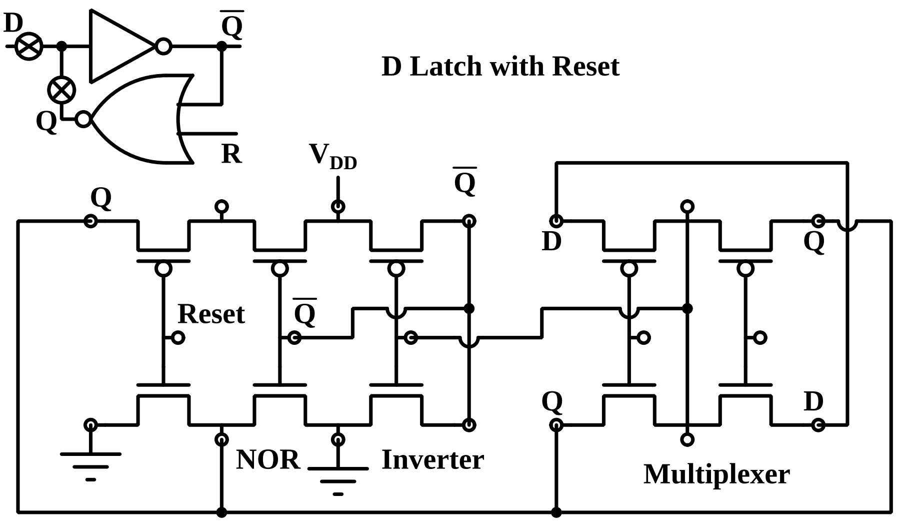


Figure 7.13: A transparent D latch implemented with "Sea of Gates"

be connected in master-slave configuration to form an edge sensitive D flip-flop.

Sea of gates based designs provide better performance compared to PLA based designs. However, these cannot be field programmed and the interconnect mask has to be customized and used during chip manufacture.

7.5 Interconnect Channels in Semi-Custom Logic

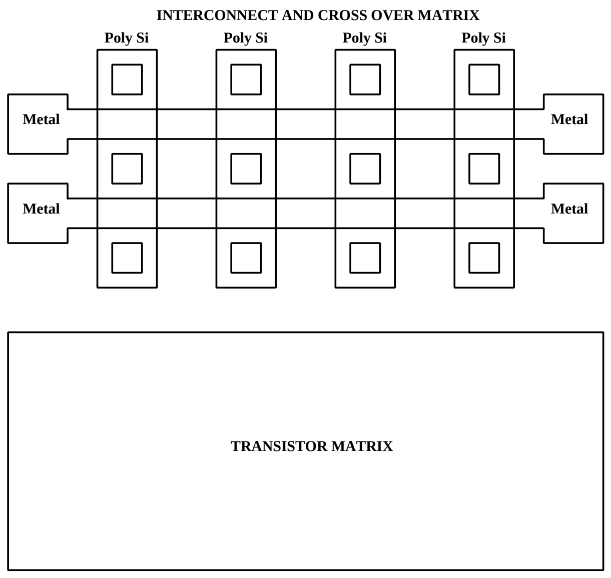


Figure 7.14: An example interconnect channel for semi-custom logic

Apart from programming the transistor matrix, we need to provide programmable interconnect between different sub-units. therefore in a semi-custom integrated circuit, prefabricated interconnect channels alternate with transistor matrices.

The exact shape and composition of these interconnect channels varies from design to design. Typically, these include facilities for local as well as global interconnects. The size of these blocks is optimized to provide a sufficient interconnect capacity without wasting too much area.

7.6 Using Memories as Logic

A logic function can be represented by a truth table. This can be stored in memory. Inputs to the logic function act as address lines. Precomputed value of the logic function are stored at the address represented by its input values. For any input combination, the address represented by these is looked up and the stored value presented as the output.

For example, any logic function of 5 inputs can be implemented by a 32×1 bit memory. If the same 32 bit memory is organized as 16×2, it can store two logic functions of up to 4 inputs.

7.7 Field Prgrammable Gate Arrays

Different types of semi-custom design chips have been introduced with various names. Here is a list:

PLAs: These were the first semi-custom devices to be introduced in the market by Philips in the early 1970's. These are used to implement generic sums of products and have a programmable AND plane as well as a programmable OR plane.
Because both AND and OR arrays are programmable in PLAs, these are rather slow.

PALs: A modification of PLAs in which the product array is programmable, but the OR plane is fixed were introduced as Programmable Array Logic or PAL. To cover for the lack of flexibility due to the OR array being fixed, these were introduced in different size combinations and multiple devices were used for different functions.

CPLDs As technology progressed, it was possible to put multiple devices on the same chip. Combinations of PALs and PLAs were introduced with programmable interconnect as complex programmable logic devices or CPLDs. Altera was one of the first companies to introduce CPLDs commercially. CPLDs were a huge success and multiple companies introduced CPLDs of different architectures.

Sea of Gates In parallel with field programmable devices, mask programmable devices were manufactured. In these, one (or more) levels of metallization could be customized at the manufacturing site, over a "sea" of pre-fabricated devices. These provided circuits with better performance, but with less flexibility as programming could not be done at the user site.

FPGAs Field Programmable Gate Arrays borrow features from sea of gates as well as CPLDs. Instead of a "sea" of transistors, these have a "sea" or a repetitive array of combinations of logic elements and memories.
In addition, these include a programming infrastructure for interconnects like CPLDs.

On the periphery of these chips, special Input Output devices are fabricated which help in establishing fast interconnection with the external world.

A field Programmable array allows the logic functions as well as the interconnect to be programmed. Transistors driven by SRAM can be used to program interconnects. Apart from

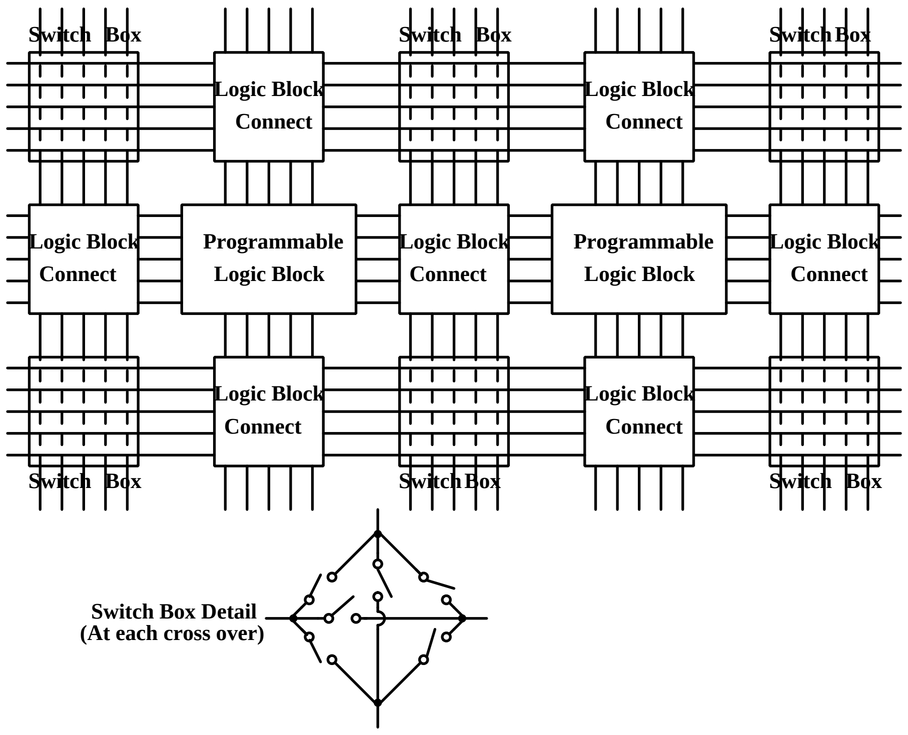


Figure 7.15: Alternating logic and interconnect boxes in an FPGA

logic blocks and interconnect fabric, modern FPGA's contain other useful structures, such as

- Block RAM,
- Processor cores (Power PC on Xilinx Vertex family),
- Fast multipliers
- I-O Blocks

A typical board, used for implementing a variety of applications will have a micro-controller and a few programmable devices for providing dedicated functions and glue logic. This is configured once at power on, and then left alone to perform its functions. This is called "static re-configurability".

But we may want to re-cofigure a structure while it is in operation! For example, one can imagine the design of digital filter, which adapts itself during operation itself depending on inputs. This kind of re-configurability is called “dynamic re-configurability”.

Obviously, dynamic configurability requires that the dead time during re-configuration should be minimized. Thus, there is a trade-off between flexibility of re-configuration and the time taken to re-configure. In fine grain re-configuration, we can re-program every gate of the design. This is the case for current FPGA and CPLD devices. This gives very good design flexibility, but takes a long time to re-configure.

In coarse grain re-configuration, relatively large functional blocks are re-configured. For example, by re-connecting a collection of shifters and adders, we can generate any combination of multipliers and adders. The effort to re-configure is smaller, because we do not alter the inner design of shifters and adders. This is compatible with dynamic re-configuration. Using coarse grain re-configurability, it becomes possible to dynamically change a circuit, *during* its operation. This has obvious advantages in adaptive circuits.

We normally do not want to re-configure the whole circuit. Indeed the control signals for doing the re-configuration will be generated by a circuit which remains unchanged. Therefore dynamic re-configurability requires circuits which can be partially re-programmed. Many FPGA are beginning to promise this feature.

Chapter 8

Multi-Stage Logic Design

We have seen how to design single stage logic in different design styles. However, typical designs need multi-stage combinational logic. In this chapter, we'll discuss techniques used for the design of multi-stage logic.

8.1 Stage Delay and Sizing

In the previous chapter 2, we had derived expressions of the type

$$\frac{K\tau_L}{C_L} = \int_{V_1}^{V_2} \frac{dV}{f(V)}$$

for the delay of a single logic stage. Here τ_L is the time taken to charge/discharge the load capacitor C_L from V_1 to V_2 and K is the conductance factor given by $\mu C_{ox}\frac{W}{L}$.

The right hand side of this equation is a definite integral. It will evaluate to some constant depending on the voltages defining the 'High' and 'Low' logic levels, the supply voltage, turn on voltages, drain saturation voltage etc. In digital design, we shall keep the channel length at its minimum value, so L is a constant. Let us initially ignore the parasitic capacitances. We can see that

$$\frac{W\tau_L}{C_L} = \text{Constant} \qquad \text{so } \tau_L \propto \frac{C_L}{W}$$

This tells us that the delay associated with a gate charging a load capacitor scales directly with C_L and inversely with W, the width of the charging/discharging transistor. This linear dependence permits us to design logic stages easily.

8.2 Tapered Buffer

As an example of multi-stage logic, let us take the case when a large capacitor is to be driven by a CMOS circuit. A minimum sized inverter will take too long to charge this capacitor. Therefore, we would like to scale up the inverter (multiply all transistor widths by a scale factor) in order to drive this large capacitor. However, the input capacitance of this scaled up inverter may be too large for a minimum sized inverter to drive. Therefore, we need a medium sized inverter to drive the large final inverter. We keep adding inverters, till the first inverter in the chain is small enough to be driven by standard CMOS logic. This kind of buffering is referred to as a *tapered buffer.*

How do we decide the number of inverters to include in this chain? And what should be the

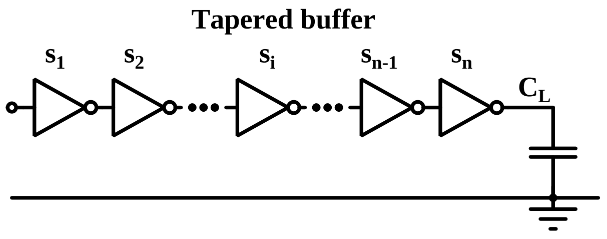


Figure 8.1: A tapered buffer

scale factors for each successive stage to minimize the total delay?

Let the i'th inverter in the chain be scaled up by a factor s_i relative to a minimum sized inverter. Let the delay of a minimum sized inverter driving another minimum sized inverter be τ. Then the i'th inverter provides charging currents which are s_i times the minimum sized inverter. However, the load it sees is s_{i+1} times the input capacitance of a minimum inverter. Therefore the delay associated with the i'th stage is $\frac{s_{i+1}}{s_i}\tau$. The total delay of the inverter chain is given by

$$d_{total} = \sum_{1}^{n} \frac{s_{i+1}}{s_i}\tau = \tau \sum_{1}^{n} \frac{s_{i+1}}{s_i} \tag{8.1}$$

In order to minimize the total delay, we should put the partial derivative with respect to each of the s_i equal to zero. Therefore,

$$\tau \frac{d}{ds_i}\left(\frac{s_2}{s_1} + \cdots + \frac{s_i}{s_{i-1}} + \frac{s_{i+1}}{s_i} + \cdots\right) = 0 \tag{8.2}$$

Only two terms in the sum contain s_i. Since all scale factors s_i are independent, the derivative of all the rest of the terms is 0. Therefore,

$$\frac{1}{s_{i-1}} - \frac{s_{i+1}}{s_i^2} = 0 \qquad \text{Which gives:} \qquad \frac{s_i}{s_{i-1}} = \frac{s_{i+1}}{s_i} \tag{8.3}$$

This means that the *stage ratio*, which is the factor by which an inverter is larger than the previous one, should be the same for all stages.

Let this constant stage ratio for the tapered buffer be ρ. The delay contributed by the i'th stage is $\frac{s_{i+1}}{s_i}\tau = \rho\tau$. The first stage is a unit inverter. Each subsequent stage has a drive capability which is ρ times the drive capability of the previous stage. Since the drive capability is being stepped up by ρ in n stages, we should have

$$\rho^n = \frac{C_L}{C_{in}} \qquad \text{so } \rho = \left(\frac{C_L}{C_{in}}\right)^{1/n} \tag{8.4}$$

We define the ratio $H \equiv C_L/C_{in}$. Then, $\rho^n = H$ and so, $n = \ln H/\ln\rho$.
The total delay is given by

$$d_{total} = \sum_1^n \frac{s_{i+1}}{s_i}\tau = \sum_1^n \rho\tau = n\rho\tau \tag{8.5}$$

We want to find the value of ρ which will minimize the total delay. Note that n and ρ are not independent since $\rho^n = H$. Taking logarithms on both sides, we get

$$n\ln\rho = \ln H, \qquad \text{so} \qquad n = \frac{\ln H}{\ln\rho} \tag{8.6}$$

The total delay can then be expressed as

$$d_{total} = \frac{\ln H}{\ln\rho}\rho\tau = \tau\ln H\frac{\rho}{\ln\rho} \tag{8.7}$$

Total delay will be minimized with respect to ρ when its derivative with respect to ρ is 0. This gives

$$\tau\ln H\left(\frac{1}{\ln\rho} - \frac{\rho}{(\ln\rho)^2}\frac{1}{\rho}\right) = 0 \tag{8.8}$$

This leads to

$$\frac{1}{\ln\rho} = \frac{1}{(\ln\rho)^2}, \qquad \text{so} \qquad \ln\rho = 1, \quad \text{or} \quad \rho = e \tag{8.9}$$

$$\text{Since} \quad \rho = e, \qquad n = \frac{\ln H}{\ln\rho} = \ln H \tag{8.10}$$

Thus we obtain the result that the optimum stage ratio for a tapered buffer is e, while the optimum number of stages in the buffer is given by $\ln(C_{out}/C_{in})$.
(The optimum stage ratio comes out higher ($\approx$ 3 to 4), if we take self loading into account.)

These results were computed for a situation where capacitors were driven only by inverters and self loading due to transistors in the gate was ignored. The general situation will differ from the tapered inverter chain in several respects:

1. Each gate will have a parasitic delay associated with it due to the capacitance of the driver transistors themselves.
2. Multi-stage logic can use gates other than inverters.
3. In the tapered inverter chain, each stage drives another inverter. In general, we can have branching, where a stage drives multiple gates. (Fanout is greater than 1).

The generalized optimization which removes these restrictions was developed by Ivan E. Sutherland and Robert F Sproull in a paper "Logical Effort: Designing for Speed on the Back of an Envelope.", published in the Proceedings of the Conference on Advanced Research in VLSI, held in March 1991. The method was later described in much greater detail in a book by Sutherland, Sproull and Davis titled "Logical Effort: Designing Fast CMOS Circuits", published by Morgan Kaufmann in 1998. The material in this chapter is largely based on that book.

8.3 Improved model for delay of a single gate

8.3.1 Considering the Effects of Self-Loading

Logic gates have to drive capacitance associated with the drains of driver transistors themselves as well as the Miller capacitance associated with these transistors. This additional capacitance is proportional to W. Thus, the total load capacitance on a logic stage is

$$C_L = C_{ext} + WC_p$$

Where C_p is the parasitic capacitance per unit width of driver transistors. Therefore the delay of the stage is

$$\tau_L \propto \frac{C_L}{W} = \frac{C_{ext} + WC_p}{W} \quad \text{Which gives} \quad \tau_L \propto \frac{C_{ext}}{W} + C_p \tag{8.11}$$

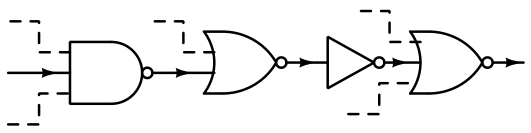
Thus the parasitic capacitance associated with driver transistors results in additional delay which is scale independent and the expression for stage delay in a multi-stage logic design needs to be modified to:

$$tau_L = \text{Const.} \times \frac{C_{ext}}{W} + \tau_p$$

Where τ_p is the parasitic delay associated with a logic gate driving its own output capacitance, and is independent of the size of the driving logic gate or the external load.

8.3.2 Using Logic Gates Other Than Inverters

We have seen how to incorporate the effect of self capacitance of the driver on the stage delay. How do we generalize the tapered inverter chain to include gates other than inverters?



For this, we observe that we are optimizing the delay along a *fixed path.* Every logic gate will have just one input and one output *on this path.* Thus, as far as the delay along this path is concerned, we can treat each gate as being equivalent to an inverter, with an appropriately modified delay model.

To model the logic gate in a multi-stage path as an equivalent inverter, let us see how the delay of a logic gate compares to that of an inverter.
Transistors sizes in a template logic gate are determined using series-parallel rules on a template inverter. These rules make the output drive from the logic gate the same as that from the template inverter. Thus final transistor sizes in a logic gate depend on:

1. Difference in mobility of pMOS and nMOS transistors. Just as in an inverter, a pMOS transistor has to be wider than an nMOS transistor to provide the same drive current because the hole mobility is lower than electrons mobility. For example, we may need to scale the width of pMOS transistors to be double that of nMOS transistors to get the same current. This ratio is represented by γ.

2. If there are n series connected transistors, their widths should be scaled up by n in order to make the output drive of the logic gate equivalent to the template inverter. For transistors in parallel, we keep the widths unchanged because each parallel path should be able to provide the same drive as the inverter *on its own.*

3. After accounting for mobility differences and series connections, we get the template logic gate. we may scale up *all* transistors of the template gate by some factor in order to drive larger loads. (This is the factor S_i used for inverters in our earlier discussion of a tapered inverter).

The template logic gate keeps the output drive the same as the template inverter, but has a different input capacitance compared to the template inverter. This changes the loading placed on the *previous* stage compared to the tapered inverter analysis, and hence its delay.

We illustrate this by taking 2 input NAND and 2 input NOR gates as examples. We take the capacitance presented to the previous gate by the nMOS in the template inverter as a unit of capacitance. Also, for this example, we take the mobility correction factor γ to be 2, so the pMOS in the template inverter has a size of $1 \times \gamma = 2$.

Sizes of transistors in the NAND and NOR gates are determined by series parallel rules.

We can see from Figure 8.2 that the template inverter places a load of $1 + 2 = 3$ units on the

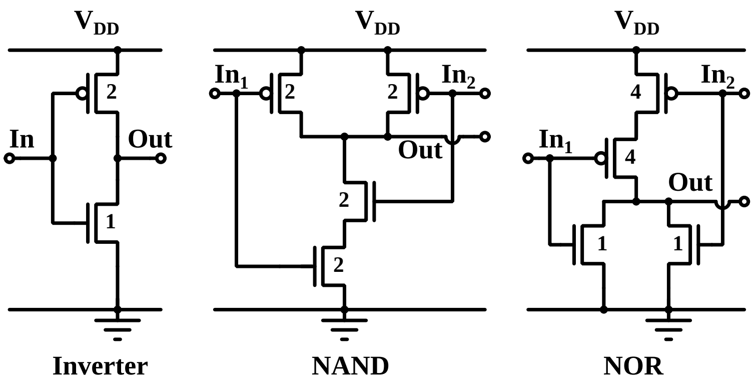


Figure 8.2: Load presented by 2 input NAND and NOR gates to their drivers.
Here we have assumed $\gamma = 2$.

previous gate, while a 2 input NAND gate (with the same output drive) loads the previous stage with a capacitive load of $2 + 2 = 4$ units and a 2 input NOR gate loads the previous stage with a capacitive load of $1 + 4 = 5$ units.

Thus a template NAND gate presents a load capacitance which is 4/3 times that presented by the template inverter. Similarly, a template NOR gate loads the previous stage with a capacitance which is 5/3 times that of a template inverter.

The template logic gates have the same output drive (and hence the stage delay) as the template inverter but increase the delay of the *previous* stage. This is not convenient for handling minimization of the delay along the logic path as single stage delays are not independent of the type of following gates. It is more convenient to set up the delay model such that each stage has the same **input capacitance** as the template inverter. (Then the logic type will affect its own delay and not that of the previous gate). To do this, we scale down all transistors of the template logic gate such that it presents the same capacitance to the previous stage as the template inverter.

This will decrease the output drive and consequently increase the delay of the logic gate. For example, if we multiply sizes of all transistors of the 2 input NAND gate by 3/4, it will present a capacitive load of 3 units to the previous stage, which is the same as the load presented by the template inverter. However, the output drive of such a NAND gate will be 3/4 times the drive of the template inverter. Thus its delay will be 4/3 times the template inverter delay.

Similarly, a 2 input NOR gate with input capacitance matched to the template inverter will have a delay 5/3 times that of the template inverter.

To set up a stage delay model such that logic type of a stage affects its own delay and keeps the delay of the previous gate unchanged, we follow the following sequence of actions:

1. We begin with the template gate obtained by series parallel rules from the template inverter.
2. We determine the ratio of input capacitance offered by the template gate to that of the template inverter. Let this be g. (g is known as the logical effort of this logic type).
3. We scale down the size of all transistors of the gate by the factor g, which matches the input capacitance of the template logic gate to that of the template inverter.
4. This results in an increase in the delay of this stage by the factor g over the delay which would have been caused by the template inverter.
5. Now we can treat this gate as an inverter with a delay scaled up by g.
6. Finally, We scale up all transistors of this reduced gate by the factor s_i just as we had done in case of the tapered inverter to optimize the total delay along the logic path.

8.3.3 Effect of Branching

Finally, we must account for the fact that a practical logic chain will have branching at certain points in the logic chain. We introduce a new correction factor b, which accounts for the fact that when branching occurs, the load on a stage is more than just the input capacitance of the next stage. Suppose there is branching at the i'th stage of the logic chain. The actual

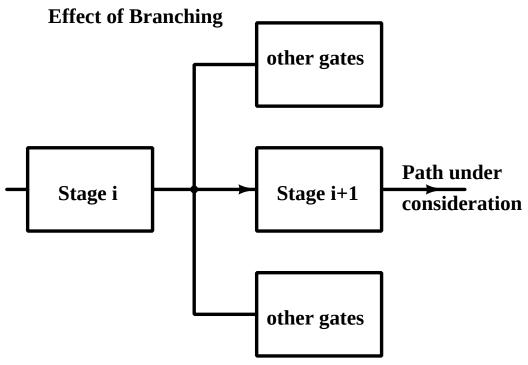


Figure 8.3: Effect of Branching

capacitive loading on the i'th stage is $C_{onpath} + C_{offpath}$, where $C_{onpath} = C_{in_{i+1}}$ and $C_{offpath}$ is the sum of all the other input capacitances connected to the output of the i'th stage. The delay of this path will be proportional to the total capacitance. We treat the logic path as if there is no branching, but the delay of this stage is corrected by the factor

$$b = \frac{C_{onpath} + C_{offpath}}{C_{onpath}} \tag{8.12}$$

If there is no branching at a stage, the corresponding b_i value is 1 since $C_{offpath} = 0$ and so, multiplication by b_i does not change any thing. If, however, there is branching at the i'th stage, multiplying by b_i corrects for the actual output capacitance.

8.4 Minimization of Path Delay

Now that we have found ways to take care of

1. Inclusion of Parasitic Delay
2. Use of logic gates other than inverters
3. Taking branching into account

We can optimize the cumulative delay of a general logic chain.
This treatment was first suggested by Ivan Sutherland, Bob Sproull and David Harris. They expanded on this technique in a book that they authored:

Logical Effort: Designing Fast CMOS Circuits
Ivan E. Sutherland, Bob F. Sproull, David L. Harris
Morgan Kaufmann Publishers, Inc.

Much of the treatment of the topic of logical effort in this chapter is based on the material in this book.

Now that we have taken care of the three departures from the tapered inverter *viz* parasitic capacitance, use of logic other than inverters and branching in the logic path, we can proceed to minimize the path delay with the new stage delay model. We can generalize the optimization made for a tapered buffer to a generic logic path by incorporating the corrections discussed above.

- We can account for the parasitic delay by adding a size independent delay. The parasitic delay depends only on the logic type and not on the size of the gate.
- To account for different types of logic, we treat a CMOS logic gate as an inverter, with a delay which is scaled up depending on its input capacitance relative to an inverter with

the same output drive. This correction factor is called the logical effort and is denoted by g.

Logical effort is also size independent and depends only on the type of gate being used and on *nothing else.* (It will depend on γ, of course - but that is a technological constant).

- We can account for branching by scaling up the charge/discharge time by the ratio of the total capacitance the gate drives with the input capacitance of the next stage.

8.4.1 Gate Delays with Logical Effort

In the method of Logical Effort, we deal with normalized delays. The unit of time is taken to be the delay of a template inverter driving another template inverter *without any parasitic elements.*

Combining the effect of self-loading with the use of logic gates other than inverters, the delay of a gate can be expressed as

$$d = f + p \tag{8.13}$$

where f is the *effort delay*, which depends on transistor currents and load capacitance, while p is the parasitic delay, which is size independent.

We further express f as a product of two quantities, g and h. Here h is the *electrical effort* of this stage. This is given by the ratio of output capacitance to input capacitance. This is equivalent to the *stage ratio* we had used while discussing the tapered inverter.

g is the *logical effort* which accounts for the extra loading caused by a gate as compared to an inverter, as discussed in the previous section. So,

$$f = gh \tag{8.14}$$

Combining Equations 8.13 and 8.14, we can express the delay introduced by a logic gate as

$$d = gh + p \tag{8.15}$$

where the delay is measured in units of τ, which is the delay of a template inverter driving another template inverter *not including the parasitic delay.*

This way of expressing delays separates the effects of different components which cause delay, so these can be handled independently.

- Dependence on technology is encapsulated in τ, which is the delay of a template inverter driving another template inverter *excluding* delay due to self loading.

- All delays are expressed as a multiple of this quantity.
- Delays in this formulation are unitless numbers and must be multiplied with τ to get the absolute value of delay.
- Dependence on sizing is encapsulated in h. This is the only component of delay which is size dependent.
- h is also a unit less quantity, because it is defined as the ratio of output capacitance to input capacitance.
$$h = \frac{C_{out}}{C_{in}}$$
- These capacitances can be measured in units of the input capacitance of an inverter.
- C_{in} is a measure of the factor by which a gate has been scaled up in order to make it faster.
- The effect of parasitic delays due to the load presented by the driver transistors themselves is expressed through p.
- p is size independent.
- It is also a dimensionless quantity as this delay is also expressed in units of τ.
- The parasitic delay does depend on logic type. It can be estimated by considering the size of transistors connected directly to the output node.

8.4.2 Logical Effort for Common CMOS Gates

The logical effort of of a gate is the ratio of input capacitance of the template logic gate obtained by series parallel rules from the template inverter and the input capacitance of the template inverter itself. The logical effort of an inverter is thus 1 by definition.

n input NAND and NOR gates are shown in the figure below. The n input NAND gate has n NMOS transistors in series and n PMOS transistors in parallel. Therefore each NMOS should have a width which is n times that of the template inverter. Each PMOS will have the same width as that of the template inverter – which is γ times the size of the nMOS (to account for lower hole mobility). Taking the capacitance of the nMOS transistor in the template inverter as the unit of capacitance, each input of the n input NAND gate loads its driver with $(n + \gamma)$ units of capacitance. The template inverter loads its driver by $(1 + \gamma)$ units. So the logical effort of an N input NAND gate is $(n + \gamma)/(1 + \gamma)$. This reduces to $(2 + \gamma)/(1 + \gamma)$ for a 2 input NAND, as expected.

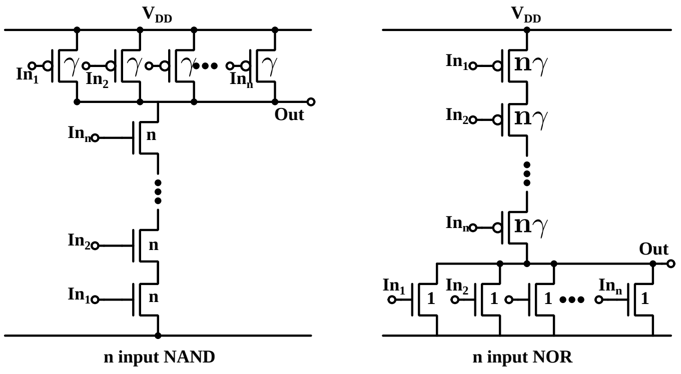


Figure 8.4: n input NAND and NOR gates

We can approximate the parasitic delay by considering only the self capacitance which is directly connected to the output. In case of an inverter, this is proportional to $(1+\gamma)$. For the n input NAND gate, the capacitance contributed by the nMOS transistor at the output node is n, while the capacitance from the n pMOS transistors is $n\gamma$. Therefore, the parasitic delay of the n input NAND is related to the parasitic delay of the inverter by the factor $(n+n\gamma)/(1+\gamma) = n$. Thus we have

$$p_{NAND} = np_{inv} \qquad \text{for an n input NAND gate} \tag{8.16}$$

The n input NOR gate has n PMOS transistors in series, each of which should have $n\gamma$ times the width of the nMOS transistor. It has n NMOS transistors in parallel, each of which can have the the same geometry as in the template inverter. (This is the unit of capacitance in our current discussion). Thus the loading on the driver of each input is $(1+n\gamma)$ units. So the logical effort of the N input NOR gate is $(1+n\gamma)/(1+\gamma)$. This reduces to $(1+2\gamma)/(1+\gamma)$ for a 2 input NOR as expected.

For the parasitic delay of the n input NOR gate, the capacitance contributed by the n nMOS transistors at the output node is n, while the capacitance from the pMOS transistor is $n\gamma$. Therefore, the parasitic delay of the n input NOR gate is related to the parasitic delay of the inverter by the factor $(n+n\gamma)/(1+\gamma) = n$. Thus we have

$$p_{NOR} = np_{inv} \qquad \text{for an n input NOR gate} \tag{8.17}$$

A 2 way multiplexer is shown in Fig. 8.5 below. It is clear that each signal input is loaded with a capacitance of $(2+2\gamma)$ units. Therefore the logical effort for either data input in this mux is $(2+2\gamma)/(1+\gamma) = 2$. The logical effort for the control inputs Sel and $\overline{Sel}$ is also 2. The parasitic delay for the 2 way mux is $2+2\gamma+2+2\gamma$ which is 4 times the parasitic delay of an inverter.

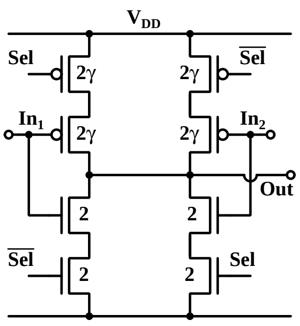


Figure 8.5: A 2 way multiplexer

It is interesting to see that the logical effort will remain 2 for every data input even when we parallel n tri-stateable inverters to form an n way mux. However the parasitic delay will increase to $2np_{inv}$ if we add n units.

The table below lists the logical effort and the parasitic delays of common logic gates. Notice that the unit of time is the delay of a minimum inverter driving another minimum inverter (excluding parasitic delay). Therefore in these units, the parasitic delay of an inverter, which is the ratio the parasitic delay (in absolute units) to the inverter delay without taking parasitic delay into account, need not be 1. It is convenient to specify the parasitic delay of other gates as multiple of the parasitic delay of an inverter p_{inv}.

Gate	Logical Effort	Parasitic Delay
Inverter	1	p_{inv}
2 input NAND	$(2+\gamma)/(1+\gamma)$	$2p_{inv}$
n input NAND	$(n+\gamma)/(1+\gamma)$	np_{inv}
2 input NOR	$(1+2\gamma)/(1+\gamma)$	$2p_{inv}$
n input NOR	$(1+n\gamma)/(1+\gamma)$	np_{inv}
2 way mux	2	$4p_{inv}$
n way mux	2	$2np_{inv}$

Table 8.1: Logical effort and parasitic delays of common gates

8.5 Design of multi-stage logic

In multi-stage logic, we consider the logical effort g_i, the electrical effort h_i and branching effort b_i of each stage. The delay of each stage is then

$$d_i = g_i h_i b_i + p_i$$

where g_i is the logical effort of this stage, h_i is the electrical effort or the size ratio of next stage with this one, b_i is the branching effort and p_i is the parasitic delay of this stage.

We define the *path logical effort* as the product of logical effort of all stages on a given path.

$$G = \prod_{i=1}^{N} g_i \tag{8.18}$$

Similarly, we define the *path electrical effort* as the product of electrical effort of all stages on a given path.

$$H = \prod_{i=1}^{N} h_i \qquad \text{where} \qquad h_i = \frac{C_{out_i}}{C_{in_i}} \quad \text{for stage i.} \tag{8.19}$$

If there is no branching, the load on a particular stage is just the input capacitance of the next stage, that is: $C_{out_i} = C_{in_{i+1}}$. When we multiply all electrical efforts along a path, all except the first and last capacitances cancel. Thus we get

$$H = \frac{C_L}{C_{in_1}} \tag{8.20}$$

Where C_L is the final load capacitance and C_{in_1} is the input capacitance of the first stage. If, however, there is branching at some stage i, we need to correct for higher loading at this stage. We do this by defining the branching effort as:

$$b = \frac{C_{onpath} + C_{offpath}}{C_{onpath}} \tag{8.21}$$

for every stage. where $C_{onpath} = C_{in_{i+1}}$, is the load capacitance of the gate being driven along the path being analyzed, while $C_{offpath}$ is the capacitance presented by the gates which are not on the path. Now, $C_{out_i} = b_i C_{in_{i+1}}$. C_{out_i} includes not only $C_{in_{i+1}}$ but also the input capacitance of other logic gates which are not on the path under consideration.

The output capacitance included in the definition of H is now multiplied by this correction factor at every stage.

$$b_i h_i = \frac{C_{onpath_i} + C_{offpath_i}}{C_{onpath_i}} \times \frac{C_{onpath_i}}{C_{in_i}} \tag{8.22}$$

If there is no branching at a stage, the corresponding b_i value is 1 since $C_{offpath} = 0$ and so, multiplication by b_i does not change any thing. If, however, there is branching at the i'th stage, multiplying h_i by b_i corrects the output capacitance, because

$$b_i h_i = \frac{C_{onpath_i} + C_{offpath_i}}{C_{onpath_i}} \times \frac{C_{onpath_i}}{C_{in_i}} = \frac{C_{onpath_i} + C_{offpath_i}}{C_{in_i}} \tag{8.23}$$

We define the branching effort of the whole path, denoted by B, as the product of the branching effort at each stage along the path.

$$B = \prod_{i=1}^{N} b_i \tag{8.24}$$

We can now define the *path effort*, F as the product of all logical efforts and branch corrected electrical efforts.

$$F = \prod_{i=1}^{N} g_i \prod_{i=1}^{N} b_i h_i = \prod_{i=1}^{N} g_i \prod_{i=1}^{N} b_i \prod_{i=1}^{N} h_i \tag{8.25}$$

So,

$$F = GBH \qquad \text{where } G = \prod_{i=1}^{N} g_i, B = \prod_{i=1}^{N} b_i, \text{ and } H = \prod_{i=1}^{N} h_i \tag{8.26}$$

The advantage of using branching correction separately from electrical effort is that H retains its cancellation property for capacitance of intermediate stages and is defined only by the final load and the input capacitance.

$$H = \frac{C_L}{C_{in_1}} \tag{8.27}$$

The equation that defines the path effort looks quite similar to the definition of the stage effort in Equation 8.14, which defined the effort for a single logic gate. Notice, however, that unlike the stage effort f, the path effort F does not define the delay of the path. The total delay is the *sum* of individual delays and not their product.

$$D = \sum d_i = \sum g_i b_i h_i + \sum p_i \tag{8.28}$$

Still, F is a useful quantity for optimisation of path delays as we shall see.

The path delay, D, is the sum of the delays of each of the N stages of logic in the path. As in the expression for delay in a single stage (Equation 8.15), we separate the contribution to path delay from effort and the delay due to parasitic capacitances.

$$D = \sum d_i = D_F + P \tag{8.29}$$

where D_F is the path effort delay, while P is the *path parasitic delay*. The path effort delay is simply $D_F = \sum g_i b_i h_i$ and the path parasitic delay is $P = \sum p_i$. In order to choose the optimum sizes of gates to minimize the total delay through the path, we need to minimize the above expression with respect to the transistor widths (and hence capacitances) of every stage. We have

$$D = \sum d_i = D_F + P = \sum g_i b_i h_i + P = \sum g_i b_i \frac{c_{i+1}}{C_i} + P \tag{8.30}$$

Notice that p_i (and hence P) are size independent and we can only minimize D_F by choosing optimum sizes of gates. Setting the derivative of D with respect to C_i to zero, and noticing that only two terms in the series expansion of D_F involve C_i, we can write

$$0 = \frac{\partial}{\partial C_i}\left(g_{i-1}b_{i-1}\frac{C_i}{C_{i-1}} + g_i b_i \frac{C_{i-1}}{C_i}\right) \tag{8.31}$$

This leads to

$$g_{i-1}b_{i-1}\frac{1}{C_{i-1}} - g_i b_i \frac{C_{i+1}}{C_i^2} = 0 \tag{8.32}$$

Which gives

$$g_{i-1}b_{i-1}\frac{C_i}{C_{i-1}} = g_i b_i \frac{C_{i+1}}{C_i} \qquad \text{hence} \qquad f_{i-1} = f_i \tag{8.33}$$

Thus, *the path delay is minimized when each stage in the path has the same stage effort, f.* The value of f which minimizes the path delay is denoted as $\hat{f}$. Since we had defined F as $F = \prod_{i=1}^{N} f_i$, in the optimum case, $F = \hat{f}^N$. In other words, the minimum delay is achieved when each stage effort is

$$\hat{f} = b_i g_i h_i = F^{1/N} \tag{8.34}$$

For this optimum effort, we obtain

$$\hat{D} = NF^{1/N} + P = N(GBH)^{1/N} + P \tag{8.35}$$

$\hat{D}$ is the minimum delay possible for this path. We achieve this minimum delay by appropriate sizing of transistors in each stage of the logic path. Transistor sizes must be so chosen that the stage delay f is the same for all stages in the logic path. Equations 8.34 and 8.14 combine to require that each logic stage be designed so that

$$\hat{h}_i \equiv \frac{C_{out_i}}{C_{in_i}} = \frac{F^{1/N}}{b_i g_i} \tag{8.36}$$

We start with the last stage, where the output capacitance is known ($=C_L$). Since the output capacitance is known, the input capacitance can be calculated from Equation 8.36. This gives the scale factor for this stage from which, geometries of transistors can be computed.

Since $C_{out_i} = b_i C_{in_{i+1}}$, we can write the recursive relation

$$C_{in_i} = g_i b_i \frac{C_{in_{i+1}}}{F^{1/N}} \tag{8.37}$$

C_{in} for every stage can now be determined using the above recursive relation (Equation 8.37). From C_{in}, we can calculate the geometry of transistors for this stage.

We can equivalently start from the input side, computing the output capacitance and hence the input capacitance of the next stage. This can be continued till we reach the final output.

8.5.1 An example: 8-input AND network

When a large number of inputs must be combined, there are several options for the structure of the circuit. Figure 3 shows three possibilities for computing the AND function of eight inputs. Which one is best? Let us take $\gamma = 2$ and $p_{inv} = 0.6$ for this example. The parasitic delay of

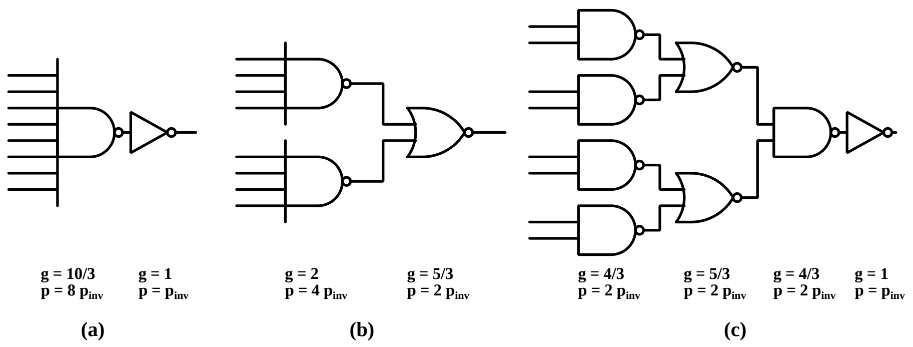


Figure 8.6: Three circuits that compute the AND function of eight inputs.

n input NANDs and NORs will then be $0.6n$. Recalling that the path logical effort, G, is the product of the logical efforts of the logic gates along the path, we find that $G = 10/3 \times 1 = 3.33$ for configuration a, $6/3 \times 5/3 = 3.33$ for configuration b, and $4/3 \times 5/3 \times 4/3 \times 1 = 2.96$ for configuration c. These figures can be used in the delay equation (8.35) to find the minimum delay that can be obtained from each circuit. The following equations also include an estimate of parasitic delays, obtained by summing the parasitic delays of each of the logic gates along the path:

$$\text{Configuration a} \quad D = \; 2(3.33H)^{1/2} + 5.4 \tag{8.38}$$

$$\text{Configuration b} \quad D = \; 2(3.33H)^{1/2} + 3.6 \tag{8.39}$$

$$\text{Configuration c} \quad D = \; 4(2.96H)^{1/4} + 4.2 \tag{8.40}$$

It is clear from these equations that configuration b will always be better than a.

Choosing between configurations b and c depends on the electrical effort H, that must be borne by the network. When $H = 1$, delay in configuration b is 7.25, and in configuration c is 9.5. Therefore in this case, configuration b will be best. However, if $H = 12$, the delay in configuration b is 16.24, while for configuration c it is 13.97. Thus, configuration c will be best in this case. The equations show that for high electrical effort, configuration c yields the least delay because the $H^{1/4}$ factor dominates. (In the current example, configuration c is best for $H > 5.68$).

To illustrate the computation of transistor sizes to achieve least delay, consider a case where $C_{in} = 4$ units and $C_{out} = 64$ (so that $H = 16$). As discussed above, configuration c - which is a 4 stage design - is best for $H > 5.68$. So we shall choose this configuration for our example design.

- Unit of capacitance is the input capacitance of the reference inverter.
- Unit of width is the width of the n channel transistor in the reference inverter.
- So $1 + \gamma$ units of width correspond to 1 unit of capacitance.
- The input capacitance of a reference logic gate, obtained from the reference inverter by series parallel rules, is g.
- Thus, $(C_{in} = g) \Rightarrow$ scale factor $= 1$ over the reference logic gate.

For computation of transistor geometries, we calculate the value of C_{in} for each stage using the optimum value of stage effort $\hat{f}$. Then, Given C_{in}, we can determine the geometry of individual transistors in each stage.

Unit of capacitance is the input capacitance of the reference inverter, while the unit of width is the width of the n channel transistor in the reference inverter.
So $1 + \gamma$ units of width correspond to 1 unit of capacitance. Also, the input capacitance of a reference logic gate, obtained from the reference inverter by series parallel rules, is g.

Thus, $(C_{in} = g) \Rightarrow$ scale factor $= 1$ over the reference logic gate. Then, for any given C_{in}, scale factor $= C_{in}/g$. So, if we know C_{in}, we can find the scale factor by dividing it by g, and then multiply all transistor widths in the reference logic gate by this scale factor to get individual transistor geometries. We can take the logic path from any of the inputs to the final output. On this path, we shall encounter:

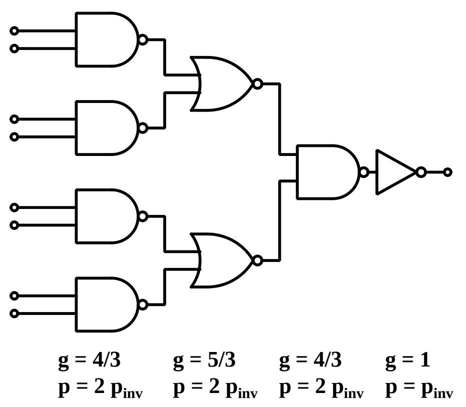


Figure 8.7: 4 stage implementation of 8 input AND

1. a 2 input NAND (g = 4/3, b = 1),
2. a 2 input NOR (g = 5/3, b = 1),
3. a 2 input NAND (g = 4/3, b = 1), and
4. an inverter (g = 1, b = 1)

$$G = \frac{4}{3} \times \frac{5}{3} \times \frac{4}{3} \times 1 = 80/27 = 2.963 \quad B = 1, \quad H = \frac{64}{4} = 16$$

$$F = GBH = 47.4074, \quad \text{So} \quad \hat{f} = 47.4074^{1/4} = 2.624$$

In this computation, We begin from the load end. (We can also start from the first stage and go towards the load for the recursive calculation).

Last stage: Inverter

The last stage is an inverter, with $g = 1, b = 1$.

$$\hat{f} = 2.624 = gbh = 1 \times 1 \times h. \quad \text{So} \quad h = 2.624$$

$$h = \frac{C_{out}}{C_{in}} = \frac{64}{C_{in}} = 2.624$$

$$\text{So} \quad C_{in} = \frac{64}{2.624} = 24.39$$

So the final stage inverter is scaled up by 24.39 compared to the reference inverter. Taking the n channel transistor width in the reference inverter as the unit,

n-channel transistor width = 24.39
p-channel transistor width = $\gamma \times 24.39 = 48.78$.

Third stage: 2 input NAND

The stage driving the inverter is a 2 input NAND.
So $g = 4/3, b = 1, C_{out} = 24.39$.

$$\hat{f} = 2.624 = gbh = \frac{4}{3} \times 1 \times h$$
$$\text{So} \quad h = \frac{2.624 \times 3}{4} = 1.968$$
$$h = \frac{C_{out}}{C_{in}} = \frac{24.39}{C_{in}} = 1.968$$
$$\text{Therefore} \quad C_{in} = \frac{24.39}{1.968} = 12.3936$$

Scale factor for this stage is $12.3936/g = (12.3936 \times 3)/4 = 9.2952$.
The reference 2 input NAND gate has n channel transistor width = 2, and p channel transistor width = 2.
Therefore the transistor geometries in this stage will be $2 \times 9.2952 = 18.59$ for both n and p channel transistors.

Second stage: 2 input NOR

The stage driving the 2 input NAND is a 2 input NOR.
So $g = 5/3, b = 1, C_{out} = 12.3936$.

$$\hat{f} = 2.624 = gbh = \frac{5}{3} \times 1 \times h. \quad \text{So} \quad h = 1.5744$$
$$h = \frac{C_{out}}{C_{in}} = \frac{12.3936}{C_{in}} = 1.5744$$
$$\text{Therefore} \quad C_{in} = \frac{12.3936}{1.5744} = 7.872$$

Scale factor for this stage is $7.872/g = (7.872 \times 3)/5 = 4.7232$.
The reference 2 input NOR gate has n channel transistor width = 1, and p channel transistor width = 4.
n-channel transistor width = 4.72,
p channel transistor width = $4 \times 4.7232 = 18.89$

First stage: 2 input NAND

Finally, we come to the first stage which is a 2 input NAND.
So $g = 4/3, b = 1, C_{out} = 7.872$.

$$\hat{f} = 2.624 = gbh = \frac{4}{3} \times 1 \times h. \quad \text{So} \quad h = 1.9680$$

$$h = \frac{C_{out}}{C_{in}} = \frac{7.872}{C_{in}} = 1.968$$

$$\text{Therefore} \quad C_{in} = \frac{7.872}{1.968} = 4$$

This agrees with our specification that the input capacitance of the first stage should be equivalent to 4 inverters.
The scale factor will be $4/g = 4/(4/3) = 3$. In the reference NAND, all transistors have a width = 2. so in the first stage, all transistors will have a width $= 3 \times 2 = 6$.

Design of 8 input AND: Summary of results

The following table gives the geometries of transistors in all stages:

Stage	I	II	III	IV
Logic type	2in NAND	2in NOR	2in NAND	Inverter
g	4/3	5/3	4/3	1
C_{in}	4	7.87	12.39	24.39
Scale Factor	3	4.7232	9.2952	24.39
n width	6	4.72	18.59	24.39
p width	6	18.89	18.59	48.78
Parasitic Delay	1.2	1.2	1.2	0.6

The total delay of the 4 stage implementation is:

$$4\hat{f} + \sum P_i = 4 \times 2.624 + 4.2 = 10.496 + 4.2 = 14.7$$

8.6 Optimizing the path length

Equation 8.34 requires that the number of stages in the logic path be known for optimizing the total delay. However, this may not be the optimum path length and we can some times get a faster circuit by buffering intermediate outputs in this logic path.

We assume that there is a logic path containing n_1 stages and we are free to add n_2 inverters to this path, if that results in a lower overall delay. We now consider the optimization of this logic path containing $N = n_1 + n_2$ stages. We shall assume that there is no requirement for n_2 to be even. (This implies that an inverted output is equally acceptable or else, the logic path and inputs can be suitably altered to produce the desired output). The optimization problem is to find the scale factors for each of the $N = n_1 + n_2$ stages, such that the delay is minimum.

The path effort $F = GBH$ for the n_1 stages of logic is known. This is because the logical effort of each of the logic gates is known to us, so that G may be evaluated. Branching, if any, in the logic chain is also known, so B can be evaluated. finally, H depends only on the load capacitance and input capacitance, which is the starting specification for the optimization.

Addition of n_2 inverters does not change the value of G, since $g = 1$ for each of the n_2 inverters. Similarly, the inverters do not introduce any branching, so B remains the same. Finally, H is defined by the final load and the input capacitance of the first logic element, which is not changed by the inserted inverters. Therefore $F = GBH$ for the $N = n_1 + n_2$ stages (including n_2 inverters) is the same as the F for n_1 logic stages.

Additional inverters in the logic chain permit sharing the effort over a larger number of stages, which can reduce the total delay. We first find the optimum value of N. If $N > n_1$, we shall have the opportunity of reducing the delay by adding inverters.

The total delay in the N stages is the sum of delays of n_1 logic stages and n_2 inverters. For optimum delay, the stage effort should be equal for all the N stages. Therefore, the stage effort of logic stages as well as the inverters is the same and is $= F^{1/N}$. So the total delay is:

$$\hat{D} = NF^{1/N} + \sum_{i=1}^{n_1} p_i \; + (N - n_1)p_{inv} \tag{8.41}$$

The first term in Equation 8.41 above is the effort delay of N stages. The sum of parasitic delays of n_1 logic gates gives us the second term. Finally, the parasitic delay of $n_2 = N - n_1$ inverters gives the third term.

We define the optimum stage effort $\rho \equiv F^{1/N}$. Then

$$\hat{D} = N(\rho + p_{inv}) + \sum_{i=1}^{n_1} p_i \; - n_1 p_{inv} \tag{8.42}$$

Since $\rho \equiv F^{1/N}$, $N = \ln F / \ln \rho$. Therefore,

$$\hat{D} = \frac{\ln F}{\ln \rho}(\rho + p_{inv}) + \sum_{i=1}^{n_1} p_i \; - n_1 p_{inv} \tag{8.43}$$

It is to be noticed that F, n_1, p_i and p_{inv} are given constants. We now optimize the total delay with respect to ρ by setting the derivative of $\hat{D}$ with respect to ρ to zero. Differentiating by parts, we get

$$\frac{\partial \hat{D}}{\partial \rho} = 0 = -\frac{\ln F}{(\ln \rho)^2} \cdot \frac{1}{\rho} \cdot (\rho + p_{inv}) + \frac{\ln F}{\ln \rho}(1) \tag{8.44}$$

Therefore,

$$\frac{\ln F}{\ln \rho} = \frac{\ln F}{(\ln \rho)^2} \cdot \frac{1}{\rho} \cdot (\rho + p_{inv}) \tag{8.45}$$

and so,

$$1 = \frac{1}{\rho \ln \rho}(\rho + p_{inv}) \tag{8.46}$$

Which gives

$$\rho + p_{inv} = \rho \ln \rho \tag{8.47}$$

Which can be written as

$$p_{inv} + \rho(1 - \ln \rho) = 0 \tag{8.48}$$

It is interesting to note that this condition is independent of F and the value of ρ is uniquely defined by p_{inv}.

Equation 8.48 cannot be solved in closed form and either iterative solutions or graphical solutions have to be used to determine ρ from p_{inv}. In the special case when $p_{inv} = 0$, we have

$$\rho(1 - \ln \rho) = 0 \quad \text{so} \quad \ln \rho = 1 \quad \text{which gives } \rho = e$$

This corresponds to the case of tapered inverter that we had solved before.

For non-zero values of p_{inv}, Equation 8.48 can be solved iteratively using the Newton Raphson technique. We can write Equation 8.48 as

$$f(\rho) \equiv \rho - \rho \ln \rho + p_{inv} = 0 \tag{8.49}$$

$$\text{Then} \qquad f'(\rho) = 1 - \ln \rho - \rho \frac{1}{\rho} = -\ln \rho \tag{8.50}$$

If we have a guess solution g for this equation, the next improved guess according to Newton Raphson technique is:

$$g_{next} = g - \frac{f(g)}{f'(g)} = g + \frac{g - g \ln g + p_{inv}}{\ln g} = \frac{g + p_{inv}}{\ln g} \tag{8.51}$$

We know that for $p_{inv} = 0$, the value of ρ is e. A good guess value to start iterations will be $\rho = 3$. Let us illustrate the iterative method by taking $p_{inv} = 1$. The successive values obtained from Equation 8.51, starting with $\rho = 3$ are:
3.0, 3.6410, 3.5914, 3.5911, 3.5911,. . .

The value converges to 4 decimal places within 3 to 4 iterations.

Similarly, for $p_{inv} = 0.6$, the successive solutions starting from 3 are:
3.2769, 3.2664, 3.2664 …

We can also solve Equation 8.47 graphically by plotting $\rho + p_{inv}$ and $\rho \ln \rho$ as functions of ρ. The value of ρ at which these two intersect is the solution of the equation.

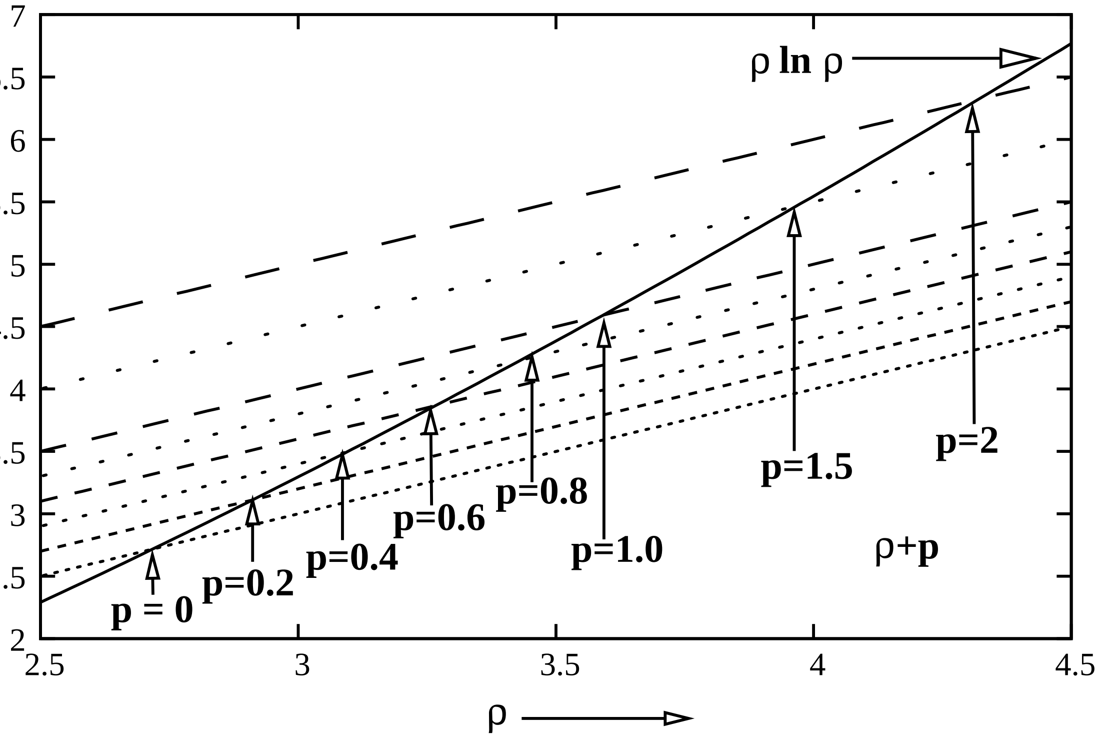


Figure 8.8: Graphical solution of the equation $\rho + p_{inv} = \rho \ln \rho$.

Either way, one can evaluate ρ if the parasitic delay of an inverter is given. Table 8.2 below gives the values of ρ and the corresponding stage delay for several values of p_{inv}.

Once ρ is known, we can evaluate the optimum number of logic stages for a given path effort by $N = \ln F / \ln \rho$. This value will be fractional in general and needs to be zapped to the nearest integer. After this rounding off, the stage effort has to be re-calculated as $f = F^(1/N)$. If $N > n_1$ (where n_1 is the number of logic stages which are necessary for implementing the desired logic function), we can insert $N - n_1$ inverter stages to optimize the overall delay.

Thus, $\hat{f}$ is the optimum stage effort when the number of logic stages is fixed and known.

p_{inv}	ρ	$\ln \rho$	$d = \rho + p_{inv}$
0	2.718 (e)	1.000	2.718
0.2	2.912	1.069	3.11
0.4	3.093	1.129	3.49
0.6	3.266	1.184	3.87
0.8	3.432	1.233	4.23
1.0	3.591	1.278	4.59
1.5	3.967	1.378	5.47
2.0	4.319	1.463	6.32

Table 8.2: ρ as a function of p_{inv}. The table also gives the optimum delay.

In this case the stage effort is F^{1/n_1} and the total delay is $n_1 F^{1/n_1} + \sum_{i=1}^{n_1} p_i$ where n_1 is the known number of logic stages.

If we have the freedom of inserting a number of inverters in the logic path to optimize delay, we follow the following procedure:

1. From p_{inv}, find the ideal stage effort ρ by solving $p_{inv} + \rho(1 - \ln \rho) = 0$. This can be done through iterative solutions or graphically as described earlier.
2. Once ρ is known, find the number of stages as $\ln F / \ln \rho$. The nearest integer to the value so found is the optimum number of stages N.
3. If $N > n_1$, insert $(N - n_1)$ inverters anywhere in the logic path. If $N \leq n_1$, take $N = n_1$.
4. The stage effort is now adjusted to $f = F^{1/N}$.
5. Given this value of f, we can start from the last stage and work backwards as earlier to calculate all transistor geometries.
6. For the last stage the output capacitance is known (=C_L). The input capacitance can be calculated from Equation 8.36.

$$C_{in_N} = b_N g_N \frac{C_L}{f}$$

   This gives the scale factor for this stage from which, geometries of transistors in the last stage can be computed.

7. For each preceding stage, we use the recursive relation 8.37

$$C_{in_i} = g_i \frac{b_i C_{in_{i+1}}}{f}$$

8. From C_{in_i}, we can calculate the scale factor, and hence the geometry of all transistors for this stage.

The optimum number of stages will depend on $F = GBH$. Because N must be an integer, a range of values of F will require the same number of stages. Table 8.6 gives the optimum number of stages for ranges of F values. This table assumes the value of p_{inv} to be 1.0. It is

Path effort F	Optimum No. of stages $\hat{N}$	Min. Delay $\hat{D}$	range of values of Stage effort f
0-5.83	1	1.0-6.8	0-5.8
5.83-22.3	2	6.8-11.4	2.4-4.7
22.3-82.2	3	11.4-16.0	2.8-4.4
82.2-300	4	16.0-20.7	3.0-4.2
300-1090	5	20.7-25.3	3.1-4.1
1090-3920	6	25.3-29.8	3.2-4.0
3920-14200	7	29.8-34.4	3.3-4.0
14200-51000	8	34.4-39.0	3.3-3.9
51000-184000	9	39.0-43.6	3.3-3.9
184000-661000	10	43.6-48.2	3.4-3.8

Table 8.3: Optimum number of stages for different F values. This table assumes $p_{inv} = 1$, so $\rho = 3.59$.

interesting to ask how much the delay for a properly optimized circuit is changed by using the wrong number of stages. The answer, as shown in Table 8.4, is that delay is quite insensitive

$N/\hat{N}$	$D/\hat{D}$
0.25	7.42
0.5	1.46
0.7	1.09
1/0	1/00

$N/\hat{N}$	$D/\hat{D}$
1.4	1.06
2.0	1.24
3.0	1.62
4.0	2.01

Table 8.4: The relative delay of a network, $D/\hat{D}$, as a function of the relative error in the number of stages used, $N/\hat{N}$. Assumes $p_{inv} = 0.6$.

to the number of stages, provided the deviation from optimum is not too large. As the table shows, doubling the number of stages from optimum increases the delay only 24%. Using half as many stages as the optimum increases the delay by 46%. Thus one need not slavishly stick to exactly the correct number of stages. It is slightly better to err in the direction of using too many stages than too few. A stage or two more or less in a design with many stages will make little difference, provided proper transistor sizes are used. Only when very few stages are required does a change of one or two stages make a large difference.

8.7 Designing Forks

We often need a signal and its complement simultaneously. When the signal changes its value, the complement should also change *at the same time*. Otherwise, there will be a short interval during which both the signal and its complement are TRUE or FALSE simultaneously. This can lead to malfunction.

The trivial solution of using an inverter to generate the complement will not meet this requirement – since in this case, the complement will switch an inverter delay later. We can meet the requirement of a signal and its complement changing almost simultaneously by using a circuit with a common input feeding two branches with different number of inversions. The delay of the two branches must be equalized as closely as possible. Such a circuit is called a "Fork" due to its shape.

In a common configuration, the two branches of a fork will have n and (n+1) inverters. A fork with 3 and 4 branches is shown in Figure 8.9. A fork is named with the number of

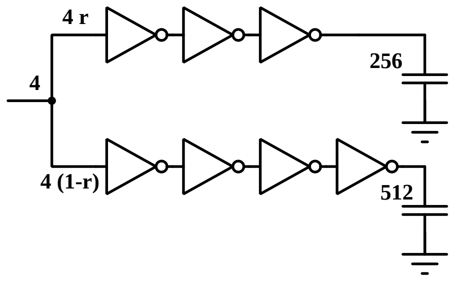


Figure 8.9: A 3-4 Fork

inverters in its branches. For example the circuit in Figure 8.9 is a 3-4 fork. Specifications for the fork will include the total capacitive load placed by both branches at the input node and the terminal load at the end of each branch. Notice that the terminal loads at the two outputs need not be the same! This is because typically, one arm of the fork drives nMOS transistors while the other drives pMOS transistors – and their sizes may not be the same.

8.7.1 Designing a Fork

It is easy to design each individual arm of the fork for minimum delay using logical effort techniques. However, how do we ensure that the optimum delays of the two arms are equal?

To equalize these delays, we need to introduce another design parameter, which determines in what ratio the drive provided by the preceding stage is split between the two branches.

Take the 3-4 fork shown in Figure 8.9 as an example. The specification demands that the two branches together should place a load of 4 on the upstream driver. If we divide the input capacitances of the first inverters in the two branches in the ratio 4r and 4(1-r), the total input capacitance will be 4 for all choices of r.

By choosing an appropriate value of r between 0 and 1, we can adjust the optimized delays of the two branches, such that these are equal. Design of a fork essentially requires the evaluation of such a suitable value for the parameter r.

Notice that for a fork with n and n+1 inverters, the difference of delay is a smaller fraction of the total delay if n is large. The choice of n is a trade off between the robustness of delay matching and power dissipation as well as complexity.

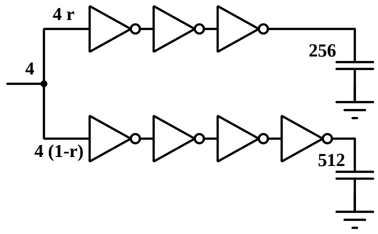
8.7.2 Design Example

Let us go through the design of the 3-4 fork shown earlier. We want to design a 3-4 fork with the total input capacitance (to be driven by the up-stream driver) equal to 4 times the unit inverter input capacitance. The final load on the branch with 3 inverters is equivalent to 256 minimum inverters, while that on the 4 inverter branch is equivalent to 512 minimum inverters. Assume $P_{inv} = 2.0, \gamma = 2.2$.

The input capacitance is divided in the ratio r:(1-r) for the 3 and 4 inverter branches of the fork respectively. We want to evaluate the value of r such that the optimum delay in the two branches is equal. Since all gates are inverters and there is no branching in either arm, all g and b values are 1 in this example. Therefore $G = 1$, and $B = 1$ for both branches.Since the input capacitance is dependent on r, the value of H is not known for either branch and depends on r.

The requirement for equal delay leads to a non-linear equation in r and we shall solve it using Newton Raphson iterations in this example. We shall use a starting guess value of $r = 0.5$.

The Upper Branch



For the upper branch, the input capacitance is $4r$, while the output capacitance is 256. We shall use the subscript 1 for the design parameters associated with the upper branch. $H_1 = 256/4r = 64/r$.
$G_1 = B_1 = 1$, so we get $F_1 = G_1B_1H_1 = 64/r$.

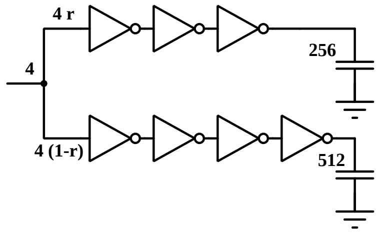
Since this a 3 stage branch, this leads to:

$$\hat{f}_1 = \left(\frac{64}{r}\right)^{1/3} = 4r^{-1/3}$$

Thus the delay through the upper branch is

$$D_1 = 3\hat{f}_1 + 3P_{inv} = 12r^{-1/3} + 3P_{inv}$$

The lower Branch



We shall use the subscript 2 for the design parameters associated with the upper branch. For this branch, the input capacitance is $4(1-r)$, while the output capacitance is 512. Thus $H_2 = 512/4(1-r) = 128/(1-r)$. $G_2 = B_2 = 1$, so we get $F_2 = G_2B_2H_2 = 128/(1-r)$.

This is a 4 stage branch, so

$$\hat{f}_2 = \left(\frac{128}{1-r}\right)^{1/4} = 3.3636(1-r)^{-1/4}$$

and the delay through the lower branch is

$$D_2 = 4\hat{f}_2 + 4P_{inv} = 13.4543(1-r)^{-1/4} + 4P_{inv}$$

Equating the two delays, we get $D_2 - D_1 = 0$, or

$$13.4543(1-r)^{-1/4} - 12r^{-1/3} + P_{inv} = 0$$

Defining $f(r) \equiv 13.4543(1-r)^{-1/4} - 12r^{-1/3} + P_{inv}$, We seek the value of r which will make $f(r) = 0$. The derivative of $f(r)$ may be written as

$$f'(r) = -\frac{13.4543}{4}(1-r)^{-5/4}(-1) + \frac{12}{3}r^{-4/3} = 3.3636(1-r)^{-5/4} + 4r^{-4/3}$$

We can now solve this non-linear equation using Newton Raphson iterations.
Taking the initial guess for r as 0.5, successive values for r can be tabulated as:

r	f(r)	f'(r)	next r
0.5	2.88095	18.0794	0.34065
0.34065	-0.251373	22.4743	0.351835
0.351835	-0.00333406	21.8878	0.351987
0.351987	-5.78369e-07	21.8803	0.351987
0.351987	-1.42109e-14	21.8803	0.351987

Thus, r = 0.352 will equalize delays.

Sizing gates in the Fork

In order to determine gate and transistor sizes, we first evaluate the input capacitance for all inverters. Since the unit of capacitance is the input capacitance of the unit inverter, this value is the scale factor for the inverter. Using this, individual transistor geometries can be determined.

For the upper branch,

$$\hat{f}_1 = \frac{4}{r^{1/3}} = \frac{4}{0.351987^{1/3}} = \frac{4}{0.70606} = 5.665232$$

All stages are inverters with $g = 1, b = 1$. Since $\hat{f} = gbh = 5.665232$, the value of $h = 5.665232$ for all stages.

- The first inverter should have an input capacitance of $4r = 4 \times 0.352 = 1.408$
- The next inverter should have an input capacitance of $1.408 \times h = 1.408 \times 5.665232 = 7.976$.
- Input capacitance for the final inverter will be $7.976 \times h = 7.976 \times 5.665232 = 45.188$.
- The final inverter can drive a load of $45.188 \times 5.665232 = 256$ as required.

Alternatively, we could have started with the output. As before:
$\hat{f} = gbh = 5.665232, g = 1, b = 1$, so $h = 5.665232$ for all stages. Final $C_{out} = 256$.

- For the last inverter, $C_{in} = 256/5.665232 = 45.188$. This becomes the output capacitance of the second inverter.
- for the second inverter, $C_{in} = 45.188/h = 45.188/5.665232 = 7.976$. This becomes the output capacitance of the first inverter.
- Finally, since the output capacitance of the first inverter is 7.976, its input capacitance is $7.976/5.665232 = 1.408$. This agrees with the value $4r$ as required.

Input capacitance of 1 corresponds to n channel width of 1 and p channel width of γ. Therefore an inverter stage with input capacitance of C_{in} will have n channel transistor width of C_{in} and p channel transistor width of $\gamma \times C_{in}$.

First Inverter		Second Inverter		Third Inverter	
$C_{in} = 1.408$		$C_{in} = 7.976$		$C_{in} = 45.188$	
n width	p width	n width	p width	n width	p width
1.408	3.10	7.976	17.548	45.188	99.413

For the lower branch, $\hat{f}_2 = 3.3636/(1-r)^{1/4} = 3.749$.
Again all stages are inverters with $g = 1, b = 1$. Therefore, for all stages, $h = 3.749$.

- Input capacitance of the first inverter is $4 \times (1 - r) = 2.592$.
- Input capacitance of the following three inverters should be $2.592 \times 3.749 = 9.717$, $9.717 \times 3.749 = 36.43$ and $36.43 \times 3.749 = 136.57$.
- The final inverter can drive a capacitance of $136.57 \times 3.749 = 512$ as expected.

Given the input capacitance values for all inverters,
the n channel widths are equal to the capacitance,
while the p channel widths are $\gamma(= 2.2)$ times this value.

Gate	C_{in}	nMOS width	pMOS width
First Inverter	2.592	2.59	5.70
Second Inverter	9.717	9.72	21.38
Third Inverter	36.43	36.43	80.15
Fourth Inverter	136.57	136.57	300.46

Verification of equality of delays

Delay for the upper branch = $3\hat{f}_1 + 3P_{inv} = 3 \times 5.665 + 6 = 22.995$.

Delay for the lower branch = $4\hat{f}_2 + 4P_{inv} = 4 \times 3.749 + 8 = 22.995$.

To see the robustness of the design, Let us assume that the actual load capacitors in both the branches are higher by 10%.

Without changing inverter sizes, what are the delays with the changed values and how much is the difference in delays of the two branches?

Since inverter sizes remain the same, all inverters except the final one see the same load. Therefore change in the final load capacitor will change the delay of the last stage only. The remaining delays will remain the same.

$$\begin{aligned} D_1 &= 2\hat{f}_1 + 1.1 \times \hat{f}_1 + 3P_{inv} = 3.1 \times 5.665 + 6 = 17.562 + 6 = 23.562 && (8.52) \\ D_2 &= 3\hat{f}_2 + 1.1 \times \hat{f}_2 + 4P_{inv} = 4.1 \times 3.749 + 8 = 17.562 + 6 = 23.371 && (8.53) \end{aligned}$$

The difference in delays is therefore $23.562 - 23.371 = 0.192$. (The upper branch is slower by this amount).

Chapter 9

Adders

9.1 Half Adder

The truth table for addition of two bits is:

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From this, we can see that

$$\text{sum} = A \cdot \overline{B} + B \cdot \overline{A} \quad (9.1)$$

$$\text{carry} = A \cdot B \quad (9.2)$$

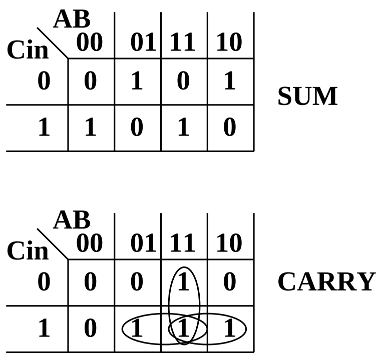
What do we do with the carry? Obviously, it must be added to more significant bits. Therefore, for subsequent stages, we need an adder with *three* inputs: A, B and Carry in. This kind of adder is called a full adder.

9.2 Full Adder

Truth Table for the addition of three bits is:

A	B	Cin	Sum	Cout
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

This leads to the Karnaugh map shown in fig. 9.1:



Cin \ AB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

SUM

Cin \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

CARRY

Figure 9.1: Karnaugh map for full adder

$$\text{So}\quad \text{sum} = \overline{A}\cdot\overline{B}\cdot C_{in} + \overline{A}\cdot B\cdot\overline{C_{in}} + A\cdot B\cdot C_{in} + A\cdot\overline{B}\cdot\overline{C_{in}}$$

$$C_{out} = A\cdot B + B\cdot C_{in} + C_{in}\cdot A = A\cdot B + C_{in}\cdot(A+B)$$

9.3 Ripple Carry adder

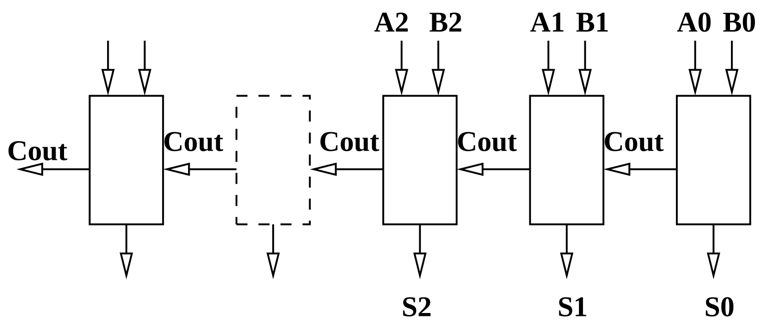


Figure 9.2: A ripple carry adder

- Carry out of one bit becomes Carry in of the next.
- This architecture is therefore called ripple carry adder.
- The critical delay path of the adder is the carry rippling from one bit to the next.

Because carry is on the critical path, Carry-out must be generated as quickly as possible. We need not optimize the delay of generating sum. We can in fact generate sum from

Carry out.

$$\overline{C_{out}} = \overline{A \cdot B + C_{in} \cdot (A+B)} \tag{9.3}$$
$$= (\overline{A} + \overline{B}) \cdot (\overline{C_{in}} + \overline{A} \cdot \overline{B}) \tag{9.4}$$
$$= \overline{A} \cdot \overline{C_{in}} + \overline{B} \cdot \overline{C_{in}} + \overline{A} \cdot \overline{B} \tag{9.5}$$
$$\overline{C_{out}} \cdot (A+B+C) = A \cdot \overline{B} \cdot \overline{C_{in}} + \overline{A} \cdot B \cdot \overline{C_{in}} + \overline{A} \cdot \overline{B} \cdot C_{in} \tag{9.6}$$

$$\text{sum} = A \cdot \overline{B} \cdot \overline{C_{in}} + \overline{A} \cdot B \cdot \overline{C_{in}} + \overline{A} \cdot \overline{B} \cdot C_{in} + A \cdot B \cdot C_{in} \tag{9.7}$$
$$= \overline{C_{out}} \cdot (A+B+C_{in}) + A \cdot B \cdot C_{in} \tag{9.8}$$

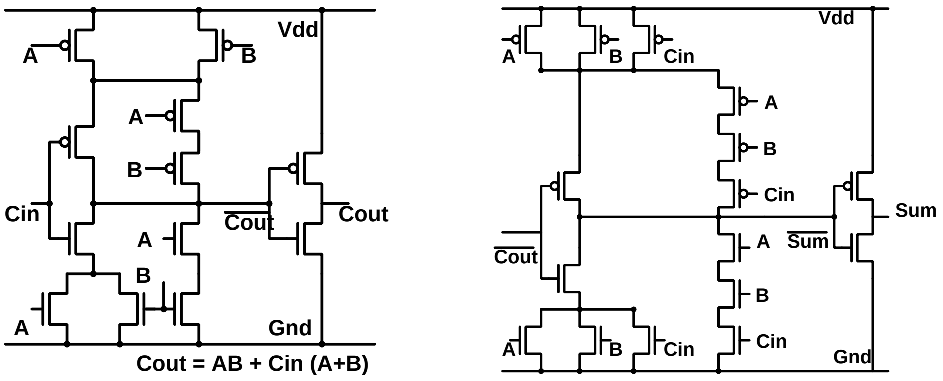


Figure 9.3: CMOS implementation for sum and carry

Both Sum and Carry show an interesting symmetry:

$$\text{sum} = \overline{A} \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C_{in}} + A \cdot \overline{B} \cdot \overline{C_{in}} + A \cdot B \cdot C_{in} \tag{9.9}$$
$$\overline{\text{sum}} = (A+B+\overline{C_{in}}) \cdot (A+\overline{B}+C_{in}) \cdot (\overline{A}+B+C_{in}) \cdot (\overline{A}+\overline{B}+\overline{C_{in}}) \tag{9.10}$$
$$= (A + A \cdot \overline{B} + A \cdot C_{in} + A \cdot B + B \cdot C_{in} + \overline{C_{in}} \cdot A + \overline{C_{in}} \cdot \overline{B}) \cdot \tag{9.11}$$
$$(\overline{A} + \overline{A} \cdot \overline{B} + \overline{A} \cdot \overline{C_{in}} + \overline{A} \cdot B + B \cdot \overline{C_{in}} + C_{in} \cdot \overline{A} + C_{in} \cdot \overline{B}) \tag{9.12}$$
$$= (A + B \cdot C_{in} + \overline{B} \cdot \overline{C_{in}}) \cdot (\overline{A} + B \cdot \overline{C_{in}} + \overline{B} \cdot C_{in}) \tag{9.13}$$
$$= A \cdot B \cdot \overline{C_{in}} + A \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot C_{in} + \overline{A} \cdot \overline{B} \cdot \overline{C_{in}} \tag{9.14}$$

This shows that the **same hardware** that produces sum from A, B and C_{in}, will produce $\overline{\text{sum}}$ if the inputs are changed to $\overline{A}$, $\overline{B}$ and $\overline{C_{in}}$

$$C_{out} = A \cdot B + C_{in} \cdot (A+B) \tag{9.15}$$
$$\overline{C_{out}} = \overline{A \cdot B + C_{in} \cdot (A+B)} \tag{9.16}$$
$$= (\overline{A} + \overline{B}) \cdot (\overline{C_{in}} + \overline{A} \cdot \overline{B}) \tag{9.17}$$
$$= \overline{A} \cdot \overline{C_{in}} + \overline{B} \cdot \overline{C_{in}} + \overline{A} \cdot \overline{B} \tag{9.18}$$

Thus the carry function also has the same property:
The same hardware which produces C_{out} from A, B and C_{in}, will produce $\overline{C_{out}}$ from $\overline{A}, \overline{B}$ and $\overline{C_{in}}$.

- In CMOS implementation, we interchange series and parallel configurations for the n and p channel transistors.
- This is to ensure that the pull up and pull down circuits are complementary.
- However, for sum and carry functions, we see that these functions are their own complements.
- Therefore, for implementing sum and carry, we can use the **same** configuration for n and p channel transistors.
- We use this to reduce the number of series connected transistors in pull up/pull down networks.

By making use of symmetry property of sum and carry, it is possible to simplify the implementations. These are called mirror gates because the n and p transistors have the **same**

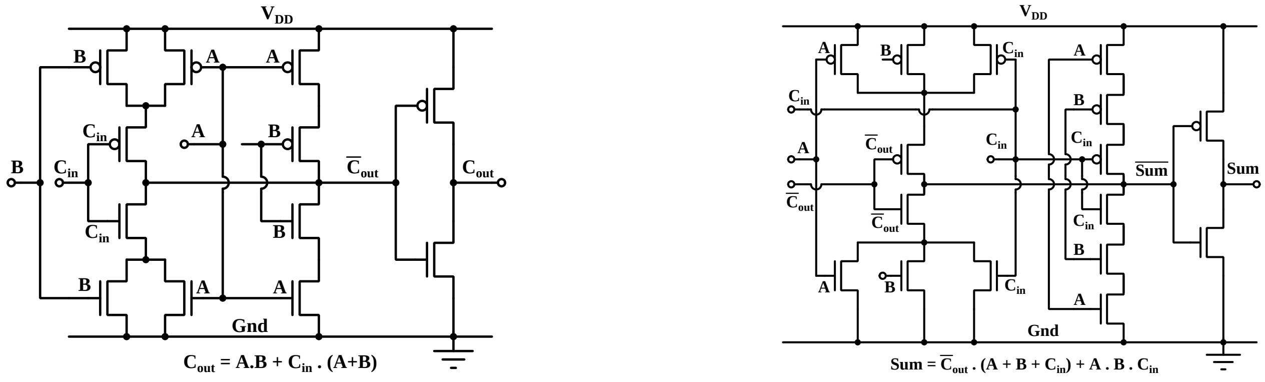

V_{DD}
B
C_{in}
A
$\overline{C}_{out}$
C_{out}
Gnd
C_{out} = A.B + C_{in} . (A+B)
$\overline{Sum}$
Sum
Sum = $\overline{C}_{out}$. (A + B + C_{in}) + A . B . C_{in}

Figure 9.4: Mirror gate implementation of sum and carry

series parallel combination. This is highly unusual.

The worst case delay of the ripple carry adder is linear in number of bits to be added. To reduce the delay per stage, we can eliminate the inverter from the carry output. All even bit adders accept a, b and C_{in} as inputs. The mirror gate gives $\overline{C_{out}}$ as the output. All odd bit adders accept $\overline{A}, \overline{B}$ and $\overline{C_{in}}$ as inputs and thus produce C_{out} as output. Outputs of all bits are now compatible with inputs of the next stage.

Extra inverters are required to produce $\overline{A}, \overline{B}$ and at the outputs to produce the proper result. However, these are not on the critical path, and do not add to the worst case delay. Extreme care needs to be taken in layout to ensure that the loading on the tree gate producing carry output is as small as possible.

9.4 Carry Look Ahead

Carry propagation is the critical path for a multi-bit adder. To speed up the adder, we would like an architecture where logic terms are classified as those dependent on carry and those which do not depend on carry. Then we can pre-compute all terms which do not depend on carry. When carry arrives, we can quickly compute the output carry and pass it on to the next stage.

Let us analyze what information can be pre-computed from A_i and B_i, which will help us in generating C_{out} quickly from C_{in}.

- When $A_i = 0$ and $B_i = 0$, C_{out} is 0, independent of C_{in}. We define this condition as 'Kill'. $K = \overline{A} \cdot \overline{B}$
- Similarly, when $A_i = 1$ and $B_i = 1$, C_{out} is 1, independent of C_{in}. We define this condition as 'Generate': $G = A.B$.
- Only when $A_i = 0$ and $B_i = 1$ or when $A_i = 1$ and $B_i = 0$,
  we need to wait for C_{in} to compute C_{out}.
  In both these cases, $C_{out} = C_{in}$.
- We call this condition as 'Propagate', and define $P = A.\overline{B} + \overline{A}.B$.

We define $K = \overline{A} \cdot \overline{B}, G = A.B$ and $P = A \oplus B$
Exactly one of K, G or P is true at any time.

When K = 1, C_{out} is 0, independent of C_{in}.
When G = 1, C_{out} is 1, independent of C_{in}.
When P = 1, $C_{out} = C_{in}$.

P is computed using an xor gate, which can be slow. However, the only difference between xor and or logic is when both inputs are 1, i.e. G = 1.

If we can ensure that G forces C_{out} to 1 irrespective of P, we can use the simpler 'or' logic to compute P.

C_{in} for bit i+1 is the C_{out} of bit i.
So we can write $C_{i+1} = G_i + P_i.C_i$
Notice that the Kill signal is not required.

If $G_i = 0$, $C_{i+1} = A \oplus B = A + B$ when $G = A.B = 0$
If $G_i = 1, C_{i+1} = 1$, and the value of P_i does not matter anyway.
So we can use P $= A + B$ instead of $P = A \oplus B$.

Now, we have the sequence:

$$C_{i+1} = G_i + P_i.C_i = G_i + P_i.G_{i-1} + P_i.P_{i-1}.C_{i-1} = \cdots$$

and so on, till we reach C_0.

Since all G_i, P_i and C_0 can be computed in parallel on arrival of the inputs, we can compute all sum and carry terms independently if we do not mind the added complexity.

$$C_{i+1} = G_i + P_i.C_i = G_i + P_i.G_{i-1} + P_i.P_{i-1}.C_{i-1} = \cdots$$

Unfortunately, static implementation of these gates has almost as much delay as the ripple carry implementation.

Therefore, the static implementation of computation of sum and carry terms as a logic expression depending on all A_i, B_i and C_0 is rarely used.

However, a dynamic implementation is still useful, and is known as the Manchester Carry Chain.

9.4.1 Manchester Carry Chain

When the clock is low, the output is unconditionally charged by the pMOS.
When the clock goes high, the output will be pulled low if G = 1 or if P = 1 and $\overline{C_{in}} = 0$.
In all other cases, the output will remain high. Thus this circuit implements the required logic.

This circuit can be concatenated for all bits and since P and G are ready before $\overline{C_{in}}$ arrives, the carry quickly ripples through from bit to bit.

Notice that the nMOS logic can be interpreted as:

$$\overline{P.Cin + G}$$

where C_{in} itself has been recursively generated by similar logic.

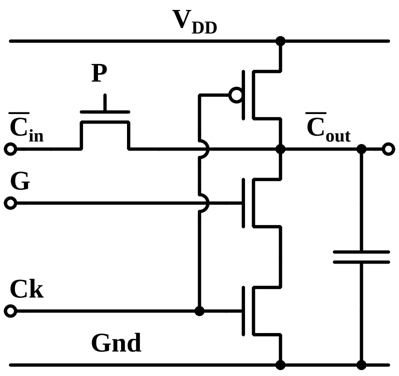
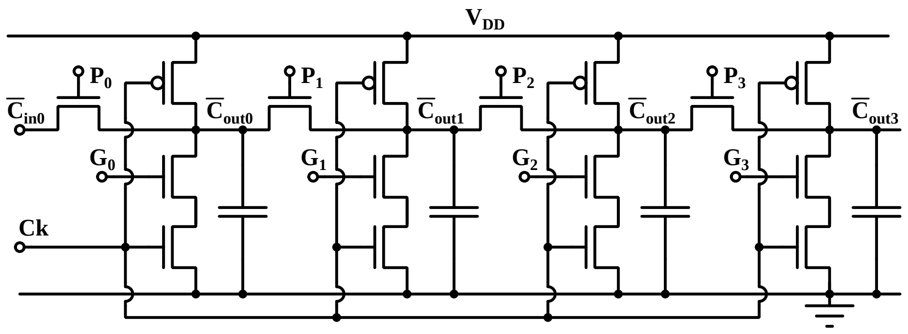


Figure 9.5: Manchester carry chain stage

As in the static case, there is a limit to the number of bits which can be so connected. If P = 1 for many successive bits, the discharge path is through series connected pass transistors of all these gates.

The circuit below shows a Manchester carry chain over 4 bits.



- If $G = 1$ for any bit, the output is brought to ‘0’. (Recall that $\overline{\text{Carry}}$ propagates – not Carry).
- The time of carry arrival for all subsequent bits is from the last bit where P = 0.
- The worst case for delay occurs when $P = 1$ for all bits. In this case, all load capacitors are shorted, so load capacitance $\propto n$.
- The discharge of capacitors is through n series connected pass transistors, so average R is $\propto n$.
- Thus in the worst case, the delay $\propto RC \propto n^2$.

9.5 Carry Bypass Adder

The worst case for addition occurs when P = 1 for all bits and carry has to ripple through all the bits. In carry bypass adder, we form groups of bits and if P = 1 for all members of a group, we pass on the carry input to this group directly to the input of the next group, without having to ripple through each bit.

This improves the worst case delay of the adder.

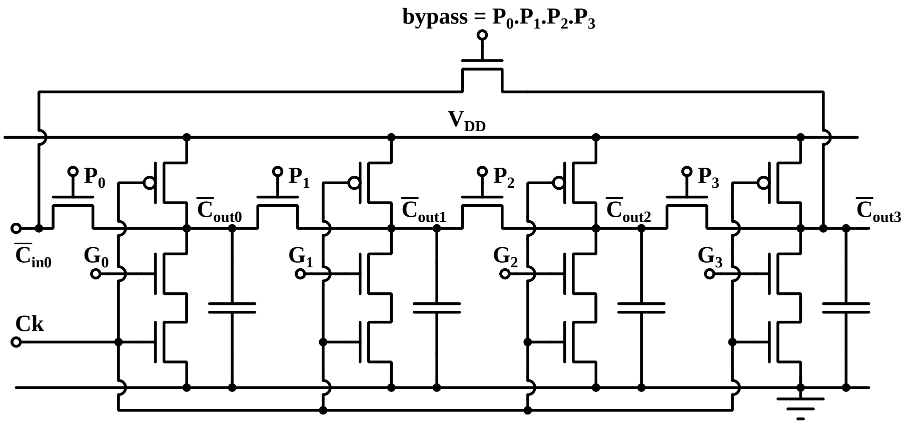


Figure 9.6: Carry By-pass adder

9.6 Carry Select Adder

An m bit carry select adder can be constructed as follows:

- We first compute the generate/propagate/kill signals for each bit (in parallel) from the input bits. Assuming unit gate delay model, this takes one unit of time.
- We use two m bit carry bypass adders. One of the adders assumes the carry input C_{in} to be 0, while the other assumes C_{in} to be 1. The two adders work in parallel and each takes m units of time.
- We now use a multiplexer controlled by the actual C_{in} to select the correct C_{out}. This takes one unit of time.
- The C_{out} of one such m bit adder will be used as the select input of the multiplexer of the next.

- The sum output of each bit is derived from P and C_{out} signals for the corresponding bit and appear one unit of time after C_{out} is available.

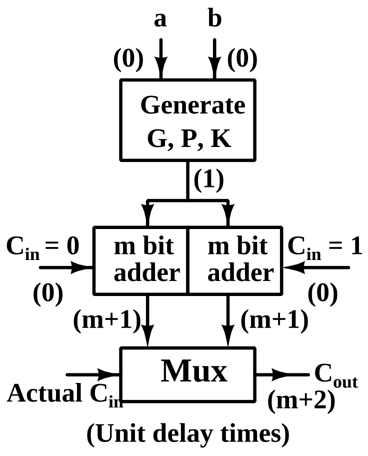


Figure 9.7: Carry select adder

Times of availability of various signals are noted in parentheses in the diagram.

- The two alternatives for the carry output are ready at (m+1) units of time.
- If the actual C_{in} is available at n units of time, the output will be available at (m+2) or (n+1), whichever is later.
- In case of 4 bit adders, this is at 6 units of time or at C_{in} arrival + 1, whichever is later.

9.6.1 Stacking Carry Select Adders

The sub-adders in carry select adder can use any architecture. They could be Manchester carry chains, carry bypass or ripple carry adders. Obviously, these sub adders should not be very long, otherwise, their outputs will be ready after a long time and we shall lose the advantage of carry select additions. Then, how do we make long adders using carry select?

This is done by stacking several smaller carry select adders.

We can stack several identical carry select adders. There is no need for carry select in the first stage, as C_{in} for this stage is available simultaneously with A_i and B_i.

Every subsequent stage will have two sub-adders, one assuming $C_{in} = 0$, the other assuming $C_{in} = 1$. The correct output will be selected by the actual C_{in} when it arrives.

Thus, after the first stage, each group of m bit adders will add only one unit of delay. This is much faster. However, the delay is still linear in number of bits.

A 32-bit adder made by cascading 8 4-bit carry select adders. Notice that the first group

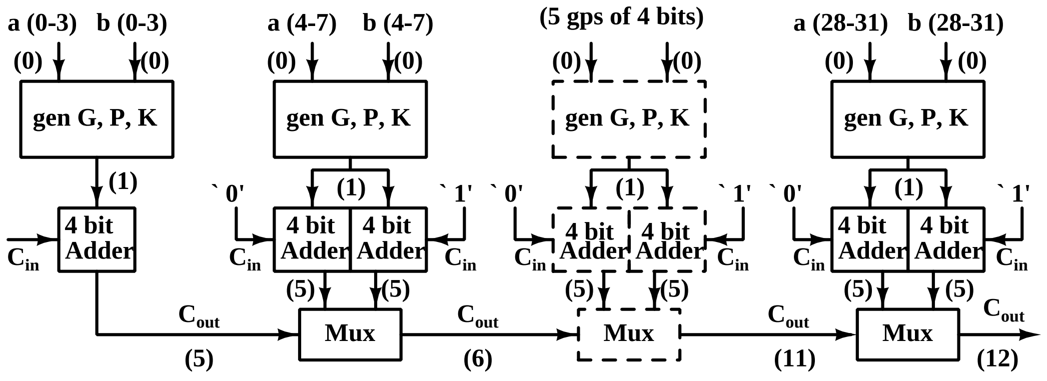


Figure 9.8: Linear stacking of carry select adders

does not need two adders and a mux, since the input carry is known to this group.

Bits	cy in	alt cy.s	cy out
0-3	0	-	5
4-7	6	5	6
8-11	7	5	7
12-15	8	5	8
16-19	9	5	9
20-23	10	5	10
24-27	11	5	11
28-31	12	5	12

The sum generation will take another unit of time, so the overall results will be available in 13 units of time.

Can we speed up the adder if we don't use the same no. of bits in every stage? In linear stacking, since all adders are identical, they are ready with their alternative outputs at the same time. But the carry arrives later and later at each successive group of carry select adders. We could have used this extra time to add up more bits in the later stages, and still be ready with the alternative results before carry arrives! Since the carry arrives one unit of time later at each successive group, each successive group could be longer by one bit.

We can do more bits of addition in the same time, if each successive stage is 1 bit longer than the previous one. Since the first stage does not add a mux delay, the first two stages will use the same number of bits. Thus, the number of bits which can be added is given by

$$m = n_0 + n_0 + (n_0 + 1) + (n_0 + 2) + \cdots = n_0 + \frac{s(n_0 + n_0 + s - 1)}{2}$$

where s is the number of stages following the first one without carry select.

The total delay will be $n_0 + 1$ for the first stage. Each subsequent stage takes just 1 unit of time since the candidates for selection are available just in time. Thus the time take is just $n_0 + s + 1$ units. When $s \gg n_0$, we have $m \approx s^2/2$, while the time taken is nearly s.

Thus the time taken to add m bits is $\approx \sqrt{2m}$
For a 32 bit adder, we could use a distribution like: 4,4,5,6,7,6.

Bits	carry in	carry alternatives	carry out
0-3	0	4	5
4-7	5	5	6
8-12	6	6	7
13-18	7	7	8
19-25	8	8	9
26-31	9	8	10

Our sum will be ready at 11 - which is much faster. This gain will be even higher for wider additions.

In square root tiling, the size of the sub-adders can become quite large for the later adders. Each of these can therefore be constructed as independent adders with any architecture. In particular, we might construct the sub-adders themselves as tiled carry select adders. Since these are faster, we can accommodate more bits in each of these stages. Thus the bits processed increase faster than square of the number of stages. In the asymptotic limit, the time for addition can be reduced to be logarithmic in the number of bits, rather than just square root.

However, it is possible to complete the addition in logarithmic time using tree adders, with much less complexity. These are the adders used in most modern systems using wide adders.

9.7 Tree Adders

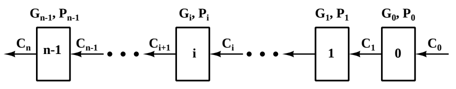
Tree adders use the idea of carry look ahead addition. However, these do not try to implement the complex logic expressions which result from looking ahead. Instead, these build up the logic in a tree like structure, where each node performs simple logic operations.

Recall that we had defined

$$K = \overline{A} \cdot \overline{B},\ G = A.B \text{ and } P = A \oplus B \tag{9.19}$$

When $K = 1,\ C_{out} = 0$, while if $G = 1,\ C_{out} = 1$, irrespective of C_{in}.
When $P = 1,\ C_{out} = C_{in}$ and this is the only case when we must wait for C_{in} to compute C_{out}

Consider an n bit adder with the least significant bit indexed as 0 and the most significant bit as $n-1$. The i'th bit accepts C_i as input carry and produces C_{i+1} as output carry.



G_{n-1}, P_{n-1}
G_i, P_i
G₁, P₁
G₀, P₀
C_n
n-1
C_{n-1}
C_{i+1}
i
C_i
1
C₁
0
C₀

We can express the output of the i'th stage as:

$$C_{i+1} = G_i + P_i \cdot C_i \tag{9.20}$$

Substituting for C_i, which is the output of cell no. (i-1),

$$\begin{aligned} C_{i+1} &= G_i + P_i \cdot (G_{i-1} + P_{i-1} \cdot C_{i-1}) \\ &= G_i + P_i \cdot G_{i-1} + P_i \cdot P_{i-1} \cdot C_{i-1} \end{aligned}$$

If we try to compute the output carry for each bit by extending this logic to a function of $G(i),\ P(i)$ and C_0, the expressions become very complex and their implementation will be slow. However, we can divide this work in a tree structure and that eventually leads to faster adders. Recall that

$$C_{i+1} = G_i + P_i \cdot G_{i-1} + P_i \cdot P_{i-1} \cdot C_{i-1}$$

We can divide the expression for the output carry in parts which are independent of input carry and parts which are dependent on it. Let us call the original single bit G and P values for the i'th cell as G_i^0 and P_i^0. Then we can define

$$G_{i:i-1}^1 \equiv G_i^0 + P_i^0 \cdot G_{i-1}^0 \quad \text{and} \quad P_{i:i-1}^1 \equiv P_i^0 \cdot P_{i-1}^0 \tag{9.21}$$

This permits us to write

$$C_{i+1} = G_{i:i-1}^1 + P_{i:i-1}^1 \cdot C_{i-1} \tag{9.22}$$

Thus we can compute C_{i+1} from C_{i-1} directly using the same logic as before, except that we use $G_{i:i-1}^1$ and $P_{i:i-1}^1$ instead of G_i^0 and P_i^0.
Notice that the logic needed to compute $G_{i,i-1}^1$ from G_i^0 and P_i^0 is the same as that used to compute the output carry from input carry using G and P values of any order.

Effectively, we have created a 2 bit cell which is equivalent to two original 1 bit adder cells and carry can be passed in groups of 2 bits across the adder. Of course, we need two logic stages to compute the second order G and P values, but these values are independent of carry and can be computed in parallel for all bit pairs.

Continuing the same process, we can combine G and P values of first order to compute second order G and P, which will permit computation of C_{i+1} directly from C_{i-3}.

$$G^2_{i:i-3} \equiv G^1_{i:i-1} + P^1_{i:i-1} \cdot G^1_{i-2:i-3} \quad \text{and} \quad P^2_{i:i-3} \equiv P^1_{i:i-1} \cdot P^1_{i-2:i-3} \tag{9.23}$$

$$C_{i+1} = G^2_{i:i-3} + P^2_{i:i-3} \cdot C_{i-3} \tag{9.24}$$

Thus carry can now be passed over groups of 4 bits. Notice that the group size over which the carry can be computed directly *multiplies* by two each time we use a higher order for G and P values, while the time to compute the required higher order G and P values increments by one gate delay for logic $A + B \cdot C$ (for G) or $A \cdot B$ (for P). This is what results in ultimate time to generate the final carry being logarithmic in the number of bits being added.

Since the generation of final carry has been speeded up substantially, we have to re-examine our assumption that carry propagation is the critical step in adder design. Addition is not complete unless all the sum bits and the terminal carry have been generated. In principle, we do not need the internal carries at each bit for the final result. The sum values at each bit are:

$$S_i = A_i \oplus B_i \oplus C_i = P_i \oplus C_i \tag{9.25}$$

For generating sums, we do need the internal carries at each bit. These can be computed using relations of the type described in equations 9.20,9.22 and 9.24 etc. Different architectures have been described in literature for the order of computation of G, P, C_{out} and Sum bits. All of these compute the final sum in times which are logarithmic functions of the number of bits. For wide adders,these can be much faster than other architectures.

9.7.1 Brent Kung adder

To illustrate the operation of tree adders, we use the architecture described by Brent and Kung for a tree adder. Many other tree adders have been described in the literature and we use this one as an example. In this adder, we compute the P and G values in a tree fashion, as shown in fig.9.9 for an 8 bit adder.

From the values of a_i, b_i, we first calculate P^0_i, G^0_i, with $i = 0 \cdots 7$. Next, using these values, we can generate $P^1_{2i+1:2i}, G^1_{2i+1:2i}$ with $i = 0 \cdots 3$. In the next step, we use these values to generate $P^2_{4i+3:4i}, G^2_{4i+3:4i}$ with $i = 0, 1$. Finally, using these values, we can compute $P^3_{7:0}, G^3_{7:0}$.

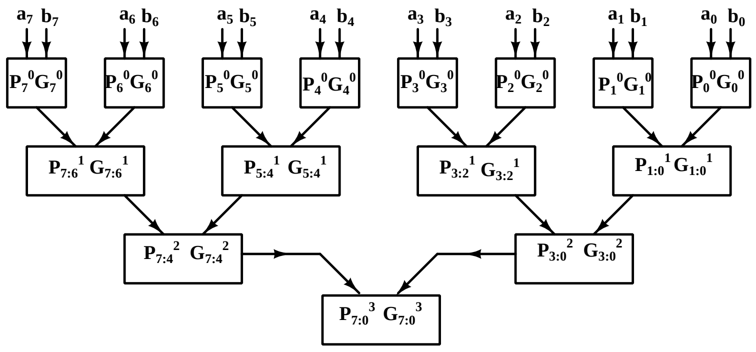


Figure 9.9: Generation of P and G values in Brent Kung adder

As soon as values of P and G terms of various orders are known, we can compute the values of carry outputs which depend on these and the input carry.

$$C_1 = G_0^0 + P_0^0 \cdot C_0, \quad C_2 = G_{1:0}^1 + P_{1:0}^1 \cdot C_0$$

$$C_4 = G_{3:0}^2 + P_{3:0}^2 \cdot C_0, \quad C_8 = G_{7:0}^3 + P_{7:0}^3 \cdot C_0$$

When these carry values are valid, the other carry values which depend on these can be generated.

$$C_3 = G_2^0 + P_2^0 \cdot C_2, \quad C_5 = G_4^0 + P_4^0 \cdot C_4 \quad C_6 = G_{5:4}^1 + P_{5:4}^1 \cdot C_4,$$

Finally, C_7 can be generated from C_6.

$$C_7 = G_6^0 + P_6^0 \cdot C_6$$

With all carry values generated, the corresponding sum values can be calculated using the relation $\text{Sum}_i = P_i^1 \oplus C_i$.

In fact, we can make use of the fact that the input carry C_0 is available right at the start along with A and B. It is possible to generate all sum and carry computations a little faster by making use of this fact, as is shown in the following tutorial for a 32 bit Brent Kung adder.

9.7.2 Tutorial: a 32 bit Brent Kung Logarthmic Adder

Terminology

First of all, let us introduce the terminology we use for understanding logarithmic adders.

- Bits are numbered from 0 to N-1 for an N bit adder.

- Thus the input operands are $A = (a_{N-1} \cdots a_0)$ and $B = (b_{N-1} \cdots b_0)$, with a possible input carry. All these bits are available at the start.
- $\mathbf{c}_i$ represents the **input** carry to the i'th bit.
- Since the ouput carry of a bit is the input carry for the next, the **output carry** from bit i is the input carry for bit (i+1), or C_{i+1}.
- Thus c_0 represents the overall input carry for the addition and c_N repreents the final output carry.
- $\mathbf{s}_i$ represents the sum output from the i'th bit.

Logarithmic adders use carry look ahead, producing Generate (G) and Propagate (P) signals over 1, 2, 4, 8 ... bits.

Generate and Propagate signals can be derived directly from the operand bits without involving carry. So these can be computed in parallel.

Single bit Generate and Propagate signals

Generate and Propagate signals over a single bit are:
$G_i = A_i \cdot B_i$, $P_i = A_i \oplus B_i$, with output carry given by $C_{i+1} = G_i + P_i \cdot C_i$. C_i is similarly generated from C_{i-1} as $C_i = G_{i-1} + P_{i-1} \cdot C_{i-1}$. To distinguish these G and P values from those defined for multiple bits, we use a superscript of 0 in addition to the subscript denoting the bit number.

Two bit P and G signals

Substituting for C_i in the expression for C_{i+1}, we get
$C_{i+1} = G_i^0 + P_i^0 \cdot (G_{i-1}^0 + P_{i-1}^0 \cdot C_{i-1}) = G_i^0 + P_i^0 \cdot G_{i-1}^0 + P_i^0 \cdot P_{i-1}^0 \cdot C_{i-1}$

Multi-bit Generate and Propagate Values

Defining $G_{(i:i-1)}^1 \equiv G_i^0 + P_i^0 \cdot G_{i-1}$ and $P_{(i:i-1)}^1 \equiv P_i^0 \cdot P_{i-1}^0$, we can write:
$C_{i+1} = G_{(i:i-1)}^1 + P_{(i:i-1)}^1 \cdot C_{i-1}$.

$G_{(i:i-1)}^1$ and $P_{(i:i-1)}^1$ are also independent of carry input and can be computed from G_i^0, P_i^0, G_{i-1}^0 and P_{i-1}^0, which in turn are computed directly from A_i, B_i, A_{i-1} and B_{i-1}.

These new G and P values permit us to compute C_{i+1} directly from C_{i-1}. We can treat bits $i-1$ and i as a block and use these G and P values as the output carry of the block from the input carry to the block.

Multi-bit P and G signals

Continuing the same process, we can combine two 2bit G and P values to compute 4 bit G and P values, which will permit computation of C_{i+1} directly from C_{i-3}.

$$\begin{aligned} G^2_{i,i-3} &\equiv G^1_{i,i-1} + P^1_{i,i-1} \cdot G^1_{i-2,i-3} \quad \text{and} \quad P^2_{i,i-3} \equiv P^1_{i,i-1} \cdot P^1_{i-2,i-3} && (9.26) \\ \text{Now } C_{i+1} &= G^2_{i,i-3} + P^2_{i,i-3} \cdot C_{i-3} && (9.27) \end{aligned}$$

All G and P values are independent of input carry and can be computed in constant time from A_i and B_i.

Carry can now be passed over groups of 4 bits using $G^2_{(i,i-3)}$ and $P^2_{(i,i-3)}$.

Extending this operation with larger and larger groups, we can compute generate and propagate signals over 8 bits (with superscript 3) and so on. G and P signals over 2^n bits will carry the superscript n and their subscripts will describe the range of bit indices over which these signals operate.

The method of computation of G and P values over larger and larger groups of bits suggests a tree structure for their computation. Different architectures have been described in literature for the order of computation of G, P, C_{out} and Sum bits.

Because the block sizes double with each computation, the time taken to compute all G and P values is logarithmic in the width of operands.As soon as G and P values of a particular order are available, the internal carry and sum values which depend on these can be computed. Time taken for this is also logarithmic in bit width of operands. For wide adders, such adders can be much faster than other architectures.

If we represents indices for upper half of the range by u and lower half by l, we can write:

$$G_{(u:l)} = G_u + P_u \cdot G_l, \quad \text{whereas} \quad C_{next} = G_{(u)} + P_{(u)} \cdot C_{prev}$$

Notice that G values are computed by the same logic relation as carry outputs. However, the input carry C_0 is known at the start itself. Whenever carry is already known, we can replace G_l by this carry. The computed value of $G_{(u:l)}$ will then be the carry output, rather than the G value. This value can be used for further G calculations and will directly give the carry each time.

This can reduce the computation required to generate the carry and sum values since some of the carry values are already available.

A few things are worth noting while implementing tree adders.

- Just producing the final carry is not enough. All internal carry values are also required to generate the sum bits. Internal bit-wise carries may be available *after* the final carry has been evaluated.
- The critical path now is not the generation of the final carry, but that of bit-wise sums.
- Different architectures have been described in literature for the order of computation of G, P, C_{out} and Sum bits.
- All of these compute the final result in times which are logarithmic functions of the number of bits.
- A simple and straightforward implementation of the tree adder is provided by Brent Kung adders.

32 bit Brent Kung adder: Order 0

We use a unit time model in which we assume that logic functions AND, XOR, A + B.C as well as A.B + C.(A+B) take the same amount of time, which defines 1 slot of time for this tutorial.

The single Bit G and P values (designated as order 0) are given by

$$P_i^0 = a_i \oplus b_i,\ G_i^0 = a_i \cdot b_i, \quad \text{except } G_0^0 = a_i \cdot b_i + c_0 \cdot (a_0 + b_0)$$

An exception is made for the least significant bit of G because for this bit, the input carry is known right at the start. We make use of this and compute effectively the carry output from bit 0 (c_1) and map the output carry as if it was due to a generate signal at this position. Thus,

$$G_0^0 = c_1 = a_0 \cdot b_0 + c_0 \cdot (a_0 + b_0)$$

All these functions can be computed in one unit of time directly from $a_i,\ b_i$ and input carry c_0. So these are all ready at the end of the first time slot.
Since $c_1 = G_0^0$, c_1 is also ready at the end of first slot.

Brent Kung adder: higher orders

As described earlier, We can define G and P functions which operate over multiple bits. Higher order G and P values are computed as

$$G = G_u + P_u \cdot G_l, \quad P = P_u \cdot P_l$$

where u and l stand for upper half range and lower half range for a range of bit indices. These can be computed within one time slot from the next lower oreder G and P values. Thus higher

orders of G and P values, (successively covering twice the range of indices for the previous order) will be available in each time slot. Internal carries are computed using functions like $C = G + P \cdot C_{in}$.
Depending on the order of G and P values, we can compute carry values whose indices are 1, 2, 4, 8 ... bits higher than the input carry. This computation also takes one time slot, but can be performed only after the needed C_{in}, P and G values are available.

- G and P values for single bits are available at the end of first slot.
- G and P values spanning groups of 2 bits are available at the end of second slot. G and P values spanning groups of 4 bits are available at the end of third slot. G and P values spanning groups of 8 bits are available at the end of fourth slot. G and P values spanning groups of 16 bits are available at the end of fifth slot.
  Finally, G and P values spanning the full word of 32 bits are available at the end of sixth slot.
- G and P values are available over spans of 2^n bits. The start bit for these spans has a granularity of 2^n bits. For example, second order values connect $0 \rightarrow 4$, $4 \rightarrow 8$ etc. We cannot connect from $1 \rightarrow 5$ using these in a Brent Kung adder.
- The lowest index G value for any order i is automatically the carry value for bit index 2^i.
- at time =0, all a_i, b_i and c_0 are available.
- at time =1, all P_i^0 and G_i^0 are available. $c_1 = G_0^0$ is also available.
- at time =2, all 2 bit P and G values ($P_{..}^1$ and $G_{..}^1$) are available. $c_2 = G_{(1:0)}^1$ has been computed.
- at time =3, all 4 bit P and G values ($P_{..}^2$ and $G_{..}^2$) are available. $c_4 = G_{(3:0)}^2$, $c_3 \leftarrow c_2$ using G_2^0, P_2^0 and c_2 have also been computed.
- at time =4, all 8 bit P and G values ($P_{..}^3$ and $G_{..}^3$) are available. $c_8 = G_{(7:0)}{}^3$ is also available.
  $c_5 \leftarrow c_4$ using G_4^0, P_4^0 and c_4; as well as $c_6 \leftarrow c_4$ using $G_{(5:4)}^1$, $P_{(5:4)}^1$ and c_4 have been computed.
- at time =5, all 16 bit P and G values ($P_{..}^4$ and $G_{..}^4$) have been computed. $c_{16} = G_{(15:0)}^4$ is also available.
  $c_7 \leftarrow c_6$ using G_6^0, P_6^0 and c_6; $c_9 \leftarrow c_8$ using $G_8 0$, P_8^0 and c_8;
  $c_{10} \leftarrow c_8$ using $G_{(9:8)}^1$, $P_{(9:8)}^1$ and c_8;
  $c_{12} \leftarrow c_8$ using $G_{(11:8)}^2$, $P_{(11:8)}^2$ and c_8 are all available.

- at time =6, $G^5_{(31:0)}$ is generated. This is the value of $c_{32} = C_{out}$.
  $P^5_{(31:0)}$ is not required.
  $c_{11} \leftarrow c_{10}$ using G^0_{10}, P^0_{10} and c_{10}; $c_{13} \leftarrow c_{12}$ using G^0_{12}, P^0_{12} and c_{12};
  $c_{14} \leftarrow c_{12}$ using $G^1_{(13:12)}$, $P^1_{(13:12)}$ and c_{12};
  $c_{17} \leftarrow c_{16}$ using G^0_{16}, P^0_{16} and c_{16};
  $c_{18} \leftarrow c_{16}$ using $G^1_{(17:16)}$, $P^1_{(17:16)}$ and c_{16};
  $c_{20} \leftarrow c_{16}$ using $G^2_{(19:16)}$, $P^2_{(19:16)}$ and c_{16}; and
  $c_{24} \leftarrow c_{16}$ using $G^3_{(23:16)}$, $P^3_{(23:16)}$ and c_{16} have all been computed.

- at time =7, all G and P values for groups of 1, 2, 4, 8 and 16 bits are available.
  $c_{15} \leftarrow c_{14}$ using G^0_{14}, P^0_{14} and c_{14}.
  $c_{19} \leftarrow c_{18}$ using G^0_{18}, P^0_{18} and c_{18}.
  $c_{21} \leftarrow c_{20}$ using G^0_{20}, P^0_{20} and c_{20}.
  $c_{22} \leftarrow c_{20}$ using $G^1_{(21:20)}$, $P^1_{(21:20)}$ and c_{20}.
  $c_{25} \leftarrow c_{24}$ using G^0_{24}, P^0_{24} and c_{24}.
  $c_{26} \leftarrow c_{24}$ using $G^1_{(25:24)}$, $P^1_{(25:24)}$ and c_{24}.
  $c_{28} \leftarrow c_{24}$ using $G^2_{(27:24)}$, $P^2_{(27:24)}$ and c_{24}.

- at time =8, we have computed:
  $c_{23} \leftarrow c_{22}$ using G^0_{22}, P^0_{22} and c_{22}.
  $c_{27} \leftarrow c_{26}$ using G^0_{26}, P^0_{26} and c_{26}.
  $c_{29} \leftarrow c_{28}$ using G^0_{28}, P^0_{28} and c_{28}.
  $c_{30} \leftarrow c_{28}$ using $G^1_{(29:28)}$, $P^1_{(29:28)}$ and c_{28} have been computed.

- at time =9, we have computed:
  $c_{31} \leftarrow c_{30}$ using G^0_{30}, P^0_{30} and c_{30}.

Fig.9.7.2 shows the sequence of generation of carry values.

32 bit Brent Kung adder: Numerical Example

Taking the example of adding B7A56893H to 506A980CH with an input carry of '1', let us list the P, G, carry and sum bits generated in each time slot.

In the first slot, we generate the single bit P and G values.

a	1011	0111	1010	0101	0110	1000	1001	0011
b	0101	0000	0110	1010	1001	1000	0000	1100
P^0	1110	0111	1100	1111	1111	0000	1001	1111
G^0	0001	0000	0010	0000	0000	1000	0000	0001[†]

h!

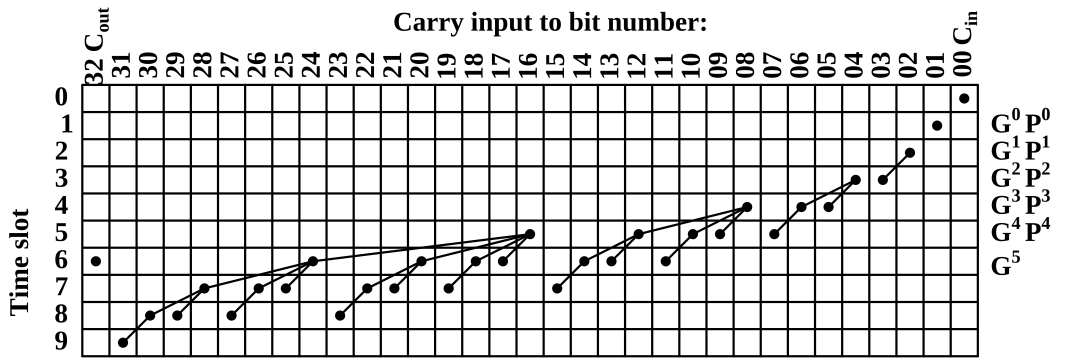


Figure 9.10: Generation of bitwise carries in a 32 bit Brent Kung adder

$P_i^0 = a_i \oplus b_i, \quad G_i^0 = a_i \cdot b_i$

$\dagger G_0^0$ is generated as $a_0 \cdot b_0 + c_0 \cdot (a_0 + b_0)$

$c_1 = G_0^0 = 1$

In the second slot, we generate P and G values spanning two bits each.

From now on,

$$P_{range}^{m+1} = P_u^m \cdot P_l^m, \quad G_{range}^{m+1} = G_u^m + P_u^m \cdot G_l^m,$$

where u represents the upper half range and l represents the lower half range.

P^0	1110	0111	1100	1111	1111	0000	1001	1111
G^0	0001	0000	0010	0000	0000	1000	0000	0001
P^1	1_0_	0_1_	1_0_	1_1_	1_1_	0_0_	0_0_	1_1
G^1	0_1_	0_0_	0_1_	0_0_	0_0_	1_0_	0_0_	0_1_

$c_2 = G_{1-0}^1 = 1$

$s_0 = P_0^0 \oplus c_0 = 1 \oplus 1 = 0, \quad s_1 = P_1^0 \oplus c_1 = 1 \oplus 1 = 0.$

In the third slot, we calculate P and G values spanning 4 bits each.

P^1	1_0_	0_1_	1_0_	1_1_	1_1_	0_0_	0_0_	1_1
G^1	0_1_	0_0_	0_1_	0_0_	0_0_	1_0_	0_0_	0_1_
P^2	0___	0___	0___	1___	1___	0___	0___	1___
G^2	1___	0___	1___	0___	0___	1___	0___	1___

$c_4 = G_{3-0}^2 = 1$. We can also compute

$c_3 = G_2^0 + P_2^0 \cdot c_2 = 0 + 1 \cdot 1 = 1,$

$s_2 = P_2^0 \oplus c_2 = 1 \oplus 1 = 0$

In the fourth slot, we calculate P and G values spanning 8 bits each.

P^2	0___	0___	0___	1___	1___	0___	0___	1___
G^2	1___	0___	1___	0___	0___	1___	0___	1___
P^3	0___	____	0___	____	0___	____	0___	____
G^3	1___	____	1___	____	1___	____	0___	____

$c_8 = G^3_{7-0} = 0$. We can also compute

$c_5 = G_4^0 + P_4^0 \cdot c_4 = 0 + 1 \cdot 1 = 1, \quad c_6 = G^1_{5-4} + P^1_{5-4} \cdot c_4 = 0 + 0 \cdot 1 = 0.$

$s_3 = P_3^0 \oplus c_3 = 1 \oplus 1 = 0, \quad s_4 = P_4^0 \oplus c_4 = 1 \oplus 1 = 0.$

In the fifth slot, we calculate P and G values spanning 16 bits each.

P^3	0___	____	0___	____	0___	____	0___	____
G^3	1___	____	1___	____	1___	____	0___	____
P^4	0___	____	____	____	0___	____	____	____
G^4	1___	____	____	____	1___	____	____	____

$c_{16} = G^4_{15-0} = 1$. We can also compute

$c_7 = G_6^0 + P_6^0 \cdot c_6 = 0 + 1 \cdot 0 = 0, \quad c_9 = G_8^0 + P_8^0 \cdot c_8 = 0 + 0 \cdot 0 = 0,$
$c_{10} = G^1_{9-8} + P^1_{9-8} \cdot c_8 = 0 + 0 \cdot 0 = 0, \quad c_{12} = G^2_{11-8} + P^2_{11-8} \cdot c_8 = 1 + 0 \cdot 0 = 1.$
$s_5 = P_5^0 \oplus c_5 = 0 \oplus 1 = 1. \quad s_6 = P_6^0 \oplus c_6 = 0 \oplus 0 = 0.$
$s_8 = P_8^0 \oplus c_8 = 0 \oplus 0 = 0.$

In the sixth slot, we compute $G^5_{31-0} = G^4_{31-16} + P^4_{31-16} \cdot G^4_{15-0}$. P^5_{31-0} is not required.

This gives $C_{out} = c_{32} = G^5_{31-0} = 1$. We can further compute:

$c_{11} = G^0_{10} + P^0_{10} \cdot c_{10} = 0 + 0 \cdot 0 = 0,$
$c_{13} = G^0_{12} + P^0_{12} \cdot c_{12} = 0 + 1 \cdot 1 = 1,$
$c_{14} = G^1_{13-12} + P^1_{13-12} \cdot c_{12} = 0 + 1 \cdot 1 = 1,$
$c_{17} = G^0_{16} + P^0_{16} \cdot c_{16} = 0 + 1 \cdot 1 = 1,$
$c_{18} = G^1_{17-16} + P^1_{17-16} \cdot c_{16} = 1 + 1 \cdot 1 = 1,$
$c_{20} = G^2_{19-16} + P^2_{19-16} \cdot c_{16} = 1 + 1 \cdot 1 = 1,$
$c_{24} = G^3_{23-16} + P^3_{23-16} \cdot c_{16} = 0 + 1 \cdot 1 = 1$

$s_7 = P_7^0 \oplus c_7 = 1 \oplus 0 = 1, \quad s_9 = P_9^0 \oplus c_9 = 0 \oplus 0 = 0,$
$s_{10} = P^0_{10} \oplus c_{10} = 0 \oplus 0 = 0, \quad s_{12} = P^0_{12} \oplus c_{12} = 1 \oplus 1 = 0,$
$s_{16} = P^0_{16} \oplus c_{16} = 1 \oplus 1 = 0,$

In the seventh slot, All the required values of P and G are already available. We can compute:
$c_{15} = G^0_{14} + P^0_{14} \cdot c_{14} = 0 + 1 \cdot 1 = 1$ $c_{19} = G^0_{18} + P^0_{18} \cdot c_{18} = 0 + 1 \cdot 1 = 1$
$c_{21} = G^0_{20} + P^0_{20} \cdot c_{20} = 0 + 0 \cdot 1 = 0$ $c_{22} = G^1_{21-20} + P^1_{21-20} \cdot c_{20} = 1 + 0 \cdot 0 = 1$
$c_{25} = G^0_{24} + P^0_{24} \cdot c_{24} = 0 + 1 \cdot 1 = 1$ $c_{26} = G^1_{25-24} + P^1_{25-24} \cdot c_{24} = 0 + 1 \cdot 1 = 1$
$c_{28} = G^2_{27-24} + P^2_{27-24} \cdot c_{24} = 0 + 0 \cdot 1 = 0$

$s_{11} = P^0_{11} \oplus c_{11} = 0 \oplus 0 = 0,$ $s_{13} = P^0_{13} \oplus c_{13} = 1 \oplus 1 = 0,$
$s_{14} = P^0_{14} \oplus c_{14} = 1 \oplus 1 = 0,$ $s_{17} = P^0_{17} \oplus c_{17} = 1 \oplus 1 = 0,$
$s_{18} = P^0_{18} \oplus c_{18} = 1 \oplus 1 = 0,$ $s_{20} = P^0_{20} \oplus c_{20} = 0 \oplus 1 = 1,$
$s_{24} = P^0_{10} \oplus c_{24} = 1 \oplus 1 = 0,$

Then in the eighth slot, we can compute:
$c_{23} = G^0_{22} + P^0_{22} \cdot c_{22} = 0 + 1 \cdot 1 = 1,$
$c_{27} = G^0_{26} + P^0_{26} \cdot c_{26} = 0 + 1 \cdot 1 = 1,$
$c_{29} = G^0_{28} + P^0_{28} \cdot c_{28} = 1 + 0 \cdot 1 = 1,$
$c_{30} = G^1_{29-28} + P^1_{29-28} \cdot c_{28} = 1 + 0 \cdot 1 = 1.$

Sums corresponding to carries computed in the previous slot can also be evaluated as:
$s_{15} = P^0_{15} \oplus c_{15} = 1 \oplus 1 = 0,$
$s_{19} = P^0_{19} \oplus c_{19} = 1 \oplus 1 = 0,$
$s_{21} = P^0_{21} \oplus c_{21} = 0 \oplus 0 = 0,$
$s_{22} = P^0_{22} \oplus c_{22} = 1 \oplus 1 = 0,$
$s_{25} = P^0_{25} \oplus c_{25} = 1 \oplus 1 = 0,$
$s_{26} = P^0_{26} \oplus c_{26} = 1 \oplus 1 = 0,$
$s_{28} = P^0_{28} \oplus c_{28} = 0 \oplus 0 = 0.$

In the ninth slot, we can compute $c_{31} = G^0_{30} + P^0_{30} \cdot c_{30} = 0 + 1 \cdot 1 = 1,$
and the sum values
$s_{23} = P^0_{23} \oplus c_{23} = 1 \oplus 1 = 0,$
$s_{27} = P^0_{29} \oplus c_{29} = 0 \oplus 1 = 1,$
$s_{29} = P^0_{29} \oplus c_{29} = 1 \oplus 1 = 0,$
$s_{30} = P^0_{30} \oplus c_{30} = 1 \oplus 1 = 0,$

Finally in the tenth slot, we can evaluate s_{31} as $s_{31} = P^0_{31} \oplus c_{31} = 1 \oplus 1 = 0$. Thus we have

C_{in}	1110	1111	1101	1111	1111	0000	0011	1111
a	1011	0111	1010	0101	0110	1000	1001	0011
b	0101	0000	0110	1010	1001	1000	0000	1100
sum	0000	1000	0001	0000	0000	0000	1010	0000

Final carry out is 1.

9.7.3 Other logarithmic adders

Brent Kung adder has low complexity and fanout from each stage. Other tree adders, (such as Kogge Stone adder) can be faster, following a similar scheme but with higher fanout and wiring congestion.

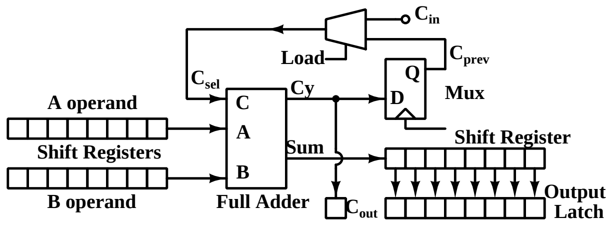
Rather than using radix 2 for generating P, G values, we can use radix 4. In this scheme, we use more complex logic for combining 4 P, G values every time to generate the next level P, G values. Thus, we can generate $P_{3,0}$ and $G_{3,0}$ directly from A_i, B_i. This reduces the depth of the tree, but at the same time, logic for each combining stage is more complex.

The other choice in logarithmic adders is for sparsity. The only use for generating bit-wise internal carries is for computing the sum bits. However, we may choose to generate only the even bit carries and compute sums for even as well odd bits using only these using more complex logic expressions.

9.8 Serial Adders

Up to now, we have been concerned with making fast adders, even at the cost of increased complexity and power. In many applications, speed is not as important as low power consumption and low cost.

Serial adders are an attractive option in such cases. A single full adder is used. If numbers to be added are available in parallel form, these can be serialized using shift registers.



A single full adder adds the incoming bits from operands A and B. Bits to be added are fed to it serially, LSB first.

- The sum bit goes to the output while carry is stored in a flip-flop.
- Carry then gets added to the next more significant bits which arrive in the next clock cycle.
- Output can be converted to parallel form if needed, using another shift register.

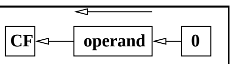
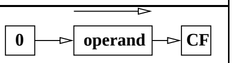
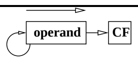
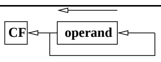
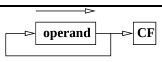
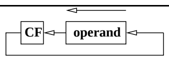
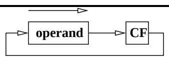
Chapter 10

Shift and Rotate Operations

We often need to shift and rotate operands in order to perform various operations such as serialization/de-serialization of data, multiplication etc.

10.1 Shift and Rotate Operations

The following operations are often required for processing of data:

SHL	op, count	
SAL	op, count	Same as SHL
SHR	op, count	
SAR	op, count	
ROL	op, count	
ROR	op, count	
RCL	op, count	
RCR	op, count	

These operations can be implemented by bi-directional shift registers with some control logic which provides circuits to choose the value and point of entry of new bits. However, this implementation can be very slow for a large number of shifts.

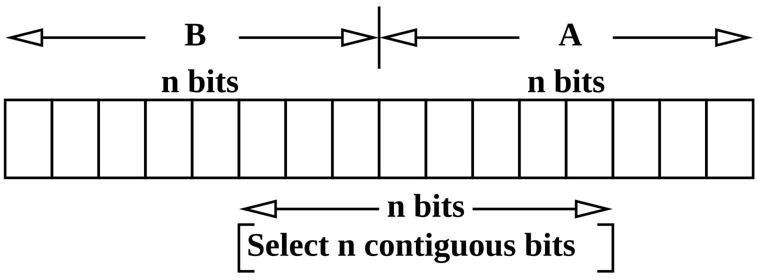
10.2 Barrel Shifters

Shift and rotate operations involve no computation. So why should we spend a large number of clock cycles for performing these operations? Ideally, we would like a shifter which produces the result in a single clock cycle.

10.2.1 Shift/Rotate as Select Operations

We can view Shift/Rotate operations as selection operations. In that case, the output can be produced directly by selecting the correct bits to appear in the desired position at the output.

Imagine two words A and B positioned as shown in the figure below.



We just have to choose B and A appropriately and select the correct range of n contiguous bits to implement various shift or rotate operations.

For all rotate operations, we choose B=A=data. For Shift Left, we choose B=data and A=0. For Logical Shift Right we choose B=0 and A=data. For Arithmetic Shift Right, we make B = replicated MSB of data, and set A = data.

Of course we do not actually copy data bits to A/B. Each output bit is produced by a mux which picks out the correct input data bit.

10.3 Barrel Shifters

Shifters which produce outputs as select operations are called barrel shifters. The name comes from viewing the inputs as well as outputs as a circular arrangement of bits. The shifter then connects the input circle to the output circle like the sections of a barrel.

A brute force implementation of such a shifter will require n multiplexers of n bits each, where the control inputs for each multiplexer are generated from the amount and type of shift/rotate that is desired. This requires quite a large number of complex n way multiplexers and puts a heavy load on data bits. Therefore rather than trying to complete the entire operation in one go, we use logarithmic barrel shifters, which are not so complex and take of the order of $\log_2 n$ operations to produce the result

10.3.1 Logarithmic Barrel Shifters

The control logic of a one step barrel shifter is complex because the amount of shift is variable. The loading on data lines and control logic complexity can be reduced if we break up the shift/rotate process into parts. We can carry out shifts in different stages, each stage corresponding to a single bit of the binary representation of the **shift amount**. Now the i'th stage needs to produce either no shift (if the corresponding control bit is '0') or a shift by a constant amount which is 2^i. Thus each stage requires only a two way mux for each bit. For example, a shift by 6 (binary: 110) will be carried out by first doing a 4 bit shift and then a 2 bit shift.

Since we need n bits to represent a maximum shift amount of $2^n - 1$ places, the number of bits to express the shift amount (and hence the number of shift stages required) is logarithmic in the maximum shift desired.

That is why such shifters are called Logarithmic Barrel Shifters. We can optionally buffer the outputs after each stage.

Bit i of the **shift amount** represents no shift (if it is 0) and a constant shift by 2^i places (if it is 1). If the amount of shift is fixed, the required bit can just be wired from the input bits.

The 2 way mux is controlled by bit i of the **shift amount**, Using this, we can choose either the unshifted operand bit or the operand bit 2^i places away from it.

This can be done for all bits of the operand in parallel. This constitutes one stage of the logarithmic shifter. The output can then be shifted again in the next stage, controlled by the next significant bit.

10.3.2 Right Rotate for an 8 bit Operand

For an 8 bit operand, the amount of rotation will be between 0 to 7 places, which can be represented by 3 bits. So we need a 3 stage implementation.

To perform a right rotate operation, we use the scheme shown in Figure 10.1.

Each input bit of the count drives eight 2-way muxes, one for each bit of the operand. At each stage, the muxes select either the unshifted bit or a bit 2^n places from it. A total of 3 stages are required for 0 to 7 bits of shift.

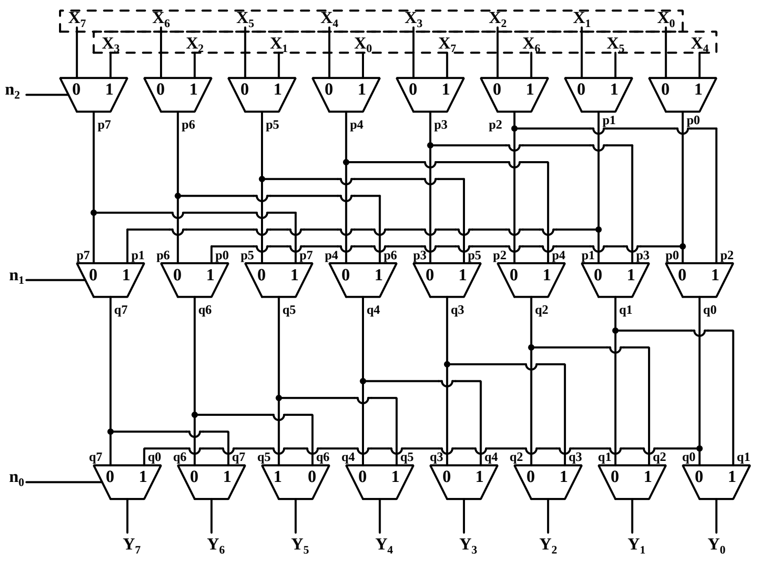


Figure 10.1: Right rotate for an 8 bit operand

10.3.3 8 bit Logical Shift Right

If we need a shift instead of a rotate, we feed a 0 instead of the corresponding data bit. We

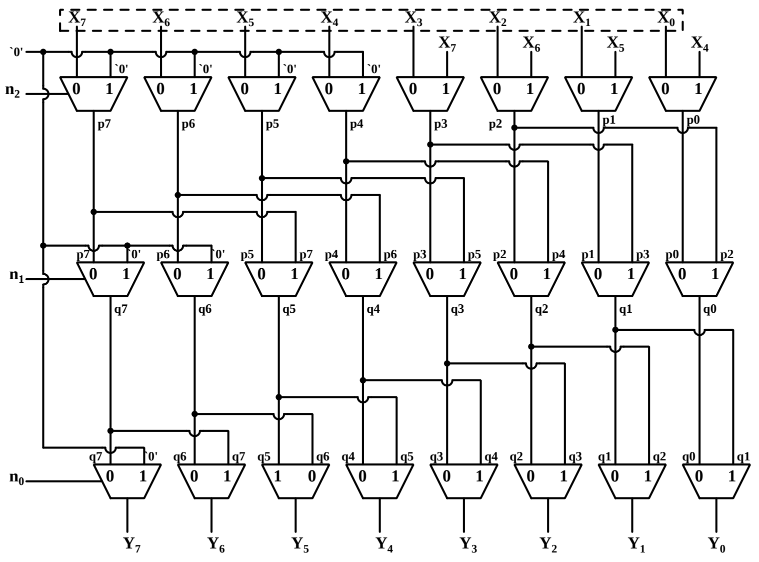


Figure 10.2: Logical Right Shift for an 8 bit operand

need to input 0's instead of data bits to the first 4 muxes in the first stage, to the first 2 muxes in the second stage and to 1 mux in the last stage.

10.3.4 Combining Rotate and Shift Operations

We can combine the circuits for rotate and shift functions by putting muxes where different inputs need to be presented for the two functions. Further, we can include the Arithmetic

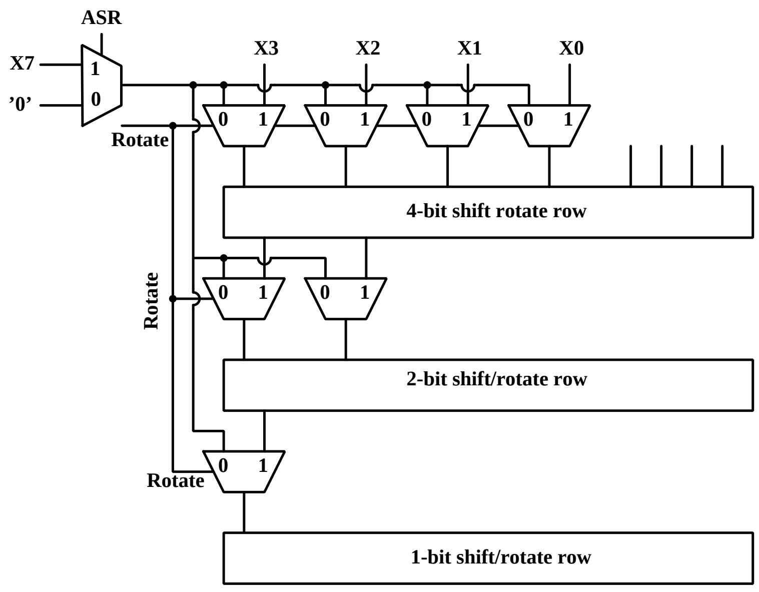


Figure 10.3: Combined Right Rotate/Shift for an 8 bit operand

Shift function by choosing between 0 or X7 as the bit to be inserted from the right.

10.3.5 Rotate and Shift by Masking

We can also combine the rotate and shift functions by masking. We use the rotate function which does not lose any information, as the primary circuit. Now we can mask n bits at the left to 0 if a right shift operation was desired instead. In case of an arithmetic shift, n bits on the left have to be set to the same value as X7.

Shift/Rotate Left case is similar, except that the Logical and Arithmetic shifts are not different operations.

10.3.6 Bidirectional Shift and Rotate Operations

It is possible to use the same hardware for left and right shift/rotate operations. This can be

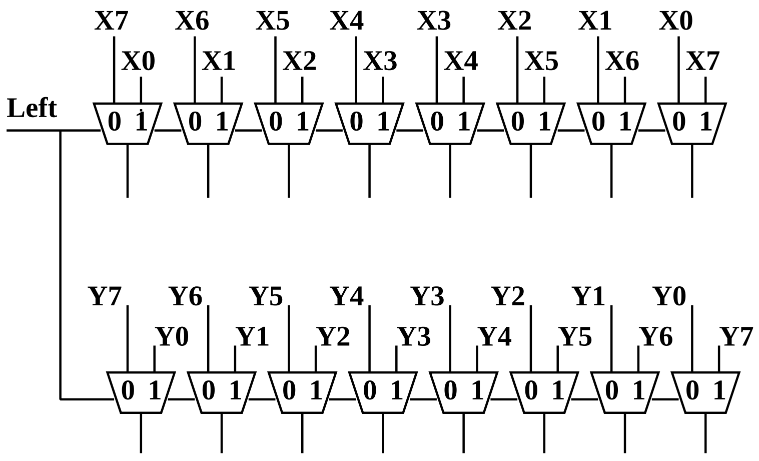


Figure 10.4: Bit reversal for bidirectional Shift and Rotate Operations

done by adding rows of muxes at the input and output which reverse the order of bits.

We can also make use of the fact that a left rotate by m places is the same as a right rotate by $2^n - m$ places. Now $2^n - m$ is just the 2's complement of m in an n bit representation. By presenting the 2's complement of m at the mux controls, we can convert a right rotate to a left rotate. This can be followed by a mask operation, if a shift operation was required, rather than a rotate.

Chapter 11

Multipliers

11.1 Shift and Add Multipliers

An obvious way for implementing multipliers is to replicate the paper and pencil procedure in hardware.

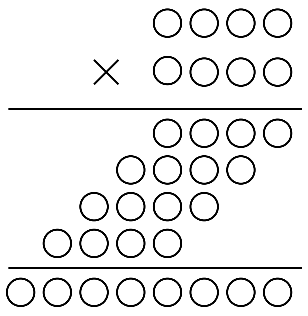


Figure 11.1: Shift and add multiplier

- Initialize the product to 0, extend multiplicand to left by n bits filled with 0s.
- If the least significant bit of the multiplier is 1, add the multiplicand to product, else do nothing.
- Shift the multiplier right by one bit.
- Shift the multiplicand left by one bit.
- Repeat for n bits

Each term being added to form the product is called a partial product. The name "partial product" is also used for individual bits of the terms being added - so beware!

The paper-pencil procedure requires n-1 additions to a 2n bit accumulator. This uses a single adder, but takes long to complete the multiplication. A 32 x 32 multiplication will require 31 addition steps to a 64 bit accumulator.

Multiplication can be made faster by using multiple adders and adding terms in a tree structure.

11.2 Array Multipliers

Suppose we want to multiply two n-bit numbers A and B, where

$$A = \sum_{i=0}^{n} 2^i a_i \qquad B = \sum_{j=0}^{n} 2^j b_j$$

We can regard all bits of the partial products as an array, whose (i,j)th element is $a_i \cdot b_j$. Notice that each element is just the AND of a_i and b_j. **All** elements of the array are available in parallel, within one gate delay of arrival of A and B. We can now use an array of full adders to produce the result. One input of each adder is the sum from the previous row, the other is the AND of appropriate a_i and b_j. This architecture is called an array multiplier.

A 4X4 array multiplier is shown in fig.11.2. Half adders can be used at the right end.

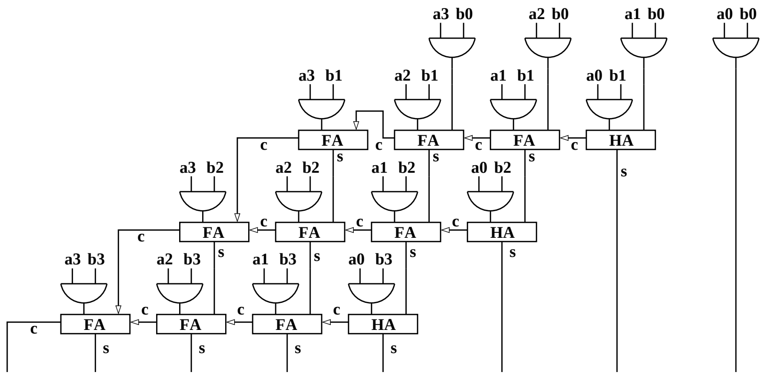


Figure 11.2: Array multiplier

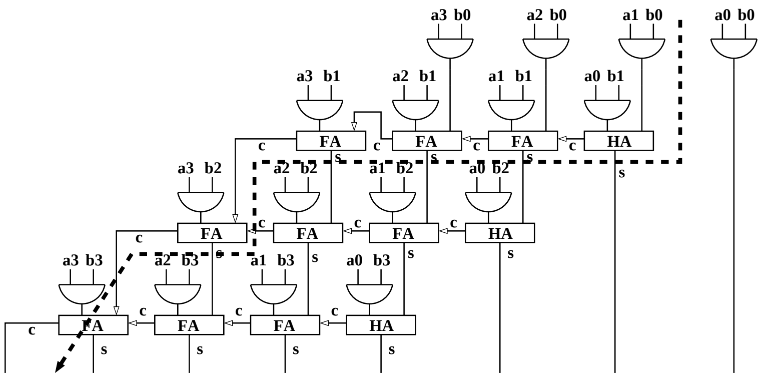


Figure 11.3: Critical path in an array multiplier

11.2.1 Critical Path through an Array Multiplier

The critical path through a 4X4 array multiplier is shown in fig.11.3 The critical path involves carry as well as sum outputs!

11.3 Speeding up Multipliers

The array multiplier has a regular layout with relatively short connections. However, it is still rather slow. How can we speed up a multiplier?

There are two possibilities:

- Somehow reduce the number of partial products to be added. For example, could we multiply 2 bits at a time rather than 1?
- Since we have to add more than two terms at a time, use an adder architecture which is optimized for this.

11.4 Booth Encoding

Booth Encoding reduces the number of partial products by multiplying 2 bits at a time.

Let the multiplicand be A and the multiplier B. Rather than multiplying A with successive bits of B, we can multiply it with two bits of B at a time. Depending on the two bits being 00, 01, 10 or 11, the partial product will be 0, A, 2A or 3A.

- 0 and A can be produced trivially.

- 2A can be produced easily by a left shift of A.
- Generating 3A presents a problem!

However, 3A can be expressed as 4A - A. The task of adding 4A is passed on to the next group of 2 bits of the multiplier. Since the place value of the next group of 2 bits is 4 times the current one, adding 4A to the product is equivalent to adding 1 to the next group of 2 bits of the multiplier. -A can be generated from A, using an adder/subtracter rather than an adder for accumulating the sum of partial products.

11.5 Modified Booth Encoding

To simplify the logic for deciding whether an additional 4A should be added on behalf of the less significant 2 bits in the multiplier, we express 2A also as 4A - 2A.

Since we anyway have an adder-subtracter, this requires no additional resources. The modified logic is: for 00, do nothing. For 01, add A.
for 10, subtract 2A, ask the next group to add 4A.
for 11, subtract A, ask the next group to add 4A.

Now the next group can just look at the more significant bit of the previous group and add 1 to the multiplier if it is '1'.

The partial product generator looks at the current 2 bits and the MSB of the previous group of 2 bits to decide its action. Thus, we scan the multiplier 3 bits at a time, with one bit overlapping. For the first group of 2 bits, we assume a 0 to the right of it. After handling the previous group, the multiplicand is shifted left by 2 positions. Thus, it has already been multiplied by 4. Therefore, adding 4 A on behalf of the previous group is equivalent to adding 1 to the multiplier corresponding to the current group.
Table 11.1 summarizes the effective multiplier for generating the partial product. The partial product generator looks at the current 2 bits and the MSB of the previous group of 2 bits to decide its action. Thus, we scan the multiplier 3 bits at a time, with one bit overlapping. For the first group of 2 bits, we assume a 0 to the right of it. After handling the previous group, the multiplicand is shifted left by 2 positions. Thus, it has already been multiplied by 4. Therefore, adding 4 A on behalf of the previous group is equivalent to adding 1 to the multiplier corresponding to the current group. Notice that a 111 in the 3 bit group being scanned requires no work at all.

What happens if there is a string of '1's in the multiplier? Consider multiplication by $111\cdots111$. As described earlier, we should add implied zeros to the right and left of this multiplier.

Current 2-bits	Multiplier for these	Previous MSBit	Pending Increment	Total Multiplier
00	0	0	0	0
01	+1	0	0	+1
10	-2	0	0	-2
11	-1	0	0	-1
00	0	1	+1	+1
01	+1	1	+1	+2
10	-2	1	+1	-1
11	-1	1	+1	0

Table 11.1: Effective multipliers for Modified Booth Algorithm

- Because the group begins with 110, there will be a -1 in the beginning.
- The group ends with 011. So there will be a +2 at the end,
- However, for the length of continuous '1's, nothing needs to be done (add zeros).

Curr. 2 bits	Prev. MSB	Multi-plier
00	0	0
00	1	+1
01	0	+1
01	1	+2
10	0	-2
10	1	-1
11	0	-1
11	1	0

Thus Booth encoding reduces the number of partial products to about half (multiplying 2 bits at a time). It also makes addition in columns of partial products fast because carry propagation during addition will be reduced.

11.6 Efficient Addition of Partial Products

Multipliers can be speeded up by using special adder architectures which are optimized for adding more than two numbers. One option is to use tree adders rather than an accumulator. Several additions proceed in parallel, since all partial products are generated together.
Fig.11.4 shows the use of tree adders in a multiplier.

11.6.1 Carry Save Adders

Ordinary adders are large and complex. Also, these are slow due to rippling of carry. Let us consider an adder which presents its output not as one word - but two. The actual result is

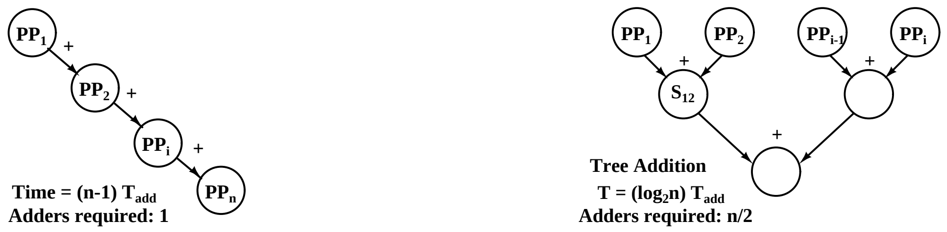


Figure 11.4: Tree additions in multipliers

the sum of these.

Obviously, this kind of adder is of no use for adding just two words! But it can be useful in a multiplier where we are adding multiple terms. For each bit column, the sum goes into one output word, while carry outs go into the other (without being added to the next more significant column). Now there is no rippling of carry and the output is available in constant time.

We do need a conventional adder in the end to add these two words. This type of adder, which reduces the product to two words which must be added using a conventional adder is called a "Carry Save Adder" or CSA.

For example, we can construct a useful CSA for adding 4 bits in the same column. The 4 input 2 output CSA uses two full adders as shown in fig.11.5. We make use of the fact that all partial product bits are available in constant time after the application of inputs.

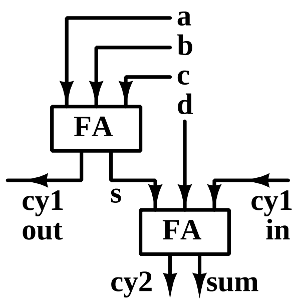


Figure 11.5: Carry save adder

Since there are 4 bits to be added, we feed three of them to a full adder. The sum and carry output of this adder is then available in constant time. The sum output of first FA goes to the second FA. The carry output (cy1) of the first FS goes as intermediate input to the CSA used in the column to the left of this one.

The second FA accepts one left over bit from the partial product column, the sum output of the first Full Adder FA1 and cy1 output coming from the CSA to its right. All inputs to this Full Adder are also available in constant time.

Notice that even though cy1 goes from one column to the next significant column, **it does not ripple all the way** horizontally. It goes to FA2 of the more significant column whose output is not required by the next column.

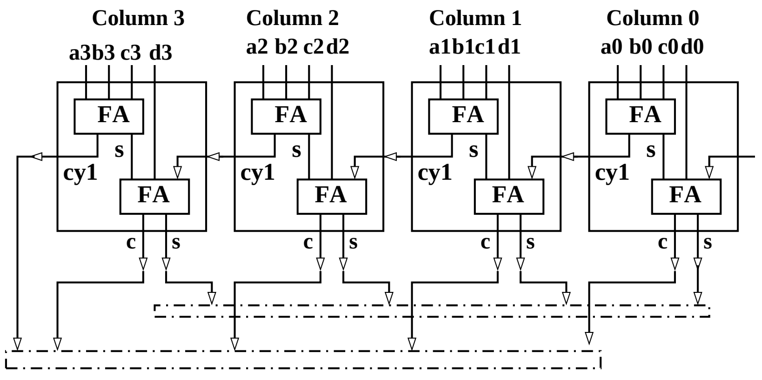


Figure 11.6: Tiling Carry Save Adders

11.6.2 Critical Path of Carry Save Adders

Figure11.6 shows how we can add 4 columns of 4 bits each.
Rows are labeled as a,b,c and d while columns are indexed as 0,1,2 and 3. Outputs are collected in two separate registers (shown in dotted lines). These must be added using a conventional adder. Critical path of a 4x4 Carry Save Adder is shown in fig.11.7. One can

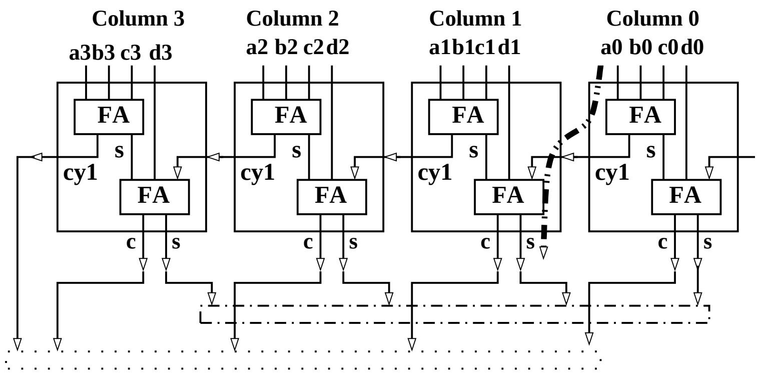


Figure 11.7: Critical path of a carry save adder

see that the critical path has been broken up. Addition of 4 words of 32 bits each will have a critical path of the same length as that for 4 words of 4 bits each.

11.7 Wallace Multipliers

Multipliers do not have the same number of bits in every column. In 1964, Wallace proposed a method for a carry save adder like reduction scheme which is valid for columns of variable size. Wallace multiplier assumes adders which take multiple inputs of the same weight and produce sum outputs of the same weight and carry outputs with higher weights. These are combined in stages to reduce the number of terms at each weight to 2 or less. These two terms are then added by a conventional adder to produce the final result.

Wallace multipliers act in three stages:

1. Generate all bits of the partial products in parallel.
2. Collect all partial products bits with the same place value in bunches of wires and reduce these in several layers of adders till each weight has no more than two wires.
3. For all bit positions which have two wires, take one wire at corresponding place values to form one number, and the other wire to form another number.

   Add these two numbers using a fast adder of appropriate size.

We assume that Full adders and Half adders will be used. A full adder takes 3 inputs and produces one output of the same weight (sum) and another of higher weight (carry). This is called a (3,2) adder. It reduces the number of wires at its own weight by 2 and adds one wire at the higher weight.

A half adder takes 2 inputs and produces one output of the same weight (sum) and another of higher weight (carry). This is a (2,2) adder. It reduces the number of wires at its own weight by 1 and adds one wire at the higher weight.

11.7.1 Reduction Stage of Wallace Multipliers

The reduction algorithm is general and can be used with any adders of type (n,m). For example, a carry save adder is of type (4,2).

- Each reduction stage looks at the number of wires for each weight and if any weight has more than 2 wires, it adds a layer of adders.
- When the numbers of wires for each weight have been reduced to 2 or less, we form one number with one of the wires at corresponding place values and another with the other wire (if present).
- These two numbers are added using a fast adder of appropriate size to generate the final product.

The original paper by Wallace did not give a reduction algorithm in detail. Subsequently, the Wallace reduction algorithm has been interpreted in many ways.We first describe the algorithm as interpreted by Dadda and other authors in many papers on multipliers. This, however, produces redundant most significant bits for multipliers of some sizes. For example, an 8x8 bit multiplier following this scheme will produce 17 bits. Of course, the 17th bit will always come out to be '0', because the product can be no wider than 16 bits. Since this scheme is widely described in literature, we shall first give the details of this scheme. Subsequently, we give a Wallace wire reduction scheme which does not lead to a redundant bit.

Wire Reduction Scheme for Wallace Multipliers

If any weight contains more than two wires, we use a reduction algorithm to reduce the number of wires. We divide the total number of rows to be added in groups of 3 rows. Any rows which are additional to these groups are passed on to the next stage as they are.

Next we take groups of 3 rows one at a time. If there are 3 wires in any column of this group, we place a full adder. The sum output of this adder goes to the same column in the next stage. The carry output of the adder goes to the column with next higher weight. If a column has two wires, it is reduced with a half adder. Again the sum output goes to the same column in the next stage and the carry wire joins the column with higher weight. If a column has a single wire, it is passed through to the next stage.

This is repeated for all groups of 3 rows.

When all groups of three rows have been reduced this way, we count wires at all weights and continue this procedure till no weight has more than two wires.

At the end, many contiguous bits at the least significant end will have a single wire. These are carried through to the result. For bits which have two wires, one is allocated to one word and the other to another word, and these two words are added using a fast adder.

11.7.2 Wallace Multiplier Example

Consider a multiplier for 4X4 bits. Partial products are generated in parallel and the number of wires at each weight are shown in the table on the right.

We shall use this example to introduce the "dot convention" used for representing wire reduction in many multipliers.

Bit	Terms	Wires
0	a0b0	1
1	a0b1, a1b0	2
2	a0b2, a1b1, a2b0	3
3	a0b3, a1b2, a2b1, a3b0	4
4	a1b3, a2b2, a3b1	3
5	a2b3, a3b2	2
6	a3b3	1

4X4 Wallace Multiplier: First Reduction

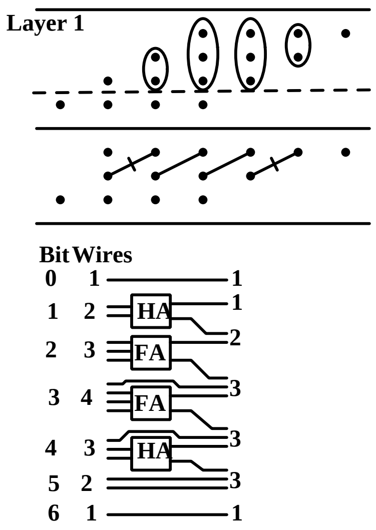


Figure 11.8: First reduction stage of a 4x4 Wallace multiplier

Figure 11.8 shows a 4x4 Wallace multiplier. This multiplier has 4 rows of partial products. The upper three rows are grouped together and can have 1, 2 or 3 wires. the bottom row is just passed through to the next stage.
Each dot in the upper diagram represents a bit at its respective weight. When we place a full adder to reduce a column, a dot representing the sum is placed at the same weight in the next stage. A dot representing carry is placed at the next higher weight. These two dots are joined by a line to show that these two are the results from a full adder. Similarly, when we use a half adder, we place a dot in the same column for the sum and a dot in the next higher weight column for the carry. These are joined with a crossed line to show that these are from a half adder. Bits which are passed through to the next stage are represented by dots not

connected to any line.
After grouping the original 4 rows of partial products in groups of 3 and 1:
Bit 0 has a single wire: which is passed through.
Bit 1 has 2 wires: which are fed to a half adder.
Bits 2 and 3 have 3 wires each, which are fed to full adders.
Bit 4 has 2 wires (in the group of 3), which are fed to a half adder.
Bit 5 has 1 wire, which is passed through.

4X4 Wallace Multiplier: Second Reduction

After first reduction, there are three rows of wires (at bits 3, 4 and 5). So we need another reduction stage. The three rows form a group and there are no passed through rows in this stage. Bits 0 and 1 have single input wires, which are passed through.

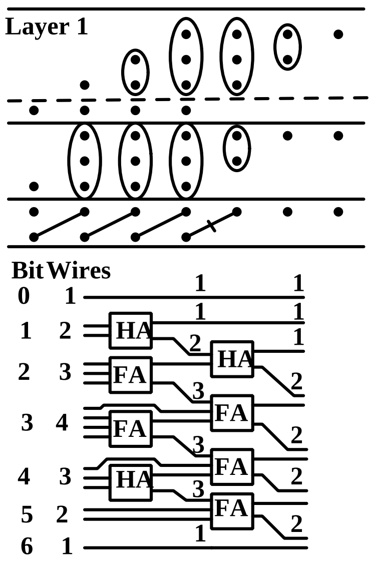


Figure 11.9: Second reduction stage of a 4x4 Wallace multiplier

Bit 2 has 2 wires. These are reduced using a half adder. The sum goes as the only output wire at this bit, while the carry goes to bit 3.
Bits 3, 4 and 5 have 3 input wires each.These are reduced using full adders. Thus these bits have two wires at their outputs – one from the sum of their full adder and the other from the carry produced by the adder at the lower weight.
Bit 6 has one input wire. This is joined by the carry of the adder at bit 5 to produce 2 output wires.

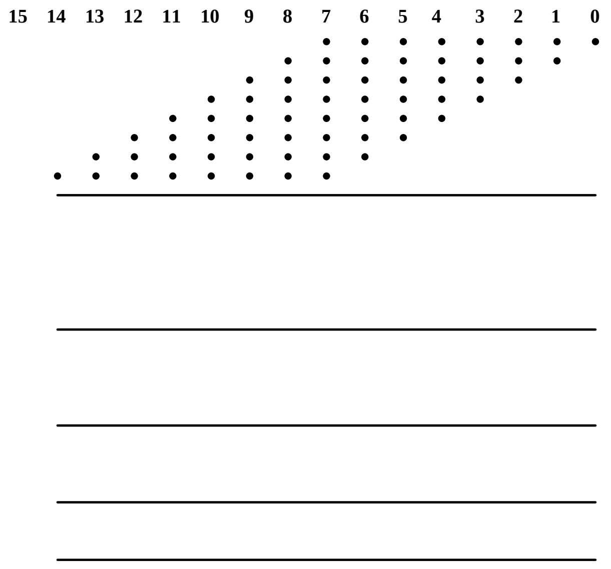
Final Addition

After the second layer, no weight has more than 2 wires. Single wires at bits 0, 1 and 2 are fed through to the output.

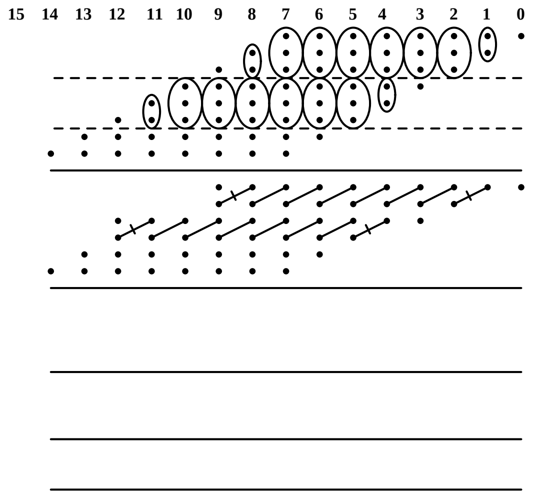
- A fast conventional adder is used to add the 2 bits each at bits 3, 4, 5 and 6.
- Notice that we do not need a full width fast adder. This is because the half adders at low weights have already rippled the carry while the rest of weights were being reduced.
- This makes the final adder smaller and faster.

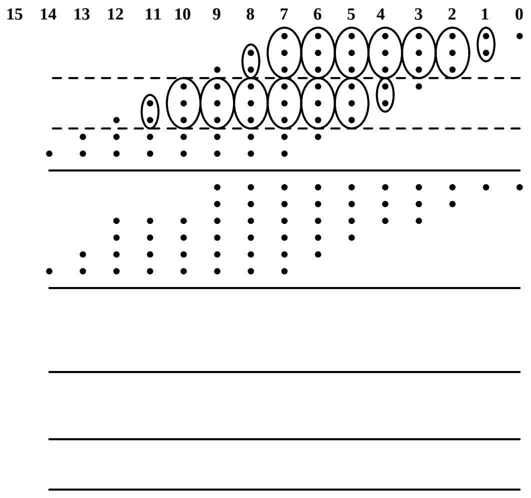
11.7.3 Redundant MSB in large Wallace Multipliers

The reduction scheme as described above sometimes produces a redundant most significant bit. The result is still correct and the redundant bit will always be zero. To see this effect, let us apply the above scheme to an 8x8 multiplier.

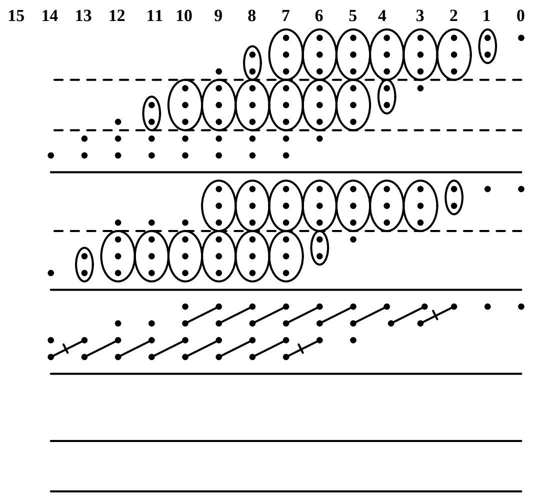


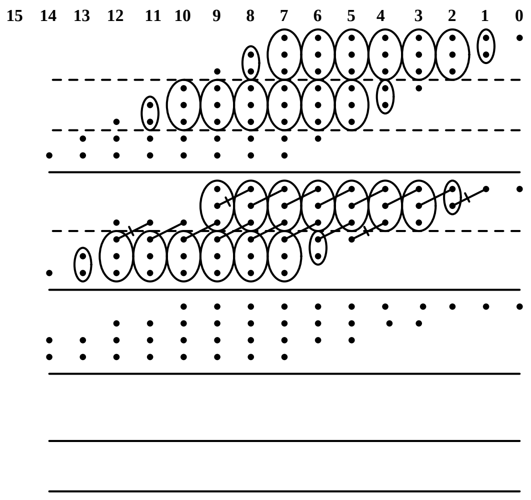
The eight rows of partial products are divided into groups 3, 3 and 2. The last two rows are just passed through to the next stage.



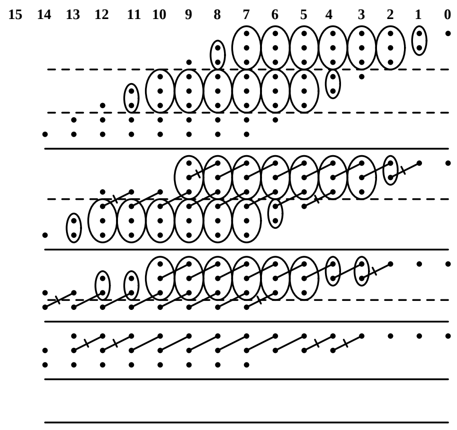


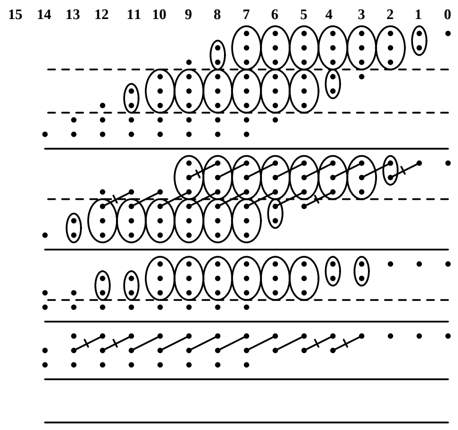
Taking each group of 3 rows, we place a full adder wherever we find three dots and a half adder where there are two dots. Single dots are passed through. This reduces the output rows to 6.



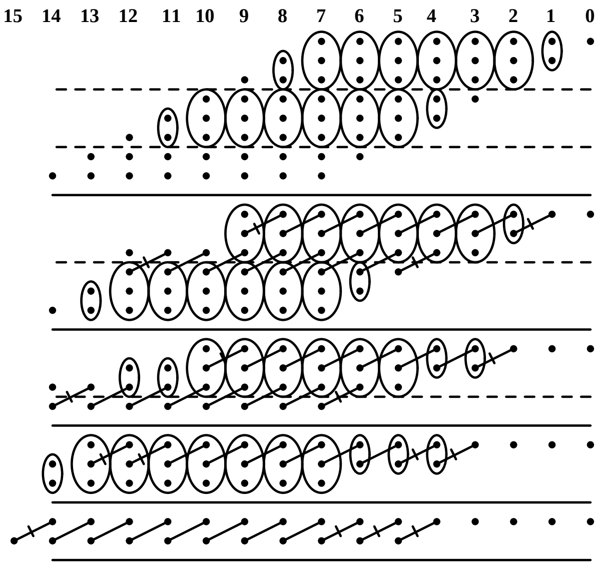


The six rows are divided into two groups of three each and we place full and half adders as shown in the figure on the left above. This reduces the number of rows to four.





The four rows output from the previous stage are grouped as 3 and 1 as shown in the figure on the left. The fourth row is just passed through, while the group of 3 rows is reduced by full and half adders. we are now left with 3 rows of output wires which need to be reduced to two by the next stage.



The 3 rows of input wires to the last stage form a single group and are reduced by full and half adders to 2 rows as shown in the figure above.

As one can see, there are two bits at bit-14, which can produce a carry during the final addition. When this carry is added to the bit at bit-15, it could produce another carry which will go to bit-16. This would be an extra bit. (Bits 0 to 16 will be 17 bits). In practice, 17 bits will not be produced, as multiplying 8 bit operands should generate at the most a 16 bit result.

11.7.4 Avoiding the Redundant MSB in Wallace Multipliers

One can avoid the redundant bit by modifying the reduction scheme:
We treat all wires in a column as equivalent. (No groups of 3 rows).
As long as there are 3 or more wires, make bunches of 3 wires and send each to a full adder. Now we can be left with 0, 1 or 2 wires. There is nothing to do for 0 wires left. If one wire is left, it is passed through to next layer.

When two wires are left, we have a more complex decision to take. For this, we first need to define the capacity of a reduction layer.

Wire capacity of a reduction layer

We define the capacity of a layer as the maximum number of wires it can accommodate. How can we determine it?

We know that the final reduction layer should have no more than 2 wires. Now we can work **backwards** from the final layer to the first. Let d_j represent the maximum number of wires for any weight in layer j, where $j = 1$ for the final adder. (Thus $d_1 = 2$). The maximum number of wires which can be handled in layer j+1 (from the end) is the integral part of $(3/2)d_j$ with $j = 1$ for the final adder. Thus $d_1 = 2$. We go up in j, till we reach a number which is just greater than or equal to the largest bunch of wires in any weight. The number of reduction layers required is this $j_{final} - 1$. Capacities of layers starting from last layer and moving towards the top are 2, 3, 4, 6, 9, 13, 19 . . . .

Reduction of two wires

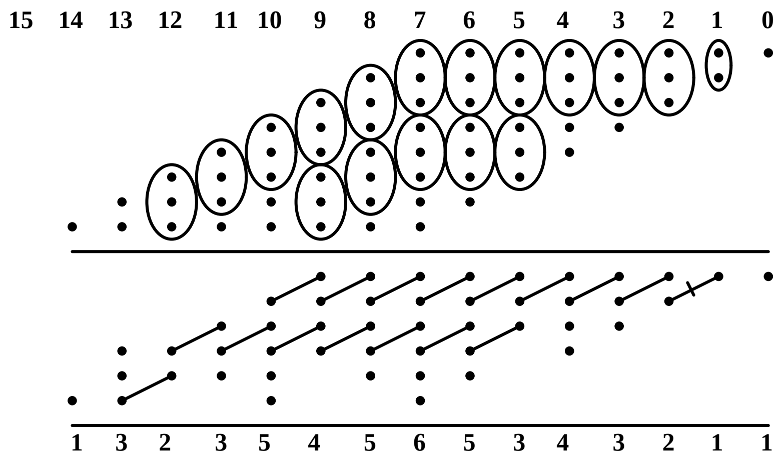
Now we can define the policy for reduction of 2 left over wires after deploying the maximum number of full adders.

- If all columns at the right have a single wire, we reduce the two wires using a half adder. (This helps in reducing the width of final adder).
- If there is a column to the right with more than one wire, we pass through the two wires to the next layer if it can accommodate these. (That is, the total number of wires do not exceed the capacity of that layer).
- If passing through the two wires would exceed the capacity of next layer, we reduce these with a half adder.

11.7.5 Wallace 8x8 Reduction without redundant MSB

We start with a maximum of 8 wires in any column. Capacity of the next layer is 6.

Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
FA	0	0	0	1	1	1	2	2	2	2	2	1	1	1	0	0
Remaining	0	1	2	0	1	2	0	1	2	1	0	2	1	0	2	1
HA	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
PT	0	1	2	0	1	2	0	1	2	1	0	2	1	0	0	1
Sums	0	0	0	1	1	1	2	2	2	2	2	1	1	1	1	0
Carries to Higher bits	0	0	0	1	1	1	2	2	2	2	2	1	1	1	1	0
Output Wires	0	1	3	2	3	5	4	5	6	5	3	4	3	2	1	1

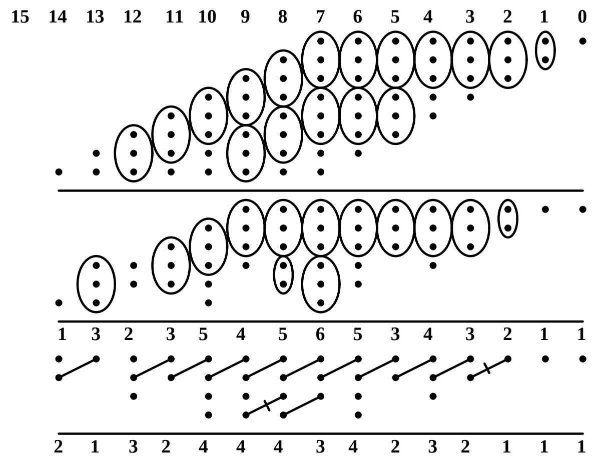


Two wires need to be handled at bits 1, 4, 7, 10 and 13. Since there is a single wire at bit 0, the two wires at bit 1 are reduced using a half adder. In all other cases, passing through the two wires will not make the number of output wires greater than 6. So we pass those through.

Second reduction layer

At this layer, maximum wires in any column is 6. Capacity of the next layer is 4.

Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	1	3	2	3	5	4	5	6	5	3	4	3	2	1	1
FA	0	0	1	0	1	1	1	1	2	1	1	1	1	0	0	0
Remaining	0	1	0	2	0	2	1	2	0	2	0	1	0	2	1	1
HA	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
PT	0	1	0	2	0	2	1	0	0	2	0	1	0	0	1	1
Sums	0	0	1	0	1	1	1	2	2	1	1	1	1	1	0	0
Carries to Higher bits	0	0	1	0	1	1	1	2	2	1	1	1	1	1	0	0
Output Wires	0	2	1	3	2	4	4	4	3	4	2	3	2	1	1	1

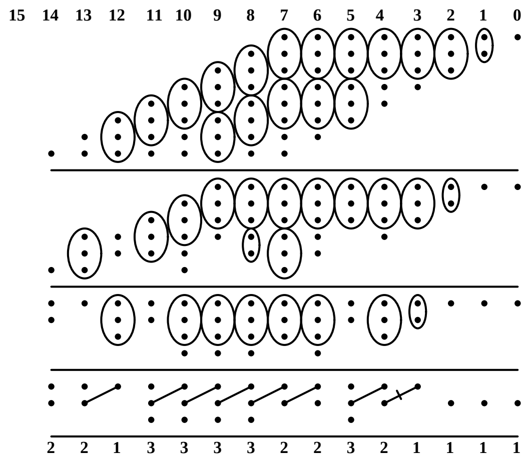


This time, we have to handle 2 wires at bits 2, 6, 8, 10 and 12. Since there is a single wire at bits 0 and 1, we reduce the two wires at bit 2 using a half adder. Two wires at bit 6 can be passed through because the number of output wires will not exceed 4. However, at bit 8, we anticipate two carry wires from bit 7 and a sum wire from the bunch of 3 wires at bit 8 itself. Passing through the 2 wires will make the number of output wires 5, which will exceed the capacity of the next layer (4). Therefore the 2 wires left at bit 8 must be reduced using a half adder. Two wires at bits 10 and 12 can be passed through because the number of output wires do not exceed 4.

Third reduction layer

Capacity of next layer is 3.

Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	2	1	3	2	4	4	4	3	4	2	3	2	1	1	1
FA	0	0	0	1	0	1	1	1	1	1	0	1	0	0	0	0
Remaining	0	2	1	0	2	1	1	1	0	1	2	0	2	1	1	1
HA	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
PT	0	2	1	0	2	1	1	1	0	1	2	0	0	1	1	1
Sums	0	0	0	1	0	1	1	1	1	1	0	1	1	0	0	0
Carries to Higher bits	0	0	0	1	0	1	1	1	1	1	0	1	1	0	0	0
Output Wires	0	2	2	1	3	3	3	3	2	2	3	2	1	1	1	1

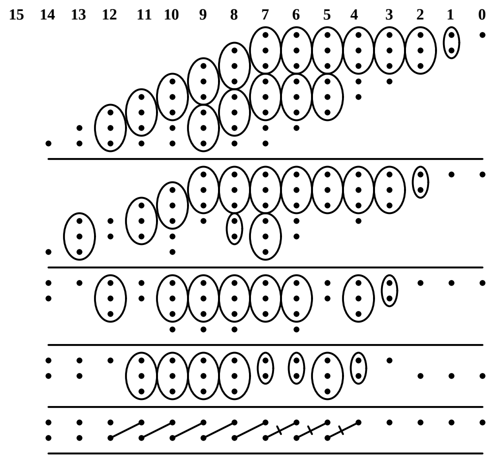


This time we have to handle two wires at bits 3, 5, 11 and 14. Since there is a single wire at bits 0, 1 and 2, we should reduce the 2 wires at bit 3 using a half adder. Passing through the 2 wires at bits 5, 11 and 14 does not exceed 3 output wires, so these are passed through.

Final reduction layer

Capacity of next layer is 2.

Bit No.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
In Wires	0	2	2	1	3	3	3	3	2	2	3	2	1	1	1	1
FA	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0
Remaining	0	2	2	1	0	0	0	0	2	2	0	2	1	1	1	1
HA	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0
PT	0	2	2	1	0	0	0	0	0	0	0	0	1	1	1	1
Sums	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
Carries to Higher bits	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0
Output Wires	0	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1



This time there are two wires at bit positions b4, b6, b7, b13 and b14. As before, we should reduce b4 wires using a half adder because there are single wires at all less significant positions. Passing through the 2 wires at bits 6 and 7 would produce 3 output wires, exceeding the target of 2 wires. Therefore here too we place half adders. However, there is no incoming carry at bits b13 and b14 and so we can pass the 2 wires through.

Thus we have reached two wires without generating a wire at b15. There are two wires at b14 and if these produce a carry, it will go to b15. However, there is no redundant b16 now.

We have reduced the number of wires at all bit positions to ≤ 2 without generating a bit at b15.
There are two wires at b14 and if these produce a carry, it will go to b15 and there is no redundant b16.

11.7.6 Dadda Multipliers

Dadda multipliers are very similar to Wallace multipliers and use the same 3 stages:

1. Generate all bits of the partial products in parallel.
2. Collect all partial products bits with the same place value in bunches of wires and reduce these in several layers of adders till each weight has no more than two wires.
3. For all bit positions which have two wires, take one wire at corresponding place values to form one number, and the other wire to form another number.

   Add these two numbers using a fast adder of appropriate size.

The difference is in the reduction stage.

Wallace multipliers reduce as soon as possible, while Dadda multipliers reduce as late as possible. Dadda multipliers plan on reducing the final number of wires for any weight to 2 with as few and as small adders as possible. We determine the number of layers required first, beginning from the **last** layer, where no more than 2 wires should be left. The number of layers in Dadda multipliers is the same as in Wallace multipliers.

We work back from the final adder to earlier layers till we find that we can manage all wires generated by the partial product generator.

We know that the final adder can take no more than 2 wires for each weight.

Let d_j represent the maximum number of wires for any weight in layer j, where $j = 1$ for the final adder. (Thus $d_1 = 2$).
The maximum number of wires which can be handled in layer j+1 (from the end) is the integral part of $3/2d_j$.

We go up in j, till we reach a number which is just greater than or equal to the largest bunch of wires in any weight. The number of reduction layers required is this final j -1.

Wire Reduction in Dadda Multipliers

At each layer we know the maximum number of wires which should be left for the next layer.

For each weight, we place the least number of smallest adders, such that the wires going out to the next layer do not exceed the maximum number of wires it can handle.

At each weight, we must consider all the incoming wires, as well as the carry wires which will be transferred from the less significant weights in the next layer.

That is why we must begin with the lowest weight and go towards higher weights in each layer.

Dadda Multiplier: Example

Take the example of 4-bit by 4-bit multiplication multiplying a3a2a1a0 by b3b2b1b0. As before, partial products are generated in parallel and we have the following wires:

Weight	Terms	Wires
1	a0b0	1
2	a0b1, a1b0	2
4	a0b2, a1b1, a2b0	3
8	a0b3, a1b2 a2b1 a3b0	4
16	a1b3, a2b2, a3b1	3
32	a2b3, a3b2	2
64	a3b3	1

Number of Reduction Layers

Maximum no. of wires for any weight in this example is 4. $d_1 = 2$, $d_2 = 3$, $d_3 = 4$. So we need 2 layers of reduction.

The first reduction layer should reduce the number of wires at any weight to a maximum of 3. The second layer will then reduce these to a maximum of 2 wires.

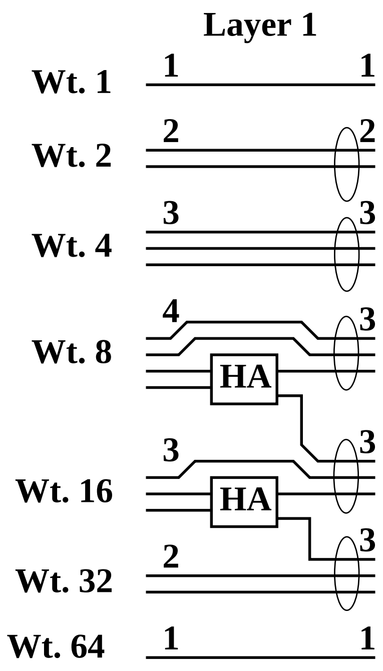
At each reduction layer, we scan from less significant weights to more significant ones, keeping track of additional carry wires which will be transferred *at the output* from lower weights to higher ones.

First Reduction Layer

Weight	Wires
1	1
2	2
4	3
8	4
16	3
32	2
64	1

- Weights 1, 2 and 4 have 3 or less wires. These are passed through.
- Weight 8 has 4 wires. No carry is anticipated from lower weights. A half adder is used to reduce the output wires to 3. (Half Adder Sum + 2 wires passed through).
- Weight 16 has 3 wires, but we anticipate a carry from the adder at weight 8. So we should reduce by 1 to keep the total **output** wires to 3. So this column is also reduced using a half adder.

After First Reduction



- Wt.1 has the single wire which was fed through.
- Wt.2 has 2 fed through wires.

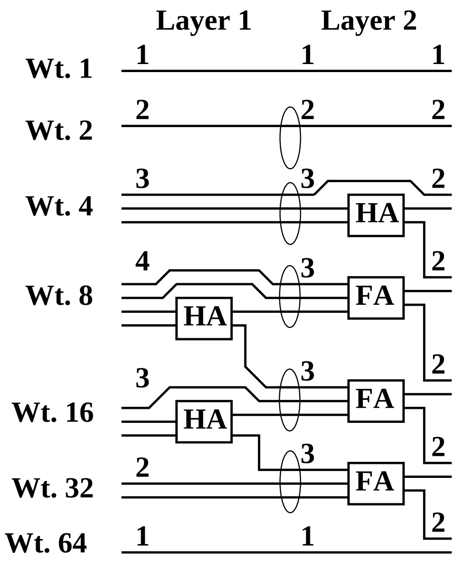
- Wt.4 has 3 wires: all passed through.
- Wt.8 has 3 wires: sum of the half adder at wt.4, and 2 passed through.
- Wt.16 has 3 wires: carry of wt. 8, sum of half adder at 16 and 1 passed through.
- Wt.32 has 3 wires: carry of wt. 16 and 2 passed through.
- Wt.64 has 1 fed through wire.

Second Reduction

In the second layer, we should leave no more than 2 wires at any weight, as this is the last stage.

- As before, we anticipate the number of carry wires transferred from the lower weight when planning reduction using half or full adders.
- In Dadda multipliers, we use minimum hardware during reduction. So the smallest adder which will reduce the output wires to 2 will be used.
- At the lowest weights, if the number of wires is less than or equal to 2, we just pass these through.
- So the single wire at Wt. 1, and the 2 wires at Wt. 2 are just fed through.

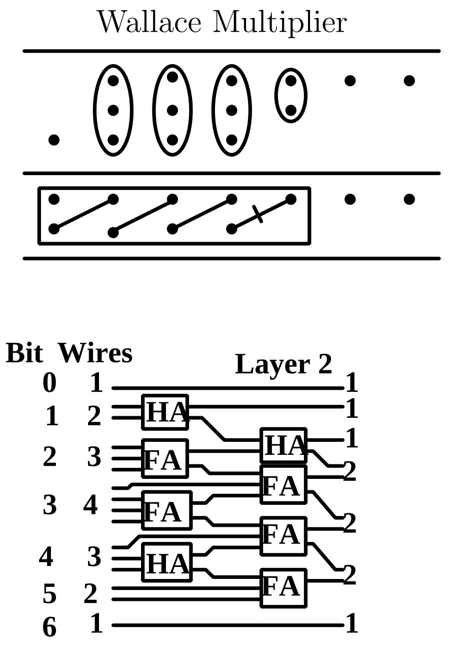
4X4 Dadda Multiplier: Second Reduction

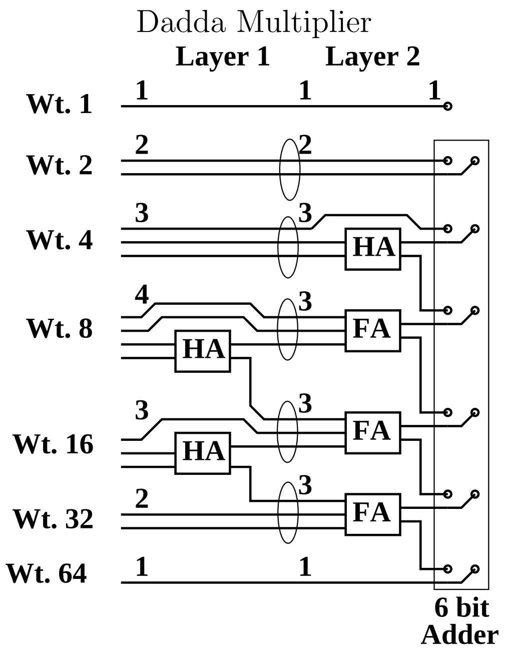


- 3 wires at Wt. 4 are reduced to 2 by a half adder: No carry in expected.

- Wt. 8 has 3 input wires. Carry will arrive from Wt. 4. Reduced using a Full adder.
- Wt. 16 has 3 input wires. Carry will arrive from Wt. 8. Reduced using a full adder.
- Wt. 32 has 3 input wires. Carry will arrive from Wt. 16. Reduced using a full adder.
- Wt. 64 has 1 input wire. Carry will arrive from Wt. 32, making it 2 output wires, which will be fed through.

4X4 Multipliers: Final Addition





Notice that we have used only 3 Full Adders and 3 Half Adders during reduction, whereas Wallace multiplier requires 5 Full Adders and 3 Half Adders.
However, we require a 6-bit final adder for Dadda multiplier, whereas Wallace multiplier needs only a 4 bit final adder.

11.8 Multiply and Accumulate circuits

A common operation required in data processing is evaluation of quantities of the type $\sum c_i X_i$. This requires that the value of the product at each term should be added to the current value of the sum to get its new value. For implementing such calculations, it would be useful if we can have a circuit in hardware which will efficiently multiply two operands as well as add a third. Notice that this third operand will be of the size of the product. Such circuits are called Multiply and Accumulate circuits.

Multiply and Accumulate circuits are easy to implement because during multiplication, we are anyway adding multiple bits in a column. The accumulator just provides an additional bit in every column and this can be easily added along with the partial product bits in a carry-save fashion using any of the wire reduction techniques described above.

Consider for example a Multiply and Accumulate circuit which multiplies two 8 bit operands and adds a 16 bit operand to it. Let us apply the Wallace wire reduction technique to compute $A \times B + C$, where A and B are 8 bit operands, while C is a 16 bit operand. The product computation will generate wires at bit positions 0 through 14 in the usual trapezoidal shape. Thus the number of wires from partial sums of the multiplier will be:

Bit pos.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Partial Sum Wires	0	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1

The operand to be added just provides one additional wire at each bit position from 0 to 15. Thus the number of wires at each bit position becomes

Bit pos.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Wire count	1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2

We can now proceed to reduce these wires as a Wallace tree till we are left with no more than 2 wires at each position. Finally, we shall add these two groups using a fast traditional adder. The tables below show the wire reduction at each stage:

Stage 1: Max. wires:9, capacity of next stage = 6

Bit pos.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input Wires	1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2
Full Adders	0	0	1	1	1	2	2	2	3	2	2	2	1	1	1	0
Half Adders	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
Pass Through	1	2	0	1	2	0	1	0	0	2	1	0	2	1	0	0
Output Wires	1	3	2	3	5	4	6	6	5	6	5	3	4	3	2	1

Stage 2: capacity of next stage = 4

Bit pos.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input Wires	1	3	2	3	5	4	6	6	5	6	5	3	4	3	2	1
Full Adders	0	1	0	1	1	1	2	2	1	2	1	1	1	1	0	0
Half Adders	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
Pass Through	1	0	2	0	2	1	0	0	0	0	2	0	1	0	0	1
Output Wires	2	1	3	2	4	4	4	4	4	3	4	2	3	2	1	1

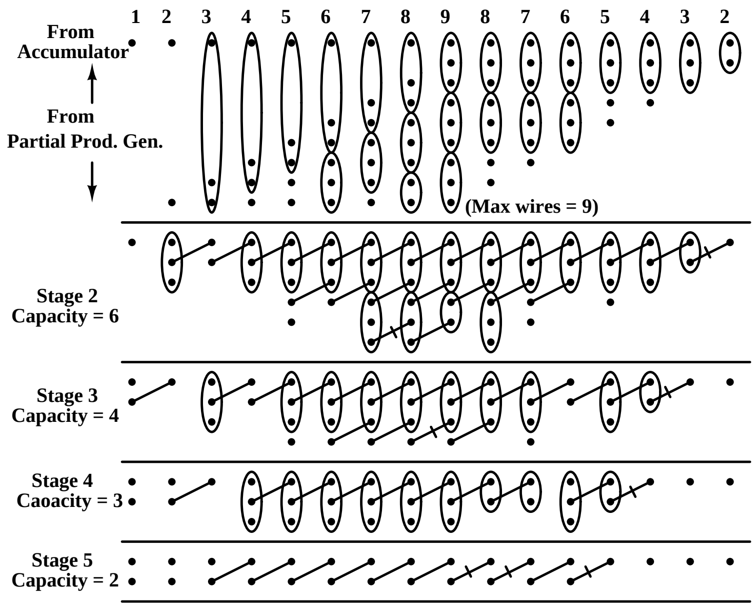
Stage 3: capacity of next stage = 3

Bit pos.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input Wires	2	1	3	2	4	4	4	4	4	3	4	2	3	2	1	1
Full Adders	0	0	1	0	1	1	1	1	1	1	1	0	1	0	0	0
Half Adders	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Pass Through	2	1	0	2	1	1	1	1	1	0	1	2	0	0	1	1
Output Wires	2	2	1	3	3	3	3	3	3	2	2	3	2	1	1	1

Stage 4: capacity of next stage = 2

Bit pos.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Input Wires	2	2	1	3	3	3	3	3	3	2	2	3	2	1	1	1
Full Adders	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0
Half Adders	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0
Pass Through	2	2	1	0	0	0	0	0	0	0	0	0	0	1	1	1
Output Wires	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1

The dot diagram for this scheme is shown below:



Finally, we need a 12 bit fast adder to get the final result.

The complexity of this circuit is not much more than a plain 8x8 multiplier and the time taken to produce the results is much less than the time taken to multiply first and then to add. This is because the latter requires two additions in which the carry ripples while the Multiply and Accumulate circuit requires only one such addition.

11.9 Serial Multipliers

Often, we need multipliers which have very low complexity or very low power consumption and speed is not very important. Serial multipliers are a good option in such cases.

Low complexity multipliers can be bit serial or row serial. Bit serial multipliers require $m \times n$ clocks for completing an $m \times n$ multiplication. Row serial multipliers require only n steps, but we require m full adders rather than just one.

11.9.1 Bit Serial Multipliers

For serial multiplication, partial product bits need not be generated in parallel. These can be generated as and when these are required. Each bit of the multiplier needs to be ANDed

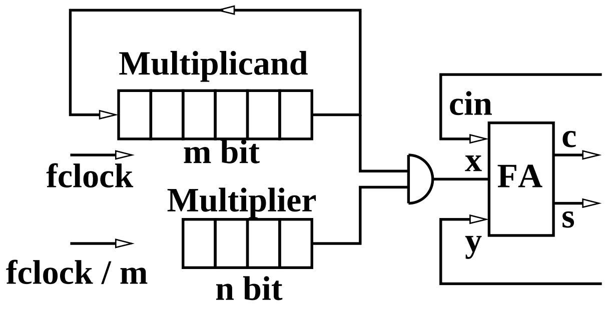
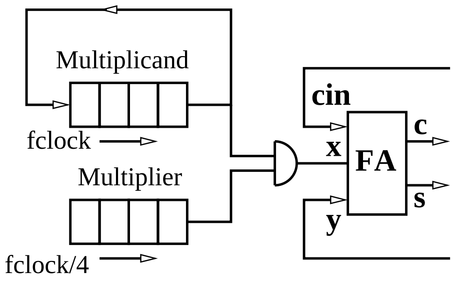


Figure 11.10: Partial product generation for bit-serial multiplication

with each bit of the multiplicand. This requires that all multiplicand bits be presented one after the other every time a new bit from the multiplier is taken up. This can be managed by using a re-circulating shift register for the multiplicand, which is clocked at a rate which is m times faster than the clock supplied to the multiplier shift register. The inputs y and Cin to the full adder have to be appropriately selected and timed to generate the correct product.

Consider a 4×4 bit serial multiplier.

The x input to the Full Adder appears in the following order:



ck	x	ck	x	ck	x	ck	x
0	a0b0	4	a0b1	8	a0b2	12	a0b3
1	a1b0	5	a1b1	9	a1b2	13	a1b3
2	a2b0	6	a2b1	10	a2b2	14	a2b3
3	a3b0	7	a3b1	11	a3b2	15	a3b3

We need additions as follows:

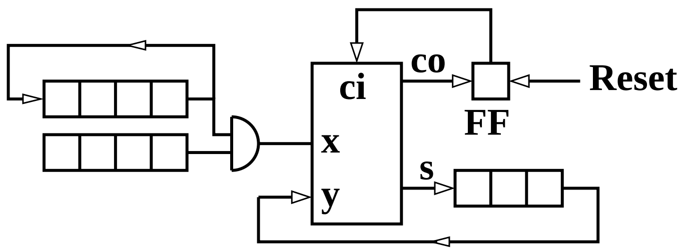
			a3	a2	a1	a0
		×	b3	b2	b1	b0
			a3b0	a2b0	a1b0	a0b0
		a3b1	a2b1	a1b1	a0b1	
	a3b2	a2b2	a1b2	a0b2		
a3b3	a2b3	a1b3	a0b3			

Let us put the arrival time of terms in parentheses next to each term.

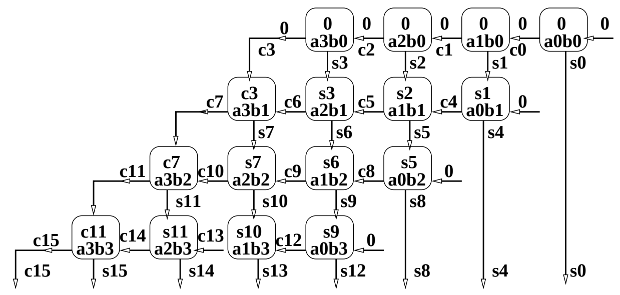
			a3	a2	a1	a0
		×	b3	b2	b1	b0
			a3b0(3)	a2b0(2)	a1b0(1)	a0b0(0)
		a3b1(7)	a2b1(6)	a1b1(5)	a0b1(4)	
	a3b2(11)	a2b2(10)	a1b2(9)	a0b2(8)		
a3b3(15)	a2b3(14)	a1b3(13)	a0b3(12)			

It is clear that for all additions, the earlier terms have to wait for 3 clock cycles before the later terms arrive.

We can manage this by putting a 3 bit shift register at the sum output and presenting the delayed output at the 'y' input of the full adder. The carry output can be added immediately in the next clock, since it should go to the next column to its left. A 3 clock delay for sum and a 1 clock delay for carry leads to the following circuit.



Unfortunately, it does not work as shown!
We need to take care of a few exceptions. Let us look at all the exceptions in detail. In the figure below, each box contains the y and x inputs presented to the adder. The sum and carry terms are indexed by the clock cycle in which these were generated. For example, $c7$ is the carry generated in the 7th clock cycle. Notice that in the first four cycles, 0 is being added to partial products and therefore the carry is always 0.



- At clocks 0, 4, 8 and 12, carry input should be forced to 0.
- At clocks 7, 11 and 15, the adder y input should receive carry terms (c3, c7 and c11) instead of sum terms (s4, s8 and s12).
- The sum terms s4, s8 and s12 should be taken out as result bits and not inserted in the 3 bit delay shift register.

We can take care of these exceptions by inserting the carry FF output (which is a 1 clock delayed version of cout) in the 3-bit shift register instead of the sum terms. Thus C3 (which is always 0), C7 and C11 will emerge from the shift register at clocks 7, 11 and 15 respectively and will be added to the correct partial product bits. The corresponding sum terms should be taken out as result bits.

11.9.2 Bit Serial Multiplier: Implementation

With exception handling at the end of rows, the serial multiplier will work. Notice the changes

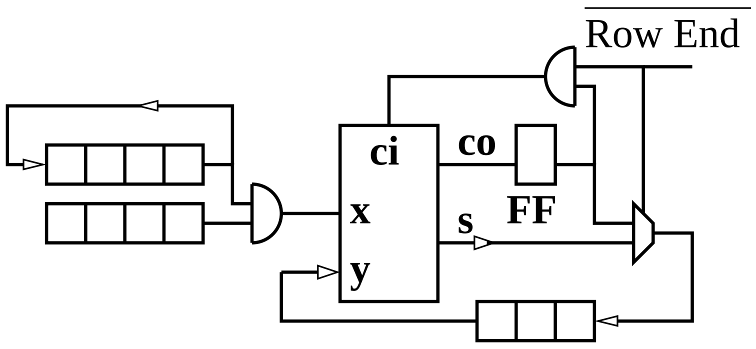


Figure 11.11: 4x4 bit serial multiplier

from the earlier suggested circuit:

- Carry input is forced to 0 at row ends.
- The mux normally inserts the sum into the shift register. However, at row ends, it inserts the delayed carry output.
- The sum terms at row ends are taken out as the low bits of the product.

- One can add another shift register at the output to collect these.
- The 2 more significant bits of the shift register and the last sum and carry provide the high bits of the product at the end.

11.9.3 Row Serial multipliers

We need not reduce the complexity all the way down to a single adder for serial multipliers. We could have a row of n adders in parallel performing n serial additions. We can use n full adders arranged in n columns. The carry output of previous addition is retained at the same column. The sum from the previous addition *in the left column* is brought to this column by a shift operation. This sum has the same weight as the carry generated during the previous clock in this column. These two are added to the partial product bit for this column.

Taking the example of 4×4 multiplication, we are trying to perform the following operations:

			a3	**a2**	**a1**	**a0**
			X b3	**b2**	**b1**	**b0**
			a3b0	**a2b0**	**a1b0**	**a0b0**
		a3b1	**a2b1**	**a1 b1**	**a0b1**	
	a3b2	**a2b2**	**a1b2**	**a0b2**		
a3b3	**a2b3**	**a1b3**	**a0b3**			

The addition process using 4 adders is represented in Figure 11.12. Notice that the same ‘a’ term is used for generating partial products for a given column. The ‘b’ term has to be shifted right every time to generate the right partial product bit.

Sums have to be shifted right to be added to the carry of the previous addition in the same column. 4 additional clock cycles will be required to ripple the carry in the last addition. During these, the partial product bits will be 0.

This scheme can be implemented as follows:

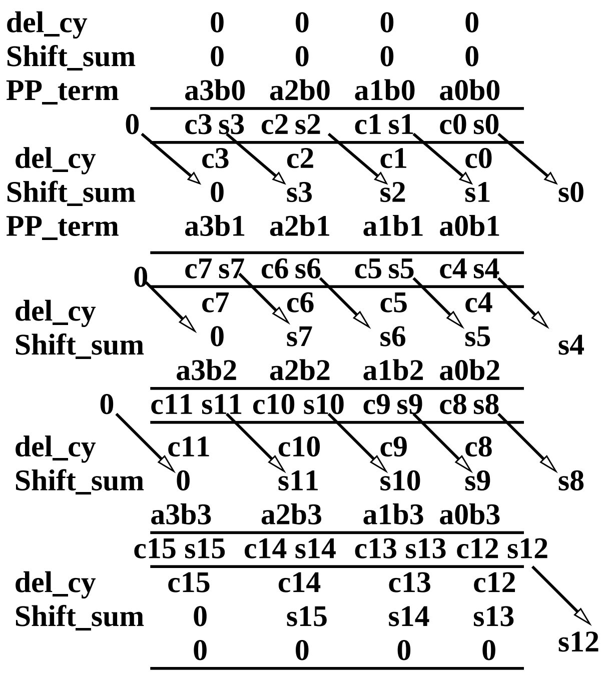

del_cy 0 0 0 0
Shift_sum 0 0 0 0
PP_term a3b0 a2b0 a1b0 a0b0
0 c3 s3 c2 s2 c1 s1 c0 s0
del_cy c3 c2 c1 c0
Shift_sum 0 s3 s2 s1 s0
PP_term a3b1 a2b1 a1b1 a0b1
0 c7 s7 c6 s6 c5 s5 c4 s4
del_cy c7 c6 c5 c4
Shift_sum 0 s7 s6 s5 s4
a3b2 a2b2 a1b2 a0b2
0 c11 s11 c10 s10 c9 s9 c8 s8
del_cy c11 c10 c9 c8
Shift_sum 0 s11 s10 s9 s8
a3b3 a2b3 a1b3 a0b3
c15 s15 c14 s14 c13 s13 c12 s12
del_cy c15 c14 c13 c12
Shift_sum 0 s15 s14 s13 s12
0 0 0 0

Figure 11.12: 4x4 row serial multiplication

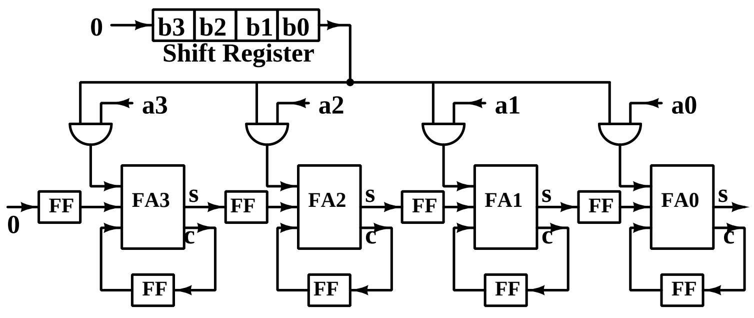


Figure 11.13: 4x4 row serial multiplier

Chapter 12

Semiconductor Memories

12.1 Introduction

Most electronic systems require storage and retrieval of data. Systems requiring a small amount of storage can use flipflops for this purpose. However, Flipflops are not efficient for large storage requirements. For large storage, we need specialised arrangements of storage elements called memories.

12.2 Memory Types

Depending on their characteristics, memories are classified as

- Static Random Access Memories (SRAM),
- Dynamic Random Access Memories (DRAM),
- Read Only Memories (ROMs)
- Electrically Programmable Memories (EPROMs)
- Electrically Programmable and Erasable Memories (EEPROMs)
- Flash Memories

Practically all memory systems these days use semiconductor memories. (Magnetic "cores" were used earlier for storage. Some memory terminology is acquired from those times).

Data stored in a memory can be accessed sequentially or in a random order. For Random Access, a new address is provided to the memory for each access. The memory then returns the stored content at that address or writes new data at this address.

Conveying the address to the memory takes time, which may slow down memory access and affect system performance adversely. Many modern memory systems use a "burst mode" in which a start address is supplied, and then data is read or written from sequential addresses in a burst. The address is automatically adjusted in the memory by incrementing the initially supplied address with each read or write in a burst.

While the storage mechanism varies from one type of memory to another, the internal organisation and the method for accessing the contents are very similar for all types of memories.

At each unique address, the memory can store a single bit or multiple bits with a width of various sizes. Correspondingly, the memory is said to be bit, byte or word oriented.

The stored data may last only as long as power is applied to the memory. Such storage is called volatile. Some memories can retain their information even when power is removed. Such memories are called non-volatile.

12.3 Memory organisation

A memory must include circuits to decode the supplied address in order to activate a specific storage location associated with this address for read or write operation. A read or write operation must start only after address decoding is complete. Otherwise, a read or write may occur from a storage cell corresponding to a transient value of the changing address. This can result in data corruption.

Figure 12.1 shows the sub-units which are used in a typical memory. Details of their operation will vary depending on the type of memory, but the overall organisation is quite similar.

- Row/Column Address decoders provide the logic used for decoding the address in order to activate the selected memory location.
- Memory array is a two dimensional array in which the storage elements are arranged. The address decoder typically selects a row and a column to identify the memory cell being addressed.
- A sense amplifier is used in every column to amplify the small signal provided by the addressed cell and to amplify it to a rail-to-rail logic level

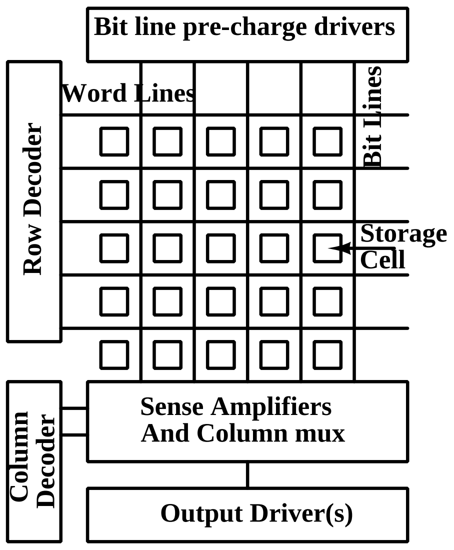


Figure 12.1: Parts of a memory

- The long bit lines in a memory would present a very heavy load for the tiny storage cells to drive. Therefore these are pre-charged to a voltage close to V_{DD}.
- A node of the addressed storage cell, when connected to these lines will not change its voltage level if the node is also at '1'.
- However, if the connected node is at '0', it will lower the voltage of the bit line by a small amount.
- The sense amplifier detects the presence or absence of this small voltage dip and amplifies it to a full scale '0' or '1'.
- For reliable operation, we usually run differential lines (bit and $\overline{\text{bit}}$) and connect these to Q and $\overline{\text{Q}}$ outputs of the storage cell.
- Output drivers provide sufficient drive to take the off-chip lines quickly to a '0' or a '1'.

12.4 Memory Timing Parameters

The speed of a memory subsystem is characterized by several timing parameters.

Read Access time: This is the delay between presentation of an address to a memory and the availability of data from that location.

Row and Column Address strobes Often the address is specified in two halves – known as Row Address and Column Address. Each half of the address is latched into the memory using strobe signals. These strobes are known as Row Address Strobe (RAS) and Column Address Strobe (CAS) signals.

RAS/CAS times The Read Access time is broken into two halves. RAS to CAS delay (T_{RCD}) is the time gap required between the Row and column address strobes. CAS latency (CL) is the delay between Column strobe and availability of data.

Write recovery time: Memory systems pre-charge long internal lines connected to a memory array and then sense the contents of a selected location by connecting it to the pre-charged lines. Write recovery time is the interval needed between a write operation and the pre-charge operation. (Write has to be complete before the next pre-charge since a pre-charge operation will forcibly take all bit lines to '1')'.

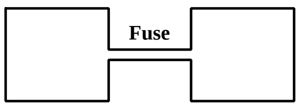
Memory cycle time: This is the time required to complete successive read or write operations. This determines the rate at which one can issue new read or write requests to a memory. The read and write cycle times need not be the same, but most systems specify a single cycle time, which is the longer of the two.

12.5 Types of Storage Cells

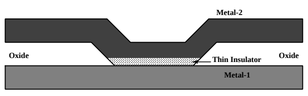
12.5.1 Read Only Memories

Read only memories are those which are programmed just once with fixed data and which will be read many times but never altered. A permanently ON or OFF switch is used as a memory element. The switch may be implemented using MOS transistors. The transistor is turned ON or OFF permanently by connecting its gate to V_{DD} or ground at the time of chip fabrication using a mask. Such memories are called Mask Programmable ROMs.

ROMs may also be made using poly-silicon fuses or anti-fuse structures as switches. A fuse is blown or an anti-fuse shorted to program data into the One Time Programmable (OTP) ROM or PROMs. Fuse and anti-fuse type structures can be used to connect or disconnect devices to implement ROM cells. Figure 12.2 shows the fuse and anti-fuse structures. The fuse structure represents an element which is normally connected, but which can be disconnected by programming. The opposite of this is true for an anti-fuse structure which is normally disconnected – but a connection can be established by programming.



- A fuse structure uses a narrow neck in an interconnect, which can be blown by passing a heavy current through it.
- A memory cell can be programmed to '0' or '1' by connecting or disconnecting a node to ground directly or through a transistor.



- An anti-fuse structure uses a structure with a thin insulator between two connecting wires.
- The insulator can be shorted by applying a high voltage across it.
- This provides the means of connecting or not connecting a node to ground programmably.

Figure 12.2: Programmable fuse and anti-fuse structures

12.5.2 Electrically programmable Read Only Memories

A variant of ROMs are Electrically Programmable ROMs or EPROMs. The data in these memories may be altered by using unusual operating conditions. The time taken to alter the contents of an EPROM is long, so the programming is carried out infrequently.

We can implement permanently ON or OFF transistors by injecting charge into the gate region. Charge can be injected into the gate region by generating energetic carriers by avalanching and applying a field of appropriate sign between the gate and substrate. The injected charge remains trapped in the insulating gate material, which changes the turn on voltage V_T. Thus an nMOS can be turned ON at zero gate bias by injecting holes into the gate region and making its V_T negative.

Such memories are erased by exposure to Ultra Violet light, which releases the trapped charge from the gate oxide back into silicon.

12.5.3 Electrically Eraseable and Programmable ROMs

Erasure of EPROMs is through exposure to UV light, which requires a specially packaged device which will let in UV light during erasure through a quartz window. This can be avoided using electrically erasable programmable ROMs or EEPROMs. EEPROMs often use a floating gate device for storage as shown in Figure 12.3. Floating gate MOS transistors are programmed in a manner similar to EPROMs, but which permit erasure through application of a high field opposite to that used for injecting charge into the gate.
A floating gate MOS device, which uses an extra floating gate embedded in the oxide to store

injected charge, can be used for this. The process of high field injection and removal of oxide

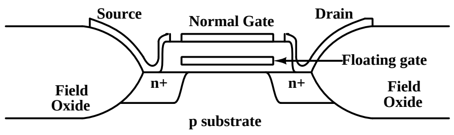


Figure 12.3: Floating gate MOS transistor for data storage

charge results in some damage to the MOS device. So the process of programming and erasing can be carried out for a limited number of times, after which the repeated charge injection and removal damages the device to such an extent that it cannot be programmed or erased any more. The number of times that a device can be programmed and erased is called the endurance of the EEPROM.

EEPROM configurations: NOR EEPROM

To address and read/write EEPROMs, programmable transistors can be connected in different ways. One such configuration is similar to NOR gates as shown in Figure 12.4. In practically

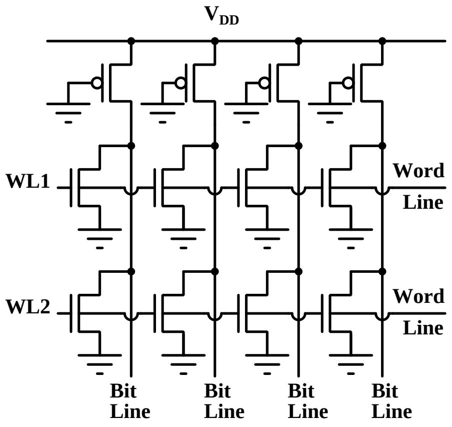


Figure 12.4: Programmable transistors configured as a NOR array

all memory devices, there are perpendicular interconnect lines called Word lines and bit lines. In the NOR configuration of EEPROMs, bit lines are pulled up using permanently ON pMOS devices, as was done in pseudo nMOS logic. A programmable nMOS transistor is connected to ground at each crossing of a word line and a bit line.

If the nMOS is ON, the bit line is pulled down to zero. If it is OFF, the bit line remains high due to the pull up transistor.

EEPROM configurations: NAND EEPROM

In a NAND EEPROM also, we have a pseudo nMOS like arrangement of transistors, with pMOS pull ups for each bit line as shown in Figure 12.5. In the NAND array, nMOS transis-

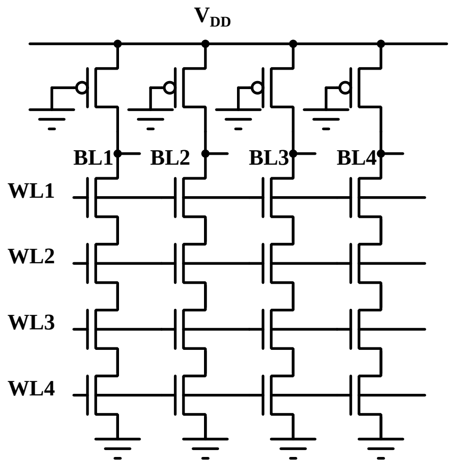


Figure 12.5: Programmable transistors configured as a NAND array

tors are all connected in series in a particular column. All word lines except the selected one are kept at '1', while the selected world line is kept at a voltage between the low and high threshold voltages of floating gate transistors.

If the selected transistor in ON at the applied gate voltage, it pulls the bit line output lower, otherwise it remains high. Unlike pseudo nMOS logic, we need not pull the output all the way down to logic '0'. This enables us to put a lot of nMOS transistors in series.

The selected column (bit line) is multiplexed to a sense amplifier, which brings the output to a proper level for logic '0' or '1'.

- NAND EEPROMs are much denser, because these don't need a contact window for every nMOS transistor.
- NOR is faster, because discharge is through a single transistor.
- Practically all thumb drives etc. use a NAND configuration these days.

12.5.4 Read/Write memories

Read/write memories provide comparable times for read and write operations. These are suitable for applications where memory has to be modified frequently – such as data storage for a micro-processor based system.

Static Read-Write memory (SRAM) uses a simple latch to store the data. The latch is implemented as two cross connected inverters and is accessed through pass transistors. Data in static Read Write memory will be retained as long as power is applied. This kind of storage is called volatile storage.

Dynamic read-write memory (DRAM) stores data on a small sized capacitor. It consists of the capacitor and a single coupling transistor. DRAM is much denser, but needs frequent refreshing of the data, since the charge stored on the capacitor can leak away.

SRAM cell

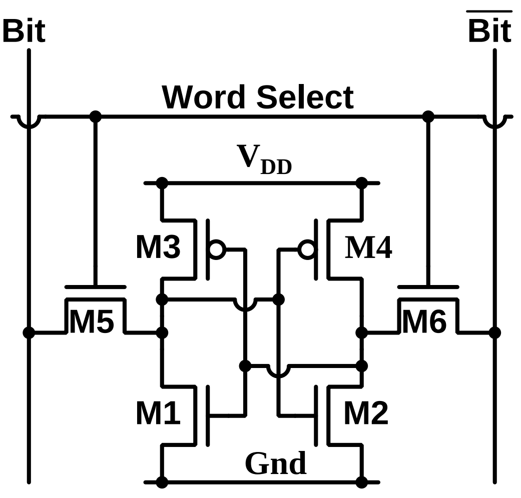


Figure 12.6: The 6 transistor SRAM cell

The 6 Transistor SRAM cell contains cross connected inverters coupled to the bit and $\overline{\text{bit}}$ lines through pass transistors as shown in Figure 12.6.
The pass transistors are enabled by the word select line.

We can analyse the behaviour of this circuit by removing the feedback and plotting the transfer characteristic of the two inverters combined, as shown in Figure 12.7.

- The solid line curve is the transfer characteristic of the first inverter (V2 vs V1).
- The dashed line curve is the transfer characteristic of the second inverter, with its input voltage V2 along the Y axis and its output voltage V3 along the X axis.

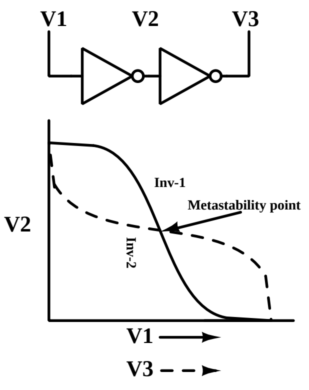


Figure 12.7: Butterfly diagram for the latch without feedback

- For any input voltage V1 (along the X axis), we read off the output of the first inverter V2 from the solid curve on the Y axis. Now, using this voltage as the input of the second inverter (on the Y axis), we read off the output of the second inverter using the dashed curve (along the X axis again).
- This curve is known as a Butterfly diagram due to its shape.
- In the actual latch, V3 is shorted to V1. So the static characteristics of the latch will be satisfied if we start with a value of V1 on the X axis and end up with the same value of V3 on the same axis (V1=V3).
- This condition will be met by the points where the solid line and the dashed line curves cross.
- The solid and dashed line curves cross at 3 points:
  - At the left most point, $V1 = 0$, so $V2 = V_{DD}$ by the solid line curve. For $V2 = V_{DD}$, $V3 = 0$. by dashed line curve. So $V1 = V3 = 0$.
  - At the right most point, $V1 = V_{DD}$, so $V2 = 0$ (solid line curve). For $V2 = 0$, $V3 = V_{DD}$ (dashed line curve). So $V1 = V3 = V_{DD}$.
  - $V1 = V3$ at the middle point as well. However, this is an unstable point. The least bit of perturbation from this point takes the circuit to one of the two stable points described above.

Let us see the effect of perturbation due to noise at the different points where circuit conditions are satisfied = *i.e.* $V1 = V3$.

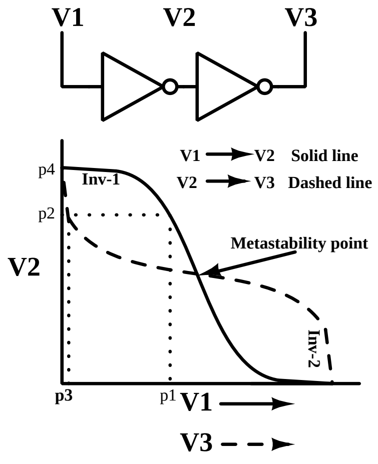


Figure 12.8: Memory latch perturbed from left stable point

Suppose the latch is stable at the left crossing point and we perturb the voltage at V1 as shown in Figure 12.8. If we perturb V1 to the point p1, the output of Inv1 will be p2 (from the solid line transfer curve for Inv1). Therefore the output of Inv2 (with its input at p2) will be p3 (from the dashed line transfer curve for Inv2).

But the output of Inv2 is shorted to input of Inv1 in the latch. So, with its input at p3, the output of Inv1 will move to p4 (solid line transfer curve for Inv1) and it will quickly revert to the stable point on the left.
In fact, perturbing V1 to any point to the left of the middle crossing will quickly recover to the left crossing point with $V1 = 0 = V3$ and $V2 = V_{DD}$.

What happens if the latch is stable at the right crossing point and we perturb the voltage from this point? This situation is shown in Figure 12.8. If we perturb V1 to the point p1, the output of Inv1 will be p2 (from solid line transfer curve for Inv1). With its input at p2, the output of Inv2 will be p3 (dashed line transfer curve for Inv2). But the output of Inv2 is shorted to input of Inv1 in the latch. With its input now at p3, the output of Inv1 will move to p4 (solid line transfer curve for Inv1). Thus the circuit will quickly recover to the right most stable point. In fact, perturbing V1 to any point to the right of the middle crossing will quickly recover to the right most crossing point where $V1 = V_{DD} = V3$ and $V2 = 0$.

Finally, what about perturbing the circuit from the middle crossing point?

Network equations are satisfied here – since the output voltage of Inv1, when applied to Inv2 gives back the same voltage.

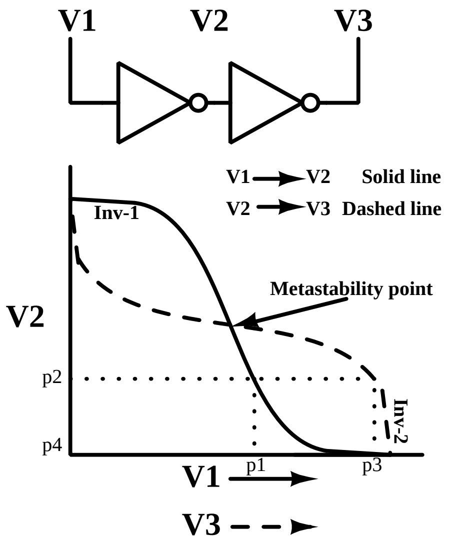
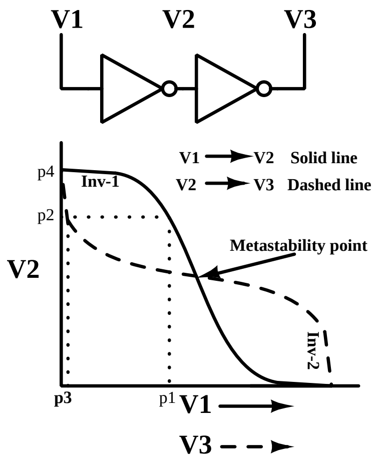
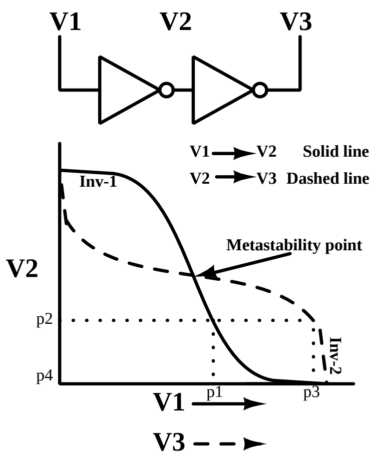


Figure 12.9: Memory latch perturbed from right stable point





However, perturbing even by a small amount to left will make the system go to the left crossing point and even by a small amount to the right will take the system to the right crossing point. So this point is not stable and is called the meta-stable point.

SRAM cell: Read cycle

In order to read the memory, the address is placed by the processor on the address lines. The row and column addresses are decoded by the memory. On receiving the $\overline{\text{Rd}}$ command, the memory generates a pre-charge pulse ($\overline{\text{PC}}$) to pre-charge the Bit and $\overline{\text{Bit}}$ lines to 'high' (see Figure 12.10). The row decoder makes the Word Select line of the addressed row 'high'. The

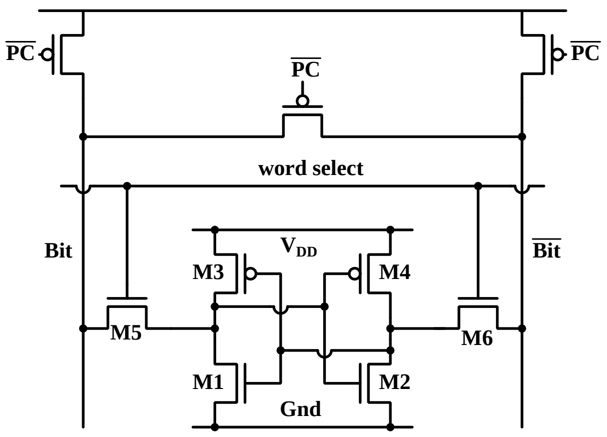


Figure 12.10: Pre-charging the bit lines

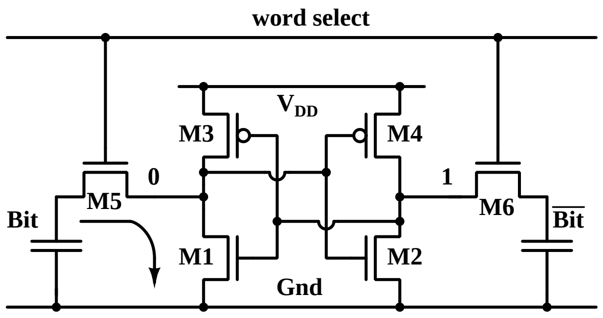


Figure 12.11: Reading the memory cell content

column address decoder connects the Bit and $\overline{\text{Bit}}$ lines of the selected column to the sense amplifier. After the $\overline{\text{PC}}$ pulse has ended, the Bit and $\overline{\text{Bit}}$ lines can be considered as charged capacitors, since the pre-charge drivers have turned off as shown in Figure 12.11. Depending on the data stored in the cell, one of the outputs of the latch will be 'high' while the other will be 'low'.

The high node is unchanged, but the low node starts discharging the Bit or the $\overline{\text{Bit}}$ line (whichever is connected to it) This creates a voltage difference between the Bit and $\overline{\text{Bit}}$ lines. The sense amplifier amplifies this voltage difference and produces a digital '0' or '1' at its output. This digital data is then buffered to the output pad by a tristateable driver, which is activated only during read cycles.

SRAM cell: Read upset

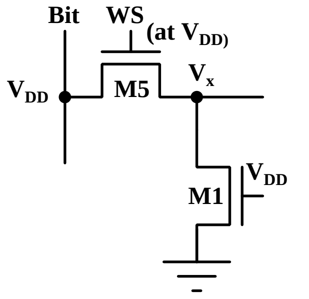
During the read operation, output nodes of the latch are connected to the Bit and $\overline{\text{Bit}}$ lines, which are like large capacitors charged to V_{DD}. This reduces the voltage on the Bit or the $\overline{\text{Bit}}$ line which is connected to the low output of the latch. However, this also raises the voltage at the low node of the latch. At the end of the read cycle, the word select line returns to 'low', disconnecting the latch from Bit and $\overline{\text{Bit}}$ lines.

Now the low node, which would have been pulled a little high due to connection with the pre-charged Bit or the $\overline{\text{Bit}}$ line is restored back to 0V by the feedback action as discussed earlier.

The geometry of the access transistors has to be carefully determined to ensure that the low node is not pulled to too high a voltage when it is connected to the pre-charged Bit or $\overline{\text{Bit}}$ line. If the voltage of the low node goes too high when it is connected to the pre-charged Bit or $\overline{\text{Bit}}$ line, it may zap to a '1' value rather than recovering back to '0'. This is called a read upset.

Consider the memory cell with a '0' stored in it. In this state, M3 and M2 are OFF (see Figure 12.11. We now find the condition that the voltage at the drain of M1 should remain below V_{Tn}. This will ensure that M2 will remain OFF during the entire read cycle.

Since M2 remains OFF, The voltage at the gate of M1 will remain at V_{DD} for the entire cycle. Conduction thus takes place only through M5 and M1.



Since the drain and gate voltages of M5 are equal (at V_{DD}), it is in saturation.

The gate of M1 is at V_{DD} while its drain is below V_{Tn}, so M1 remains in linear regime.

Equating currents through M5 and M1:

$$\frac{K_{n5}}{2}\,(V_{DD} - V_x - V_{Tn})^2 = K_{n1}\left((V_{DD} - V_{Tn})V_x - \frac{1}{2}V_x^2\right)$$

We define $\beta \equiv K_{n1}/K_{n5}$. This gives:

$$(V_{DD} - V_x - V_{Tn})^2 = 2\beta V_x(V_{DD} - V_{Tn} - \frac{1}{2}V_x)$$

In the limiting case, $V_x = V_{Tn}$. Then:

$$(V_{DD} - 2V_{Tn})^2 = 2\beta V_{Tn}(V_{DD} - \frac{3}{2}V_{Tn}) = \beta V_{Tn}(2V_{DD} - 3V_{Tn})$$

$$\text{Hence,} \quad \beta = \frac{(V_{DD} - 2V_{Tn})^2}{V_{Tn}(2V_{DD} - 3V_{Tn})}$$

$$\beta = \frac{(V_{DD} - 2V_{Tn})^2}{V_{Tn}(2V_{DD} - 3V_{Tn})}$$

If we take $V_{Tn} = V_{DD}/5$, we get

$$\beta = \frac{(5V_{Tn} - 2V_{Tn})^2}{V_{Tn}(10V_{Tn} - 3V_{Tn})} = \frac{9V_{Tn}^2}{7}V_{Tn}^2 = 9/7 = 1.2857$$

- Thus the value of β should be ≥ 1.3 to keep V_x below V_{Tn}.
- Notice that the geometry ratio of M5 and M1, as well as the maximum voltage rise at V_x is fixed by this consideration.

This must be kept in mind while discussing the write operation.

SRAM cell: Write cycle

During the write cycle, the address is placed by the processor on the address lines and the row and column address are decoded by the memory.

When the word line of the selected row goes HIGH, the access transistors are turned on for **all** cells in this row. When $\overline{\text{Wr}}$ is asserted, the Bit and $\overline{\text{Bit}}$ lines are driven to Data and $\overline{\text{Data}}$ respectively **in the selected column only**. Please see Figure 12.12 Thus, in the selected

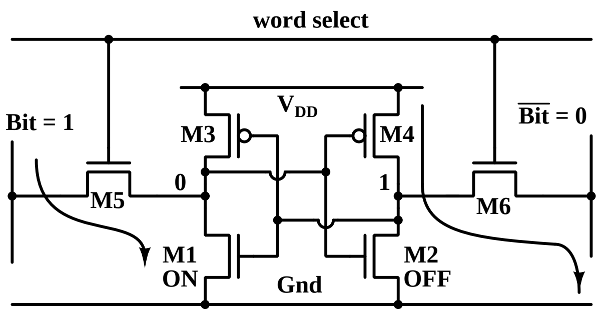


Figure 12.12: Static RAM cell: write operation

column of the selected row, the cell goes through a write cycle.

For all the remaining columns in the selected row, cells go through a read cycle.

In the selected column, the complementary values of Data and $\overline{\text{Data}}$ are copied to the latch through the pass transistors. Once the voltage at the node connected to the pass transistors passes the meta-stability point, the positive feed back ensures that it will go all the way to '0' or '1'. After the write process is complete, the Word line is brought low again.

Considerations of Read upset fix the ratio between the pass transistor and pull down transistor width. The voltage rise at the node which is at '0' is limited to $\approx V_{Tn}$, when the bit line connected to it is at '1'. Then how do we write a '1' to the cell – which requires the node to be taken above $\approx V_{DD}/2$?

The major work for writing a '1' has to be done at the 'high' node which must be pulled down to well below $V_{DD}/2$. his consideration fixes the geometry ratio of M4 and M6.

Notice that M1, M3 and M5 should have the same size as M2, M4 and M6 respectively.

SRAM cell: Full Data path

The data path during read and write cycles is shown in Figure 12.13

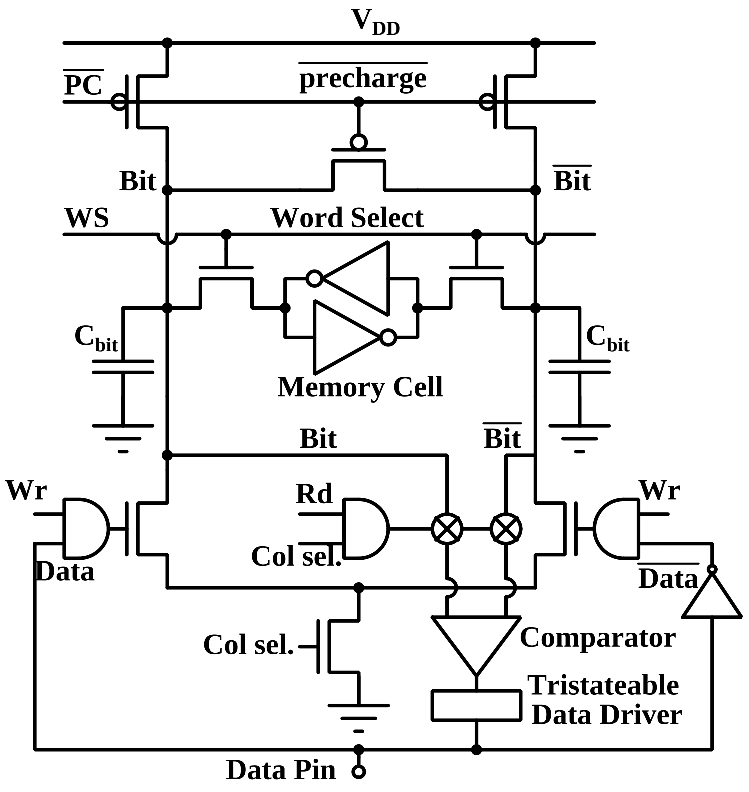


Figure 12.13: Full data path in the RAM

- All 3 pMOS transistors on the top turn on during pre-charge. The weak shorting transistor ensures that the voltages on Bit and $\overline{\text{Bit}}$ are equal.
- The pass gates connected to Bit and $\overline{\text{Bit}}$ lines are activated only if this is a read cycle and this column has been selected.
- When activated, these feed the sense amplifier/comparator, which then drives a tri-stateable buffer to the data pin.

SRAM cell: Alternate Bit line pull up

We can use nMOS transistors instead of pMOS to pre-charge the Bit and $\overline{\text{Bit}}$ lines as shown in Figure 12.14. The pre-charge pulse should now be a positive pulse.

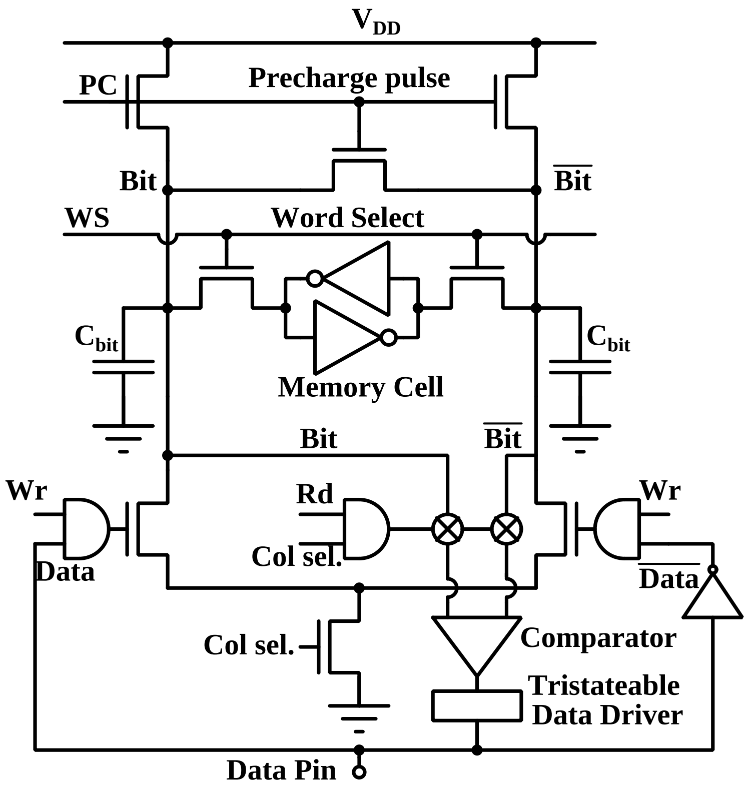


Figure 12.14: Alternative pre-charge in the RAM

- Mobility of n channel transistors is higher and the source follower configuration has lower output impedance. So charging is much faster.

- However, the maximum voltage to which the Bit and $\overline{\text{Bit}}$ lines can be charged is now $V_{DD} - V_{Tn}$.
- Transistor geometries have to be adjusted due to lower voltage on the Bit and $\overline{\text{Bit}}$ lines.
- This lower voltage is actually an advantage for the design of sense amplifier, whose common mode range does not have to go all the way up to V_{DD}.

12.6 DRAM cell

Dynamic RAM uses a single transistor per cell, so it is much denser than SRAM. The storage is on a capacitor – so it needs periodic refreshing to avoid data loss due to leakage. To store sufficient charge on the capacitor, special technological steps are required. So it is not process compatible with other CMOS circuits. A DRAM cell is shown in Figure **??**. In the read cycle,

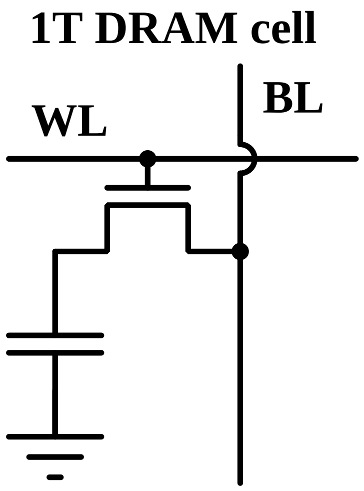


Figure 12.15: 1 Transistor DRAM cell

the bit line is pre-charged 'high'. If the capacitor stores a '0' (it is discharged), it pulls the voltage on the bit line a little lower due to charge sharing. If it stores a '1', the voltage on the bit line remains unchanged. Presence or absence of a voltage change is detected by the sense amplifier and driven to the output as in the case of an SRAM.

During a read cycle, the storage capacitor is connected to the bit line for all cells in the enabled word line. Since the storage capacitor is much smaller than the bit line capacitor, it gets charged to '1' during charge sharing. All cells which had a '0' stored in them have their data destroyed!

So the Read operation takes place on the entire row, and the data which has been read needs to be written back to the entire row. Thus a read cycle is always followed by an internal write operation for the whole row. This also refreshes the data in the entire row, restoring any leaked charges.

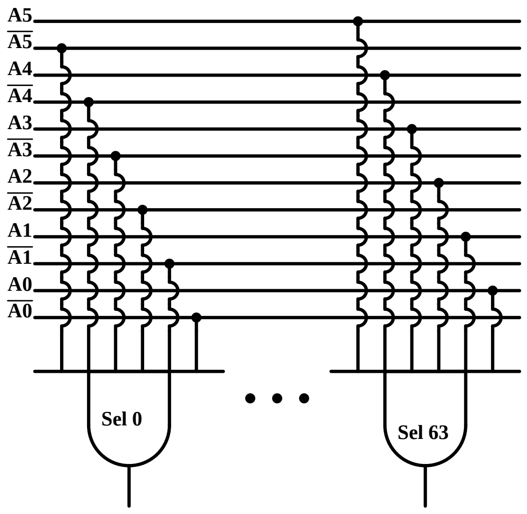
Write Cycle in a DRAM cell

During a write cycle, the bit line is driven to the data value in the selected column. (Refer to Figure **??**). When the word select signal comes, the data is copied to the storage capacitor. The columns which are not selected by the current address effectively go through a read cycle, including a refresh operation.

- The data is refreshed for the entire word line in a DRAM whenever it is accessed for a read or a write operation.
- Banks of DRAM memories need a controller, which periodically performs a dummy read on every row in order to refresh the data stored in the memory.
- Modern DRAM chips have refresh circuitry on chip, which internally do the refresh operation if some word lines are not being accessed. These memories are also called SDRAMs.

12.6.1 Address decoding in Memories

A one step decoder for 6 bit row or column address is shown below.



This kind of decoding is not practical. All address lines need to run over the entire length of the memory chip, with makes their capacitance large. A more practical approach is shown in

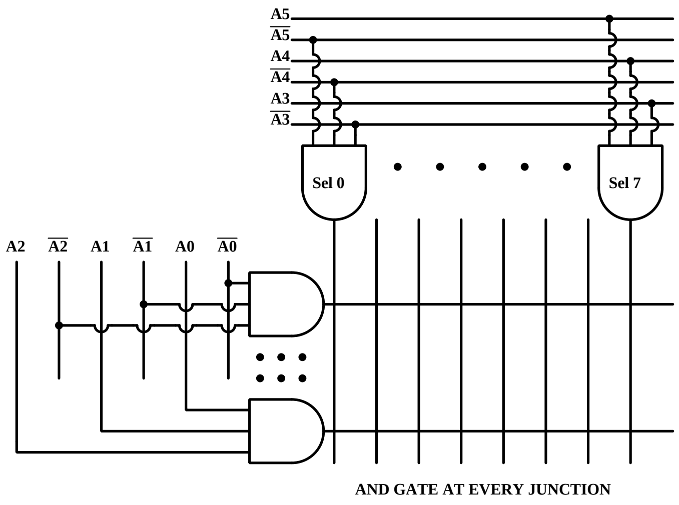


Figure 12.16: Two step address decoding in a memory

Figure 12.16. It decodes different blocks of memory separately. In this particular example, We could separately decode 3 bits each and then combine the select outputs using AND gates.

Chapter 13

IO circuits

Circuits used for I-O require special design and layout techniques. While sizes of devices and on-chip interconnects have been aggressively scaled over the years, sized of wires used for connections to the external world have remained practically unscaled. A wire attached to the chip to connect it to the external circuits cannot be made much thinner due to considerations of mechanical strength and the soldering techniques used to attach it.

As a result, connections to the outside circuits have to be through I-O pads which are large and not scaled with the technology. Wires are attached to these pads through ultrasonic or thermo-compression bonding. The core of the circuit has to be protected from the thermal and electrical stress produced by this process.

13.1 Protection at the I-O interface

ICs need to be handled by human beings and machines before being inserted into circuits. Human beings, as well as machines may carry huge electrostatic charges. Since CMOS circuitry has high input impedance and very thin gate oxides, input transistors will breakdown immediately when exposed to high voltages.

A human body is often modeled as a 100 pF capacitor charged to a voltage between 1.5KV

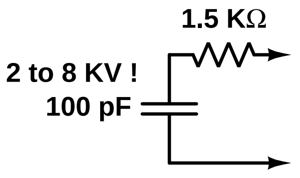


Figure 13.1: Human Body Model

to 8KV and which will discharge through a resistance of 1.5KΩ. Special protection circuits have to be placed at inputs to protect delicate MOS devices from destruction due to these high voltages. Notice that this protection should work even when the chip is not powered!

13.1.1 Input pad protection

Protection from external charges is provided through careful design of pads and other protection elements.

The resistors and diodes are actually the same device on the chip – a p+ diffused line in an

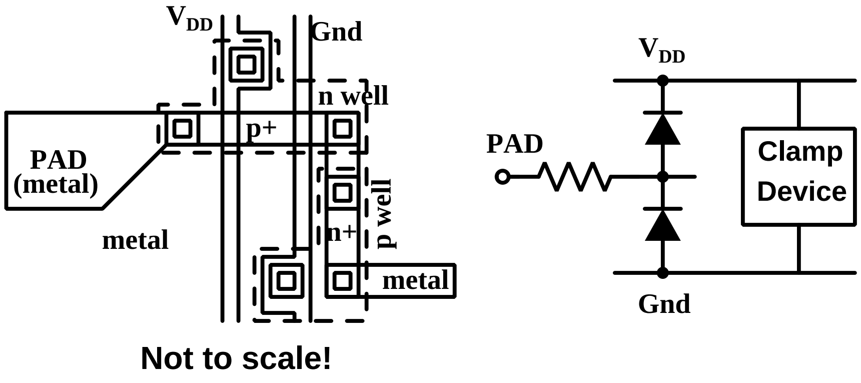


Figure 13.2: I-O Pad for electro-static protection

n-well and an n+ diffused line in a p-well. Circuits at I-O pads need to be put in separate n and p wells, with good n+ and p+ guard bands around these to prevent latch up.
The diodes ensure that any negative voltage is shorted by the diode to ground and any voltage above V_{DD} forward biases the upper diode, clamping the voltage to within about 0.6V of V_{DD}. This, however, assumes that a supply is connected to the circuit. How can the chip be protected when it is not connected to a supply?
That is done through an internal clamp device.

13.1.2 Designing the clamp device

The diodes at the input will just direct the excess voltage to the supply node or ground depending on the polarity of the high input voltage. A device is need to discharge this when the chip is not connected to a supply voltage. Such devices are called “Clamp” devices.

The clamp device must *not* turn on as long as the voltage between the supply rails is within specifications. However, it *must* turn on if the voltage exceeds the specified supply voltage even by a relatively small amount. Also, the clamping action should be non-destructive – so that it will continue to work after it discharges the excess voltage.

This puts severe limits on the turn on voltage of the clamp device in the presence of process and temperature variations and scaled down supply voltages. Typically, SCR devices (using the latch up structure!) or thick oxide MOS devices are used for this purpose.

13.2 I-O interface considerations

The supply voltage in the core of VLSI needs to be scaled with the technology. On the other hand, system voltages outside the chip are changed much less frequently. Thus, a system may need to integrate multiple chips which work at different internal supply voltages. As a result, we may need to provide voltage translation from external signals to internal signals and *vice versa.*

13.2.1 I-O transistors

Since higher than usual voltages may exist at the I-O terminals, special transistors with thicker gate oxides and lightly doped extensions of he drain to stand higher than usual voltages at their gates and drains are provided in most CMOS processes these days

These transistors are called I-O transistors or H-V transistors. Because of higher oxide

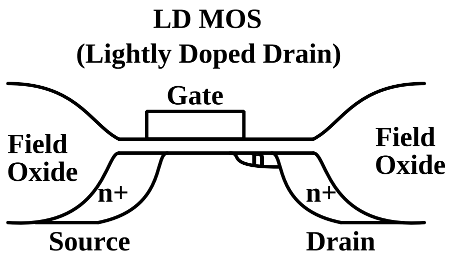


Figure 13.3: I-O or HV transistors

thickness, their transconductance and speed is lower. So these are not often used in the core of the VLSI.

13.2.2 Voltage Level Translation at input or output

Logic levels outside the chip may not be the same as the ones used inside it. This necessitates voltage transformation at the I-O interface for proper functioning of the system.

To take an example, a chip might use an internal supply voltage of 1.8V. However, the system in which it is placed may work with a supply voltage of 3.3V – with correspondingly high logic swings. Obviously, the low swing logic voltages inside the chip need to be amplified to high swing values for proper interpretation by the system. Assume that the ‘low’ level is around 0V for both systems. An internal inverter with a supply of 3.3V will be a poor choice for voltage translation, because the logic ‘high’ level of 1.8V will not be able to turn off the pMOS transistor of the inverter working with a supply voltage of 3.3V. This will result in high static power consumption.

The circuit shown in figure 13.4 can be used for translating low swing logic levels to high swing logic. The circuit shows a low swing to high swing translator using CVSL. The inverter

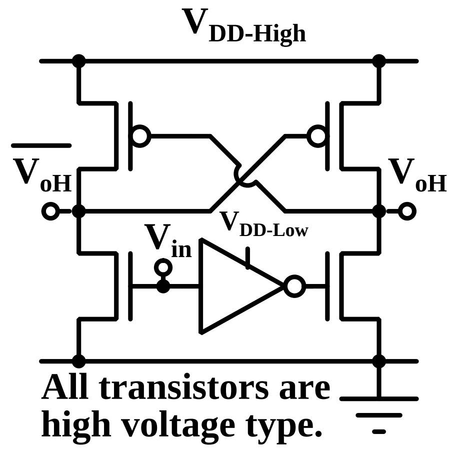


Figure 13.4: CVSL circuit for translating low swing to high swing

uses low supply voltage compatible with low swing logic. Transistors are high voltage devices.

High swing to Low swing conversion is much easier and can be managed with a simple inverter using thick oxide devices.

13.3 Input buffers with hysteresis

Signals coming in from outside may have slower rise and fall times with noise imposed on them. This results in the input hovering around the trip point of inverters at the input for considerable times. This is shown in Figure 13.5.

Any noise superimposed on the input as it moves slowly through the trip point of the input inverter will take the input voltage above and below the trip point, resulting in the inverter output switching to a ‘1’ or a ‘0’ depending on the polarity of the noise.

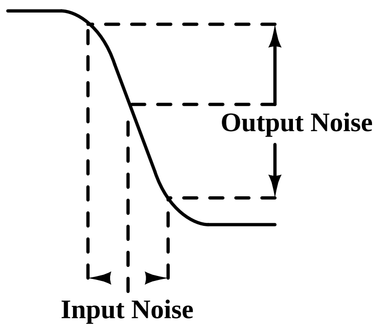


Figure 13.5: Noise amplification by an inverter

Even if the output does not swing rail to rail, the amplification factor at the trip point is very high and the noise will be amplified at the output of the inverter – which may exceed the designed noise margin of the main circuit.

Input buffers use hysteresis to avoid false transitions from '0' to '1' and back due to noise while the input transitions through the trip voltage.
The circuit in Figure 13.6 provides hysteresis to produce clean transitions at the output.
The circuit is an inverter formed by MP1 and MN1, whose supply and ground voltages are

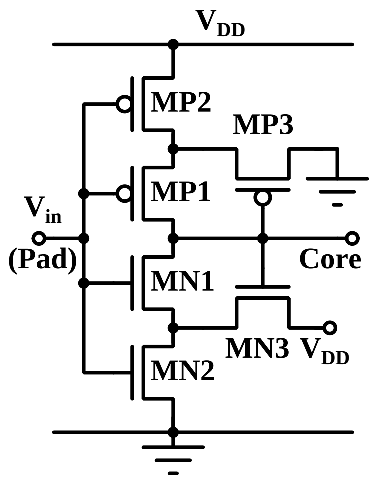


Figure 13.6: Inverter with hysteresis

set by voltage division between MP2/MP3 and MN2/MN3 respectively. These voltages are manipulated by the circuit so that the trip point is high when the input voltage rises from 0 to V_{DD} and low when it drops from V_{DD} to 0.

Consider that the input is initially low, so the inverter output is high. Then MP3 is OFF while MN3 is ON. Because of this, the source of MN1 is above ground, which makes the trip point of the inverter high.

Now consider a slow rise on the input from 0 to V_{DD} with noise riding on it. When the input voltage (+ noise) exceeds the high trip point, the output drops low, turning MP3 ON and turning MN3 OFF. The source of MN1 now drops to a low voltage. simultaneously, MP3 pulls the source of MP1 lower. Thus there is a voltage translation downwards for the supply and "ground" voltage of the inverter formed by MN1 and MP1, which pulls its trip point low.

Because the trip point of the inverter has been made much lower now, noise on the input will not be able to take the input voltage lower than this changed value, so the output will not switch again due to noise.

A similar argument applies when the input voltage falls slowly from high to low.Initially the output is low, so the source of MP1 is much lower than V_{DD}, which makes the trip point of the inverter low. Once the input crosses below this low trip point, the output goes high, turning MN3 ON and turning MP3 OFF. this raises the source voltages of MP1 and MN1, making the trip point high. This higher trip point is out of reach of the noise voltage, so there is no multiple switching due to the noise and the output remains high.

13.4 Output Drivers

Output drivers of the chip are required to drive external loads which are much higher than internal loads. Output drivers therefore use tapered inverters with geometries calculated through logical effort techniques to optimise the delay through the buffers.
Protection mechanisms are required at outputs as well as at inputs. However, drains of large transistors in the output buffer provide diode connections to substrate (connected to ground) and to n-well (connected to V_{DD}) which are similar to the protection diodes described at the beginning of this chapter.

13.4.1 Bi-directional ports

We often need I-O paths which can be used for input as well as output. One possible way of implementing these would be to use tri-stateable drivers for outputs. These drivers can then be disabled during input and enabled when we want to output data. However, this arrangement is not a satisfactory solution. This is because tri-stateable inverters use series connected n and p transistors, which doubles their size. Since these are very large devices, doubling of size has a high area and dynamic power dissipation cost.

Figure 13.7 shows a tri-stateable inverter being used as the output buffer.

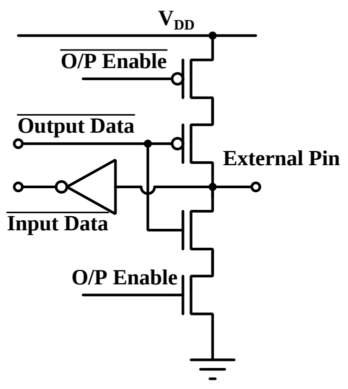

V_{DD}
$\overline{\text{O/P Enable}}$
$\overline{\text{Output Data}}$
External Pin
$\overline{\text{Input Data}}$
O/P Enable

Figure 13.7: A possible bi-directional I-O port

Can we use a final driver using single p and n transistors with tri-stating function implemented through circuits which precede the final stage?
Indeed we can! – This can be done using a NAND-NOR combination.
The circuit in Figure 13.8 uses a NAND-NOR combination to drive output transistors.

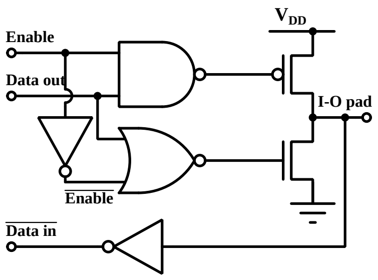


Figure 13.8: Bi-directional I-O using NAND-NOR combination

When Enable is ‘0’, output of the NAND gate (with Enable as one of its inputs) is unconditionally ‘1’. This turns off the p channel transistor in the output stage.
The output of the NOR gate (with $\overline{\text{Enable}}$ = ‘1′ as one of its inputs) is unconditionally ’0’. This turns off the n channel transistor in the output stage.
Thus when enable is ‘0’ both the n and p channel transistors in the output stage are off and the output stage is tri-stated, as desired.

When enable is ‘1’, both NAND and NOR act as inverters, so the p and n channel transistors in the output receive $\overline{\text{Data out}}$ as their input. Thus the output stage acts like an inverter and

drives the pin to the “Data out” value.
A compact version of the NAND-NOR logic as shown in Figure 13.9 is often used in bi-directional I-O.
When the enable input is ‘0’:

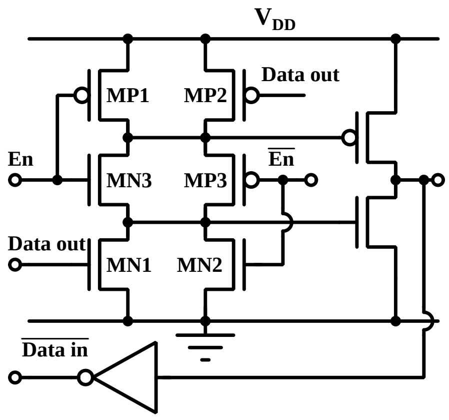


Figure 13.9: A compact driver for bidirectional I-O

- The middle n and p channel transistors (MN3 and MP3) in the logic circuit driving the output stage are OFF.
- The p channel transistor on the left top (MP1) is ON, which makes the upper output ’1’, driving the pMOS in the output stage OFF.
- The nMOS transistor in the lower right (MN2) is ON, pulling the lower output to ‘0’, which turns the nMOS in the output stage OFF.

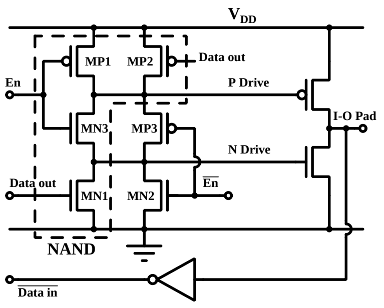
Thus the output stage is tri-stated when Enable is ‘0’. When the enable input is ‘1’:

- The pMOS transistor on the left top (MP1) and the nMOS transistor at the right bottom (MN2) are OFF.
- The middle nMOS and pMOS (MN3 and MP3) are ON, shorting the drains of MP2 and MN1.
- This reduces the logic circuit to an inverter formed by MP2 and MN1, with “Data out” as its input.
- Its output is therefore $\overline{\text{Data out}}$, which drives the gates of both the transistors in the output stage. Thus the output stage also acts like an inverter.

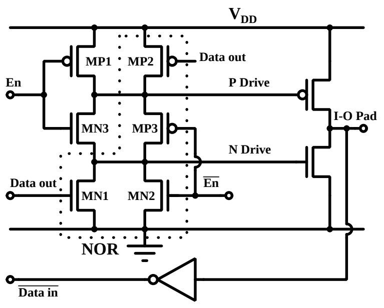
So the value of Data out is driven to the output pin when enable = ‘1’.

In actual circuits, non-inverting driver chains with tapered geometries (computed using logical effort) will be inserted between the two outputs of the first stage and the gates of n and p channel transistors in the final stage.

The equivalence of this driver to the NAND-NOR circuit is not surprising because it encompasses NAND and NOR logic in its circuit as shown in Figure 13.10:



MN1, MN3, MP1 and MP2 form the NAND gate.



MN1, MN2, MP3 and MP2 form the NOR gate.

Figure 13.10: NAND and NOR functions in the compact driver

The dashed and dotted regions indicate the NAND and NOR functions included in the compact driver.