Chapter

3

Low-Voltage Low-Power Adders

3.1 Introduction

Addition is an obligatory operation that is crucial to processing the fundamental arithmetic operations [1–3]. It is used extensively in many VLSI design paradigms and is by far the most frequently used operation in a general-purpose system and in application-specific processors. Also, because the operations of subtraction, multiplication, division and address calculation usually rely on the operation of addition, addition is often seen as an indispensable part of the arithmetic unit. It is dubbed the heart of any microprocessor, DSP architecture, and data processing system.

In addition, each of the resulting output bits depends on all the input bits to be of equal or lower magnitude. Addition is a very crucial operation because it usually involves a carry ripple step which must propagate a carry signal from each bit to its higher bit position. This results in a substantial circuit delay. The adder, therefore, which lies in the critical delay path, effectively determines the system's overall speed. On the other hand, the option of reducing the power consumption of the designed adder, which for many years has been a narrow specialty, has recently been gaining prominence. This can be attributed to the emergence and increasing popularity of smaller and more durable mobile computing and communication systems. Low-power dissipation allows a portable system to operate longer with the same battery.

Due to the potential versatility of adders in this contemporary research field, this chapter focuses solely on describing the existing adders and adder designs currently intended for future low-voltage and low-power environments. The chapter begins by describing the

Scanned with CamSc

standard adder cells (in Sec. 3.2) as the basic building blocks used in designing and fabricating of different kinds of adder architectures. Those different adder architecture types are described in greater detail in the subsections that follow. The half adder (HA) is included, as is the full adder (FA), along with an assortment of their schematic configurations.

Section 3.3 presents an overview of a range of representative architectures of the adder families, namely the Ripple Carry Adder (RCA), the Carry Look-Ahead Adder (CLA), the Carry Select Adder (CSL), the Carry Save Adder (CSA), the Carry Skip Adder (CSK), and the Conditional Sum Adder (COS). It concludes with a summary that appraises and compares the different classes of adders in terms of silicon area, worst-case delay, average power dissipation and power-delay product.

Section 3.4 covers a BiCMOS implementation of a 64-bit CLA mechanism that utilizes Pass Transistor BiCMOS (PT-BiCMOS) gates. The BiCMOS device offers higher driving capability than its CMOS counterpart and is therefore more appropriate for driving heavy load capacitance. Also included is a general propagation delay comparison between the BiCMOS and CMOS adders, thus revealing that the BiCMOS adder attains significant speed-up over the CMOS adder when the fan-out factor is more than five.

Section 3.5 presents an illustration pointing out the trends of the technology and of power supply voltage over the past decade. As the technology matured, the channel length was scaled drastically, from 1.7 μm (in 1989) to 0.13 μm (in 2002) and eventually reaching the nanometer regime (in 2004). On the other hand, power supply voltage has undergone a tremendous drop, from 5 V (in 1989) to 1.2 V (in 2002) and all the way down to the sub-1 V regime (in 2004). In view of the earlier-mentioned developments, this section reveals some low-voltage low-power design techniques.

Very often, high levels of sophistication do not come without the price of increased complexity and its corresponding problems. Therefore, Sec. 3.6 introduces future adders based on Multiple-Valued Logic (MVL) that reduce wiring complexity and lower the device count. Here, the current-mode logic acts as a replacement for the conventionally employed voltage-mode logic. The CMOS/BiCMOS Current-Mode Multiple-Valued Logic (CMMVL), in combination with the carefully chosen arithmetic number representations, such as the Residue Number system and the New Signed-Digit Number system, would offer great potential for high-speed addition.

3.2 Standard Adder Cells

3.2.1 Half adders

The HA is the simplest and most fundamental kind of adder [4]. It consists of two binary operands that have a pair of single-bits as inputs

Scanned with CamSc

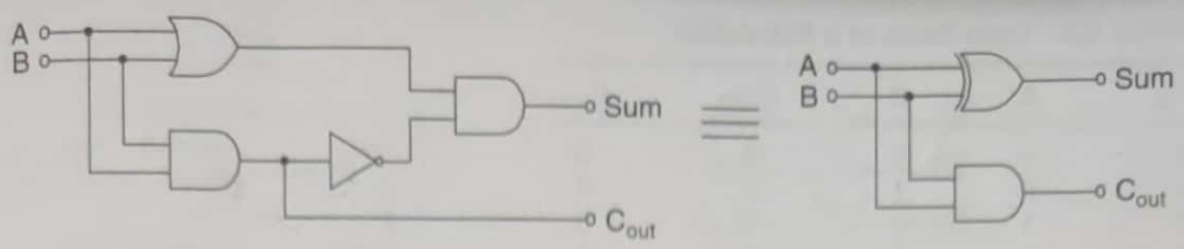


Figure 3.1 Logic gate of a half adder.

TABLE 3.1 Truth Table of a Half Adder

A	B	Sum	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

and produces a two-bit binary number as its resultant. The lower order bit of the resultant is known as Sum, whereas the higher order bit is named C_{out}. The logic gate of the HA is shown in Fig. 3.1 and the truth table representing its operation is depicted in Table 3.1 [5].

From the truth table, the logic functions for the Sum and C_{out} are generalized as

$$\text{Sum} = A \oplus B \tag{3.1}$$

$$C_{out} = A \bullet B \tag{3.2}$$

The HA can only be used to add two single-bit binary numbers since it does not permit a carry-in. To add operands that process more than one bit, the adder would have to have three inputs rather than two, which would enable it to process the carry-in. An adder with a carry-in is called a *full adder* and is described next.

3.2.2 Full adders and their various schematic configurations

A full adder adds two binary numbers with a carry-in. The structure representation of the conventional CMOS full adder appears in Fig. 3.2. It is constructed using two HAs and an OR gate. There are a total of three inputs for the FA, two for the input numbers A and B, and one

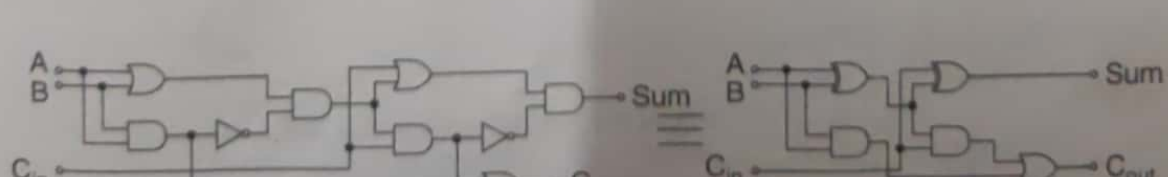


Figure 3.2 Logic circuit of the conventional CMOS full adder.

Scanned with CamSc

TABLE 3.2 Truth Table of a Full Adder

C_{in}	A	B	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

for the carry-in, C_{in}. The outputs are the Sum and carry-out C_{out}. The truth table for the FA logic circuit is shown in Table 3.2.

According to Table 3.2, the logic functions corresponding to terminals Sum and C_{out} are as follows:

$$\text{Sum} = A \oplus B \oplus C_{in} \tag{3.3}$$

$$C_{out} = (A \oplus B) \bullet C_{in} + A \bullet B \tag{3.4}$$

The logic symbol for the FA appears in Fig. 3.3 and subsequently in the following subsections.

Figure 3.4 shows the transistor level implementation of a conventional CMOS full adder cell design using a total of 32 transistors [6]. Its modified version, based on CMOS transmission gates and inverters, uses only 20 transistors [1]. Note how its schematic diagram, which appears in Fig. 3.5, is logically equivalent to the transistor level diagram depicted in Fig. 3.4.

Although the modified conventional CMOS full adder configuration has been widely accepted and utilized in numerous applications, it often exhibits a critical delay that actually limits the system's total performance. Specifically, wherever two or more of these FAs are cascaded together to perform multiple-bit addition, the system's speed takes a hit.

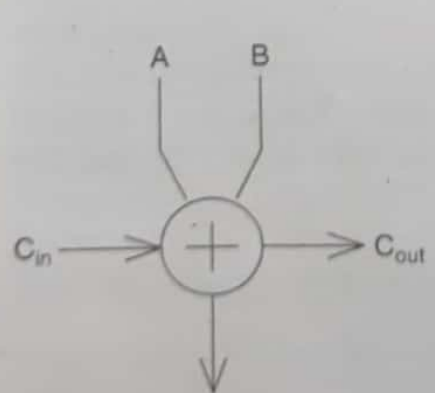


Figure 3.3 Logic symbol of a full adder.

Scanned with CamSc

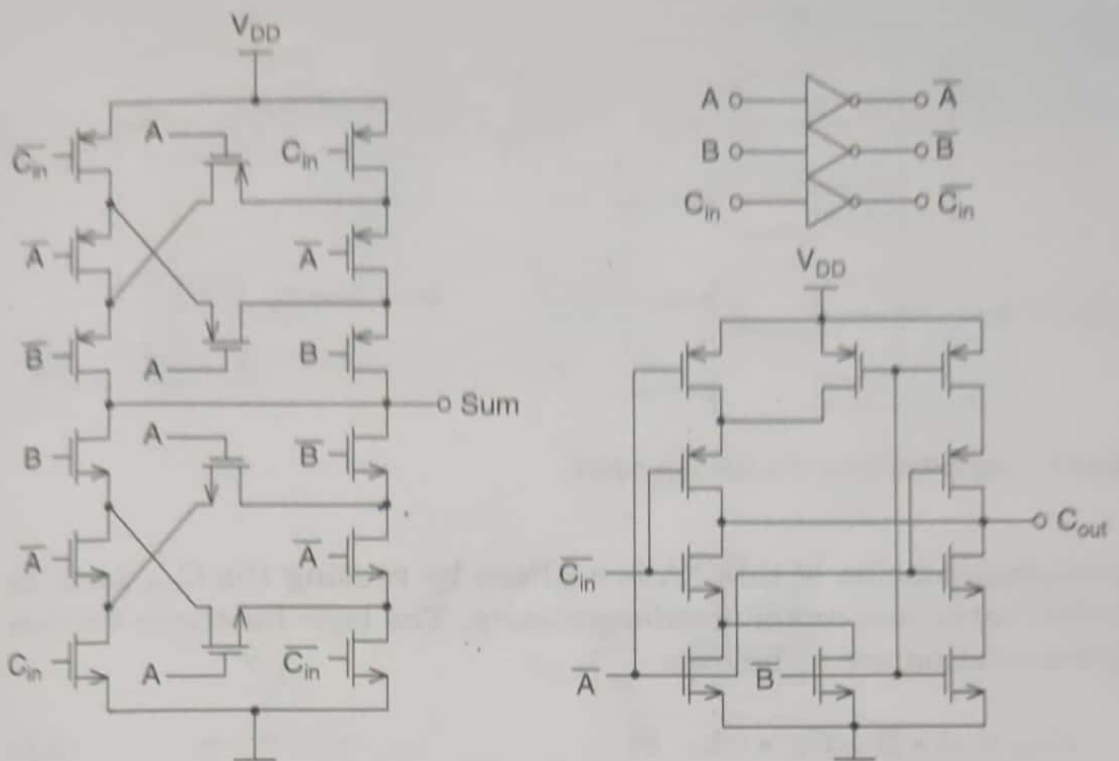


Figure 3.4 The conventional CMOS full adder.

Therefore, to ensure better speed performance, a fast FA has been designed and is illustrated in Fig. 3.6 in its logic realization form.

There is an alternative implementation of the FA cell that does not use XOR gates, but instead uses 28 transistors [6]. Figure 3.7 shows its logic circuit and its transistor diagram. As shown in Fig. 3.7(*a*),

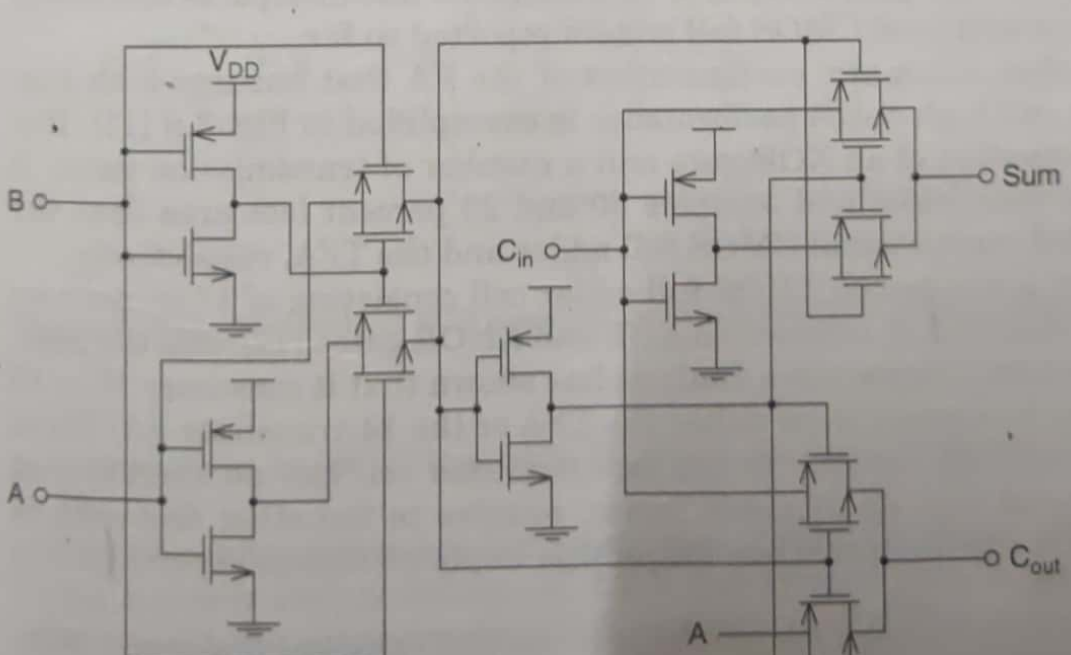


Figure 3.5 The modified conventional CMOS full adder.

Scanned with CamSc

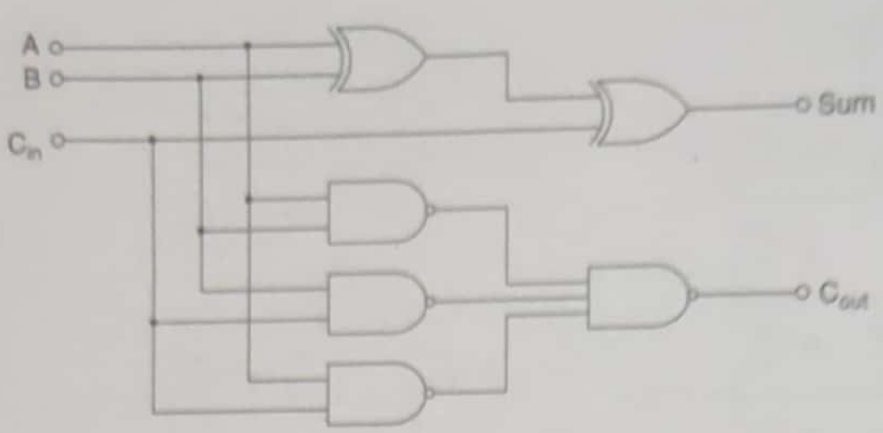


Figure 3.6 Logic structure of a fast full adder.

the implementation of this FA is realized by reusing the C_{out} term in the Sum term as a common subexpression. The logic functions for this implementation are as follows:

$$C_{out} = A \bullet B + C_{in} \bullet (A + B) \tag{3.5}$$

$$\begin{aligned} \text{Sum} &= A \bullet B \bullet C_{in} + (A + B + C_{in}) \bullet \overline{C_{out}} \\ &= A \bullet B \bullet C_{in} + (A + B + C_{in}) \bullet \overline{(A \bullet B + C_{in} \bullet (A + B))} \end{aligned} \tag{3.6}$$

In the never-ceasing process to further simplify the FA, which is the fundamental unit of the arithmetic unit, a CMOS full adder based on transmission function theory [7–9], namely the Transmission Function Adder (TFA), was developed [10]. Its schematic diagram is illustrated in Fig. 3.8. The TFA consists of 16 transistors and dissipates less power than conventional CMOS full adders reported so far.

Another schematic configuration of the FA that ensures both low-power and high-speed performance is exemplified in Fig. 3.9 [11]. It is a combination of an XOR gate and a number of transmission gates. It has 14 transistors and occupies 30 and 20 percent less area than the modified conventional CMOS full adder and the TFA, respectively.

Next, a low-power CMOS full adder cell consisting of 17 transistors [1] is described. It is based on XOR and XNOR gates [12], and the pass-transistors. Comparative analysis has shown that it consumes 10 to 15 percent less power than either the TFA or the 14-transistor FA. These power savings are due to the fact that this cell has no short circuit power and that its dynamic power, relative to the other two cells, is lower. The transistor schematic of this implementation is portrayed in Fig. 3.10.

Another promising FA prototype is the 10-Transistor low-power high-speed FA cell (10-T FA), which employs an XOR gate, an inverter and a

Scanned with CamSc

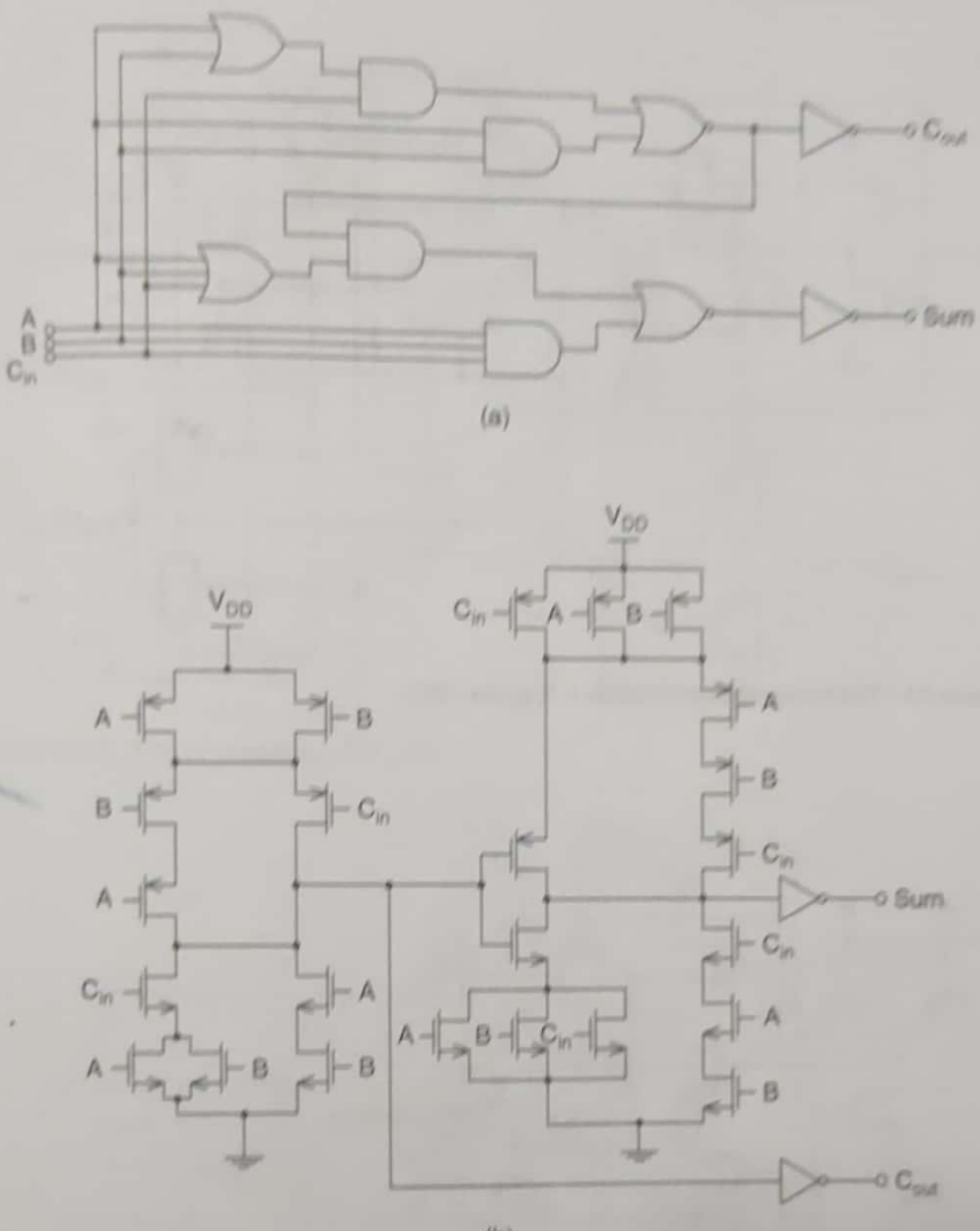


Figure 3.7 Full adder without XOR gates: (*a*) logic diagram; (*b*) transistor diagram.

pass transistor in its critical path [3]. The schematic diagram appears in Fig. 3.11.

This FA configuration has been compared to the TFA reported earlier. Using a power supply voltage of 3.3 V, the critical path delay of the 10-transistor FA measures at 0.086 ns while in the TFA it measures at 0.12 ns. Also, with the same supply voltage and running a clock

Scanned with CamSc

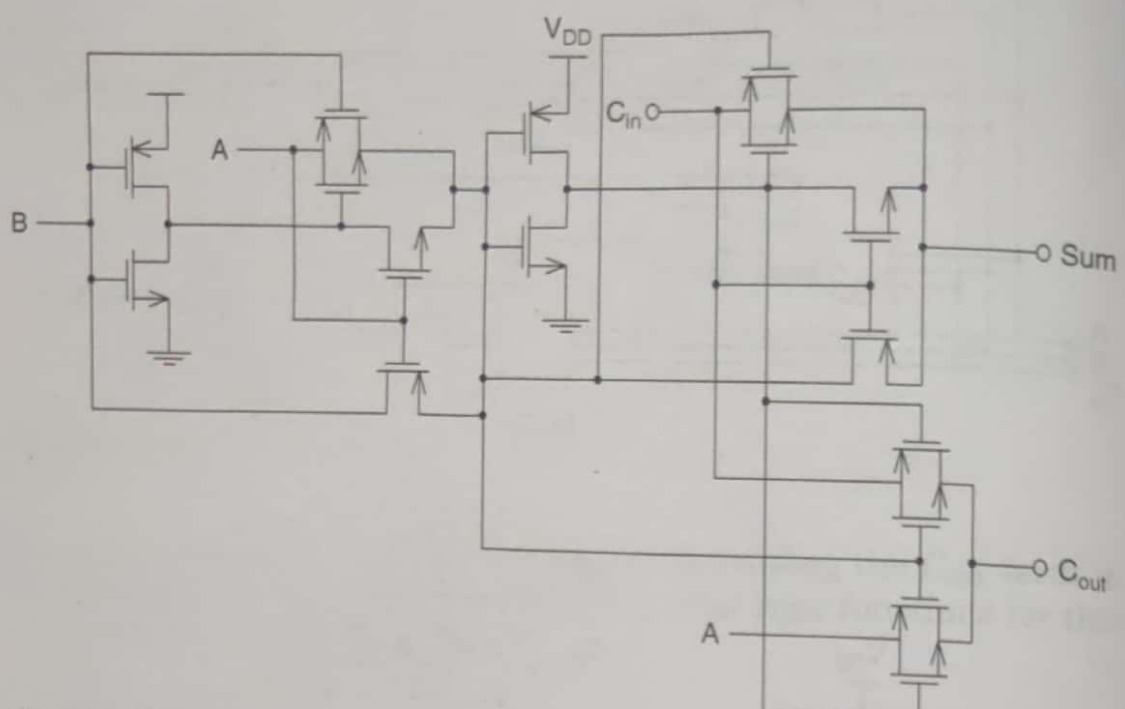


Figure 3.8 The transmission function full adder (TFA).

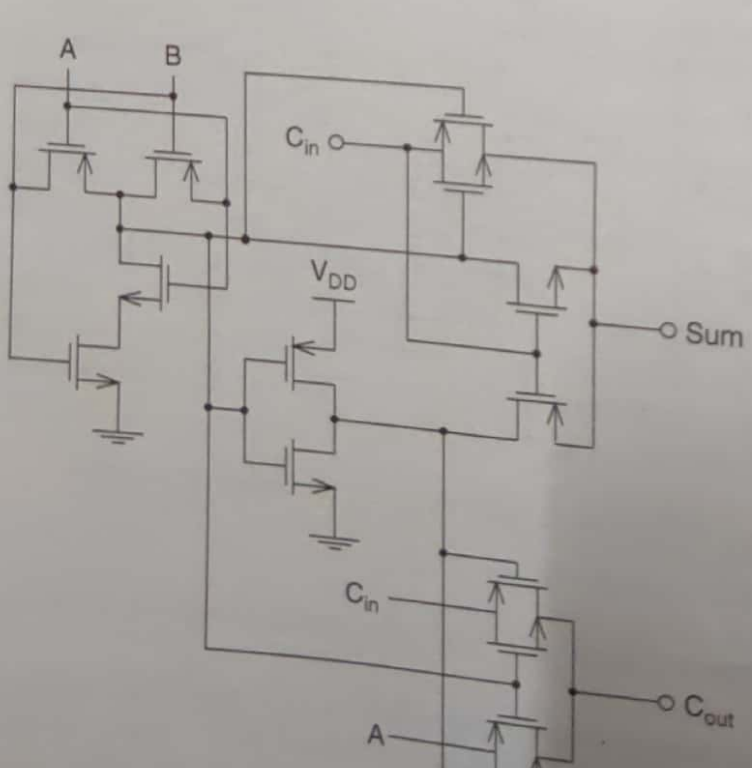


Figure 3.9 The 14-transistor full adder.

Scanned with CamSc

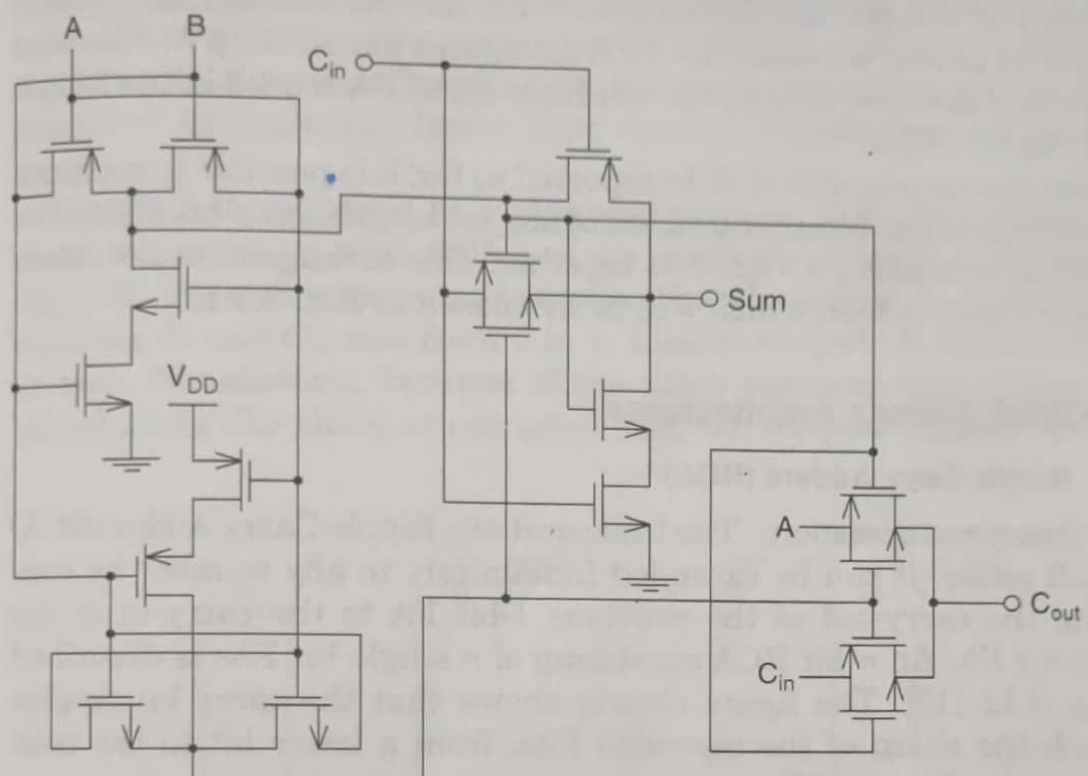


Figure 3.10 The 17-transistor full adder.

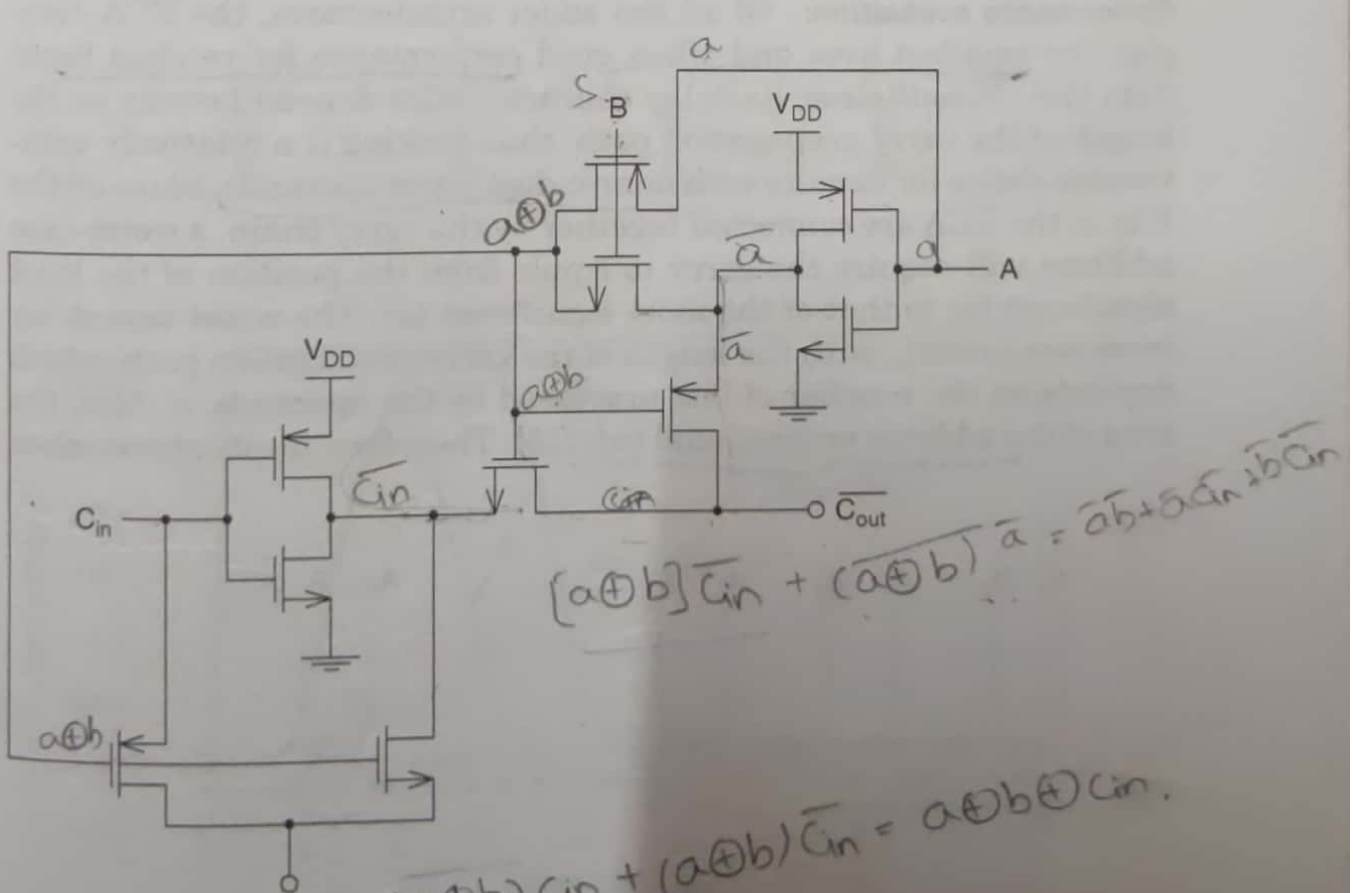


Figure 3.11 The 10-transistor full adder (10-T FA).

Scanned with CamSc

frequency of 1 GHz, the 10-transistor FA has an average dissipation of 81 μW of power, whereas the TFA dissipates about 170 μW. The next section explains how this low-power high-speed FA is used in the Ripple Carry Adder (RCA).

Given the variety of 1-bit FAs reported so far, it is possible to compute the addition of two binary numbers of any bit length. An n-bit adder can be built by cascading n 1-bit FAs together. This arrangement results in a Ripple Carry Adder, which will be discussed in Sec. 3.3.1.

3.3 CMOS Adder's Architectures

3.3.1 Ripple Carry Adders (RCA)

Basic theory and operation. The basic unit of a Ripple Carry Adder (RCA) is a full adder. It can be extended indefinitely to any number by connecting the carry-out of the previous 1-bit FA to the carry-in of the next 1-bit FA. An n-bit RCA consisting of n single-bit FAs is described in Fig. 3.12 [13]. The figure clearly shows that the carry bit ripples through the chain of the cascaded FAs, from a lower bit to the next higher order FA [14, 15].

Performance evaluation. Of all the adder architectures, the RCA occupies the smallest area and offers good performance for random input data [16]. Nonetheless, its delay characteristics depend heavily on the length of the carry propagation path, thus making it a relatively unfavorable choice for circuits with nonrandom input operands. Since all the FAs in the RCA are connected together by the carry chain, a worst-case addition will require the carry to ripple from the position of the least significant bit to that of the most significant bit. The worst-case delay increases linearly with the length of the carry propagation path, which depends on the number of bits processed by the operands, n. Also, the area of the adder is proportional to n [15]. Therefore, in situations when

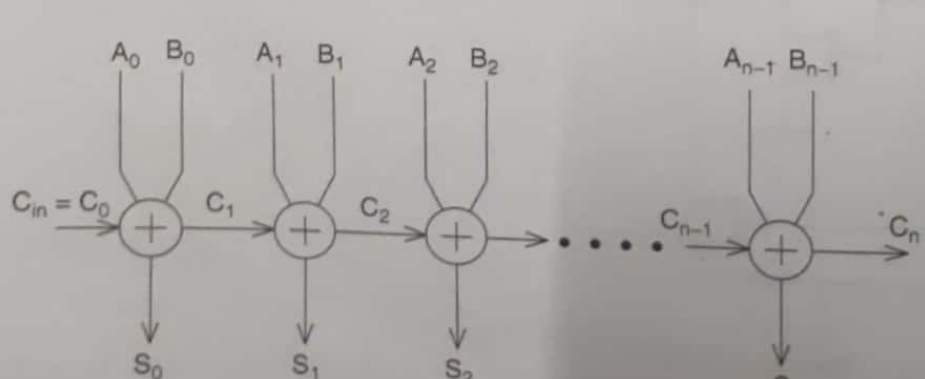
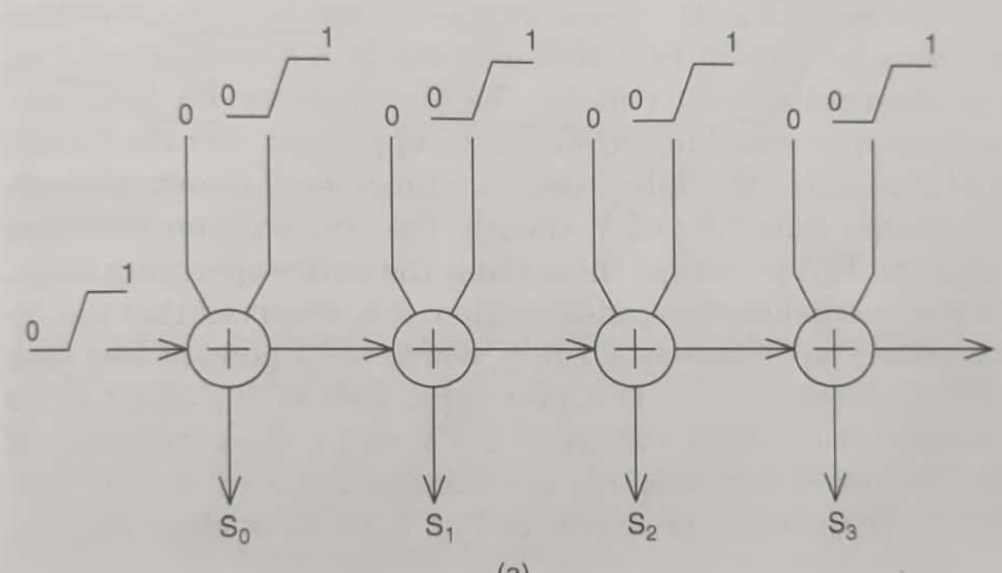


Figure 3.12 The Ripple Carry Adder (RCA).

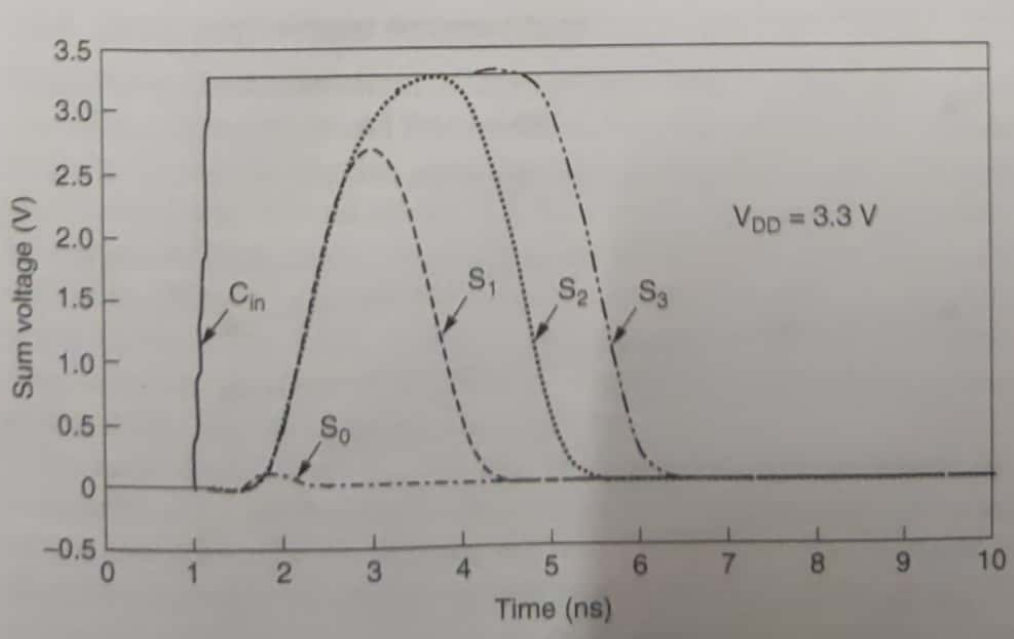
Scanned with CamSc

high speed performance is crucial and the minimum amount of hardware is underperforming, using an RCA in the arithmetic operation would be detrimental. Despite this, however, carry propagation can be enhanced by exploiting faster logic circuit technologies and faster FA designs.

The RCA is subjected to a glitching problem [13]. To further illustrate this issue, a 4-bit RCA and its static simulation is depicted in Fig. 3.13. Here we make an assumption that the inputs A_i are set to zero, whereas B_i and C_{in} rise from 0 to 1. Ideally, outputs S_i should remain at zero. Nonetheless, because of the delay characteristic of the carry signal along the chain of cascaded FAs, the outputs display spurious



(a)



(b)

Figure 3.13 (*a*) The 4-bit RCA. (*b*) Static simulation of the 4-bit RCA.

Scanned with CamSc

transitions, as shown in Fig. 3.13(*b*). This is known as the glitching phenomenon. These dynamic transitions cause extra power dissipation, which can take up a significant portion of the total power. However, this setback can be minimized by means of careful circuit design and optimization.

As mentioned earlier, the carry propagation time for an RCA can be minimized by utilizing various implementations of enhanced FA architectures. One example of such optimization is illustrated as follows. Two prototypes of 32-bit RCAs [3] are constructed. One prototype uses a Transmission gate Full Adder (TFA), whereas the other prototype, which uses a 10-transistor FA, is constructed with a two-transistor inverter driver (10-T FA). Both of these FA schematic configurations have been covered in Sec. 3.2.2. At a power supply voltage of 2.8 V, the critical path delay time for a 32-bit RCA that uses the TFA prototype is 7.2 ns, while it is observed to be 4.1 ns for the 10-transistor FA prototype, thereby exhibiting a speed improvement of 44 percent over the former. Figure 3.14 illustrates the delay time as a function of the supply voltage, which ranges from 2.8 to 5 V. Clearly, the 10-transistor prototype outperforms the TFA prototype throughout the entire operating range.

For the power consumption consideration, it is observed that the 10-transistor prototype dissipates 2.1 mW, which is 81 percent less than the 11 mW dissipated by the TFA prototype. Both of the 32-bit RCAs were simulated at a supply voltage of 2.8 V and a clock frequency of 125 MHz. The power consumption at different values of supply voltage and clock frequencies is shown in Fig. 3.15. It is clear that the

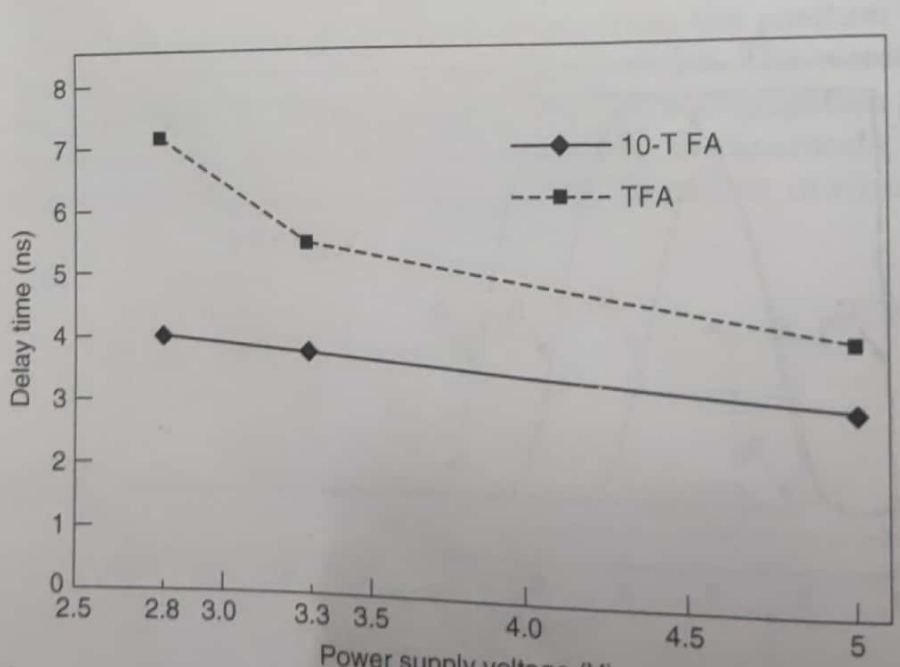


Figure 3.14 Delay time versus power supply voltage.

Scanned with CamSc

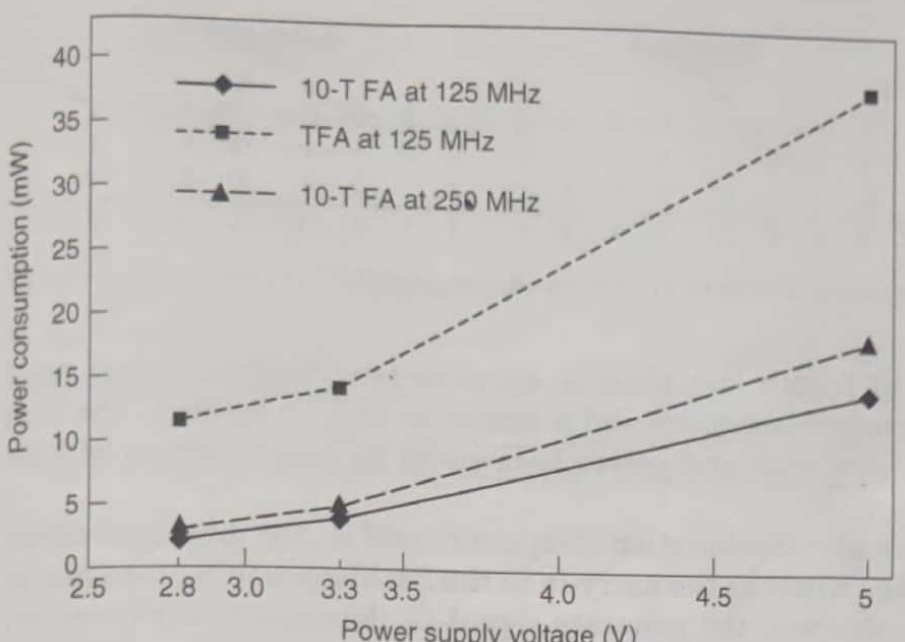


Figure 3.15 Power dissipation versus power supply voltage at various clock frequencies.

10-transistor prototype displays enhanced power dissipation over the TFA for the operating range of 2.8 V to 5 V. It can operate satisfactorily at frequencies up to 350 MHz at a supply voltage of 5 V. This means that large architectures can be built to operate at very high frequencies without compromising the small-area and low-power characteristics, which are the main criteria for today's evolving technology.

3.3.2 Carry Look-Ahead Adders (CLA)

Basic theory and operation. Carry-ripple delays grow linearly with the size of the input operand for the RCA, but these delays can be shortened by generating the carries of each stage in parallel [2, 6]. For example, in the Carry Look-Ahead adder (CLA), carries are generated concurrently by means of look-ahead logic. It is an adder with time propagation duration in $O(\log n)$ and whose area size requirement is in $O(n \log n)$. The delay time of the CLA architecture therefore exhibits logarithmic dependency on the size of the adder, which allows the propagation delay of the carry signal to be minimized [17].

Because forming carries in the RCA sequentially makes it necessary to specify C_{i+1} as a specific function of C_i, limitations arise according to the number of C_{out} signals. In the CLA, however, a carry does not depend explicitly on the preceding one. It can, however, be expressed as a function of the relevant propagate and generate signals, P_i and G_i as well as the initial carry-in, C_{in}. Therefore, the CLA comes in handy for better delay-reduction performance. It is not advantageous, however,

Scanned with CamSc

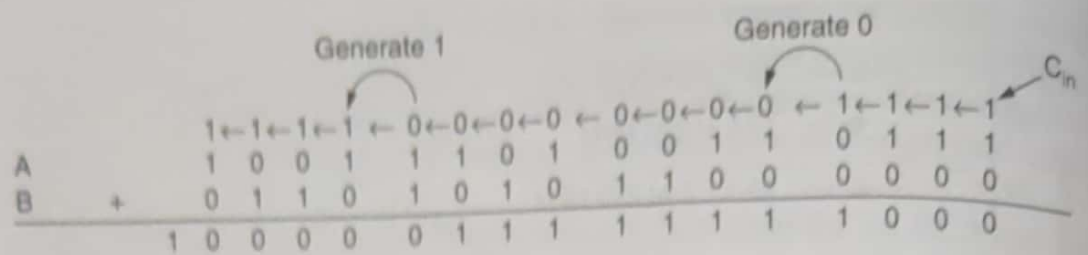


Figure 3.16 An example of carry generation and propagation.

for adders of length less than or equal to four because a four-stage RCA requires less hardware and is almost as fast. In addition, the CLA consumes more area and power because of its large number of logic gates.

For a particular combination of inputs A_i and B_i, the propagate signal P_i determines whether the carry-in to the ith block would propagate to the output, whereas the generate signal G_i determines if a carry-out would be set from inside the block independently from the inputs [14, 18]. The expressions for G_i and P_i with two input binary operands, A_i and B_i are as follows:

$$G_i = A_i \bullet B_i \tag{3.7}$$

$$P_i = A_i \oplus B_i \tag{3.8}$$

Carry generation occurs when $A_i = B_i$, so when $A_i = B_i = 1$, a carry of 1 is produced at the ith position, yet when $A_i = B_i = 0$, a carry of 0 gets generated instead. On the other hand, carry propagation occurs when $A_i \neq B_i$. Hence, when the carry-in (C_{in}) is 1 and $A_i \neq B_i$ for some $i = 0, 1, 2, 3, 4, 5$, then C_{in} is said to propagate to the fifth bit position. Figure 3.16 illustrates an example that shows carry propagation when $A_i \neq B_i$ with carry generation occurring at the 4th and 12th bits where $A_i = B_i$.

Besides the P_i and G_i signals, the Boolean variables for the CLA adder are

$$S_i = P_i \oplus C_i \tag{3.9}$$

$$C_{i+1} = G_i + P_i \bullet C_i \tag{3.10}$$

where S_i and C_{i+1} are the sum and carry recurrence for the ith stage, respectively.

The logic schematic of a 4-bit carry generator and the block diagram of a 4-bit CLA are depicted in Figs. 3.17 and 3.18, respectively. As seen in Fig. 3.17, the carry generation requires only two gate delays. This makes the addition of two n-bit operands extremely fast as compared to the RCA. However, it costs more gates to implement this logic circuit, because for large values of n, a huge number of gates and very big fan-in

Scanned with CamSc

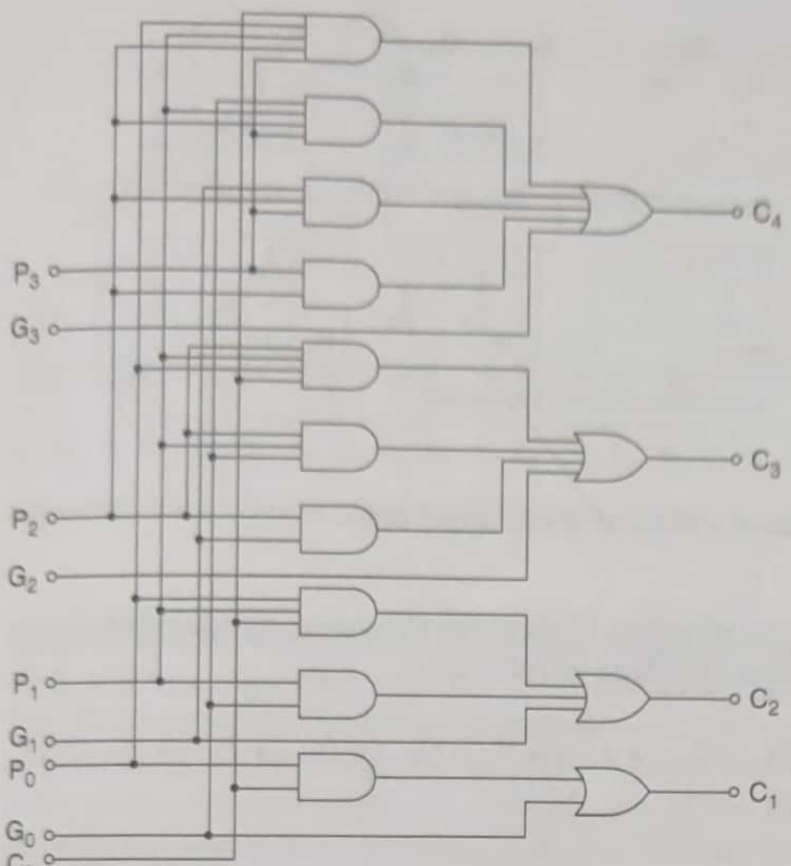


Figure 3.17 Logic schematic of 4-bit carry.

gates are required [14]. Therefore, to reduce the span of the carry look-ahead, the n-bit operands are divided into equal-sized groups, which are then interconnected by incorporating the concept of RCA. This method inevitably reduces the speed of carry propagation.

The block diagram of a 16-bit CLA is shown in Fig. 3.19 [14, 19]. It is divided into four 4-bit groups and comprises a combinatorial circuit, namely the look-ahead carry generator. The term G_k^* denotes the

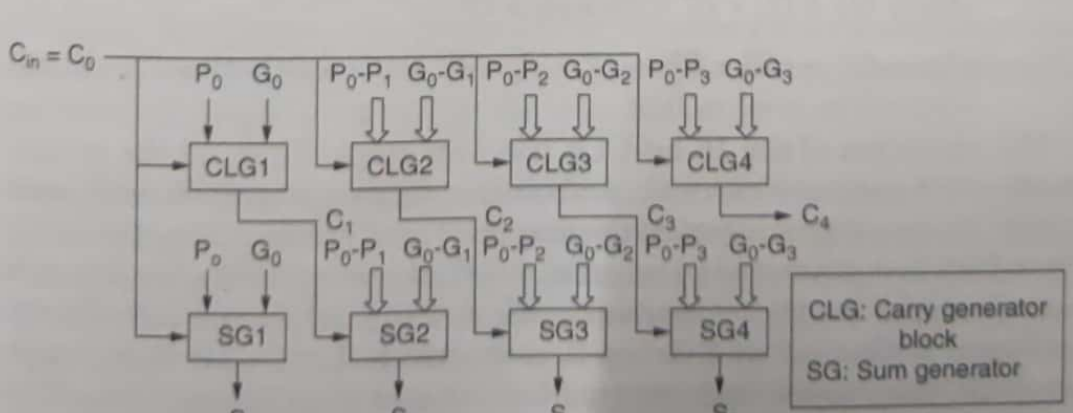


Figure 3.18 Block diagram of the 4-bit CLA.

Scanned with CamSc

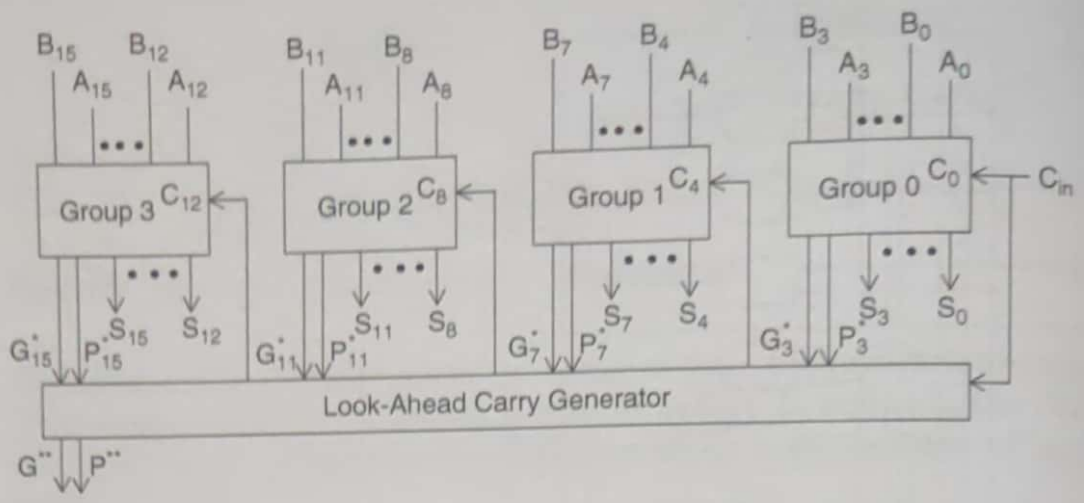


Figure 3.19 Block diagram of a 16-bit carry look-ahead adder.

group-generated carry, whereas P_k^* denotes the group-propagated carry. The Boolean equations for these carries are

$$G_k^* = G_k + G_{k-1} \bullet P_k + G_{k-2} \bullet P_{k-1} \bullet P_k + G_{k-3} \bullet P_{k-2} \bullet P_{k-1} \bullet P_k \quad (3.11)$$

and

$$P_k^* = P_{k-3} \bullet P_{k-2} \bullet P_{k-1} \bullet P_k \quad (3.12)$$

where $k = 3, 7, 11, 15$.

The implementation of these equations is carried out by the look-ahead carry generator, where both terms G_k^* and P_k^* are used to generate the group carry-ins. The outputs of the look-ahead carry generator, C_4, C_8, and C_{12} serve as inputs to the subsequent groups.

$$C_4 = G_3^* + C_0 \bullet P_3^* \quad (3.13)$$

$$C_8 = G_7^* + G_3^* \bullet P_7^* + C_0 \bullet P_3^* \bullet P_7^* \quad (3.14)$$

$$C_{12} = G_{11}^* + G_7^* \bullet P_{11}^* + G_3^* \bullet P_7^* \bullet P_{11}^* + C_0 \bullet P_3^* \bullet P_7^* \bullet P_{11}^* \quad (3.15)$$

The operation of the 16-bit CLA has four steps. First, all the groups produce bit-generate-carry, G_i, and bit-propagate-carry, P_i. Second, each group produces group-generate-carry, G_k^*, and group-propagate-carry, P_k^*, which are generated in parallel. Next, the carry look-ahead generator produces the group carries C_4, C_8 and C_{12}, which are fed directly to Group 1, Group 2 and Group 3, respectively. Lastly, all four groups generate their individual internal carries and then the sum bits. The section-carry generate, G^{**}, and section-carry propagate, P^{**}, are also computed.

Scanned with CamSc

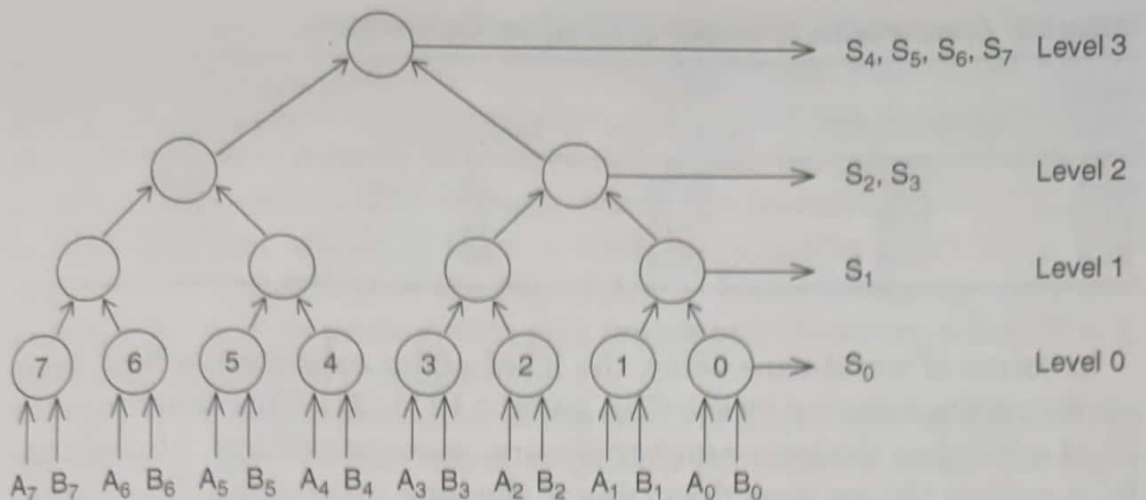


Figure 3.20 Block diagram of the 8-bit ELM adder.

Variation of the basic CLA. In this subsection, a variation of the basic CLA addition algorithm, namely the ELM adder [20], will be analyzed. The ELM addition algorithm incorporates a binary tree of simple processors running in $O(\log n)$ time and it is also based on the concept of carry propagate and generate. The novelty of the ELM adder is that it directly computes the sum bits in parallel, thereby reducing the number of inter-cell interconnects [2]. Figure 3.20 illustrates the block diagram of an 8-bit ELM adder.

The leaves at the lowest level of the tree receive the binary inputs, A_i and B_i. The rightmost node at each level computes that level's output (S_i). Levels 0 and 1 produce one bit of the total sum, S_0 and S_1, respectively. Meanwhile, the rightmost nodes of the remaining levels compute 2^{k-1} bits, where k is the level of the tree. The leaves of the tree compute the carry generate signals, the carry propagate signals and the partial sums. At higher levels of the tree, the nodes receive the generated partial sums from the adjacent lower tree level, as well as the generate and propagate information required to update the partial sums at that particular level of the tree. When all the necessary information is available, new partial sums and the carry generate and propagate signals get calculated and passed up to the next higher level of the tree.

Performance evaluation. The 32-bit CLA and ELM adders have been simulated using the static CMOS circuit design methodology [2]. Both of these architectures use up more area and require more transistors than the RCA. Table 3.3 draws a clear explanation of the relative aspects of the adders. The comparison accounts not only for the transistors, but for the interconnections as well [2]. Note that even though the CLA has more transistors than the ELM adder, it has shorter interconnects and hence occupies a smaller area.

Scanned with CamSc

TABLE 3.3 Comparisons of Various 32-bit Adder Architectures

Adder type	Area ($\times 10^6 \lambda^2$)	No. of transistors	Delay (ns)	Average power dissipation per addition (mW)
CLA	2.27	2132	15	114.6
ELM	2.36	2078	10	104.1
RCA	0.80	1204	55	87.2

In terms of worst-case delay, the ELM adder exhibits the best delay performance, followed by the CLA and the RCA [21]. This is because the ELM adder has the least number of worst-case gate delays, whereas the RCA suffers the greatest delay due to the long carry chain. Meanwhile, the CLA dissipates the most power when compared to its ELM and RCA counterparts.

Manchester Carry Chain (MCC) and Manchester Adder. The Manchester Carry Chain (MCC) [10] is a carry propagation network that is characterized by the well-known carry generate, propagate and annihilate (absorb) functions:

$$\text{Carry generate: } G_i = A_i \bullet B_i \tag{3.16}$$

$$\text{Carry propagate: } P_i = A_i \oplus B_i \tag{3.17}$$

$$\text{Carry annihilate: } AN_i = \overline{A_i} \bullet \overline{B_i} = \overline{A_i + B_i} \tag{3.18}$$

The Manchester Adder uses the MCC as its carry network. The carry recurrence function in Eq. (3.10) serves as the basis for the MCC. The conceptual representation and CMOS realization of a one-stage MCC are depicted in Fig. 3.21 [14]. Referring to Fig. 3.21(a), a one-stage MCC

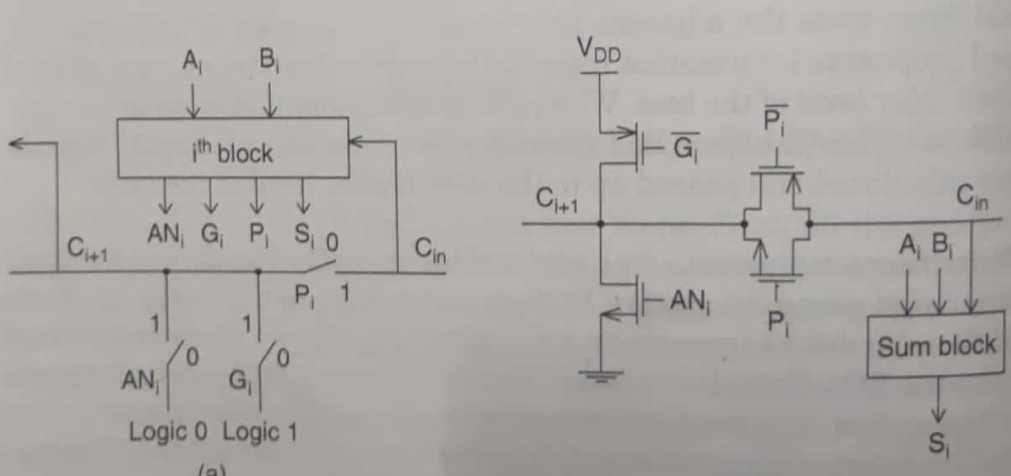


Figure 3.21 (a) Conceptual representation. (b) CMOS realization of the one-stage MCC.

Scanned with CamSc

can be conceptually analyzed as having three switches, each manipulated by controlling signals, G_i, P_i, and AN_i. From Eqs. (3.16), (3.17), and (3.18), it is clear that at any time, only one of the three signals G_i, P_i, and AN_i is at logic 1. The carry-out signal, C_{i+1}, is connected to 0 if AN_i is high, or to 1 if G_i is high, and to the incoming carry, C_{in}, if P_i is high. In other words, if $G_i = 1$, then an outgoing carry ($C_{i+1} = 1$) is generated regardless of the C_{in}. If $AN_i = 1$, any incoming carry gets annihilated and its propagation gets truncated. However, when $P_i = 1$, the C_{in} is allowed to propagate [6].

The CMOS implementation of the MCC is illustrated in Fig. 3.21(*b*) [2]. Once a carry is generated, it quickly propagates along the carry chain composed of transmission gates, until it is finally absorbed. A multiple-bit MCC adder can be constructed by concatenating the circuit in Fig. 3.21(*b*). Buffers are usually inserted between them to partition the n bits into separate groups in order to reduce the delay and strengthen the carry signal.

3.3.3 Carry Select Adders (CSL)

Basic theory and operation. The Carry Select Adder (CSL) [23] provides a substantial compromise between the RCA, which occupies a small area and has a longer delay, and the CLA, which occupies a larger area and has a shorter delay [24]. In the CSL, both the n-bit operands, A_i and B_i are divided into k blocks of possibly different sizes. An example of an 8-bit implementation of a CSL comprising constant-sized blocks is portrayed in Fig. 3.22 [13]. The first block with $C_{in} = C_0$ is implemented by a 4-bit adder, typically the RCA, while the second block is evaluated

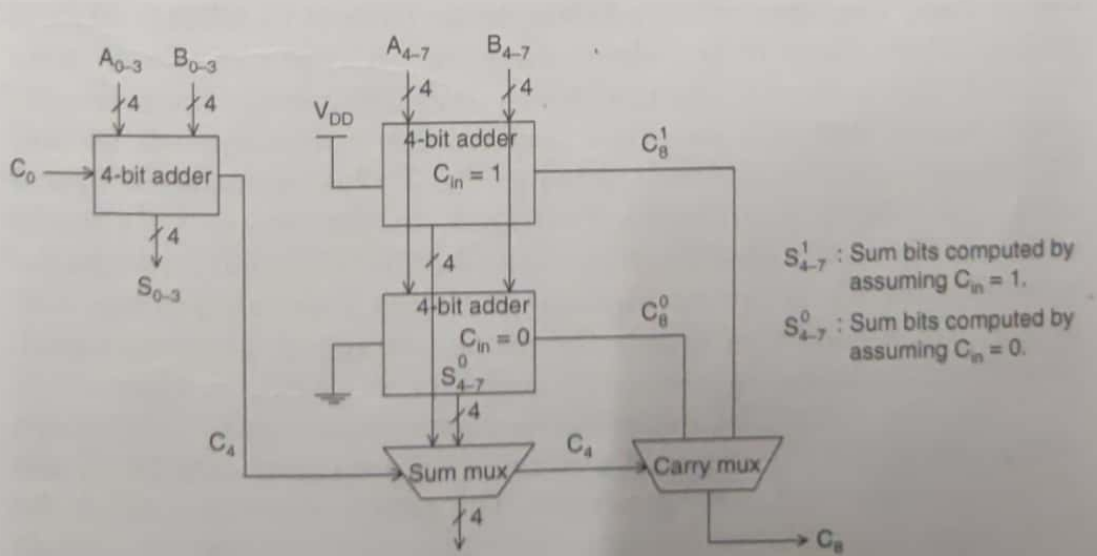


Figure 3.22 An 8-bit architecture of CSL.

Scanned with CamSc

TABLE 3.4 Area and Delay Comparison of CSL and RCA

Number of bits	Area, λ^2		% Change	Delay, ns		% Change
	RCA	CSL		RCA	CSL	
8	154624	290160	87.6	11.5	8.5	−26.0
16	382720	717889	87.5	25.0	11.5	−54.0
32	914400	1779904	94.6	52.5	21.5	−59.0
64	2439168	4667608	91.3	108.0	33.0	−69.4
128	7115072	13536432	90.2	226.0	54.0	−76.1
Average			90.2			−56.9

conditionally with a pair of adders, thus permitting two additions to be executed in parallel. One addition assumes that the carry-in (C_{in}) is zero while the other addition assumes that the C_{in} is one [2]. Accordingly, two sums and two carry-outs get generated. In order to select the correct set of sum bits (S_{4-7}) and the carry-in for the next block (C_8) from the respective multiplexers, a control signal, C_4, which is computed from the preceding block is required. The additional cost of the CSL over the RCA is the duplicate carry chain and the select logic [25].

Performance evaluation. Layouts for the RCA and CSL adders were generated for the following sizes: 8-bit, 16-bit, 32-bit, 64-bit, and 128-bit [25]. The comparisons of the area sizes and performance delays for both types of adders are summarized in Table 3.4. It can be seen that classical CSL averages a 56.9 percent performance delay reduction at the cost of using a much larger area as compared to that of the RCA.

Hybrid Carry Look-Ahead/Carry-Select Adder (Hybrid CLA/CSL). Hybrid adders, which refer to the elementary combinations of two or more "pure" design methods, aim to reduce power dissipation, improve cost-effectiveness, and achieve other performance enhancements as well [22]. In fact, the CSL adder depicted in Fig. 3.22 is essentially a hybrid adder, since the RCAs are coupled within it. Another example of a hybrid adder that is widely used combines the CLA and CSL and is named the hybrid CLA/CSL. Its 16-bit implementation is illustrated in Fig. 3.23 [22]. P_i, G_i, S_i, and C_{i+1} denote the propagate signal, generate signal, sum signal, and carry-out signal for each bit i, respectively, where $i = 0, 1, \ldots, 15$. The pair of CSL adder blocks may be based on the MCC adder, which will supply the required generate and propagate signals, G_i and P_i, to the look-ahead carry generator. The multiplexers then select the final carry, C_{16}, and also the sum bits when the block carry-in signals are known.

Scanned with CamSc

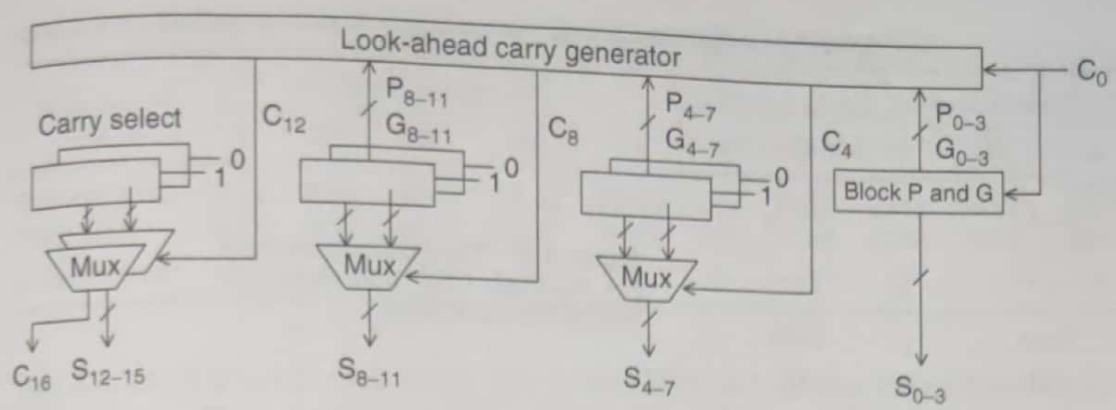


Figure 3.23 A 16-bit hybrid CLA/CSL.

3.3.4 Carry Save Adders (CSA)

Basic theory and operation. As described in Sec. 3.3.1, the RCA makes use of a row of cascaded binary FAs to compute the summation of two operands. In fact, with slight modification, this row of FAs can also be viewed as a mechanism to reduce three binary numbers into two binary numbers in multi-operand (three or more operands) addition [22]. This method is used in the Carry Save Adder (CSA) where it is indeed an RCA with its carries saved rather than propagated. Therefore, the CSA operator is often called a (3:2) counter. The block diagrams for RCA and CSA are depicted in Fig. 3.24 for comparison.

Unlike RCA, CLA, and CSL adders, the CSA realizes concurrent addition of multiple operands, which is a basic requirement of multiplication. Instead of using the 2-operand adders that necessitate the time-consuming carry propagation to repeat several times, depending on the number of operands, the CSA's timing could be improved with a little increase or even a reduction in area [26].

A CSA tree consists of CSA operators and one adder at the root of the tree. The CSA operators are used to transform an arbitrary number of operands in the addition process to produce two adding operands, after which the adder at the root of the CSA tree computes the final sum [27]. Figure 3.25 shows the addition of three 1-bit binary numbers, A, B and C, implemented without the CSA operator and with the CSA

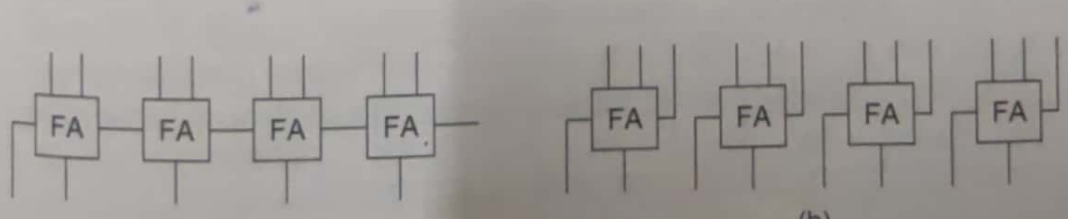


Figure 3.24 Sample block diagram of (*a*) the RCA; (*b*) the CSA.

Scanned with CamSc

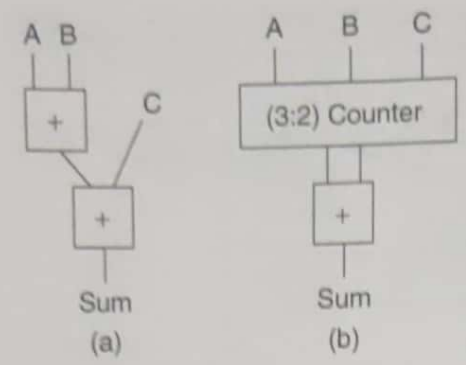


Figure 3.25 The addition process (*a*) without CSA operation; (*b*) with CSA operation.

operator, respectively. The performance evaluation of both additions is reported in the next section.

The 1-bit multi-operand addition mentioned earlier can be extended to an n-bit multi-operand addition by cascading the CSA operators. An n-bit CSA consists of n disjoint FAs operating in parallel. Each FA has three ith bit inputs and generates two outputs, namely an ith bit partial sum, S, and an ith bit carry, C. As for adding more than three operands, there is a second or further subsequent levels of the CSA operators. They receive the S and C from the previous CSA operator level, together with another input operand, and produce a set of new S and C values [14]. The levels with CSA operators contain no carry propagation. The carries propagate only in the last step. Consequently, as compared to the RCA's propagation delay of n FAs, the CSA has the same propagation delay as only one FA, and it remains constant for any value of n [26, 27, 28].

Figure 3.26 shows a CSA for the addition of four 4-bit binary numbers A, B, C and D, with an initial carry-in, C_0 [14]. The upper two levels of

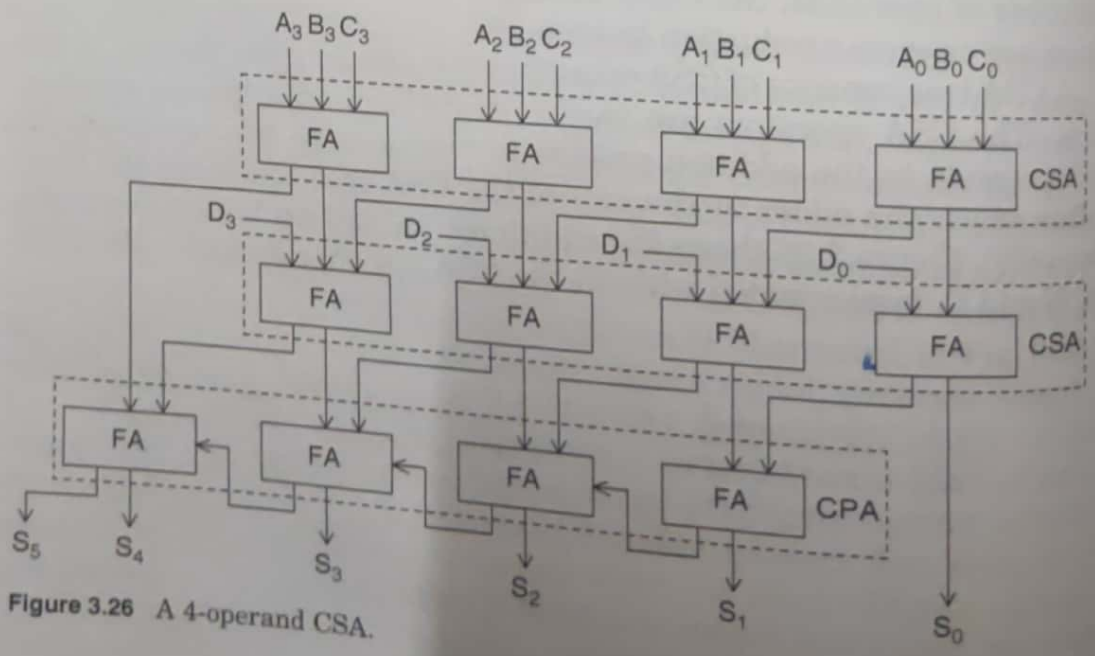


Figure 3.26 A 4-operand CSA.

Scanned with CamSc

TABLE 3.5 Timing and area comparison of the operation in Figure 3.25

Number of bits, n	Best timing (ns)					Cell area under the best timing (λ^2)				
	8	24	40	56	64	8	24	40	56	64
Without CSA	3.12	8.46	13.16	17.17	19.33	402	1337	2245	3412	3873
With CSA	2.72	8.06	12.77	15.57	18.12	364	1186	1993	2934	3341
% Reduction	13	5	3	9	6	9	11	11	14	14

the FA or (3:2) counters form the 4-bit CSA, while the third level is the 4-bit Carry Propagating Adder (CPA). The CPA used in this case is an RCA, but a CLA or any other fast adder can substitute for it. Each CSA operator takes three bits of the same significance and then computes the sum of the same significance and the carry for the bit of a higher significance [19].

The implementation of the CSA can be further expanded to add k operands. Here, $(k-2)$ CSA levels and one CPA are required to realize the addition operation. The time to obtain the summation is

$$T = (k-2) \bullet T_{\text{CSA}} + T_{\text{CPA}} \tag{3.19}$$

where T_{CSA} and T_{CPA} are the execution times for a CSA level and CPA, respectively.

Performance evaluation. The timing and area comparison for the two operation trees, without and with CSA implementation, is illustrated in Table 3.5 [27]. The table shows that there is a substantial reduction in timing when the CSA method is used. Meanwhile, the area comparison implies that by using the CSA, we can reduce the cell area quite significantly.

3.3.5 Carry Skip Adders (CSK)

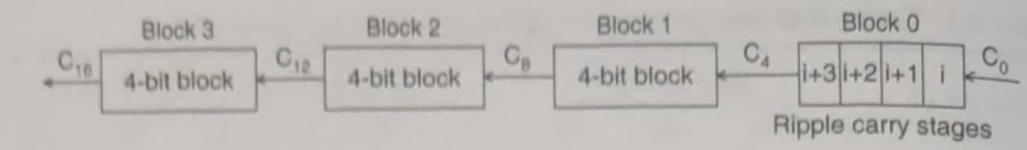
A Carry Skip Adder is similar in concept to how the MCC adder works, which is to reduce the carry propagation time by skipping over groups of successive adder stages [14]. A 16-bit RCA divided into four 4-bit blocks is portrayed in Fig. 3.27(a) [22]. Each block has four stages ranging from $4k$ to $4k+3$, where k is the number of the block, from 0 to 3. In the meantime, each stage consists of a 1-bit FA with two input binary numbers, A_i and B_i and the following Boolean definitions:

$$P_i = A_i \oplus B_i \tag{3.20}$$

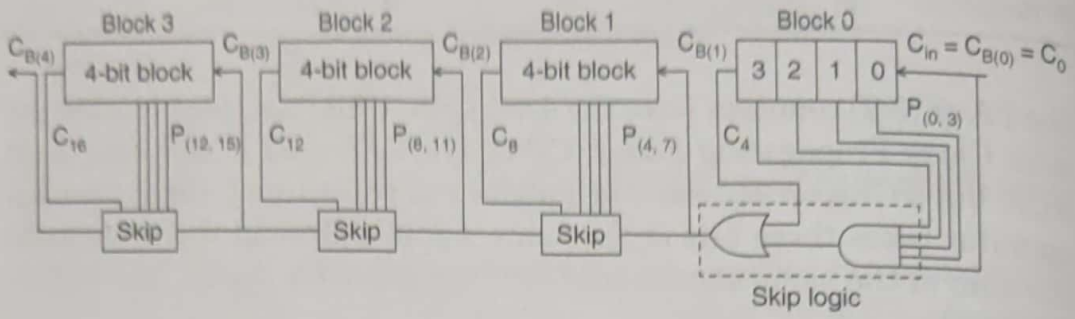
$$S_i = P_i \oplus C_i \tag{3.21}$$

$$C_{i+1} = A_i \bullet B_i + P_i \bullet C_i \tag{3.22}$$

Scanned with CamSc



(a)



(b)

Figure 3.27 (*a*) A 16-bit RCA. (*b*) A 16-bit CSK with 4-bit skip blocks.

Here, i is the stage position, P_i is the carry propagate signal, S_i is the sum, C_{i+1} is the carry-out and C_i is the carry-in for each stage [29]. We denote the block carry propagate signal as

$$P_{[4k,4k+3]} = P_{4k} \bullet P_{4k+1} \bullet P_{4k+2} \bullet P_{4k+3} \tag{3.23}$$

The time-consuming carry-ripple phenomenon in the RCA can be improved by establishing bypasses or skip paths around the 4-bit blocks, as depicted in Fig. 3.27(*b*). The carry can skip any consecutive stages if the corresponding bits in the two operands are not equal ($A_i \neq B_i$ or $P_i = 1$). Meanwhile, every block computes a block carry propagate signal, $P_{[4k,4k+3]}$, which is equal to one if all stages within the block satisfy $P_i = 1$[14]. With this signal, a carry-in into a block k can bypass all the internal stages and straightaway produce a block carry-out. This block carry-out signal will then become the carry-in for the next block. As shown in Fig. 3.27(*b*), the skip logic consists of a 5-input AND gate and a 2-input OR gate with the following Boolean expression for the block carry-out, C_B:

$$C_{B(k+1)} = C_{4k+4} + P_{[4k,4k+3]} \bullet C_{B(k)} \tag{3.24}$$

where block $k = 0, 1, 2, 3$, denotes the particular group taken into consideration.

For an n-bit CSK with fixed block size K, the worst-case carry propagation occurs when a carry is generated in stage 0 of block 0 and then ripples through stages $1, 2, \ldots, (K-1)$ within the first block, skips blocks $1, 2, \ldots, (\frac{n}{k} - 2)$, and propagates through the last block $(\frac{n}{k} - 1)$.

Scanned with CamSc

In general, with the assumption that the delay of the skip logic is equal to the carry propagation delay from one stage to another (in the first and the last block), the overall time of this carry propagation chain is [22] calculated as follows:

$$T_{\text{carry}} = (K-1) + \left(\frac{n}{K} - 2\right) + (K-1) \quad \text{stages}$$

$$\text{first block} \quad \text{skips} \quad \text{last block}$$

$$= \left(2K + \frac{n}{K} - 4\right) \quad \text{stages} \tag{3.25}$$

Assuming that a straightforward 2-gate level implementation is used for both the RCA stage and skip block, then one stage will have a $2\Delta_G$ gate delay. Equation (3.25) becomes

$$T_{\text{carry}} = \left(2K + \frac{n}{K} - 4\right) \bullet 2\Delta_G \tag{3.26}$$

Differentiating Eq. (3.26) to K and equating it to 0 results in an optimal fixed block width.

$$\frac{dT_{\text{carry}}}{dK} = \left(2 - \frac{n}{K^2}\right) \bullet 2\Delta_G = 0 \tag{3.27}$$

$$K_{\text{opt.}} = \sqrt{\frac{n}{2}} \tag{3.28}$$

Therefore, for a 32-bit CSK, the best design will be eight blocks of $K_{\text{opt.}} = \sqrt{\frac{32}{2}} = 4$ block width. This leads to a worst-case carry propagation of $24\Delta_G$ instead of $32 \times 2\Delta_G = 64\Delta_G$ for the 32-bit RCA. The fixed block-width CSK can be modified to a variable block-width CSK with multiple levels of skip logic associated with it to create a faster circuit [30].

3.3.6 Conditional Sum Adders (COS)

The Conditional Sum Adder is a fast adder implementation with logarithmic speedup [14, 31, 32]. The underlying principle of this architecture is the parallel computation of two sets of conditional sums and conditional carries for every bit position. To minimize the overall computation time, the given n bits are divided into smaller groups so that the serial carry propagation within the groups can be carried out in parallel. The COS approach is very similar to that of the CSL. The major difference is that in COS the division of the n bits into subblocks continues until the extreme of having only single-bit adders if n is an

Scanned with CamSc

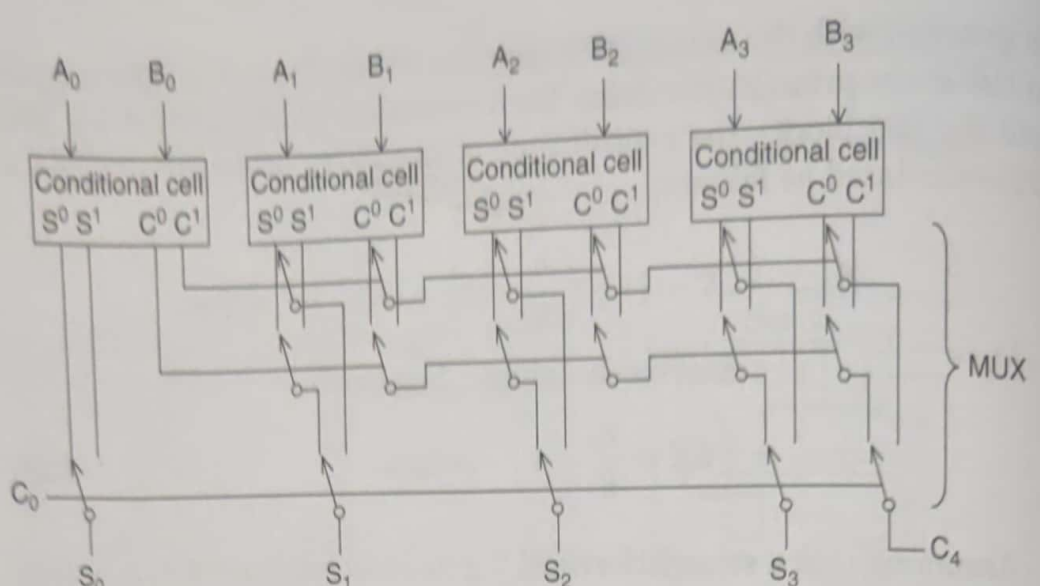


Figure 3.28 Schematic diagram of a 4-bit COS.

integer power of 2. The COS scheme can still be implemented even if n is not a power of 2 because subblocks of equal sizes are not compulsory. The operation of the COS can be further illustrated with the aid of a 4-bit COS depicted in Fig. 3.28 [13, 33].

Figure 3.28 shows that the 4-bit COS makes use of two types of cells, namely the conditional cells and the Multiplexers (MUXs). Each conditional cell generates two sums and two carries consecutively. One set of the results, S_0 and C_0, assumes that the carry-in is a 0, while the other set, S_1 and C_1, is based on the assumption that the carry-in is a 1 [32]. Therefore, the block results will be ready for both carry conditions. The real sums are selected by the initial carry-in (C_0) and the carry-outs from the previous cells, which act as the control signals for the multiplexers. One possible logic realization of the conditional cell is shown in Fig. 3.29 [22].

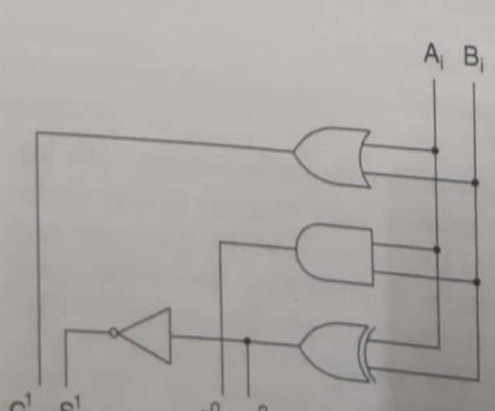


Figure 3.29 Logic realization of the conditional cell.

Scanned with CamS

The Boolean expressions for the above logic implementation are as follows:

$$S_i^0 = A_i \oplus B_i \tag{3.29}$$

$$C_{i+1}^0 = A_i \bullet B_i \tag{3.30}$$

$$S_i^1 = \overline{A_i \oplus B_i} \tag{3.31}$$

$$C_{i+1}^1 = A_i + B_i \tag{3.32}$$

3.3.7 Performance evaluation of various adder architectures

An overall performance evaluation and comparison, to rank the adders reported in the preceding subsections, has been carried out based on a 1.2 μm scalable CMOS technology. Table 3.6 [2, 34] shows the area and number of transistors for each type of adder. The RCA occupies the smallest area, as compared to the rest of the adders, at the expense of the longest worst-case delay, as shown in Table 3.7. Therefore, the RCA is particularly suitable for applications where the area savings are of critical importance, while high-speed performance is not as crucial [13].

All the parallel adders have been simulated using HSPICE at a power supply voltage of 5 V [2, 34]. The experimental results obtained using a clock frequency of 10 MHz are listed in Table 3.7. The delay time is typically considered as the time taken for the carry to ripple from the input to the slowest output, which is from the least significant bit position to the most significant bit position. As mentioned earlier, the RCA has the lowest speed because of its long carry chain and is therefore inappropriate for high-speed applications. As for the MCC, it is undoubtedly an attractive choice for systems of lower precisions (up to 16-bit), since

TABLE 3.6 Comparison of Different Types of Adders in Terms of Area and Number of Transistors

	Area, ($\times 10^6 \lambda^2$)			No. of transistors		
Adder type	16-bit	32-bit	64-bit	16-bit	32-bit	64-bit
RCA	0.40	0.80	1.60	596	1204	2420
CLA	1.14	2.27	4.55	1038	2132	4348
ELM	1.08	2.36	5.38	892	2078	4752
MCC	0.48	0.96	1.90	642	1298	2610
CSL	0.76	1.45	2.75	914	1982	4128
CSA	1.05	2.03	3.90	1176	2360	4728
CSK	0.82	1.62	3.22	682	1410	2866
COS	2.34	—	—	—	—	—

Scanned with CamSc

3.5 Low-Voltage Low-Power Design Techniques

3.5.1 Trends of technology and power supply voltage

Figure 3.38 depicts the technology trends of the Microprocessor (MPU) printed gate length and power supply voltage, beginning with the year 2001 and projecting up to year 2016 [36]. Most of the process technology

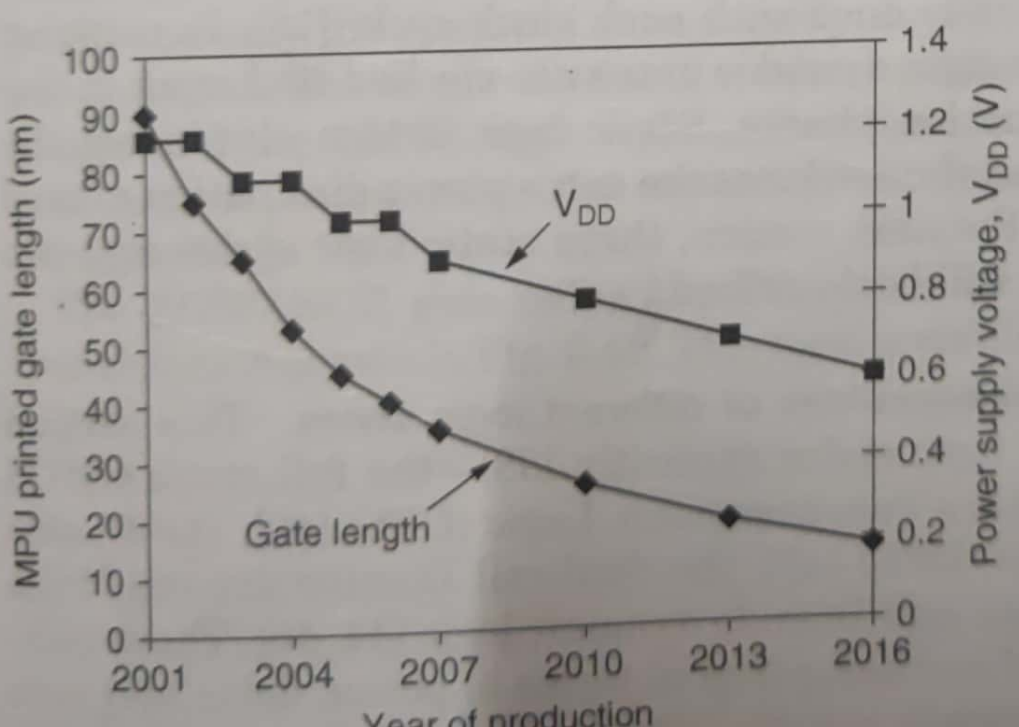


Figure 3.38 Trends of technology and power supply voltage.

Scanned with CamS

studies for low-voltage and low-power applications converge to the conclusion that scaled BiCMOS/CMOS technology will remain the dominant solution in the future. The technology was at 95 nm in 2001 and it reduced to 65 nm by 2003. It is conceivable that once the problem in manufacturing yield is overcome, by 2016, the gate length will reduce to 13 nm. As for the power supply voltage, it was at 1.2 V in 2001 and is expected to experience a ladder-like reduction to 0.9 V by 2007. In the long term, it is predicted that it will continue to reduce to 0.6 V by 2016 due to portability and reliability issues.

3.5.2 Low-voltage low-power logic styles

Low-power applications have emerged as an arena of prime concern for VLSI system designers. Together with that, the high-speed adder that uses low-power consumption has indisputably become one of the most crucial components of a processor because it is heavily used in the Arithmetic Logic Unit (ALU), the floating-point unit, and for address generation during cache or memory access.

The relentless drive for adders with low-power dissipation can be addressed at various design levels, namely the architecture, circuit, layout, device and process technology [11, 37, 38]. At the circuit level, to achieve considerable power savings, the designer can use a myriad of different adder types, as described in Secs. 3.3 and 3.4. Another potential approach is by implementing a proper choice of logic styles for a given adder type. This approach will be illustrated in the next section.

Static and dynamic logic styles. CMOS logic styles can be categorized into static and dynamic circuits. Static logic families evaluate the output whenever there is an input variation, while the dynamic logic gates evaluate the output only once with each clock cycle [18]. In contrast to the static gates design, dynamic gates are clocked and work in the precharge and evaluation phases. Static logic design eliminates the precharging stage and thus reduces the extra power dissipation caused by clocking [39]. In the next section, three static logic styles and two dynamic logic styles will be described.

XOR/XNOR Gate implementations of different logic styles. This section covers five different logic styles explicitly [38]—the full static CMOS [6], the Complementary Pass-transistor Logic (CPL) [40], the Double Pass-transistor Logic (DPL) [41], the dual-rail Domino dynamic logic [42, 43], and the single-rail Domino dynamic logic [44, 45]. These techniques are evaluated and compared in terms of power, delay and area efficiency at power supply voltage ranging from 1.5 V to 3.5 V. In the succeeding paragraphs, each of the earlier mentioned logic families will

Scanned with CamSc

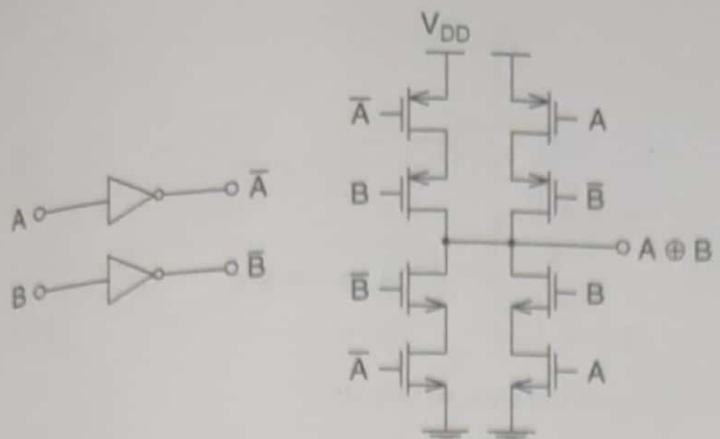


Figure 3.39 An XOR gate using full static CMOS.

be briefly explained using their respective realization of the XOR/XNOR gate, which is the most fundamental gate in adder circuits.

Figure 3.39 portrays the full static CMOS XOR gate. The serial connections of the nMOS or pMOS transistors require increased width in order to acquire a reasonable conducting current to drive capacitive loads. This is because connecting the pMOS or nMOS devices in series can be visualized as a number of cascaded transistors. The delay time imposed by these devices is defined by

$$\tau = C \bullet R \tag{3.33}$$

$$\frac{1}{R} \infty \frac{W}{L} \tag{3.34}$$

where C is the capacitance, R is the resistance, L is the channel length, and W is the channel width, which is inversely proportional to R. Therefore, to minimize the delay time, W must be increased.

A multitude of different pass-transistor logic designs have been proposed and only the essential ones, namely the CPL and the DPL will be summarized. The major distinction between the pass-transistor logic family and the CMOS logic style is that the source node of the MOS transistor is connected to the input signals rather than to the power supply voltage [37].

The XOR/XNOR gate using the CPL with an array of nMOS pass transistors is shown in Fig. 3.40. This logic style eliminates the problem

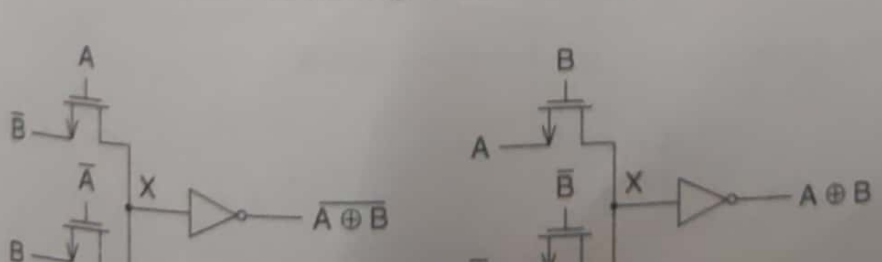


Figure 3.40 XOR/XNOR implementation of the CPL.

Scanned with CamSc

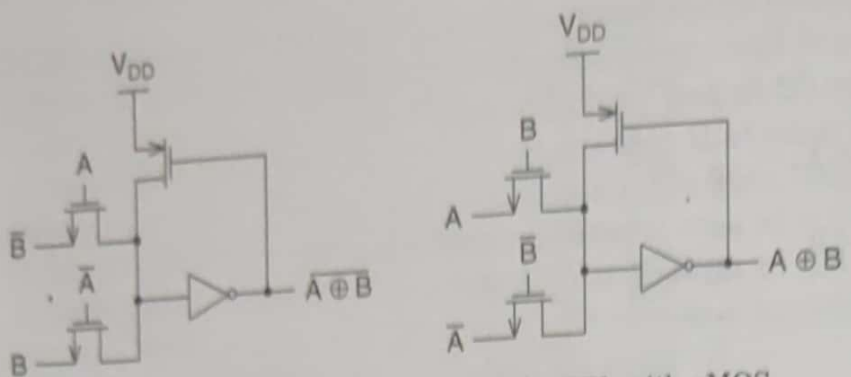


Figure 3.41 XOR/XNOR implementation of the CPL with pMOS feedback.

of vigilantly sizing the serial transistors, thereby requiring only half as many transistors as compared to the full static CMOS XOR gate. When the output of the nMOS pass transistor network at node X is logically high, at $(V_{DD}-V_{th})$, where V_{th} is the threshold voltage, it causes a major setback by inducing an incomplete turn-off of the pMOS in the inverter, thus incurring a high short circuit current. To restrain this current, a pMOS device is then coupled across the output of the inverter gate in order to pull up the output node X to full V_{DD}, as illustrated in Fig. 3.41.

Another logic design that uses pass transistors is the DPL, which is a verification of the CPL. The XOR/XNOR gate using the DPL is depicted in Fig. 3.42. By using both the pMOS and nMOS devices, the DPL prevents the problem of the nMOS threshold voltage dropping in CPL logic design.

Figure 3.43 shows the dual-rail Domino logic style utilized to construct the XOR/XNOR gate. Contrary to static techniques, dynamic design requires a precharge and an evaluation phase. The precharge stage occurs when the CLK signal is at a low value, while the evaluation stage takes place when the CLK signal is at a high value. Because of the

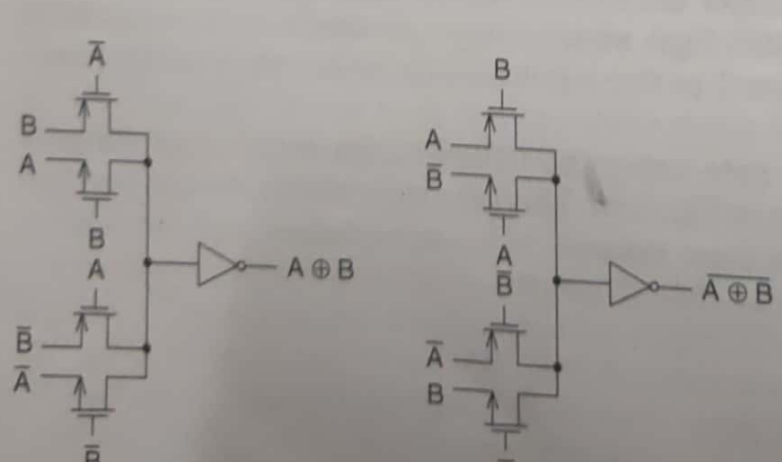


Figure 3.42 XOR/XNOR using DPL.

Scanned with CamSc

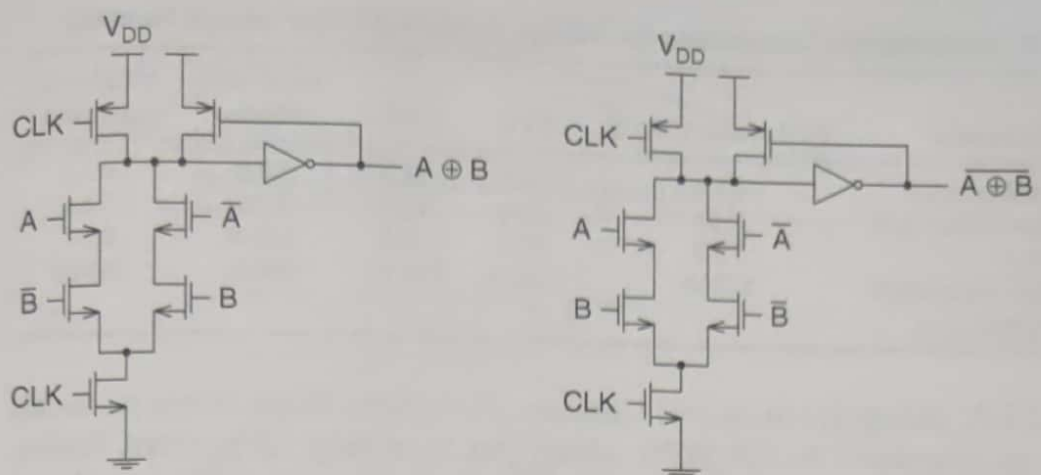


Figure 3.43 XOR/XNOR gate using dual-rail Domino logic style.

precharge and evaluation phases, the dynamic design abolishes all the spurious transitions and its corresponding power consumption, which is intrinsically present in any static logic designs. Nevertheless, devices such as the clock driver actually dissipate additional power.

As reported in Ref. [38], the dual-rail inputs are only necessary for XOR/XNOR gate implementation. For other logic gate functions, such as NAND and NOR, the single-rail inputs are possible. Accordingly, power dissipation is alleviated. The NOR gate realization employing the single-rail Domino logic is demonstrated in Fig. 3.44.

32-bit CLA Adder Implementations. A 32-bit CLA is constructed using three static logic styles and two dynamic Domino circuit families described earlier. Table 3.8 shows the power, delay and area summary for these adders that were simulated using SPICE, at a power supply voltage of 3.3 V, and with an input clock frequency of 100 MHz.

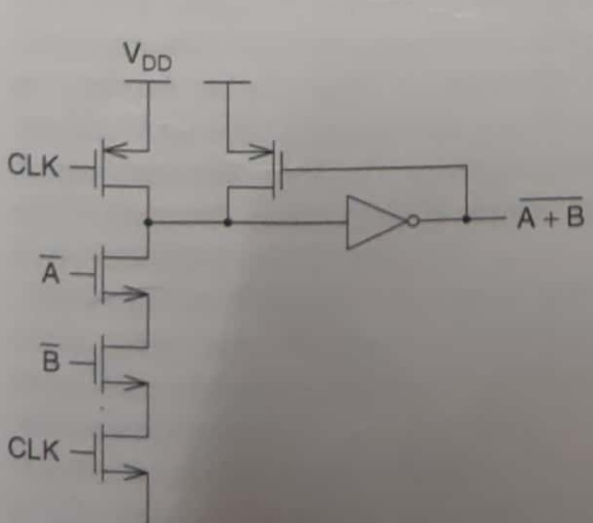


Figure 3.44 NOR gate realization of the single-rail Domino logic style.

Scanned with CamSc

TABLE 3.8 Performance Comparison for Adders of Various Logic Design Families

Parameters	Full static CMOS	CPL	DPL	Dual-rail domino	Single-rail domino
Power (mW)	34.3	34.5	27.5	82.5	60.2
Worst case delay (ns)	2.33	2.24	1.98	1.78	1.64
Energy (pJ)	79.9	77.3	54.5	146.9	98.7
Area: total transistor width (μm)	27355	17437	19117	32444	26926

The CLA, using the dual-rail Domino, dissipates three times as much power as compared to the DPL adder, yet it is only 12 percent faster. As for its single-rail Domino counterpart, it consumes about two times more power than the DPL and exhibits speed that is 21 percent faster. The static styles are more energy-efficient than the dynamic styles. More precisely, among all the adders realized based on static circuit styles, the DPL adder is the most energy-efficient; it occupies 47 percent less area and barely 10 percent more area than the full static CMOS and the CPL adders, respectively.

3.6 Current-Mode Adders

3.6.1 Current-Mode CMOS Adders using multiple-valued logic

In today's state-of-the-art VLSI technology, the common approach used in designing arithmetic circuits is the voltage-mode operation for both digital and analog parts of the chip. With the increasing need to develop more advanced circuits, by using reduced wiring complexity and lower device count per function, the interest in current-mode design methods has revitalized [46, 47]. The ultimate aim is to realize applications that are both high in speed and low in power consumption.

It is well known that the binary number system is a logical and dominant choice for conventional voltage-mode design of digital computers. However, there are also significant drawbacks to a binary number in traditional system implementations. For instance, in a typical VLSI circuit, about 70 percent of the chip area is occupied by interconnection, 20 percent by insulation and the remaining 10 percent is devoted to the devices. Interconnection occupies a large portion of the physical area even when it is not in use. Thus, the interconnect will be more efficient if several levels of logic then get injected into a single wire, as in the Multiple-Valued Logic (MVL). MVL requires more than two discrete levels of logic signals that are better represented in Current Mode (CM) by specific current values, thus resulting in the birth of the Current-Mode Multiple-Valued Logic (CMMVL) [46–48]. The exploitation of this design technique with a proper choice of numerical representation of

Scanned with CamSc