

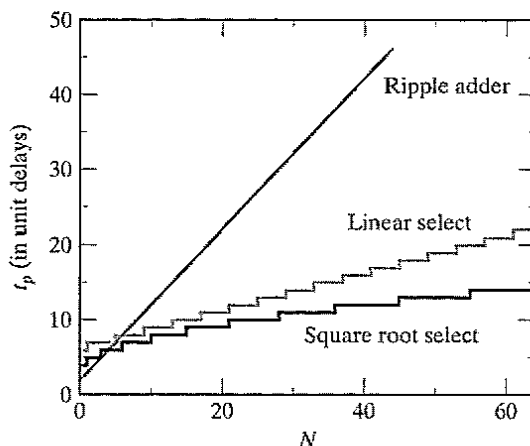


Figure 11-19 Propagation delay of square-root carry-select adder versus linear ripple and select adders. The unit delay model is used to model the cell delays.

or

$$P \approx \sqrt{2N} \quad (11.13)$$

Equation (11.13) can be used to express t_{add} as a function of N by rewriting Eq. (11.10):

$$t_{add} = t_{setup} + Mt_{carry} + (\sqrt{2N})t_{mux} + t_{sum}. \quad (11.14)$$

The delay is proportional to $\sqrt{N}$ for large adders ($N \gg M$), or $t_{add} = O(\sqrt{N})$. This square-root relation has a major impact, which is illustrated in Figure 11-19, where the delays of both the linear and square-root select adders are plotted as a function of N . It can be observed that for large values of N , t_{add} becomes almost a constant.

Problem 11.3 Unequal Bypass Groups in Carry-Bypass Adder

A careful reader might be interested in applying the previous technique to carry-bypass adders. We saw earlier that their delay is a linear function of a number of bits. Can they be modified to achieve better than linear delay by using variable group sizes?

It does make sense to make the consecutive groups gradually larger. However, the technique used in carry-select adders does not directly apply to this case, and a progressive increase in stage sizes eventually increases the delay. Consider a carry-bypass adder in which the last stage is the largest: The carry signal that propagates through that stage and gets consumed at the *msb* position (with no chance of bypassing it) is on the critical path for the sum generation. Increasing the size of the last group does not help the problem.

Based on this discussion and assuming constant delays for carry and bypass gates, sketch the profile of the carry bypass network that achieves a delay that is better than linear.

The Carry-Lookahead Adder*

The Monolithic Lookahead Adder When designing even faster adders, it is essential to get around the rippling effect of the carry that is still present in one form or another in both the carry-bypass and carry-select adders. The *carry-lookahead* principle offers a possible way to do so [Weinberger56, MacSorley61]. As stated before, the following relation holds for each bit position in an N -bit adder:

$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1} \quad (11.15)$$

The dependency between $C_{o,k}$ and $C_{o,k-1}$ can be eliminated by expanding $C_{o,k-1}$:

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2}) \quad (11.16)$$

In a fully expanded form,

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}(\dots + P_1(G_0 + P_0 C_{i,0}))) \quad (11.17)$$

with $C_{i,0}$ typically equal to 0.

This expanded relationship can be used to implement an N -bit adder. For every bit, the carry and sum outputs are independent of the previous bits. The ripple effect has thus been effectively eliminated, and the addition time should be independent of the number of bits. A block diagram of the overall composition of a carry-lookahead adder is shown in Figure 11-20.

Such a high-level model contains some hidden dependencies. When we study the detailed schematics of the adder, it becomes obvious that the constant addition time is wishful thinking and that the real delay is at least increasing linearly with the number of bits. This is illustrated in Figure 11-21, where a possible circuit implementation of Eq. (11.17) is shown for $N = 4$. Note that the circuit exploits the self-duality and the recursivity of the carry-lookahead equation to build a mirror structure, similar in style to the single-bit full adder of Figure 11-6.⁴ The large fan-in of the circuit makes it prohibitively slow for larger values of N . Implementing it with simpler gates requires multiple logic levels. In both cases, the propagation delay increases. Further-

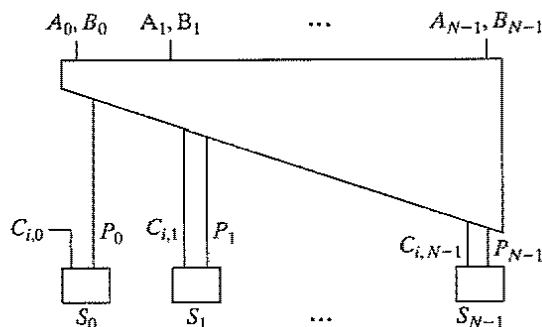


Figure 11-20 Conceptual diagram of a carry-lookahead adder.

⁴Similar to the mirror adder, this circuit requires that the *Propagate* signal be defined as $P = A + B$.

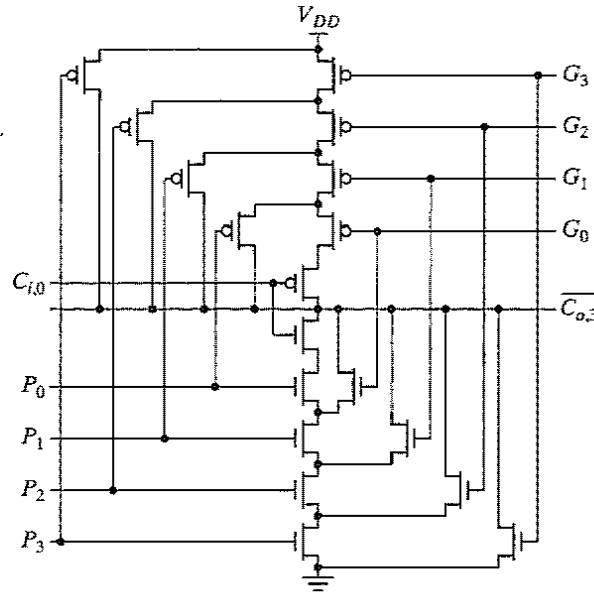


Figure 11-21 Schematic diagram of mirror implementation of four-bit lookahead adder (from [Weste85]).

more, the fan-out on some of the signals tends to grow excessively, slowing down the adder even more. For instance, the signals G_0 and P_0 appear in the expression for every one of the subsequent bits. Hence, the capacitance on these lines is substantial. Finally, the area of the implementation grows progressively with N . Therefore, the lookahead structure suggested by Eq. (11.16) is only useful for small values of N (≤ 4).

The Logarithmic Lookahead Adder—Concept For a carry-lookahead group of N bits, the transistor implementation has $N + 1$ parallel branches with up to $N + 1$ transistors in the stack. Since wide gates and large stacks display poor performance, the carry-lookahead computation has to be limited to up to two or four bits in practice. In order to build very fast adders, it is necessary to organize carry propagation and generation into recursive trees. A more effective implementation is obtained by hierarchically decomposing the carry propagation into subgroups of N bits:

$$\begin{aligned}
 C_{o,0} &= G_0 + P_0 C_{i,0} \\
 C_{o,1} &= G_1 + P_1 G_0 + P_1 P_0 C_{i,0} = (G_1 + P_1 G_0) + (P_1 P_0) C_{i,0} = G_{1:0} + P_{1:0} C_{i,0} \\
 C_{o,2} &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{i,0} = G_2 + P_2 C_{o,1} \\
 C_{o,3} &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{i,0} \\
 &= (G_3 + P_3 G_2) + (P_3 P_2) C_{o,1} = G_{3:2} + P_{3:2} C_{o,1}
 \end{aligned} \tag{11.18}$$

In Eq. (11.18), the carry-propagation process is decomposed into subgroups of two bits. $G_{i:j}$ and $P_{i:j}$ denote the generate and propagate functions, respectively, for a group of bits (from bit positions i to j). Therefore, we call them *block generate* and *propagate* signals. $G_{i:j}$ equals 1 if the group generates a carry, independent of the incoming carry. The block propagate $P_{i:j}$ is true if an incoming carry propagates through the complete group. This condition is equivalent to the carry bypass, discussed earlier. For example, $G_{3:2}$ is equal to 1 when a carry either is generated at bit position 3 or is generated at position 2 and propagated through position 3, or $G_{3:2} = G_3 + P_3G_2$. $P_{3:2}$ is true when an incoming carry propagates through both bit positions, or $P_{3:2} = P_3P_2$.

Note that the format of the new expression for the carry is equivalent to the original one, except that the generate and propagate signals are replaced with block generate and propagate signals. The notation $G_{i:j}$ and $P_{i:j}$ generalizes the original carry equations, since $G_i = G_{i:i}$ and $P_i = P_{i:i}$. Another generalization is possible by treating the generate and propagate functions as a pair $(G_{i:j}, P_{i:j})$, rather than considering them as separate functions. A new Boolean operator, called the *dot* operator ($\cdot$), can be introduced. This operator on the pairs and allows for the combination and manipulation of blocks of bits:

$$(G, P) \cdot (G', P') = (G + PG', PP') \quad (11.19)$$

Using this operator we can now decompose $(G_{3:2}, P_{3:2}) = (G_3, P_3) \cdot (G_2, P_2)$. The dot operator obeys the associative property, but it is not commutative.

Example 11.4 Ripple-Carry Adder Expressed by Using the Dot Operator

With the dot operator, a four-bit ripple carry adder can be re-written as

$$(C_{a,3}, 0) = [(G_3, P_3) \cdot (G_2, P_2) \cdot (G_1, P_1) \cdot (G_0, P_0)] \cdot (C_{i,0}, 0)$$

The associative property allows us to rewrite this function and express $C_{a,3}$ as a function of 2 group carries:

$$\begin{aligned} (G_{3:0}, P_{3:0}) &= [(G_3, P_3) \cdot (G_2, P_2)] \cdot [(G_1, P_1) \cdot (G_0, P_0)] \\ &= (G_{3:2}, P_{3:2}) \cdot (G_{1:0}, P_{1:0}) \end{aligned}$$

By exploiting the associative property of the dot operator, a tree can be constructed that effectively computes the carries at all $2^i - 1$ positions (that is, 1, 3, 7, 15, etc.) for $i = 1 \dots \log_2(N)$. The crucial advantage is that the computation of the carry at position $2^i - 1$ takes only $\log_2(N)$ steps. In other words, the output carry of an N -bit adder can be computed in $\log_2(N)$ time. This is a major improvement over the previously described adders. For example, for an adder of 64 bits, the propagation delay of a linear adder is proportional to 64. For a square-root select adder, it is reduced to 8, while, for a logarithmic adder, the proportionality constant is 6. This is illustrated in Figure 11-22, which shows the block diagram of a 16-bit logarithmic adder. The carry at position 15 is computed by combining the results of blocks (0:7) and (8:15). Each of these, in turn, is

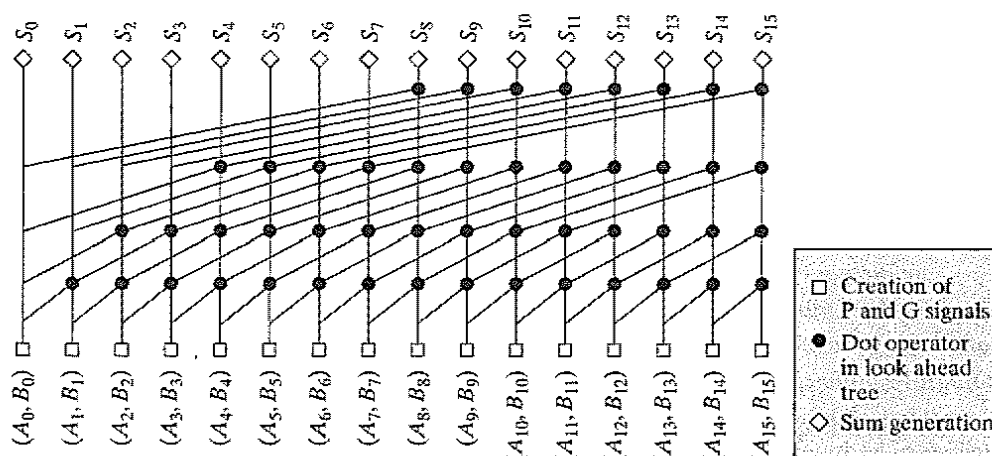


Figure 11-22: Schematic diagram for Kogge–Stone 16-bit lookahead logarithmic adder.

composed hierarchically. For instance, (0:7) is the composition of (0:3) and (4:7), while (0:3) consists of (0:1) and (2:3), etc.

Computing the carries at just the $2^i - 1$ positions is obviously not sufficient. It is necessary to derive the carry signals at the intermediate positions as well. One way to accomplish this is by replicating the tree at every bit position, as illustrated in Figure 11-22 for $N = 16$. For instance, the carry at position 6 is computed by combining the results of blocks (6:3) and (2:0). This complete structure, which frequently is referred to as a *Kogge–Stone tree* [Kogge73], is a member of the *radix-2* class of trees. Radix-2 means that the tree is binary: It combines two carry words at a time at each level of hierarchy. The total adder requires 49 complex logic gates each to implement the dot operator. In addition, 16 logic modules are needed for the generation of the propagate and generate signals at the first level (P_i and G_i), as well as 16 sum-generation gates.

Design Example—Implementing a Lookahead Adder in Dynamic Logic

The combination of carry-lookahead (CLA) techniques and dynamic logic seems to be ideal when very high performance is the ultimate goal. It is therefore useful to walk through the complete design of a dynamic CLA.

The first module generates the propagate and generate signal, as shown in Figure 11-23. The addition of a separate inverter to drive the keeper represents a small twist. This approach is beneficial in gates that drive a sizable fan-out. By decoupling its driver from the fan-out it allows for a quick disengagement of the keeper after the transition starts. The inverter that is driving successive logic gates, on the other hand, is optimized to drive a fan-out of two (for G outputs) or three (for P outputs) NMOS pull-down networks.

Each of the black dots in Figure 11-22 represents two gates that compute the block-level propagate and generate signals, as shown in Figure 11-24. Since these gates are not located at the beginning of the pipeline, the evaluation transistor (also called the *foot switch*) is optional, as discussed in Chapter 6. This approach is commonly used in dynamic datapaths. During the precharge phase, all the outputs of the domino gates are guaranteed to be low, turning off any discharge path in the succeeding domino stages. Elimination of the foot switch in any stage other than the first lowers the logical effort of the gates and speeds up

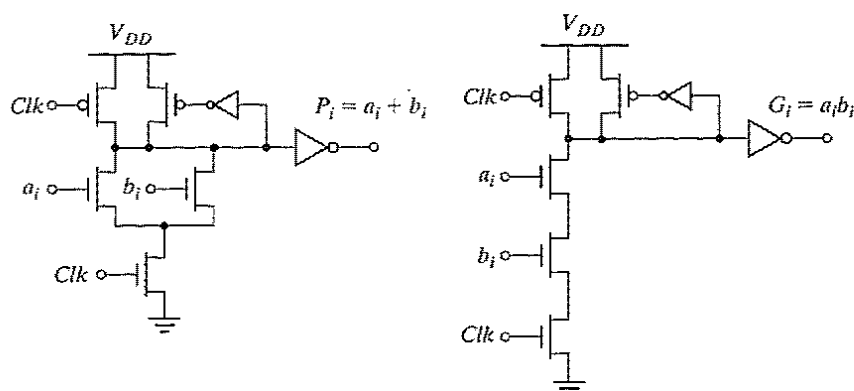


Figure 11-23 Dynamic Implementation of propagate and generate signals.

the evaluation. For example, the propagate gate in Figure 11-24 has a logical effort of two-thirds instead of unity (assuming that the inverter is symmetrical). However, there is a drawback to this method. During the precharge phase, a short-circuit current exists in all gates without a foot switch until their inputs get precharged. To avoid the short-circuit current, the clock at each stage Clk_k is delayed from the previous stage. This approach is called *clock-delayed domino* as was introduced in Chapter 10. Note that the clock is delayed by the same amount for all bit-slices per stage, thus simplifying the implementation.

By putting together seven stages of logic in a bit-sliced fashion— P - G generation, followed by six dot operators—a 64-bit adder can be constructed. The only logic stage that is missing to complete the dynamic adder design is the final sum generation. The sum generation requires an XOR function, which is not easily built in domino logic. Static XOR gates could be used, but these produce nonmonotonic transitions and thus cannot be used to drive other domino gates. This might not be a problem per se, since the sum generation typically is the final stage of the addition operation. However, the latch that follows the sum generation cannot be transparent, because this could cause a violation of the transition rule for the succeeding domino gates.

One way of implementing the sum in domino logic is through *sum selection*, in which both care for the sum are computed as $S_i^0 = a_i \oplus b_i$ and $S_i^1 = a_i \odot b_i$. The dynamic gate of Figure 11-25 is then used to select one of these possibilities, based on the incoming carry.

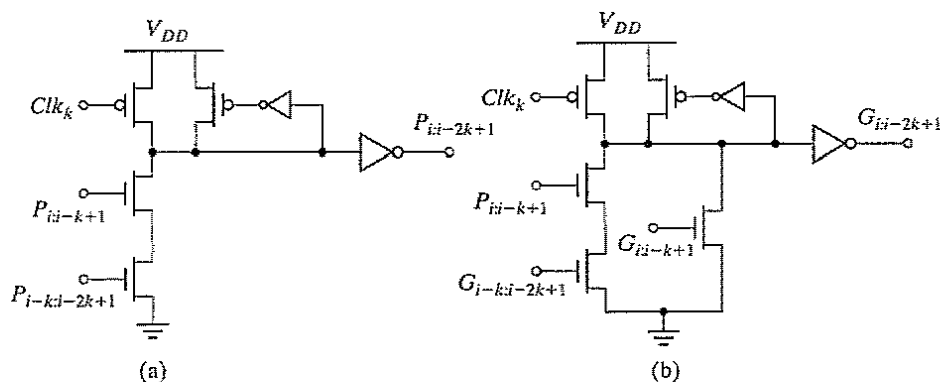


Figure 11-24 Implementation of the dot operator in dynamic logic: (a) propagate and (b) generate logic at stage k and bit position i (see Figure 11-22).

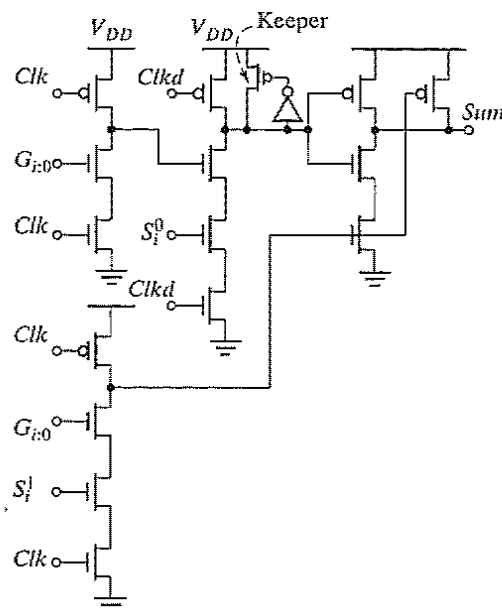


Figure 11-25 Sum select in dynamic logic.

The implementation of the multiplexer gate requires three logic levels, because no complementary carry is available in domino logic. As usual, keepers should be placed at all dynamic nodes, but the one shown in the figure is absolutely critical. The first two dynamic stages (on top) violate the dynamic logic design rules: two dynamic gates are cascaded without introducing an inverter. A glitch might happen at the output of the second gate if both gates are evaluated with the same clock. Delaying the clock of the second gate (*Ckd*) helps to address this issue, although the delayed clock presents a *hard-timing edge*—all the inputs to the second gate must have finished their transition before the clock raises.

It should be emphasized that when a designer chooses to use these design techniques, circuit robustness cannot be compromised. All clocks are subject to random skew, and the delayed clock must have enough margin to absorb the worst case skew. If *Clkd* arrives early, the adder will malfunction. However, proper sizing of the keeper may add a kind of safety net to the design. It can suppress the glitch at the output of the second stage, in case the clock arrives early. The keeper must be sized large enough to minimize this glitch, but not so large that it would compromise performance.

Another option is to implement the adder in differential domino logic, where both signal polarities are available, and the inversion problem is avoided. However, the overall design would be quite power hungry. ■

Problem 11.4 Static Adder Tree Design

Design a 16-bit carry-lookahead adder tree in static complementary CMOS. Design the lookahead tree by using inverted logic gates, avoiding the addition of inverters. Highlight the critical path of this adder. What are the logical efforts of gates along the critical path? How would you size it for minimal delay?

Logarithmic Lookahead Adder—Alternatives Designers of fast adders sometimes revert to other styles of tree structures as they trade off for area, power, or performance. We briefly discuss the Brent–Kung adder and the radix-4 adder, two of the more common alternative structures.

The Kogge–Stone tree of Figure 11-22 has some interesting properties. First, its interconnect structure is regular, which makes implementation quite easy. Furthermore, the fan-out throughout the tree is fairly constant, especially on the critical paths. The task of sizing the transistors for optimal performance is therefore simplified. At the same time, however, the replication of the carry trees to generate the intermediate carries comes at a large cost in terms of both area and power. Designers sometimes trade off some delay for area and power by choosing less complex trees. A simpler tree structure computes only the carries to the powers-of-two bit positions [Brent82], as illustrated in Figure 11-26 for $N = 16$.

The forward binary tree realizes the carry signals only at positions $2^N - 1$:

$$\begin{aligned}
 (C_{0,0}, 0) &= (G_0, P_0) \cdot (C_{i,0}, 0) \\
 (C_{0,1}, 0) &= [(G_1, P_1) \cdot (G_0, P_0)] \cdot (C_{i,0}, 0) = (G_{1:0}, P_{1:0}) \cdot (C_{i,0}, 0) \\
 (C_{0,3}, 0) &= [(G_{3:2}, P_{3:2}) \cdot (G_{1:0}, P_{1:0})] \cdot (C_{i,0}, 0) = (G_{3:0}, P_{3:0}) \cdot (C_{i,0}, 0) \\
 (C_{0,7}, 0) &= [(G_{7:4}, P_{7:4}) \cdot (G_{3:0}, P_{3:0})] \cdot (C_{i,0}, 0) = (G_{7:0}, P_{7:0}) \cdot (C_{i,0}, 0) \\
 &\dots
 \end{aligned} \tag{11.20}$$

The forward binary-tree structure is not sufficient to generate the complete set of carry bits. An *inverse binary tree* is needed to realize the other carry bits (shown in gray lines in Figure 11-26). This structure combines intermediate results to produce the remaining carry bits. It is left for the reader to verify that this structure produces the correct expressions for all carry

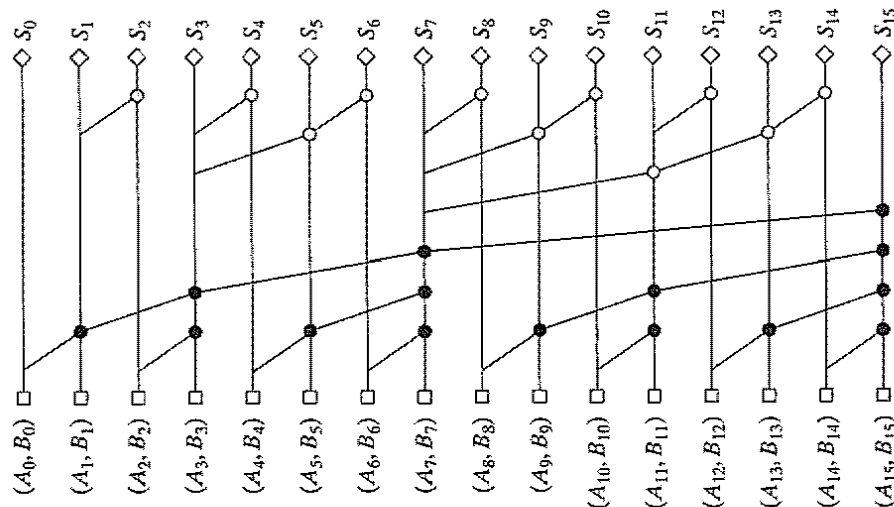


Figure 11-26 A 16-bit Brent–Kung tree. The reverse binary tree is colored gray.

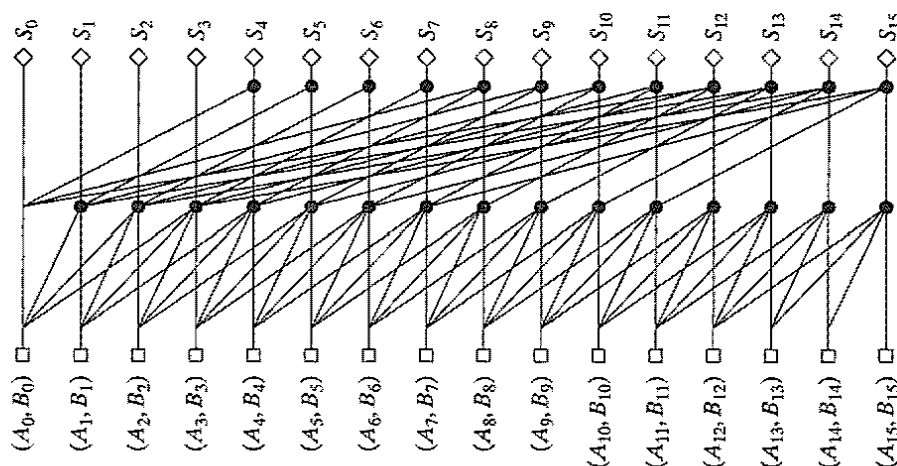


Figure 11-27 Radix-4 Kogge-Stone tree for 16-bit operands.

bits. The resulting structure, commonly called the *Brent-Kung adder*, uses 27 dot gates, or almost half of the 49 needed for a full radix-2 tree, and it needs fewer wires as well. The wiring structure is less regular, however, and fan-out varies from gate to gate, making performance optimization more difficult. Especially the fan-out of the middle node ($C_{0,7}$), which equals one sum and five dot operations for this example, is of major concern. This observation makes the Brent-Kung adder rather unsuited for very large adders (> 32 bits).

An option to reduce the depth of the tree is to combine four signals at a time at each level of the hierarchy. The resulting tree is now of class *radix-4*, because it uses building blocks of order 4 as shown in Figure 11-27. A 16-bit addition needs only two stages of carry logic. Be aware that each gate is more complex and that having less logic stages may not always result in faster operation (as we saw in Chapter 5).

Problem 11.5 Radix-4 Dot Operator in Dynamic Logic

Design the radix-4 dot operators in dynamic logic. Use the radix-2 circuits as reference. How is the sum select implemented for radix-4? Using the method of logical effort, compare the delays of radix-2 and radix-4 designs for 16-bit adders. Which one is faster?

On average, a lookahead adder is several times larger than a ripple adder, but has dramatic speed advantages for larger operands. The logarithmic behavior makes it preferable over bypass or select adders for larger values of N . The exact value of the cross point depends heavily on technology and circuit design factors.

The discussion of adders is by no means complete. Due to its impact on the performance of computational structures, the design of fast adder circuits has been the subject of many publications. It is even possible to construct adder structures with a propagation delay that is *independent of the number of bits*. Examples of those are the carry-save structures and the redundant

binary arithmetic structures [Swartzlander90]. These adders require number-encoding and -decoding steps, whose delay is a function of N . Therefore, they are only interesting when embedded in larger structures such as multipliers or high-speed signal processors.

11.4 The Multiplier

Multiplications are expensive and slow operations. The performance of many computational problems often is dominated by the speed at which a multiplication operation can be executed. This observation has, for instance, prompted the integration of complete multiplication units in state-of-the-art digital signal processors and microprocessors.

Multipliers are, in effect, complex adder arrays. Therefore, the majority of the topics discussed in the preceding section are of value in this context as well. The analysis of the multiplier gives us some further insight into how to optimize the performance (or the area) of complex circuit topologies. After a short discussion of the multiply operation, we discuss the basic array multiplier. We also discuss different approaches to partial product generation, accumulation and their final summation.

11.4.1 The Multiplier: Definitions

Consider two *unsigned* binary numbers X and Y that are M and N bits wide, respectively. To introduce the multiplication operation, it is useful to express X and Y in the binary representation

$$X = \sum_{i=0}^{M-1} X_i 2^i \quad Y = \sum_{j=0}^{N-1} Y_j 2^j \quad (11.21)$$

with $X_i, Y_j \in \{0, 1\}$. The multiplication operation is then defined as follows:

$$\begin{aligned} Z = X \times Y &= \sum_{k=0}^{M+N-1} Z_k 2^k \\ &= \left(\sum_{i=0}^{M-1} X_i 2^i \right) \left(\sum_{j=0}^{N-1} Y_j 2^j \right) = \sum_{i=0}^{M-1} \left(\sum_{j=0}^{N-1} X_i Y_j 2^{i+j} \right) \end{aligned} \quad (11.22)$$

The simplest way to perform a multiplication is to use a single two-input adder. For inputs that are M and N bits wide, the multiplication takes M cycles, using an N -bit adder. This *shift-and-add* algorithm for multiplication adds together M *partial products*. Each partial product is generated by multiplying the multiplicand with a bit of the multiplier—which, essentially, is an AND operation—and by shifting the result on the basis of the multiplier bit's position.

A faster way to implement multiplication is to resort to an approach similar to manually computing a multiplication. All the partial products are generated at the same time and organized in an array. A multioperand addition is applied to compute the final product. The approach is illustrated in Figure 11-28. This set of operations can be mapped directly into hardware. The

this format, we have to add only two partial products, but the final adder has to be able to perform subtraction as well. This type of transformation is called *Booth's recoding* [Booth51], and it reduces the number of partial products to, at most, one half. It ensures that for every two consecutive bits, at most one bit will be 1 or -1. Reducing the number of partial products is equivalent to reducing the number of additions, which leads to a speedup as well as an area reduction. Formally, this transformation is equivalent to formatting the multiplier word into a base-4 scheme, instead of the usual binary format:

$$Y = \sum_{j=0}^{(N-1)/2} Y_j 4^j \text{ with } (Y_j \in \{-2, -1, 0, 1, 2\}) \quad (11.23)$$

Note that 1010...10 represents the worst case multiplier input because it generates the most partial products (one half). While the multiplication with $\{0, 1\}$ is equivalent to an AND operation, multiplying with $\{-2, -1, 0, 1, 2\}$ requires a combination of inversion and shift logic. The encoding can be performed on the fly and requires some simple logic gates.

Having a variable-size partial-product array is not practical for multiplier design, and a *modified Booth's recoding* is most often used instead [MacSorley61]. The multiplier is partitioned into three-bit groups that overlap by one bit. Each group of three is recoded, as shown in Table 11-2, and forms one partial product. The resulting number of partial products equals half of the multiplier width. The input bits to the recoding process are the two current bits, combined with the upper bit from the next group, moving from *msb* to *lsb*.

Table 11-2 Modified Booth's recoding.

Partial Product Selection Table	
Multiplier Bits	Recoded Bits
000	0
001	+ Multiplicand
010	+ Multiplicand
011	+ 2 × Multiplicand
100	- 2 × Multiplicand
101	- Multiplicand
110	- Multiplicand
111	0

In simple terms, the modified Booth's recoding essentially examines the multiplier for strings of ones from *msb* to *lsb* and replaces them with a leading 1, and a -1 at the end of the string. For example, 011 is understood as the beginning of a string of ones and is therefore replaced by a leading 1 (or 100), while 110 is seen as the end of a string and is replaced by a -1 at the least significant position (or 0 $\bar{1}$ 0).

Example 11.5 Modified Booth's Recoding

Consider the eight-bit binary number 01111110 shown earlier. This can be divided into four overlapping groups of three bits, going from *msb* to *lsb*: 00 (1), 11 (1), 11 (1), 10 (0). Recoding by using Table 11-2 yields: 10 (2 $\times$), 00 (0 $\times$), 00 (0 $\times$), $\bar{1}$ 0 ($-2\times$), or, in combined format, 1000000 $\bar{1}$ 0. This is equivalent to the result we obtained before.

Problem 11.7 Booth's Recoder

Design the combinational logic that implements a modified Booth's recoding for a parallel multiplier, using Table 11-2. Compare its implementation in complementary and pass-transistor CMOS.

11.4.3 Partial-Product Accumulation

After the partial products are generated, they must be summed. This accumulation is essentially a multioperand addition. A straightforward way to accumulate partial products is by using a number of adders that will form an array—hence, the name, *array multiplier*. A more sophisticated procedure performs the addition in a tree format.

The Array Multiplier

The composition of an array multiplier is shown in Figure 11-30. There is a one-to-one topological correspondence between this hardware structure and the manual multiplication shown in Figure 11-28. The generation of N partial products requires $N \times M$ two-bit AND gates (in the style of Figure 11-29).⁵ Most of the area of the multiplier is devoted to the adding of the N partial products, which requires $N - 1$ M -bit adders. The shifting of the partial products for their proper alignment is performed by simple routing and does not require any logic. The overall structure can easily be compacted into a rectangle, resulting in a very efficient layout.

Due to the array organization, determining the propagation delay of this circuit is not straightforward. Consider the implementation of Figure 11-30. The partial sum adders are implemented as ripple-carry structures. Performance optimization requires that the critical timing path be identified first. This turns out to be nontrivial. In fact, a large number of paths of almost identical length can be identified. Two of those are highlighted in Figure 11-31. A closer

⁵This particular implementation does not employ Booth's recoding. Adding recoding does not substantially change the implementation. The number of adders is reduced by half, and the partial-product generation is slightly more complex.

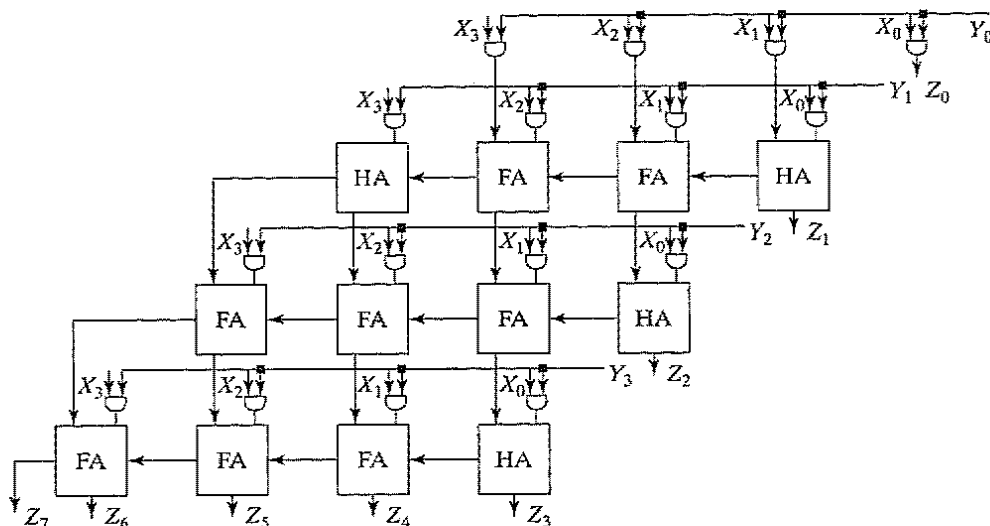


Figure 11-30 A 4×4 bit-array multiplier for unsigned numbers—composition. HA stands for a half adder, or an adder cell with only two inputs. The hardware for the generation and addition of one partial product is shaded in gray.

look at those critical paths yields an approximate expression for the propagation delay (derived here for critical path 2). We write this as

$$t_{mult} \approx [(M-1) + (N-2)]t_{carry} + (N-1)t_{sum} + t_{and} \quad (11.24)$$

where t_{carry} is the propagation delay between input and output carry, t_{sum} is the delay between the input carry and sum bit of the full adder, and t_{and} is the delay of the AND gate.

Since all critical paths have the same length, speeding up just one of them—for instance, by replacing one adder by a faster one such as a carry-select adder—does not make much sense from a design standpoint. All critical paths have to be attacked at the same time. From Eq. (11.24), it can be deduced that the minimization of t_{mult} requires the minimization of both t_{carry}

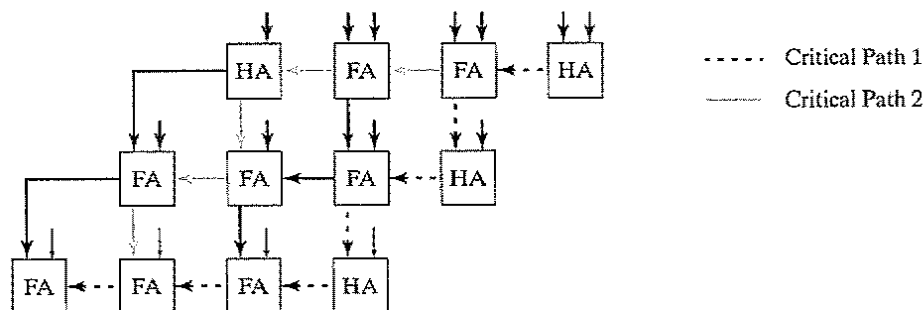


Figure 11-31 Ripple-carry based 4×4 multiplier (simplified diagram). Two of the possible critical paths are highlighted.

and t_{sum} . In this case, it could be beneficial for t_{carry} to equal t_{sum} . This contrasts with the requirements for adder cells discussed before, where a minimal t_{carry} was of prime importance. An example of a full-adder circuit with comparable t_{sum} and t_{carry} delays was shown in Figure 11-7.

Problem 11.8 Signed-Binary Multiplier

The multiplier presented in Figure 11-30 only handles unsigned numbers. Adjust the structure so that two's-complement numbers are also accepted.

Carry-Save Multiplier

Due to the large number of almost identical critical paths, increasing the performance of the structure of Figure 11-31 through transistor sizing yields marginal benefits. A more efficient realization can be obtained by noticing that the multiplication result does not change when the output carry bits are passed diagonally downwards instead of only to the right, as shown in Figure 11-32. We include an extra adder called a *vector-merging* adder to generate the final result. The resulting multiplier is called a *carry-save multiplier* [Wallace64], because the carry bits are not immediately added, but rather are “saved” for the next adder stage. In the final stage, carries and sums are merged in a fast carry-propagate (e.g., carry-lookahead) adder stage. While this structure has a slightly increased area cost (one extra adder), it has the advantage that its worst case critical path is shorter and uniquely defined, as highlighted in Figure 11-32 and is expressed as

$$t_{mult} = t_{and} + (N - 1)t_{carry} + t_{merge} \quad (11.25)$$

still assuming that $t_{add} = t_{carry}$.

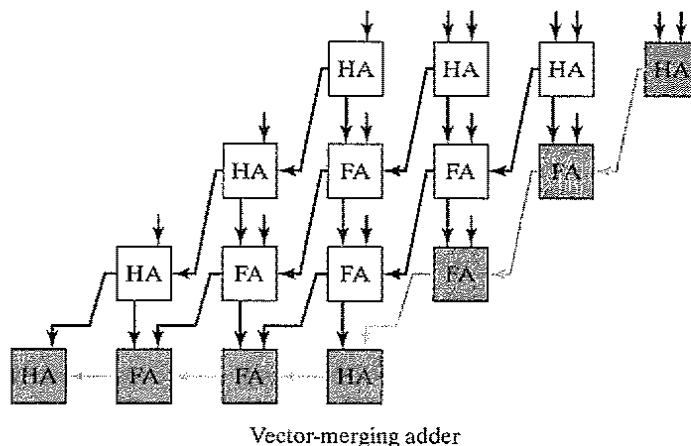


Figure 11-32 A 4×4 carry-save multiplier. The critical path is highlighted in gray.

Example 11.6 Carry-Save Multiplier

When mapping the carry-save multiplier of Figure 11-32 onto silicon, one has to take into account some other topological considerations. To ease the integration of the multiplier into the rest of the chip, it is advisable to make the outline of the module approximately rectangular. A floor plan for the carry-save multiplier that achieves this goal is shown in Figure 11-33. Observe the regularity of the topology. This makes the generation of the structure amenable to automation.

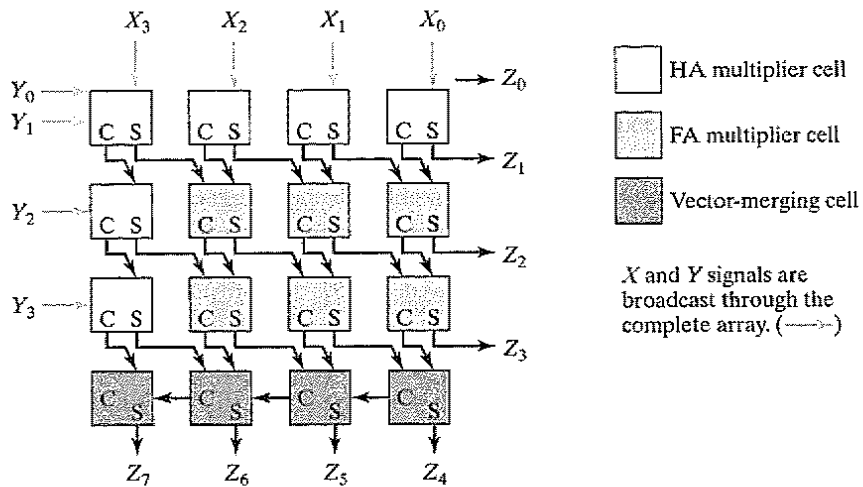


Figure 11-33 Rectangular floorplan of carry-save multiplier. Different cells are differentiated by shades of gray. X and Y signals are AND'ed before being added. The leftmost column of cells is redundant and can be eliminated.

The Tree Multiplier

The partial-sum adders can also be rearranged in a treelike fashion, reducing both the critical path and the number of adder cells needed. Consider the simple example of four partial products each of which is four bits wide, as shown in Figure 11-34a. The number of full adders needed for this operation can be reduced by observing that only column 3 in the array has to add four bits. All other columns are somewhat less complex. This is illustrated in Figure 11-34b, where the original matrix of partial products is reorganized into a tree shape to visually illustrate its varying depth. The challenge is to realize the complete matrix with a minimum depth and a minimum number of adder elements. The first type of operator that can be used to cover the array is a full adder, which takes three inputs and produces two outputs: the sum, located in the same column and the carry, located in the next one. For this reason, the FA is called a *3-2 compressor*. It is denoted by a circle covering three bits. The other operator is the half-adder, which takes two input bits in a column and produces two outputs. The HA is denoted by a circle covering two bits.

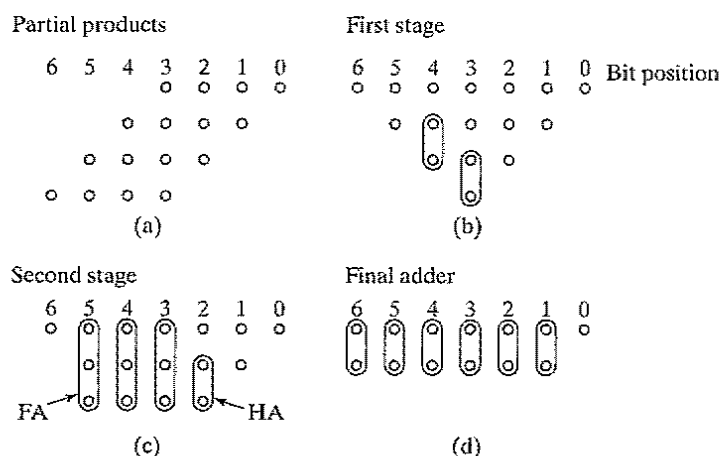


Figure 11-34 Transforming a partial-product tree (a) into a Wallace tree (b,c,d), using an iterative covering process. The example shown is for a four-bit operand.

To arrive at the minimal implementation, we iteratively cover the tree with FAs and HAs, starting from its densest part. In a first step, we introduce HAs in columns 4 and 3 (Figure 11-34b). The reduced tree is shown in Figure 11-34c. A second round of reductions creates a tree of depth 2 (Figure 11-34d). Only three FAs and three HAs are used for the reduction process, compared with six FAs and six HAs in the carry-save multiplier of Figure 11-32! The final stage consists of a simple two-input adder, for which any type of adder can be used (as discussed in the next section, “Final Addition”).

The presented structure is called the *Wallace tree multiplier* [Wallace64], and its implementation is shown in Figure 11-35. The tree multiplier realizes substantial hardware savings for larger multipliers. The propagation delay is reduced as well. In fact, it can be shown that the propagation delay through the tree is equal to $O(\log_{3/2}(N))$. While substantially faster than the carry-save structure for large multiplier word lengths, the Wallace multiplier has the disadvantage of being very irregular, which complicates the task of coming up with an efficient layout. This irregularity is visible even in the four-bit implementation of Figure 11-35.

There are numerous other ways to accumulate the partial-product tree. A number of compression circuits has been proposed in the literature, a detailed discussion of which is beyond the scope of this book. They are all based on the concept that when full adders are used as 3:2 compressors, the number of partial products is reduced by two-thirds per multiplier stage. One can even go a step further and devise a 4–2 (or higher order) compressor, such as in [Weinberger81, Santoro89]. In fact, many of today’s high-performance multipliers do just that.

11.4.4 Final Addition

The final step for completing the multiplication is to combine the result in the final adder. Performance of this “vector-merging” operation is of key importance. The choice of the adder style depends on the structure of the accumulation array. A carry-lookahead adder is the preferable

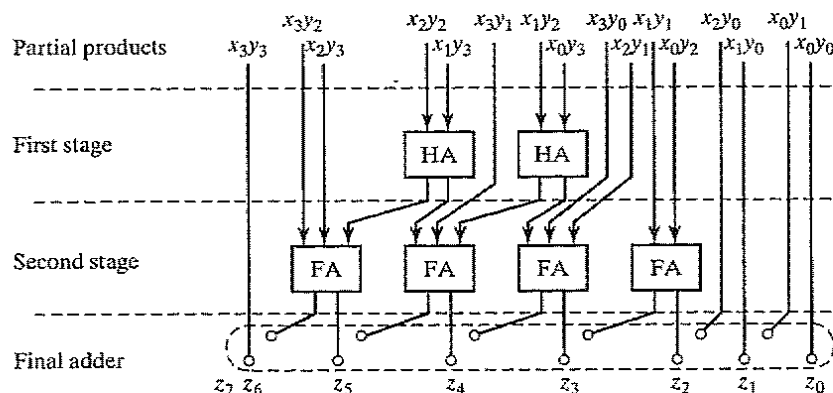


Figure 11-35 Wallace tree for four-bit multiplier.

option if all input bits to the adder arrive at the same time, as it yields the smallest possible delay. This is the case if a pipeline stage is placed right before the final addition. Pipelining is a technique frequently used in high-performance multipliers. In nonpipelined multipliers, the arrival-time profile of the inputs to the final adder is quite uneven due to the varying logic depths of the multiplier tree. Under these circumstances, other adder topologies, such as carry select, often yield performance numbers similar to lookahead at a substantially reduced hardware cost [Oklobdzija01].

11.4.5 Multiplier Summary

All the presented techniques can be combined to yield multipliers with extremely high performance. For instance, a 54×54 multiplier can achieve a propagation delay of only 4.4 ns in a 0.25- μm CMOS technology by combining Booth encoding with a Wallace tree by using 4–2 compression in pass-transistor implementation and by using a mixed carry-select, carry-lookahead topology for the final adder [Ohkubo95]. More information on these multipliers (and others) can be found in the references [e.g., Swartzlander90, Oklobdzija01].

11.5 The Shifter

The shift operation is another essential arithmetic operation that requires adequate hardware support. It is used extensively in floating-point units, scalars, and multiplications by constant numbers. The latter can be implemented as a combination of add and shift operations. Shifting a data word left or right over a constant amount is a trivial hardware operation and is implemented by the appropriate signal wiring. A programmable shifter, on the other hand, is more complex and requires active circuitry. In essence, such a shifter is nothing less than an intricate multiplexer circuit. A simple one-bit left–right shifter is shown in Figure 11-36. Depending on the control signals, the input word is either shifted left or right, or else it remains unchanged. Multi-bit shifters can be built by cascading a number of these units. This approach rapidly becomes complex, unwieldy, and ultimately too slow for larger shift values. Therefore, a more structured

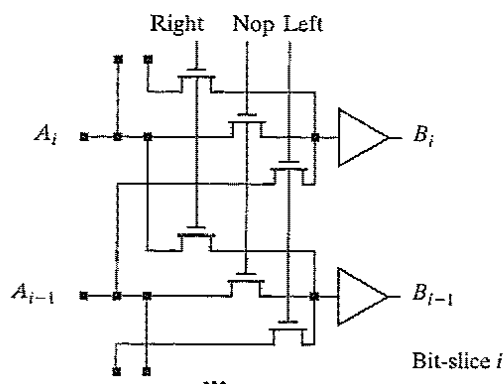


Figure 11-36 One-bit (left-right) programmable shifter. The data passes undisturbed under the nop condition.

approach is advisable. Next, we discuss two commonly used shift structures, the *barrel shifter* and the *logarithmic shifter*.

11.5.1 Barrel Shifter

The structure of a barrel shifter is shown in Figure 11-37. It consists of an array of transistors, in which the number of rows equals the word length of the data, and the number of columns equals the maximum shift width. In this case, both are set equal to four. The control wires are routed diagonally through the array. A major advantage of this shifter is that the signal has to pass through at most one transmission gate. In other words, the propagation delay is theoretically constant and independent of the shift value or shifter size. This is not true in reality, however, because the capacitance at the input of the buffers rises linearly with the maximum shift width.

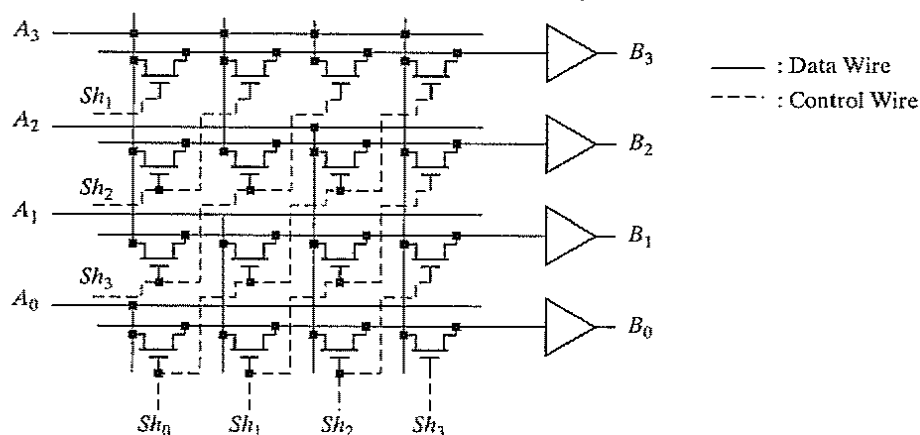


Figure 11-37 Barrel shifter with a programmable shift width from zero to three bits to the right. The structure supports automatic repetition of the sign bit (A_3), also called *sign-bit extension*.

An important property of this circuit is that the layout size is not dominated by the active transistors as in the case of all other arithmetic circuits, but by the number of wires running through the cell. More specifically, the size of the cell is bounded by the pitch of the metal wires!

Another important consideration when selecting a shifter is the format in which the shift value must be presented. From the schematic diagram of Figure 11-37, we see that the barrel shifter needs a control wire for every shift bit. For example, a four-bit shifter needs four control signals. To shift over three bits, the signals $Sh_3:Sh_0$ take on the value 1000. Only one of the signals is high. In a processor, the required shift value normally comes in an encoded binary format, which is substantially more compact. For instance, the encoded control word needs only two control signals and is represented as 11 for a shift over three bits. To translate the latter representation into the former (with only one bit high), an extra module called a *decoder* is required. (Decoders are treated in detail in Chapter 12.)

Problem 11.9 Two's Complement Shifter

Explain why the shifter shown in Figure 11-37 implements a two's complement shift.

11.5.2 Logarithmic Shifter

While the barrel shifter implements the whole shifter as a single array of pass transistors, the logarithmic shifter uses a staged approach. The total shift value is decomposed into shifts over powers of two. A shifter with a maximum shift width of M consists of a $\log_2 M$ stages, where the i th stage either shifts over 2^i or passes the data unchanged. An example of a shifter with a maximum shift value of seven bits is shown in Figure 11-38. For instance, to shift over five bits, the first stage is set to shift mode, the second to pass mode, and the last stage again to shift. Notice that the control word for this shifter is already encoded, and no separate decoder is required.

The speed of the logarithmic shifter depends on the shift width in a logarithmic way, since an M -bit shifter requires $\log_2 M$ stages. Furthermore, the series connection of pass transistors slows the shifter down for larger shift values. A careful introduction of intermediate buffers is therefore necessary, as discussed in Chapter 6.

In general, we conclude that a barrel shifter is appropriate for smaller shifters. For larger shift values, the logarithmic shifter becomes more effective, in terms of both area and speed. Furthermore, the logarithmic shifter is easily parameterized, allowing for automatic generation. The most important concept of this section is that the exploitation of regularity in an arithmetic operator can lead to dense and high-speed circuit implementations.

11.6 Other Arithmetic Operators

In the previous sections, we only discussed a subset of the large number of arithmetic circuits required in the design of microprocessors and signal processors. Besides adders, multipliers, and shifters, others operators such as comparators, dividers, counters, and goniometric operators (sine, cosine, tangent) are often needed. A full analysis of these circuits is beyond the scope of

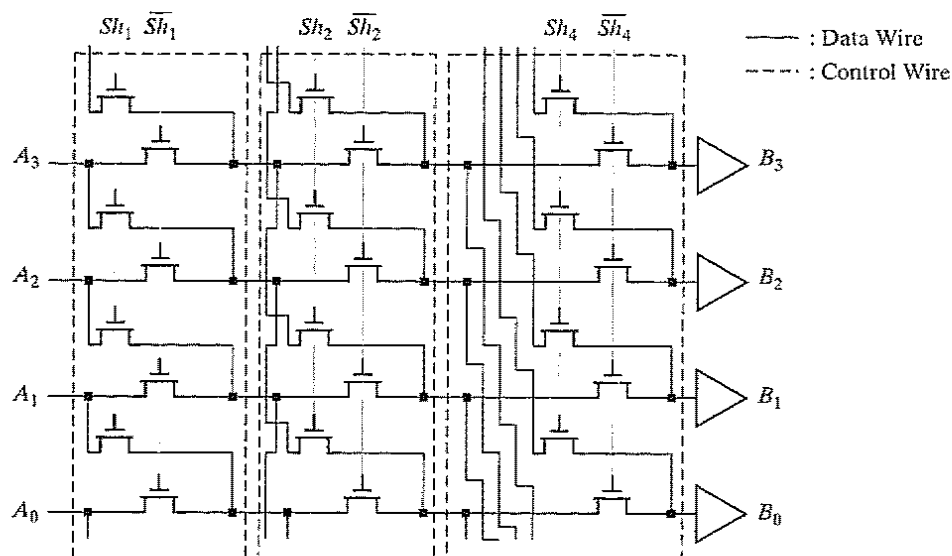


Figure 11-38 Logarithmic shifter with maximal shift width of seven bits to the right. (Only the four least significant bits are shown.)

this book. We refer the interested reader to some of the excellent references available on the topic (e.g., [Swartzlander90], [Oklobdzija01]).

The reader should be aware that most of the design ideas introduced in this chapter apply to these other operators as well. For instance, comparators can be devised with a linear, square root, and logarithmic dependence on the number of bits. In fact, some operators are simple derivatives of the adder or multiplier structures presented earlier. For example, Figure 11-39 shows how a two's complement subtractor can be realized by combining a two's complement adder with an extra inversion stage, or how a subtractor can be used to implement $A \geq B$.

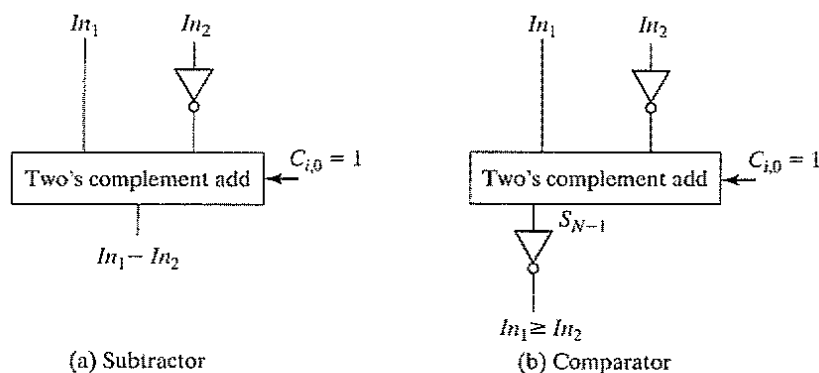


Figure 11-39 Arithmetic structures derived from a full adder.

Problem 11.10 Comparator

Derive a logic diagram for a comparator that implements the following logic functions: $\geq$, $=$, and $\leq$.

Case Study—Design of an Arithmetic-Logic Unit (ALU)

The core of any microprocessor or microcontroller is the arithmetic-logic unit (or ALU). The ALU combines addition and subtraction with other operations, such as shifting and bitwise logic operations (AND, OR, and XOR).

An ALU taken from a 64-bit high-end microprocessor is shown in Figure 11-40 [Mathew01.] It consists of two levels of wide multiplexers, a 64-bit adder, a logic unit, an operator-merging multiplexer and the write-back bus driver. The first pair of 9:1 multiplexers selects the from nine different input sources, three of which are from register files and caches. The other six are bypass paths that come directly from the six ALUs—the microprocessor in question can issue six integer instructions in one cycle. One of the bypass loops actually feeds the output of the ALU back to its own input, creating a single cycle loop that defines the critical path. The second 5:1 multiplexer on the A input performs a partial shifting of the operand, while the 2:1 multiplexer on the B input simply inverts the operand to implement subtraction. The adder executes two's complement addition or subtraction on the operands A and B . Two's complement subtraction is performed by inverting all the bits of the operand B (using the 2:1 MUX) and by setting the carry-in to one. The sum-selection block not only implements the sum selection, but also merges the outputs of arithmetic and logic units. Finally, a strong buffer drives the loop-back bus, which presents a large load to the ALU. For example, for a six-issue Intel/HP Itanium processor [Fetzer02], the bus has to connect to all six units, translating into more than 2 mm of wire length in a 0.18- μm technology. When added to the load from the register file, it presents over 0.5 pF of capacitive load.

An ALU represents a typical example of a bit-sliced design. Each slice in each block is pitch matched, which minimizes the vertical routing. A floor plan of the 64-bit ALU is shown in Figure 11-41. Intercell routing is done horizontally in metal-3, except for the long horizontal wires between adder stages 1 and 2 and 2 and 3, which are laid out in a combination of metal-3 and metal-4. For example, if the adder is implemented as a radix-4 CLA, the longest wire after the second PG stage crosses 48 cells.

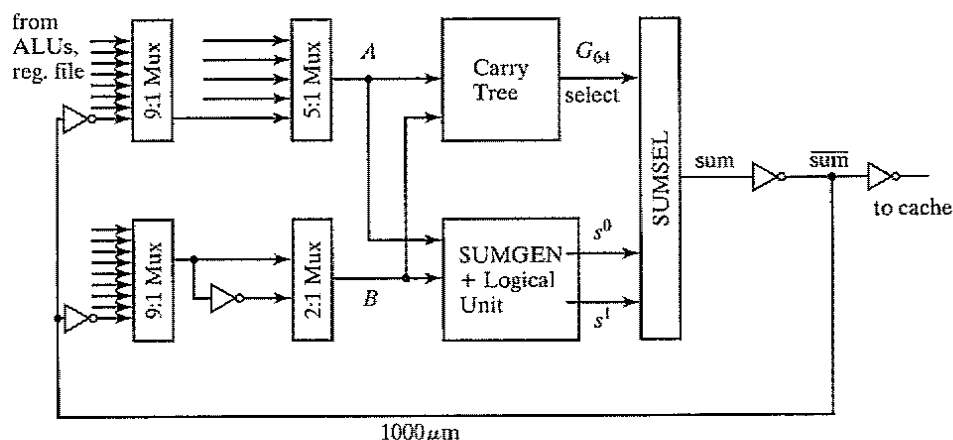


Figure 11-40 A 64-bit arithmetic-logic unit.

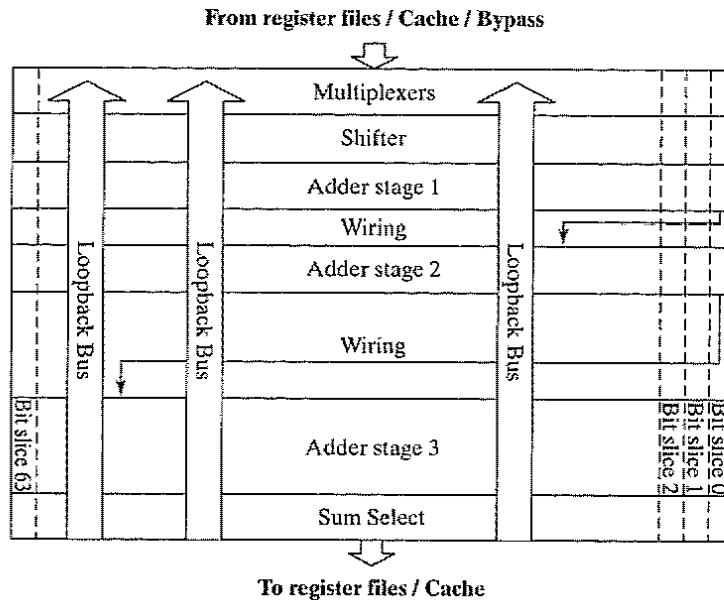


Figure 11-41 Floor plan of a 64-bit ALU.

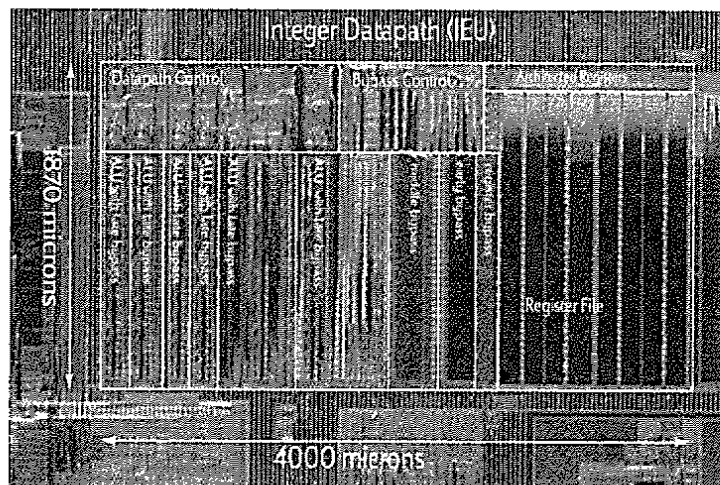


Figure 11-42 Microphotograph of the Itanium processor, showing bit-sliced design of six ALUs together with bypasses, register file, and control logic. (Courtesy of Intel.)

The loop-back wiring is routed in the top-level metal (e.g., metal-5). In case of a multiple-issue microprocessor, these busses span across all ALUs, as can be seen in Figure 11-42. The number of loop-back busses frequently determines the bit-pitch in multiissue ALUs. In our example, nine 64-bit-wide busses have to cross back over the ALU. To avoid needing to add extra wiring space, the bit width is set to nine

bus-wire pitches. Since these wires are long; they are designed with double width and at least double spacing to reduce their resistance and capacitance. ■

11.7 Power and Speed Trade-Offs in Datapath Structures*

In the preceding discussion on adders, multipliers, and shifters, we mainly explored the trade-off between speed and area and ignored power considerations. In this section, we briefly analyze the third dimension of the design exploration space. Since most of the approaches to minimize power were already introduced in Chapter 6, the discussion that follows serves mostly as an illustration of the concepts advanced there.

Typical digital designs are either *latency* or *throughput* constrained. A latency-constrained design has to finish computation by a given deadline. Interactive communication and gaming are examples of such. Throughput-constrained designs, on the other hand, must maintain a required data throughput. A 1000BaseT Gigabit Ethernet connection has to maintain a constant throughput of 1 Gigabit/s. The architectural optimization techniques that are available to the designer differ based on the design constraints. Pipelining or parallelization, for instance, works effectively for the throughput-constrained scenario, but it may not be applicable to the latency-constrained case. For example, a throughput of 1 Gb/s over copper wires is achieved in Gigabit Ethernet by processing four 250 Mb/s streams of data in parallel.

With a fixed architecture of the datapath, speed, area, and power can be traded off through the choice of the supply voltage(s), transistor thresholds, and device sizes. This opens the door for a large variety of power minimization techniques, which are summarized in Table 11-3. They are classified as follows:

- **Enable Time**—Some design techniques are implemented (or enabled) at *design time*. Transistor widths and lengths, for instance, are fixed at the time of design. Supply voltage and transistor thresholds, on the other hand, can be either assigned statically during the design phase, or changed dynamically at *run time*. Other techniques primarily address the time that a function or module in a digital design is in idle mode (or standby). It is only logical to require that the power dissipation of a module in *sleep mode* should be absolutely minimal.

Table 11-3 Power minimization techniques.

	Constant Throughput/Latency		Variable Throughput/Latency
	Design Time	Sleep Mode	Run Time
Active	Lower V_{DD} , Multi- V_{DD} , Transistor Sizing Logic Optimizations	Clock gating	Dynamic voltage scaling
Leakage	Multi- V_{th} + Active Techniques	Sleep transistors, Variable V_{th}	Variable V_{th} + Active Techniques

- **Targeted Dissipation Source**—Another classification of the power-management techniques concerns the source of power dissipation they address: active (dynamic) power or leakage (static) power. Lowering the supply voltage, for example, is a very attractive technique: It not only reduces the energy consumed per transition in a quadratic way—albeit at the expense of performance—but also reduces the leakage current. On the other hand, increasing the threshold voltage mainly impacts the leakage component.

The sleep mode of operation deserves some special attention. If a digital block still receives a clock while in idle mode, its clock distribution network, together with the attached flip-flops, continues to consume energy, even while no computation is performed. Recall that, typically, one third of total energy of a digital system is consumed in clock distribution network. A common method to reduce power in idle mode is the *clock gating* technique introduced in Chapter 10. In this approach, the main clock connection to a module is turned off (or *gated*) whenever the block is idle. However, clock gating does not reduce the leakage power of the idle block. More complicated schemes to lower the standby power have to be used, such as increasing the transistor thresholds or switching off the power rails.

In the following sections, we discuss each of these design-time and run-time techniques in detail.

11.7.1 Design-Time Power-Reduction Techniques

Reducing the Supply Voltage

A reduction in supply voltage results in quadratic power savings and thus is the most attractive approach. On the negative side, the delay of CMOS gates increases inversely with supply voltage. At the datapath level, this loss of performance can be compensated for by other means, such as logical and architectural optimizations. For example, a ripple-carry adder can be replaced by a faster structure, such as a lookahead adder. The latter implementation is larger and more complex, which translates into a larger physical and switching capacitance. This is more than offset by the fact that the faster adder can run at a lower supply voltage for the same performance.

The trade-off between ripple and lookahead operates at the logical level. Similarly, and even more effectively, architectural optimizations can be employed to compensate for the effect of a reduced V_{DD} , as illustrated in Example 11.7.

Example 11.7 Minimizing the Power Consumption by Using Parallelism

To illustrate how architectural techniques can be used to compensate for reduced speed, a simple eight-bit datapath consisting of an adder and a comparator is analyzed, assuming a 2- μm CMOS technology [Chandrakasan92]. As shown in Figure 11-43, inputs A and B are added, and the result is compared with input C . Assume that the worst case delay through the adder, comparator, and latch is approximately 25 ns at a supply voltage of 5 V. At best, the system can be clocked with a clock period of $T = 25$ ns. When required to run at this maximum possible throughput, it is clear that the operating

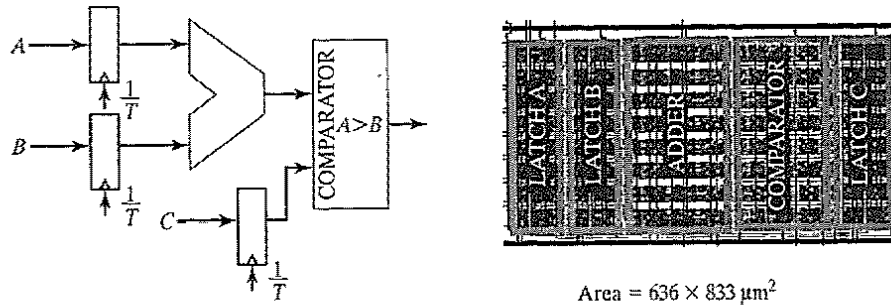


Figure 11-43 A simple datapath with corresponding layout.

voltage cannot be reduced any further because no extra delay can be tolerated. We use this topology as the reference datapath for our study and present power improvement numbers with respect to this reference. The average power consumed by the reference datapath is given by

$$P_{ref} = C_{ref} V_{ref}^2 f_{ref} \quad (11.26)$$

where C_{ref} is the total effective capacitance being switched per clock cycle. The effective capacitance can be determined by averaging the energy over a sequence of random input patterns with a uniform distribution.

One way to maintain throughput while reducing the supply voltage is to utilize a parallel architecture. As shown in Figure 11-44, two identical adder-comparator datapaths connect in parallel, allowing each unit to work at half the original rate while maintaining the original throughput. Since the speed requirements for the adder, comparator, and latch have decreased from 25 ns to 50 ns, the voltage can be dropped from 5 V to 2.9 V—the voltage that doubles the delay. While the datapath capacitance has increased by a factor of two, the operating frequency has correspondingly decreased by a factor of two. Unfortunately, there also is a slight increase in the total effective capacitance due to the extra routing and data multiplexing. This results in an increased capacitance by a factor of 2.15. The power for the parallel datapath is thus given by

$$P_{par} = C_{par} V_{par}^2 f_{par} = (2.15 C_{ref}) (0.58 V_{ref})^2 \left(\frac{f_{ref}}{2} \right) \approx 0.36 P_{ref} \quad (11.27)$$

The approach presented *trades off area for power*, as the resulting area is approximately 3.4 times larger than the original design. This technique is only applicable when the design is not area constrained. Furthermore, parallelism introduces extra routing overhead, which might cause additional dissipation. Careful optimization is needed to minimize this overhead.

Parallelism is not the only way to compensate for the loss in performance. Other architectural approaches, such as the use of pipelining, can accomplish the same goal. The most impor-

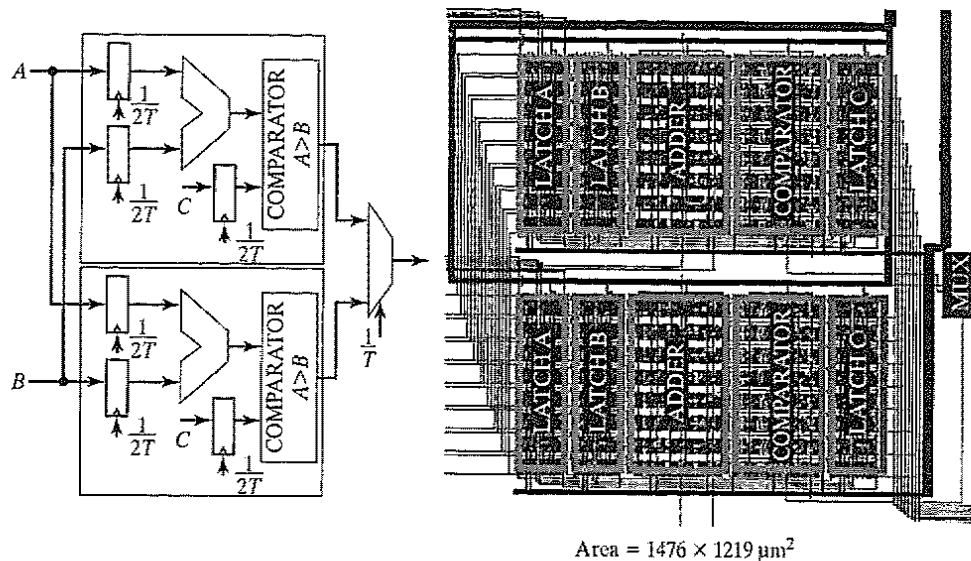


Figure 11-44 Parallel implementation of the simple datapath.

tant message in the preceding analysis is that if power dissipation is the prime concern, dropping the supply voltage is the most effective means to achieve that goal. The subsequent loss in performance can be compensated for, if necessary, by an increase in area. Within certain bounds, this is acceptable, since area is no longer the compelling issue it used to be due to the dramatic increase in integration levels of the last decade.

Problem 11.11 Reducing the Supply Voltage, Using Pipelining

A pipeline stage is introduced between the adder and the comparator of the reference datapath of Figure 11-44. You may assume that this roughly halves the propagation delay of the logic, while it increases the capacitance by 15%. Obviously, an extra pipeline register is needed on input C as well. Determine how much power can be saved by this approach, given that the throughput has to remain constant compared with the reference datapath. Comment on the area overhead.

Using Multiple Supply Voltages

Reducing the supply voltage is fairly straightforward, but it may not be optimal. Reduced supply evenly lowers the power dissipation of all logic gates, while evenly increasing their delay. A better approach is to selectively decrease the supply voltage on some of the gates:

- those which correspond to fast paths and finish the computation early, and
- those with gates that drive large capacitances, have the largest benefit for the same delay increment.

This approach, however requires the use of more than one supply voltage. Multiple supply voltages are already employed frequently in today's ICs. A separate supply voltage is provided for the I/O for compatibility reasons, where the I/O ring is designed with transistors with thicker gate oxides to sustain higher voltages. The logic core is powered from lower voltage supplies and uses transistors with thinner oxides. This approach can be extended to lower the power dissipation of a circuit. For instance, every module could select the most appropriate voltage from two (or more) supply options. Even more extreme, multiple voltages can be assigned on a gate-by-gate basis.

Module-Level Voltage Selection Consider, for instance, the digital system shown in Figure 11-45, which consists of a datapath block with a critical path of 10 ns and a control block with a much shorter critical path of 4 ns, operating from the same supply voltage of 2.5 V. Also, assume that the datapath block has a fixed latency and throughput and that no architectural transformation can be applied to lower its supply. Since the control block finishes early (in other words, it has *timing slack*), its supply voltage can be lowered. A reduction to $V_{DDL} = 1$ V increases its critical path delay to 10 ns, and lowers its power dissipation by more than five times.⁶ We effectively exploit the discrepancy in the critical-path length of the various modules (called the *slack*) to selectively lower the power consumption of the modules with the larger slack.

When combining multiple supply voltages on a die, *level converters* are required whenever a module at the lower supply has to drive a gate at the higher voltage. If a gate supplied by V_{DDL} drives a gate at V_{DDH} , the PMOS transistor never turns off, resulting in static current and reduced output swing as illustrates in Figure 11-46. A level conversion performed at the boundaries of supply voltage domains prevents these problems. An asynchronous level converter, based on the DVSL template (Chapter 6) and similar to the low-swing signalling gate of Figure 9-32, is shown in Figure 11-46. The cross-coupled PMOS transistors perform the level conversion by using positive feedback. The delay of this level converter is quite sensitive to transistor-sizing and supply-voltage issues. The NMOS transistors operate with a reduced overdrive, $V_{DDL} - V_{Th}$, compared with the PMOS devices. They have to be made large to be able to overpower the positive feedback. For a low value of V_{DDL} , the delay can become very long. Due to

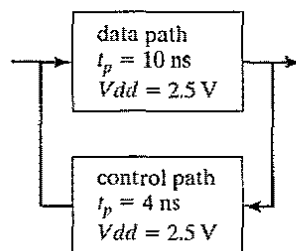


Figure 11-45 Design with diverging critical-path lengths.

⁶This uses the simplifying assumption that the propagation delay is inversely proportional to the supply voltage.

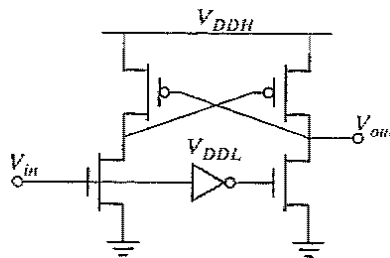


Figure 11-46 Level converter.

the reduced overdrive, the circuit is also very sensitive to variations in the supply voltage. Note that level converters are not needed for a step-down change in voltage.

The overhead of the level conversion can be somewhat mitigated by observing that most conversions are performed at a register boundary. For instance, the control inputs to the datapath block of Figure 11-45 are commonly sampled in a register. A practical way to perform the level conversion is, then, to embed it inside the register. A level-converting register is shown in Figure 11-47. It is a conventional transmission gate implementation of a master-slave latch pair, where a cross-coupled PMOS pair is embedded in the slave latch to perform level conversion.

Multiple Supplies Inside a Block The same approach can be applied at much smaller granularity by individually setting the supply voltage for each cell inside a block [Usami95, Hamada01]. Examining the histogram of the critical-path delays for a typical digital block reveals that only a few paths are critical or near critical and that many paths have much shorter delays. The shorter paths essentially waste energy, as there is no reward for finishing early. For each of these paths, the supply voltage could be lowered to the optimum level. Minimum energy consumption would be achieved if all paths become critical. However, this is not easily achievable, as many logic gates are shared between different paths, and it is impractical to generate and

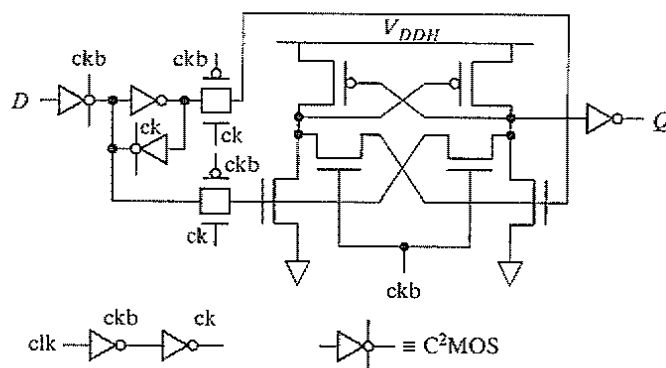


Figure 11-47 Level-converting register. Shaded gates are supplied from V_{DDL} [Usami95].

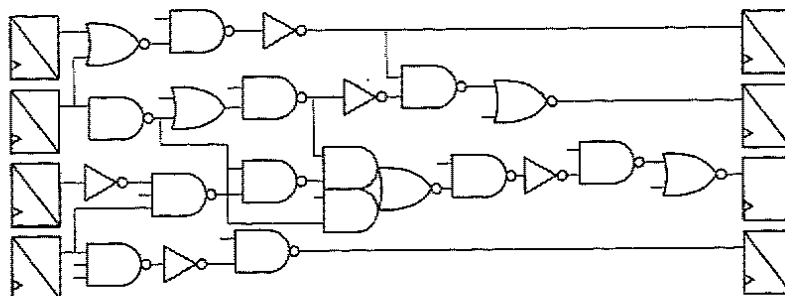


Figure 11-48 Dual-supply design using clustered voltage scaling. Each path starts with a high supply and switches to a low supply (gray logic gates) if there is a delay slack. Level conversion is performed in the registers.

distribute a wide range of supply voltages. A more practical implementation, employing only two supplies, is shown in Figure 11-48. Using a *clustered voltage-scaling* technique, each path starts with the high supply voltage and switches to the low supply when delay slack is available. Level conversion is performed in the registers at the end of the paths, using circuits such as the one introduced in Figure 11-47. Note that the level conversion is necessary only if the logic section that follows cannot run entirely off the low supply.

The impact of this approach is illustrated in Figure 11-49, which plots the path-length distribution of a typical logic block for single and dual supplies. When a single supply is used, a large fraction of the paths is substantially shorter than critical. Adding the second supply shifts the delay distribution closer to the target delay, making many more paths critical. If the design has very few critical paths and most of paths finish early, energy savings would be large. On the other hand, if most of the paths inside a block are critical, introducing the second supply does not yield any energy savings. You may wonder if adding a third or even a fourth supply voltage may yield even greater benefits to equalize the larger delay spreads. Unfortunately, a number of studies have shown that for typical path-delay distributions, adding more supplies only yields marginal savings [Hamada01, Augsburg02].

Similarly, one may ask whether this dual-supply technique yields a larger benefit, compared with a uniform reduction the supply. In using clustered voltage scaling, the dual-supply approach is **more effective when large switched capacitances are concentrated toward the**

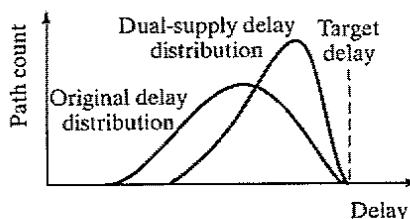


Figure 11-49 Typical path-delay distribution before and after applying the second supply.

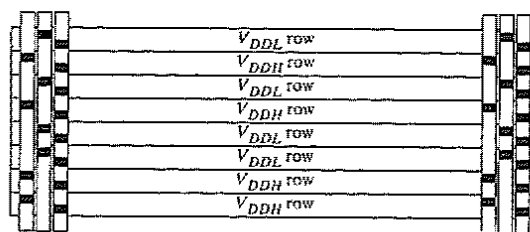


Figure 11-50 Layout of a dual-supply logic block where all cells at different supplies are placed in different rows.

end of the block, such as in buffer chains [Stojanovic02]. This observation is in accordance with the concept of low-swing busses, introduced in Chapter 9. A bus typically presents the largest capacitance, and it is more advantageous to lower the supply voltage on its driver than that on any other logic gate. Since it has the largest capacitance, the payoff in power savings is the largest for the same increase in delay.

Distributing multiple supply voltages on a die complicates the design of the power-distribution network and tends to tax the traditional place-and-route tools. As we saw many times before, a structured approach can help to minimize the impact. One simple option is to place cells with different supplies into different rows of a standard cell layout, as shown in Figure 11-50. The second supply can be brought in from outside the chip, or it can be generated on the die, using an internal DC–DC converter. Step-down switching DC–DC converters have a conversion efficiency of well over 90% and yield only a minimal overhead. Still the V_{DDL} distribution network has to meet all necessary design requirements, such as decoupling to minimize the voltage variations and immunity to electromigration.

Using Multiple Device Thresholds

The use of devices with multiple thresholds offers another way of trading off speed for power. Most sub-0.25- μm CMOS technologies offer two types of n - and p -type transistors, with thresholds differing by about 100 mV. This higher threshold device features a leakage current that is about one order of magnitude lower than that of the lower threshold transistor, at the expense of a ~30% reduction in active current. Therefore, the low-threshold devices are preferably used in timing-critical paths, while the high thresholds are used anywhere else. The assignment can be done on a per-cell basis, rather than per individual transistor [Wei99, Kato00]. Note that the use of multiple thresholds does not require level converters or any other special circuits, as shown Figure 11-51. Clustering of the logic is not required either; as many gates as possible should be converted to high threshold until the timing slack is completely consumed. A careful assignment of the thresholds can reduce the leakage power by as much as 80%–90%.

While the multiple-threshold approach primarily addresses the leakage current, it yields a small reduction in active power as an additional benefit. This is primarily due to a reduced gate-channel capacitance in the off state and a small reduction in signal swing on the internal nodes of a gate ($V_{DD} - V_{ThH}$). Such a reduction is partially offset by increased source and drain junction sidewall capacitances. The overall active power reduction turns out to be only about 4%.

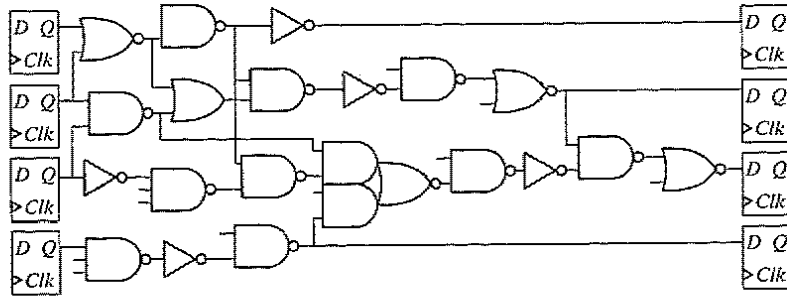


Figure 11-51 Use of dual-threshold gates. Gray-shaded gates use low thresholds and are employed only in critical paths.

Reducing Switching Capacitance through Transistor Sizing

The input capacitance of a complementary CMOS gate is directly proportional to its size and, therefore, also to its speed. In Chapter 6, we examined the optimal sizing of gates and found that each gate in a logic path should be sized to have an effective fan-out of approximately 4 to achieve the minimum delay for that path. An interesting question is how to size a circuit for minimum energy when the allowable delay is longer than minimal?

We know that for an inverter chain with a given load and number of stages, the minimum delay is achieved when the size of each inverter in the chain is the geometric mean of its neighbors:

$$s_i^2 = s_{i+1}s_{i-1} \quad (11.28)$$

When the minimum delay that is achieved by this sizing is below the desired delay, an optimization problem can be formulated that minimizes the switching capacitance under delay constraints. One option is to reduce the number of stages and increase the tapering factor, as suggested in Chapter 9. Even better, an analytical solution exists for this problem [Ma94], which establishes that the optimal approach is to adjust the tapering factor at each stage according to the following equation:

$$s_i^2 = \frac{s_{i+1}s_{i-1}}{1 + \mu s_{i-1}} \quad (11.29)$$

The parameter μ is a nonnegative number that depends on the amount of timing slack available (with respect to the chain sized for the minimum delay). This solution is intuitively clear—the last stages in the inverter chain are the largest; hence, they are the prime candidates for downsizing. Since downsizing any of the inverters in the chain by a given percentage causes the same delay increment. Hence, we rather do it at the stage where we get the largest impact, which is the largest one.

The same principle applies to a logic path. When it is optimized for minimum delay, the delay of each stage is the same (while the intrinsic delays of gates may differ). Therefore,

the delay of the largest energy consuming gates should be increased first. This idea can be extrapolated to the energy–delay optimization of a general combinational logic block. However, applying it in practice is not trivial. Downsizing one path affects the delay of all paths that share logic gates with it, making it difficult to isolate and optimize one particular path. In addition, many paths in a general combinational block are reconvergent.

Example 11.8 Energy-Delay Trade-Offs in Adder Design

Let us examine the energy-delay optimization of the 64-bit Kogge–Stone tree adder. There are many paths through an adder, and not all of these paths are balanced. A crucial question is how to identify the paths for resizing or initial sizing. One option is to select all the paths in the adder equal to the critical path. Since the paths through an adder roughly correspond to different bit slices, we allocate each gate in the adder to a bit slice. There are 64 bit slices, and a total of nine stages of logic. This partition works well for Figure 11-52a shows the resulting energy map for the minimum delay. It can be seen that the adder consumes the largest energy along the longest carry paths. Figure 11-52b shows the energy profile of the same adder, this time resized, allowing for a 10% delay increase. The energy dissipation is reduced by 54%. In contrast, a dual-supply solution saves only 27%, while a single reduced supply yields 22% savings [Stojanovic02]. In summary, **sizing is an effective power-reduction method for datapaths, where the majority of energy is consumed inside the block, rather than in driving the external load.**

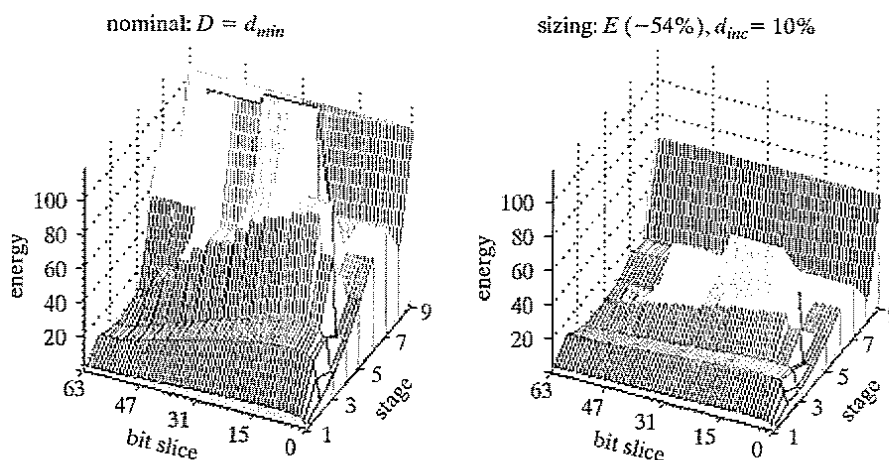


Figure 11-52 Energy profiles of a 64-bit adder: (a) sized for minimum delay, (b) sized for 10% delay increment and minimized energy.

Reducing Switching Capacitance through Logic and Architecture Optimizations

As the effective capacitance is the product of the physical capacitance and the switching activity, minimization of both factors is recommended. A wide variety of logic and architectural optimizations exist that reduce the activity at no expense in performance. Some of those were already introduced in Chapter 6. For the sake of brevity, we only show a small number of representative optimizations. (Refer to [Rabaey96] and [Chandrakasan98] for an in-depth overview.)

Reducing Switching Activity by Resource Allocation Multiplexing multiple operations on a single hardware unit has a detrimental effect on the power consumption. Besides increasing the physical capacitance, it can also increase the switching activity. This is illustrated with a simple experiment in Figure 11-53, which compares the power consumption of two counters running simultaneously. In the first case, both counters run on separate hardware, while in the second case, they are multiplexed on the same unit. Figure 11-53b plots the number of switching events as a function of the skew between the two counters. The nonmultiplexed case is always superior, except when both counters run in a completely synchronous fashion. The multiplexing tends to randomize the data signals presented to the operational unit, which results in increased switching activity. When power consumption is a concern, it is often beneficial to avoid the excessive reuse of resources. Observe that CMOS hardware units consume only negligible amounts of power when idle. Providing dedicated, specialized operators only presents an extra cost in area, while being generally beneficial in terms of speed and power.

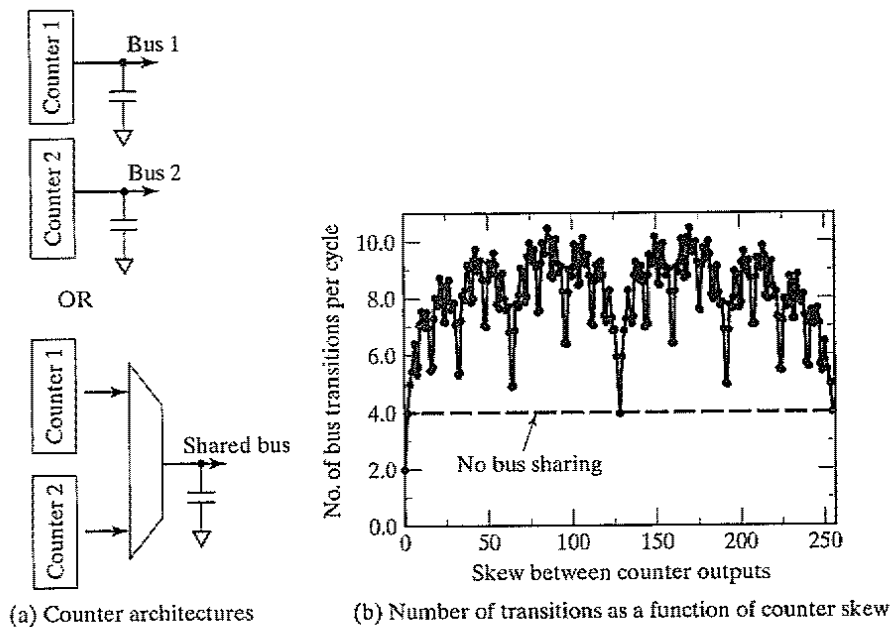


Figure 11-53 Multiplexing increases the switching activity.

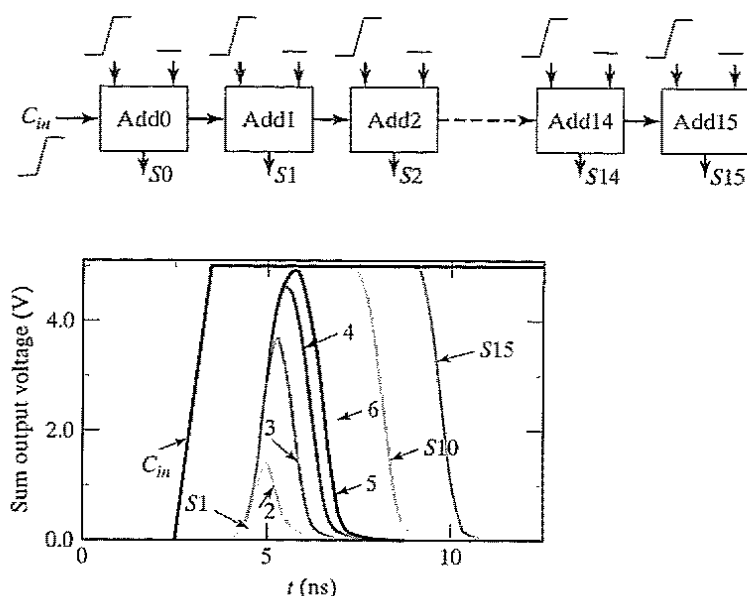


Figure 11-54 Glitching in the sum bits of a 16-bit ripple-carry adder.

Reducing Glitching through Path Balancing Dynamic hazards, or *glitching*, are a major contribution to the dissipation in complex structures such as adders and multipliers. The wide discrepancy in the lengths of the signal paths in some of those structures can be the cause of spurious transients. This is demonstrated in Figure 11-54, which displays the simulated response of a 16-bit ripple adder for all inputs going simultaneously from 0 to 1. A number of the sum bits are shown as a function of time. The sum signals should be zero for all bits. Unfortunately, a 1 appears briefly at all of the outputs, since the carry takes a significant amount of time to propagate from the first bit to the last. Notice how the glitch becomes more pronounced as it travels down the chain.

A dramatic reduction in glitching activity can be obtained by selecting structures with balanced signal paths. The tree lookahead adder structures (such as Kogge–Stone) and the tree multipliers have this property; therefore, they should be more attractive from a power point of view, even in the presence of a larger physical capacitance. An inspection of the lookahead structure of Figure 11-22 reveals that the timing paths to the inputs of dot operators are of a similar length, although some deviations may occur due to differences in loading and fan-out.

11.7.2 Run-Time Power Management

Dynamic Supply Voltage Scaling (DVS)

A static reduction of the supply voltage, as discussed in the preceding paragraphs, lowers the energy per operation and extends the battery life at the expense of performance. This performance penalty is often not acceptable, especially in applications that are latency constrained.

Substantial power reductions are still obtainable, based on the realization that the peak performance is not continuously required. Consider, for instance, a general-purpose processor to be used in portable applications, such as notebooks, electronic organizers, and cellular phones. The computational functions to be executed on such a processor fall into three major categories: compute-intensive tasks, low-speed functions, and idle-mode operation. Compute-intensive and short-latency tasks need the full computational throughput of the processor to achieve real-time constraints. MPEG video and audio decompression are examples of such. Low-throughput and long-latency tasks, such as text processing, data entry, and memory backups, operate under far more relaxed completion deadlines and require only a fraction of the maximum throughput of the microprocessor. There is no reward for finishing the computation early, and if a task is completed early, it can be considered a waste of energy. Finally, portable processors spend a large fraction of their time on idle, waiting for a user action or an external wake-up event. In sum, the computational throughput and latency expected from a mobile processor vary drastically over time.

Even compute-intensive operations, such as MPEG decoding, show variable computational requirements while processing a typical stream of data. For example, the number of times an MPEG decoder computes an inverse discrete cosine transform (IDCT) per video frame varies widely, depending upon the amount of motion in the video scenes. This is illustrated in Figure 11-55, which plots the distribution of the number of IDCTs/frame for a typical video sequence. The processor that is executing this algorithm experiences a different computational workload from frame to frame.

Lowering the clock frequency when executing the reduced workloads reduces the power, but does not save on energy—every operation is still executed at the high voltage level. However, if both supply voltage and frequency are lowered simultaneously, the energy is reduced. In order to maintain the required throughput for high workloads and minimize energy for low workloads, both supply and frequency must be dynamically varied according to the requirements application that is currently being executed. This technique is called *dynamic voltage scaling* (DVS). The concept is illustrated in Figure 11-56 [Burd00, Gutnik97]. It operates under

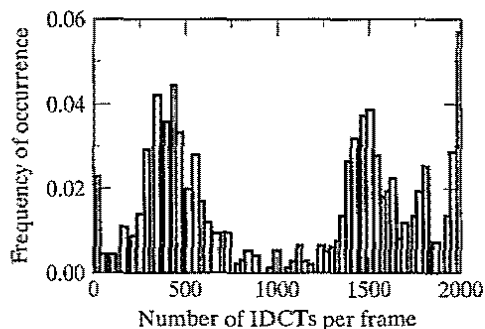


Figure 11-55 Typical IDCT histogram for MPEG decoding.

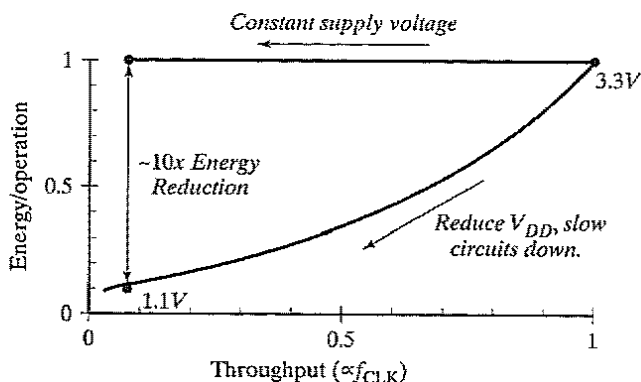


Figure 11-56 Energy/operation versus throughput ($1/T$) for constant and variable supply voltage operation.

the guideline that a function should always be operated at the lowest supply voltage that meets the timing constraints.

The DVS concept is enabled by the observation that the delay of most CMOS circuits and functions track each other well over a range of supply voltages, which is a necessity for system operation under varying supply conditions. Figure 11-57 shows the delays of a number of representative CMOS blocks (such as NAND gates, ring oscillators, register files, and SRAM), over a supply voltage range from 1 V to 4 V [Burd00]. Excellent performance tracking can be observed. Note that some circuit families, such as NMOS-only pass-transistor logic do not follow this behavior over the complete range of supplies.

A practical implementation of a dynamic-voltage scaling system now consists of the following components:

- a processor that can operate under a wide variety of supply voltages,

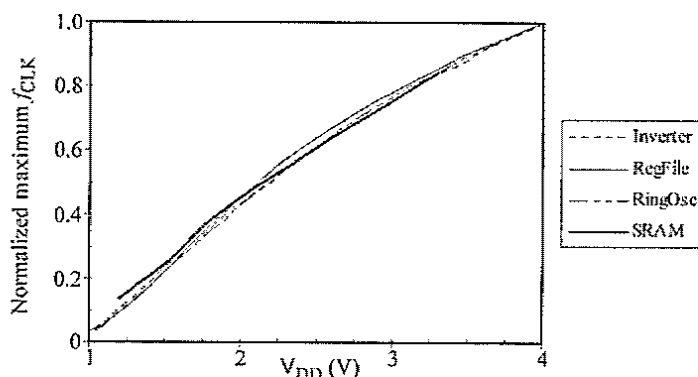


Figure 11-57 Delay of representative CMOS functions as a function of supply voltage for a 0.6- μm CMOS technology.

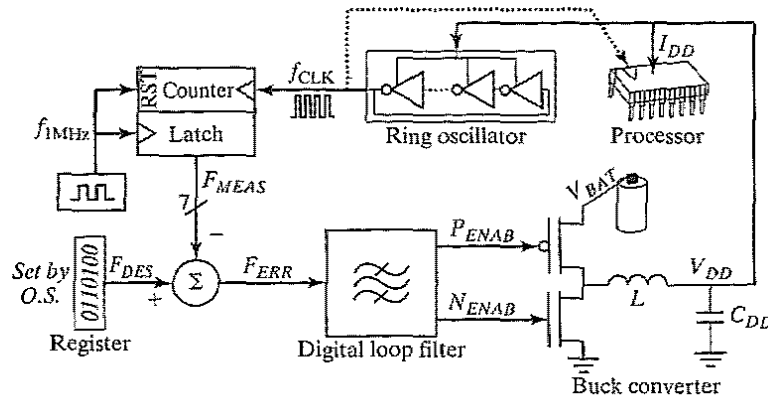


Figure 11-58 Block diagram of a dynamic voltage-scaled system.

- a supply-regulation loop that sets the minimum voltage necessary for operation at a desired frequency, and
- an operating system that calculates the required frequencies to meet requested throughputs and task completion deadlines.

One possible implementation is shown in Figure 11-58. The core of the DVS system is a ring oscillator, whose oscillating frequency matches the microprocessor critical path. When included inside the power-supply control loop, this ring oscillator provides the translation between the supply voltage and the clock frequency. The operating system digitally sets the desired frequency (F_{DES}). The current value of ring oscillator frequency is measured and compared with the desired frequency. The difference is used as a feedback error. By adjusting the supply voltage, the supply-voltage loop changes the ring oscillator frequency to set this error value to 0.

The task of the scheduler (or real-time operating system) is to determine dynamically the optimal frequency (or voltage) as a function of the combined computational requirements of all active tasks in the system. In the more complex case of a general-purpose processor, each task should supply a completion deadline (e.g., video frame rate) or a desired execution frequency. The voltage scheduler then estimates the number of processor cycles necessary for completing each of the tasks and computes the optimal processor frequency [Pering99]. In the case of a single task with varying performance requirements, a queue can be used to determine the computational load and to adjust the voltage accordingly. This is illustrated in Figure 11-59, where the depth of the input queue is used to set the supply voltage (and frequency) of a stream-based signal processor [Gutnik97].

Dynamic Threshold Scaling (DTS)

In analogy to the dynamic variation of the supply voltage, it is attractive to adjust the threshold voltage of the transistors dynamically. For low-latency computation, the threshold should be lowered to its minimal value; for low speed computation, it can be increased; and in the standby

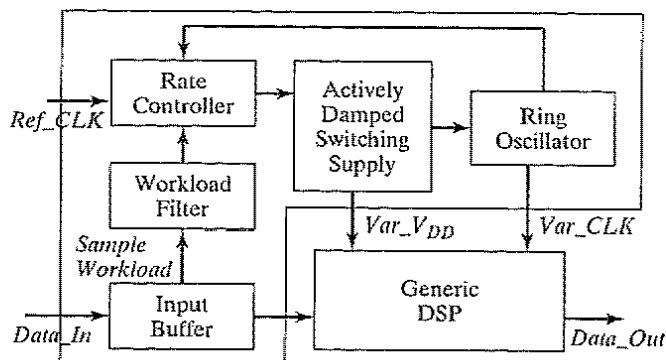


Figure 11-59 Using a queue to determine the workload in a signal processor [Gutnik97].

mode, it should be set to the highest possible value to minimize the leakage current. Substrate bias is the control knob that allows us to vary the threshold voltages dynamically. In order to do so, we have to operate the transistors as four-terminal devices. This is only possible in a triple-well process, as independent control of all four terminals of both n - and p -type devices is required, as shown in Figure 11-60. Substrate biasing can be implemented for a complete chip, on a block-by-block or a cell-by-cell basis. Per cell granularity of substrate biasing, however, has a large layout cost.

Similar to dynamic supply-voltage scaling scheme, the variable threshold voltage scheme is based on a feedback loop, which can be set to accomplish a variety of goals:

- It can lower the leakage in the standby mode
- It can compensate for threshold variations across the chip during normal operation of the circuit
- It can throttle the throughput of the circuit to lower both the active and leakage power based on performance requirements.

Since the current flow into the substrate is much smaller than into the supply lines, DTS has a smaller circuit overhead than DVS. A feedback system, designed to control the leakage in

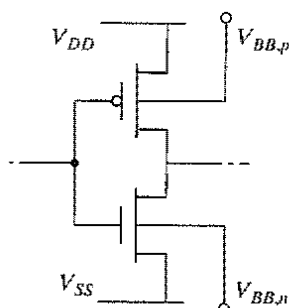


Figure 11-60 An inverter with body terminals used for threshold control.

To minimize the power penalty of the monitoring, the leakage monitor should be made as small as possible. However, if it is too small, the overall leakage monitoring response gets slower and the substrate biasing does not track variations closely. The optimal value can be set very low, with the monitor transistor being as small as 0.001% of the total transistor width on the chip.

A conventional charge pump siphons current in and out of the substrate. The control circuit monitors the leakage current. If it is above the preset value—corresponding to the mode of operation, set externally by the user or the operating system—the charge pump increases the negative back bias by pumping current out of the substrate. The charge pump shuts off when the leakage current reaches the target value. Junction leakages and impact ionization in the circuit will eventually raise the substrate bias voltage again, which activates the feedback loop new. Since it does not have to provide large supply currents on a continuous basis, this scheme is simpler to implement than dynamic voltage scaling.

Unfortunately, the effectiveness of adaptive body biasing is decreasing with further technology scaling. This is due to inherently lower body-effect factors and increased junction leakage attributable to band-to-band tunneling.

11.7.3 Reducing the Power in Standby (or Sleep) Mode

The idle mode represents an extreme corner of the dynamic power-management space. As no active switching occurs, all power dissipation is due to leakage—assuming that appropriate clock and input gating is in place. One option to reduce the leakage during standby is the DTS technique presented in the previous section. A simpler power-down scheme utilizes large sleep transistors to switch off the power supply rails when the circuit is in the sleep mode. This straightforward approach significantly reduces the leakage current, but increases the design complexity. It can be implemented by using a power switch on the supply rail only, or, even better, on both supply and ground rails, as shown Figure 11-62.

In normal operation mode, the *SLEEP* signal is high, and the sleep transistors must present as small a resistance as possible. The finite resistance of these transistors results in noise on supply rails, attributable to changes in supply current drawn by the logic. The sizing and the selection of the thresholds of the sleep transistors is subject to a trade-off process. To minimize fluctuations in the supply voltage, the sleep transistors should have a very low on-resistance, and therefore be very wide. However, increasing their size brings with it a major layout penalty. When transistors with a higher threshold are available, a better leakage suppression can be achieved. However, high-threshold devices must be even larger to yield the same resistance as low-threshold devices. The principles of leakage reduction are a direct extension of principles introduced in Section 6.4.2. Adding the sleep transistor effectively increases the transistor stack height, resulting in leakage reductions of the order of tens (for low threshold switches) to a thousand (for high threshold switches) times.

As opposed to simple clock gating, switching off the power supplies erases the state of the registers inside the block. In some applications, this is acceptable, such as when a completely

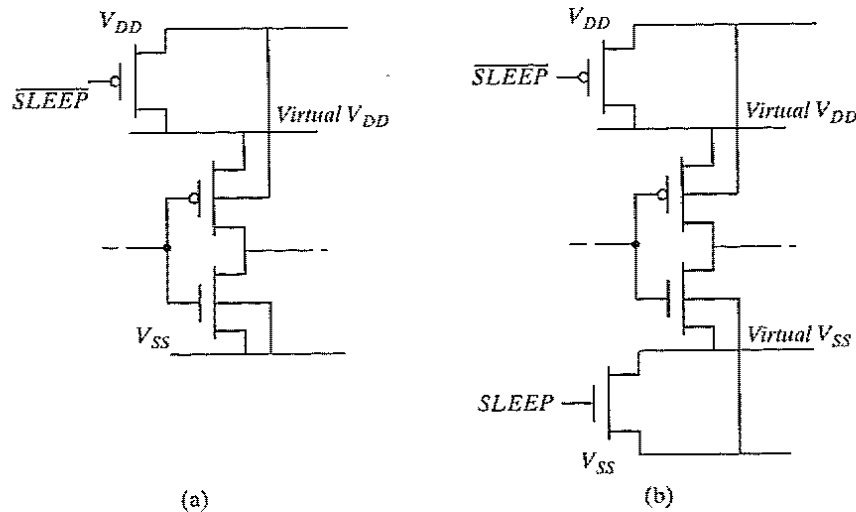


Figure 11-62 Sleep transistors used (a) only on the supply rail, (b) on both supply and ground.

new task is executed after the block wakes up again. However, additional effort is required when the state needs to be preserved. One option is to connect all the registers to the nongated supply rails, V_{DD} and V_{SS} , as discussed in Chapter 7 and shown in Figure 7-18. An alternative is to use the operating system to save the state of all registers to a nonvolatile memory.

11.8 Perspective: Design as a Trade-off

The analysis of the adder and multiplier circuits makes it clear again that digital circuit design is a trade-off between area, speed, and power requirements. This is demonstrated in Figure 11-63, which plots the normalized area and speed for some of the adders discussed earlier as a function of the number of bits.⁷ The overall project goals and constraints determine which factor is dominant.

The die area has a strong impact on the cost of an integrated circuit. A larger chip size means that fewer parts fit on a single wafer as discussed in Chapter 1. Reducing the area can help the viability of a product. Ultimate performance is what makes the newest microprocessor sell, and the lowest possible power consumption is a great marketing argument for a cellular phone. Understanding the market of a product is therefore essential when deciding on how to play the trade-off game. One should be aware that all design constraints—speed, power, and area—contribute to the feasibility or market success of a design.

In this context, it is worth summarizing some of the important design concepts that have been introduced in the course of this chapter:

⁷Be aware that these results are for a particular implementation in a particular technology. Extrapolation to other technologies should be done with care.

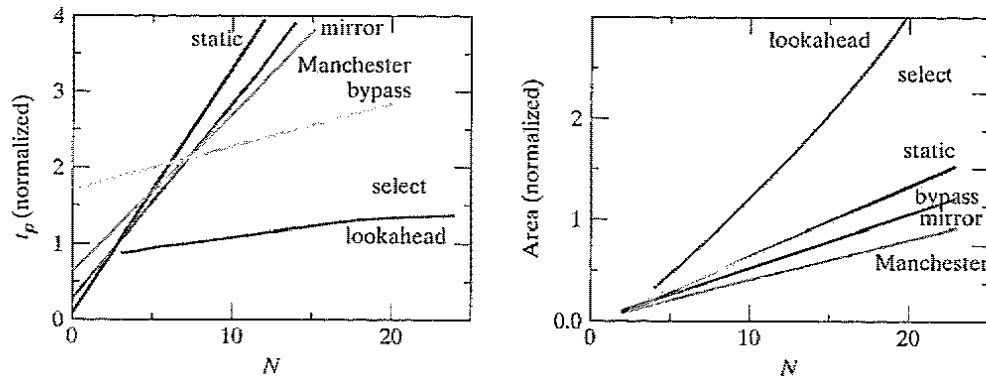


Figure 11-63 Area and propagation delay of various adder structures as a function of the number of bits N . Based on results from [Vermassen86].

1. The most important rule is to select the *right structure* before starting an elaborate circuit optimization. Going for the optimal performance of a complex structure by rigorously optimizing transistor sizes and topologies probably will not give you the best result. Optimizations at higher levels of abstraction, such as the logic or architectural level, can often generate more dramatic results. Simple first-order calculations can help give a global picture on the pros and cons of a proposed structure.
2. Determine the *critical timing path* through the circuit, and focus most of your optimization efforts on that part of the circuit. In addition to hand analysis, computer-aided design tools are available to help determine the critical paths and size the transistors appropriately. Be aware that some noncritical paths can be downsized to reduce power consumption.
3. Circuit size is not only determined by the number and size of transistors, but also by other factors such as *wiring and the number of vias and contacts*. These factors are becoming even more important with shrinking dimensions or when extreme performance is a goal.
4. Although an obscure optimization can sometimes help to get a better result, be wary if this results in an irregular and convoluted topology. *Regularity and modularity* are a designer's best friend.
5. Power and speed can be traded off through a choice of circuit sizing, supply voltages, and transistor thresholds.

11.9 Summary

In this chapter, we have studied the implementation of arithmetic datapath operators from a performance, area, and power perspective. Special attention was devoted to the development of *first-order performance models* that allow for a fast analysis and comparison of various logic structures before diving into the tedious transistor-level optimizations.

- A datapath is best implemented in a *bit-sliced* fashion. A single layout slice is used repetitively for every bit in the data word. This regular approach eases the design effort and results in fast and dense layouts.
- A *ripple-carry* adder has a performance that is linearly proportional to the number of bits. Circuit optimizations concentrate on reducing the delay of the carry path. A number of circuit topologies were examined, showing how careful optimization of the circuit topology and the transistor sizes helps to reduce the capacitance on the carry bit.
- Other adder structures use logic optimizations to increase the performance. The performance of *carry-bypass*, *carry-select* and *carry-lookahead* adders depends on the number of bits in square root and logarithmic fashion, respectively. This increase in performance comes at a cost in area, however.
- A *multiplier* is nothing more than a collection of cascaded adders. Its critical path is far more complex, and performance optimizations proceed along vastly different routes. The carry-save technique relies on a logic manipulation to turn the adder array into a regular structure with a well-defined critical timing path that can easily be optimized. Booth recoding and partial product accumulation in a tree reduces the complexity and delay of larger multipliers.
- The performance and area of a programmable shifter are dominated by the *wiring*. The exploitation of regularity can help to minimize the impact of the interconnect wires. This is exemplified in the barrel and the logarithmic shifter structures.
- *Power consumption* can be reduced substantially by the proper choice of circuit, logical, or architectural structure. This might come at the expense of area, but area might not be that critical in the age of submicron devices.
- A wide range of design-time and run-time techniques are at the disposition of a designer to minimize the power consumption. At *design time*, power and delay can be traded off through the choice of supply voltages and thresholds in addition to transistor sizing and logic optimization. The use of parallelism and pipelining can help to *reduce the supply voltage*, while maintaining the same throughput. The *effective capacitance* can be reduced by avoiding waste, as introduced by excessive multiplexing, for example.
- Some applications operate under variable throughput or latency conditions. Using *variable supplies and transistor thresholds* can lower the active or leakage power in such systems. Minimization of the standby energy consumption is essential for portable battery-operated devices.

11.10 To Probe Further

The literature on arithmetic and computer elements is vast. Important sources for newer developments are the *Proceedings of the IEEE Symposium on Computer Arithmetic*, the *IEEE Transactions on Computers* and the *IEEE Journal of Solid-State Circuits* (for integrated circuit implementation). An excellent collection of the most significant papers in the area can be found in some IEEE Press reprint volumes [Swartzlander90]. A number of other references, such as [Omondi94], [Koren98], and [Oklobdzija01], are provided for further reading.

References

- [Augsburger02] S. Augsburger, "Using Dual-Supply, Dual-Threshold and Transistor Sizing to Reduce Power in Digital Integrated Circuits," M.S. Project Report, University of California, Berkeley, April 2002.
- [Bedrij62] O. Bedrij, "Carry Select Adder," *IRE Trans. on Electronic Computers*, vol. EC-11, pp. 340-346, 1962.
- [Booth51] A. Booth, "A Signed Binary Multiplication Technique," *Quart. J. Mech. Appl. Math.*, vol. 4., part 2, 1951.
- [Brent82] R. Brent and H.T. Kung, "A Regular Layout for Parallel Adders," *IEEE Trans. on Computers*, vol. C-31, no. 3, pp. 260-264, March 1982.
- [Burd00] T.D. Burd, T.A. Pering, A.J. Stratakos, R.W. Brodersen, "A Dynamic Voltage Scaled Microprocessor System," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, November 2000.
- [Chandrakasan92] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low Power CMOS Digital Design," *IEEE Journal of Solid State Circuits*, vol. SC-27, no. 4, pp. 1082-1087, April 1992.
- [Chandrakasan98] A. Chandrakasan and R. Brodersen, Eds., *Low-Power CMOS Design*, IEEE Press, 1998.
- [Fetzer02] E.S. Fetzer, J.T. Orton, "A Fully-Bypassed 6-Issue Integer Datapath and Register File on an Itanium Microprocessor," 2002 *IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, San Francisco, CA, pp. 420-421, February 2002.
- [Gutnik97] V. Gutnik, and A. P. Chandrakasan, "Embedded Power Supply for Low-Power DSP," *IEEE Transactions on Very Large Integration (VLSI) Systems*, vol. 5, no. 4, pp. 425-435, December 1997.
- [Hamada01] M. Hamada, Y. Ootaguro, and T. Kuroda, "Utilizing Surplus Timing for Power Reduction," *Proceedings of the IEEE 2001 Custom Integrated Circuits Conference*, San Diego, CA, pp. 89-92, May 2001.
- [Kato00] N. Kato *et al.*, "Random Modulation: Multi-Threshold-Voltage Design Methodology in Sub-2-V Power Supply CMOS," *IEICE Transactions on Electronics*, vol. E83-C, no. 11, p. 1747-1754, November 2000.
- [Katz94] R.H. Katz, *Contemporary Logic Design*, Benjamin Cummings, 1994.
- [Kilburn60] T. Kilburn, D.B.G. Edwards and D. Aspinall, "A Parallel Arithmetic Unit Using a Saturated-Transistor Fast-Carry Circuit," *Proceedings of the IEE*, Pt. B, vol. 107, pp. 573-584, November 1960.
- [Kogge73] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, pp. 786-793, August 1973.
- [Koren98] I. Koren, *Computer Arithmetic Algorithms*, Brookside Court Publishers, 1998.
- [Kuroda96] T. Kuroda and T. Sakurai, "Threshold Voltage Control Schemes Through Substrate Bias for Low-Power High-Speed CMOS LSI Design," *Journal on VLSI Signal Processing*, vol. 13, no. 2/3, pp. 191-201, 1996.
- [Kuroda01] T. Kuroda, and T. Sakurai, "Low-Voltage Technologies," in *Design of High-Performance Microprocessor Circuits*, Chandrakasan, Bowhil, Fox (eds.), IEEE Press, 2001.
- [Lehman62] M. Lehman and N. Burla, "Skip Techniques for High-Speed Carry Propagation in Binary Arithmetic Units," *IRE Transactions on Electronic Computers*, vol. EC-10, pp. 691-698, December 1962.
- [Ma94] S. Ma, P. Franzon, "Energy Control and Accurate Delay Estimation in the Design of CMOS Buffers," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 9, pp. 1150-1153, September 1994.
- [MacSorley61] O. MacSorley, "High Speed Arithmetic in Binary Computers," *IRE Proceedings*, vol. 49, pp. 67-91, 1961.
- [Mathew01] S. K. Mathew *et al.*, "Sub-500-ps 64-b ALUs in 0.18- μ m SOI/Bulk CMOS: Design and Scaling Trends," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1636-1646, November 2001.
- [Ohkubo95] N. Ohkubo *et al.*, "A 4.4 ns CMOS 54*54-b Multiplier Using Pass-Transistor Multiplexer," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 251-257, March 1995.
- [Oklobdzija01] V. G. Oklobdzija, "High-Speed VLSI Arithmetic Units: Adders and Multipliers," in *Design of High-Performance Microprocessor Circuits*, Chandrakasan, Bowhil, Fox (eds.), IEEE Press, 2001.
- [Omondi94] A. Omondi, *Computer Arithmetic Systems*, Prentice Hall, 1994.
- [Pering00] T. A. Pering, *Energy-Efficient Operating System Techniques*, Ph.D. Dissertation, University of California, Berkeley, 2000.

- [Rabaey96] J. Rabaey, and M. Pedram, Eds., "Low-Power Design Methodologies," Kluwer Academic Publishers, 1996.
- [Stojanovic02] V. Stojanovic, D. Markovic, B. Nikolic, M. A. Horowitz, and R. W. Brodersen, "Energy-Delay Tradeoffs in Combinational Logic using Gate Sizing and Supply Voltage Optimization," *European Solid-State Circuits Conference, ESSCIRC'2002*, Florence, Italy, September 24–26, 2002.
- [Suzuoki99] M. Suzuoki *et al.*, "A Microprocessor with a 128-bit CPU, Ten Floating-Point MAC's, Four Floating-Point Dividers, and an MPEG-2 Decoder," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 11, pp. 1608–1618, November 1999.
- [Swartzlander90] E. Swartzlander, ed., *Computer Arithmetic—Part I and II*, IEEE Computer Society Press, 1990.
- [Usami95] K. Usami, and M. Horowitz, "Clustered Voltage Scaling Technique for Low-Power Design," *Proceedings, 1995 International Symposium on Low Power Design*, Dana Point, CA, pp. 3–8, April 1995.
- [Vermassen86] H. Vermassen, "Mathematical Models for the Complexity of VLSI," (in dutch), Master's Thesis, Katholieke Universiteit, Leuven, Belgium, 1986.
- [Wallace64] C. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. on Electronic Computers*, EC-13, pp. 14–17, 1964.
- [Weinberger56] A. Weinberger and J. L. Smith, "A One-Microsecond Adder Using One-Megacycle Circuitry," *IRE Transactions on Electronic Computers*, vol. 5, pp. 65–73, 1956.
- [Wei99] L. Wei, Z. Chen, K. Roy, M.C. Johnson, Y. Ye, and V. K. De, "Design and Optimization of Dual-Threshold Circuits for Low-Power Applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 16–24, March 1999.
- [Weinberger81] A. Weinberger, "A 4:2 Carry-Save Adder Module," *IBM Technical Disclosure Bulletin*, vol. 23, January 1981.
- [Weste85&93] N. Weste and K. Esragian, *Principles of CMOS VLSI Design: A Systems Perspective*, 2d ed., Addison-Wesley, 1985 and 1993.

Exercises

For the latest problem sets, design challenges and design projects related to arithmetic modules, log in to <http://bwrc.eecs.berkeley.edu/IcBook>.

H

Validation and Test of Manufactured Circuits

Manufacturing test

Design for testability

Test pattern generation

H.1 Introduction

While designers tend to spend numerous hours on the analysis, optimization, and layout of their circuits, one issue is often overlooked: When a component returns from the manufacturing plant, how does one know if it actually works? Does it meet the functionality and performance specifications? The customer expects a delivered component to perform as described in the specification sheets. Once a part is shipped or deployed in a system, it is expensive to discover that it does not work. The later a fault is detected, the higher the cost of correction. For instance, replacing a component in a sold television set means replacement of a complete board as well as the cost of labor. Shipping a nonworking or partially functional device should be avoided if at all possible.

A correct design does not guarantee that the manufactured component will be operational. A number of manufacturing defects can occur during fabrication, either due to faults in the base material (for instance, impurities in the silicon crystal), or as a result of variations in the process, such as misalignment. Other faults might be introduced during the stress tests that are performed after the manufacturing. These tests expose a part to cycles of temperature and mechanical stress to ensure its operation over a wide range of working conditions. Typical faults include short circuits between wires or layers and broken interconnections. This translates into network nodes that are either shorted to each other or to the supply rails, or that may be floating.

Making sure a delivered part is operating correctly under all possible input conditions is not as simple as it would seem at a first glance. When analyzing the circuit behavior during the design phase, the designer has unlimited access to all the nodes in the network. He is free to apply input patterns and observe the resulting response at any node he desires. This is not the case once the part is manufactured. The only access one has to the circuit is through the input-output pins. A complex component such as a microprocessor is composed of tens to hundreds of millions of transistors and contains an uncountable number of possible states. It is a very lengthy process—if it is possible at all—to bring such a component into a particular state and to observe the resulting circuit response through the limited bandwidth offered by the input-output pads. Hardware testing equipment tends to be very expensive and every second a part spends in the tester adds to its price.

It is therefore advisable to consider the testing early in the design process. Some small modifications in a circuit can help make it easier to validate the absence of faults. This approach to design has been dubbed *design for testability (DFT)*. While often despised by circuit designers who prefer to concentrate on the exciting aspects of design, such as transistor optimization, DFT is an integral and important part of the design process and should be considered as early as possible in the design flow. “*If you don’t test it, it won’t work! (Guaranteed)*” [Weste93]. A DFT strategy contains two components:

1. Provide the necessary *circuitry* so that the test procedure can be swift and comprehensive.
2. Provide the necessary *test patterns* (excitation vectors) to be employed during the test procedure. For reasons of cost, it is desirable that the test sequence be as short as possible while covering the majority of possible faults.

In the subsequent sections, we briefly cover some of the most important issues in each of these domains. Before doing so, a short description of a typical test procedure helps to put things in perspective.

H.2 Test Procedure

Manufacturing tests fall into a number of categories depending upon the intended goal:

- **The diagnostic test** is used during the debugging of a chip or board and tries to accomplish the following: Given a failing part, identify and locate the offending fault.
- **The functional test** (also called *go/no go* test) determines whether or not a manufactured component is functional. This problem is simpler than the diagnostic test since the only answer expected is yes or no. As this test must be executed on every manufactured die and has a direct impact on the cost, it should be as simple and swift as possible.
- **The parametric test** checks on a number of nondiscrete parameters, such as noise margins, propagation delays, and maximum clock frequencies, under a variety of working conditions, such as temperature and supply voltage. This requires a different set-up from

the functional tests that only deal with 0 and 1 signals. Parametric tests generally are subdivided into static (dc) and dynamic (ac) tests.

A typical manufacturing test proceeds as follows. The predefined test patterns are loaded into the tester that provides excitations to the *device under test* (DUT) and collects the corresponding responses. The test patterns are defined in a *test program* that describes the waveforms to be applied, voltage levels, clock frequency, and expected response. A probe card, or DUT board, is needed to connect the outputs and inputs of the tester to the corresponding pins on the die or package.

A new part is automatically fed into the tester. The tester executes the test program, applies the sequence of input patterns to the DUT, and compares the obtained response with the expected one. If differences are observed, the part is marked as faulty (e.g., with an ink spot), and the probes are automatically moved to the next die on the wafer. During the scribing process that divides the wafer into the individual dies, spotted parts will be automatically discarded. For a packaged part, the tested component is removed from the test board and placed in a good or faulty bin, depending upon the outcome of the test. The whole procedure takes in the range of a few seconds per part, making it possible for a single tester to handle thousands of parts in an hour.

Figure H-1 shows a picture of a high-end tester for logic circuits. Automatic testers are very expensive pieces of equipment. The increasing performance requirements, imposed by the high-speed ICs of today, have aggravated the situation, causing the cost of the test equipment to skyrocket. Reducing the time that a die spends on the tester is the most effective way to reduce the test cost. Unfortunately, with the increasing complexity of ICs, an opposite trend can be observed. Design approaches to reduce the test burden are thus very desirable.

H.3 Design for Testability

H.3.1 Issues in Design for Testability

As mentioned, a high-speed tester that can adequately handle state-of-the-art components comes at an astronomical cost. Reducing the test time for a single component can help increase the

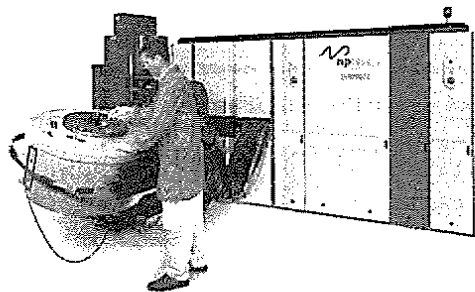


Figure H-1 Automatic tester (Courtesy Shlumberger).

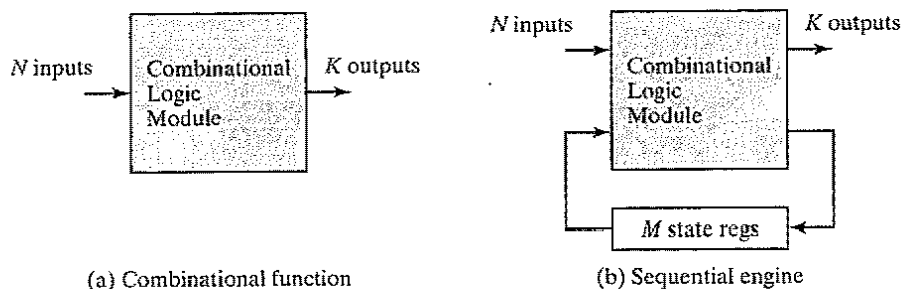


Figure H-2 Combinational and sequential devices under test.

throughput of the tester, and has an important impact on the testing cost. By considering testing from the early phases of the design process, it is possible to simplify the whole validation process. In this section, we describe some approaches that achieve that goal. Before detailing these techniques, we should first understand some of the intricacies of the test problem.

Consider the combinational circuit block of Figure H-2a. The correctness of the circuit can be validated by exhaustively applying all possible input patterns and observing the responses. For an N -input circuit, this requires the application of 2^N patterns. For $N = 20$, more than 1 million patterns are needed. If the application and observation of a single pattern takes 1 μ sec, the total test of the module requires 1 sec. The situation gets more dramatic when considering the sequential module of Figure H-2b. The output of the circuit depends not only upon the inputs applied, but also upon the value of the state. To exhaustively test this finite state machine (FSM) requires the application of 2^{N+M} input patterns, where M is the number of state registers [Williams83, Weste93]. For a state machine of moderate size (e.g., $M = 10$), this means that 1 billion patterns must be evaluated, which takes 16 minutes on our 1 μ sec/pattern testing equipment. Modeling a modern microprocessor as a state machine translates into an equivalent model with over 50 state registers. Exhaustive testing of such an engine would require over a billion years!

Obviously, an alternative approach is required. A more feasible testing approach is based on the following premises.

- An exhaustive enumeration of all possible input patterns contains a substantial amount of *redundancy*; that is, a single fault in the circuit is covered by a number of input patterns. Detection of that fault requires only one of those patterns, while the other patterns are superfluous.
- A substantial reduction in the number of patterns can be obtained by relaxing the condition that all faults must be detected. For instance, detecting the last single percentage of possible faults might require an exorbitant number of extra patterns, and the cost of detecting them might be larger than the eventual replacement cost. Typical test procedures only attempt a 95–99% fault coverage.

By eliminating redundancy and providing a reduced fault coverage, it is possible to test most combinational logic blocks with a limited set of input vectors. This does not solve the sequential problem, however. To test a given fault in a state machine, it is not sufficient to apply the correct input excitation; the engine must be brought to the desired state first. This requires that a sequence of inputs be applied. Propagating the circuit response to one of the output pins might require another sequence of patterns. In other words, testing for a single fault in an FSM requires a sequence of vectors. Once again, this might make the process prohibitively expensive.

One way to address the problem is to turn the sequential network into a combinational one by breaking the feedback loop in the course of the test. This is one of the key concepts behind the *scan-test* methodology described later. Another approach is to let the circuit test itself. Such a test does not require external vectors and can proceed at a higher speed. The concept of *self-test* will be discussed in more detail later. When considering the testability of designs, two properties are of foremost importance:

1. **Controllability**, which measures the ease of bringing a circuit node to a given condition using only the input pins. A node is easily controllable if it can be brought to any condition with only a single input vector. A node (or circuit) with low controllability needs a long sequence of vectors to be brought to a desired state. It should be clear that a high degree of controllability is desirable in testable designs.
2. **Observability**, which measures the ease of observing the value of a node at the output pins. A node with a high observability can be monitored directly on the output pins. A node with a low observability needs a number of cycles before its state appears on the outputs. Given the complexity of a circuit and the limited number of output pins, a testable circuit should have a high observability. This is exactly the purpose of the test techniques discussed in the sections that follow.

Combinational circuits fall under the class of easily observable and controllable circuits, since any node can be controlled and observed in a single cycle.

Design-for-test approaches for the sequential modules can be classified in three categories: ad hoc test, scan-based test, and self-test.

H.3.2 Ad Hoc Testing

As suggested by the title, ad hoc testing combines a collection of tricks and techniques that can be used to increase the observability and controllability of a design and that are generally applied in an application-dependent fashion.

An example of such a technique is illustrated in Figure H-3a, which shows a simple processor with its data memory. Under normal configuration, the memory is only accessible through the processor. Writing and reading a data value into and out of a single memory position requires a number of clock cycles. The controllability and observability of the memory can be dramatically improved by adding multiplexers on the data and address busses (Figure H-3b).

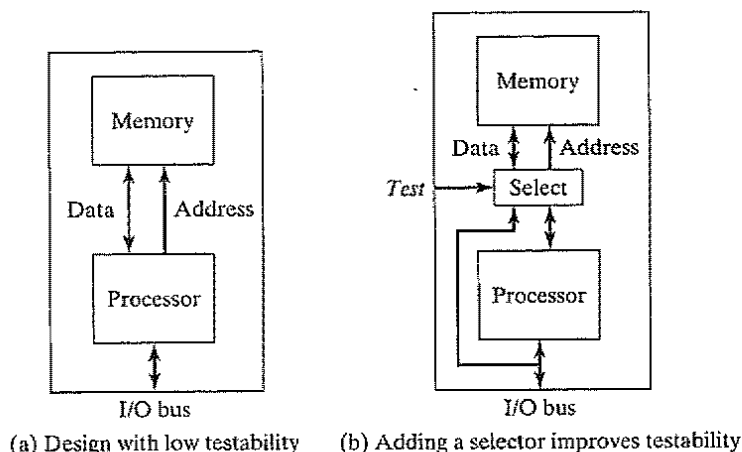


Figure H-3 Improving testability by inserting multiplexers.

During normal operation mode, these selectors direct the memory ports to the processor. During test, the data and address ports are connected directly to the I/O pins, and testing the memory can proceed more efficiently. The example illustrates some important design-for-testability concepts.

- It is often worthwhile to introduce *extra hardware* that has no functionality except improving the testability. Designers are often willing to incur a small penalty in area and performance if it makes the design substantially more observable or controllable.
- Design-for-testability often means that extra I/O pins must be provided besides the normal functional I/O pins. The *test* port in Figure H-3b is such an extra pin. To reduce the number of extra pads that would be required, one can multiplex test signals and functional signals on the same pads. For example, the I/O bus in Figure H-3b serves as a data bus during normal operation and provides and collects the test patterns during testing.

An extensive collection of ad hoc test approaches has been devised. Examples include the partitioning of large state machines, addition of extra test points, provision of reset states, and introduction of test busses. While very effective, the applicability of most of these techniques depends upon the application and architecture at hand. Their insertion into a given design requires expert knowledge and is difficult to automate. Structured and automatable approaches are more desirable.

H.3.3 Scan-Based Test

One way to avoid the sequential-test problem is to turn all registers into externally loadable and readable elements. This turns the circuit-under-test into a combinational entity. To control a node, an appropriate vector is constructed, loaded into the registers and propagated through the logic. The result of the excitation propagates to the registers and is latched, after which the con-

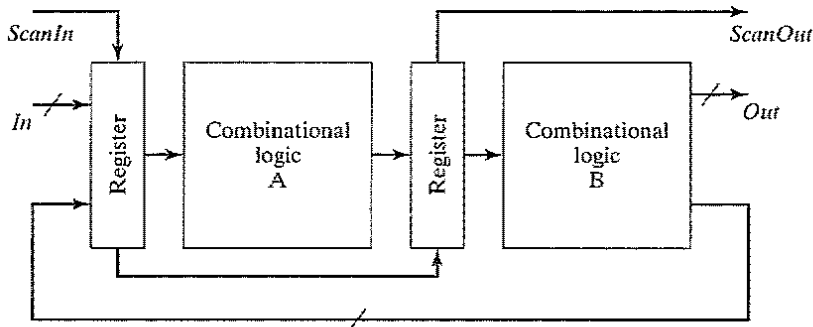


Figure H-4 Serial-scan test.

tents are transferred to the external world. Connecting all the registers in a design to a test bus regrettably introduces an unacceptable amount of overhead. A more elegant approach is offered by the serial-scan approach illustrated in Figure H-4.

The registers have been modified to support two operation modes. In the normal mode, they act as N -bit-wide clocked registers. During the test mode, the registers are chained together as a single serial shift register. A test procedure now proceeds as follows.

1. An excitation vector for logic module A (and/or B) is entered through pin *ScanIn* and shifted into the registers under control of a test clock.
2. The excitation is applied to the logic and propagates to the output of the logic module. The result is latched into the registers by issuing a single system-clock event.
3. The result is shifted out of the circuit through pin *ScanOut* and compared with the expected data. A new excitation vector can be entered simultaneously.

This approach incurs only a minimal overhead. The serial nature of the scan chain reduces the routing overhead. Traditional registers are easily modified to support the scan technique, as demonstrated in Figure H-5, which shows a 4-bit register extended with a scan chain. The only addition is an extra multiplexer at the input. When *Test* is low, the circuit is in normal operation

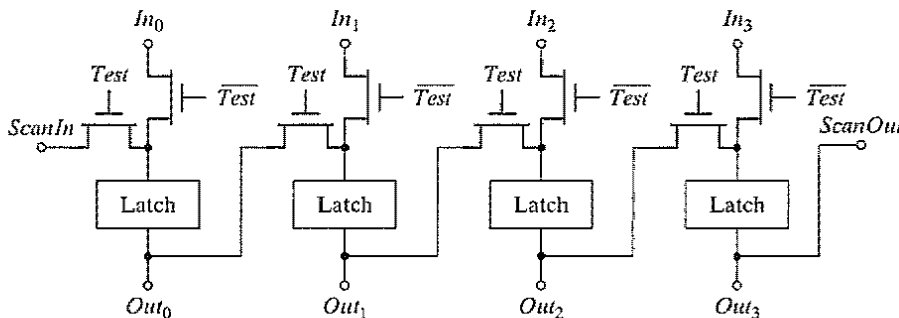


Figure H-5 Register extended with serial-scan chain.

mode. Setting *Test* high selects the *ScanIn* input and connects the registers into the scan chain. The output of the register *Out* connects to the fan-out logic, but also doubles as the *ScanOut* pin that connects to the *ScanIn* of the neighboring register. The overhead in both area and performance is small and can be limited to less than 5%.

Problem H.1 Scan-Register Design

Modify the static, two-phase master-slave register of Figure 7-10 to support serial scan.

Figure H-6 depicts the timing sequence that would be employed for the circuit in Figure H-4 under the assumption of a two-phase clocking approach. For a scan chain N registers deep, the *Test* signal is raised, and N clock pulses are issued, loading the registers. *Test* is lowered, and a single clock sequence is issued, latching the results from the combinational logic into the registers under normal circuit-operation conditions. Finally, N extra pulses (with *Test* = 1) transfer the obtained result to the output. Note again that the scan-out can overlap with the entering of the next vector.

Many variants of the serial-scan approach can be envisioned. A very popular one, which was actually the pioneering approach, was introduced by IBM and is called *level-sensitive scan design* (LSSD) [Eichelberger78]. The basic building block of the LSSD approach is the *shift-register latch* (SRL) shown in Figure H-7. It consists of two latches $L1$ and $L2$, the latter being

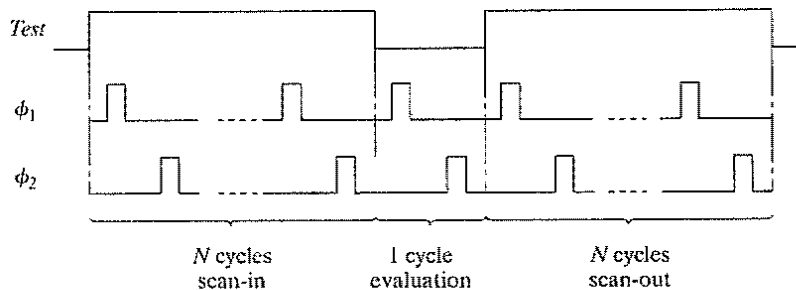


Figure H-6 Timing diagram of test-sequence. N represents the number of registers in the test chain.

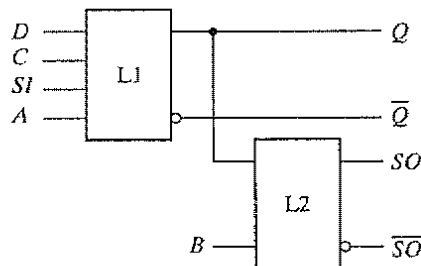


Figure H-7 Shift-register latch.

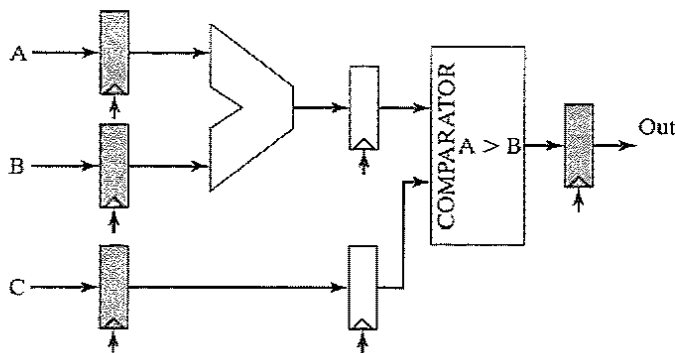


Figure H-8 Pipelined datapath using partial scan. Only the shaded registers are included in the chain.

present only for testing purposes. In normal circuit operation, signals D , Q (Q), and C serve as latch input, output, and clock. The test clocks A and B are low in this mode. In scan mode, SI and SO serve as scan input and scan output. Clock C is low, and clocks A and B act as nonoverlapping, two-phase test clocks.

The LSSD approach represents not only a test strategy, but also a complete clocking philosophy. By strictly adhering to the rules implied by this methodology, it is possible to automate to a large extent the test generation and the timing verification. This is why the use of LSSD was obligatory within IBM for a long time. The prime disadvantage of the approach is the complexity of the SRL latch.

It is not always necessary to make all the registers in the design scannable. Consider the pipelined datapath of Figure H-8. The pipeline registers in this design are only present for performance reasons and do not strictly add to the state of the circuit. It is, therefore, meaningful to make only the input and output registers scannable. During test generation, the adder and comparator can be considered together as a single combinational block. The only difference is that during the test execution, two cycles of the clocks are needed to propagate the effects of an excitation vector to the output register. This approach is called *partial scan* and is often employed when performance is of prime interest. The disadvantage is that deciding which registers to make scannable is not always obvious and may require interaction with the designer.

H.3.4 Boundary-Scan Design

Until recently, the test problem was most compelling at the integrated circuit level. Testing circuit boards was facilitated by the abundant availability of test points. The through-hole mounting approach made every pin of a package observable at the back side of the board. For test, it was sufficient to lower the board onto a set of test probes (called “bed-of-nails”) and apply and observe the signals of interest. The picture changed with the introduction of advanced packaging techniques such as surface-mount or multichip modules (Chapter 2). Controllability and observability are not as readily available anymore, because the number of probe points is dramatically

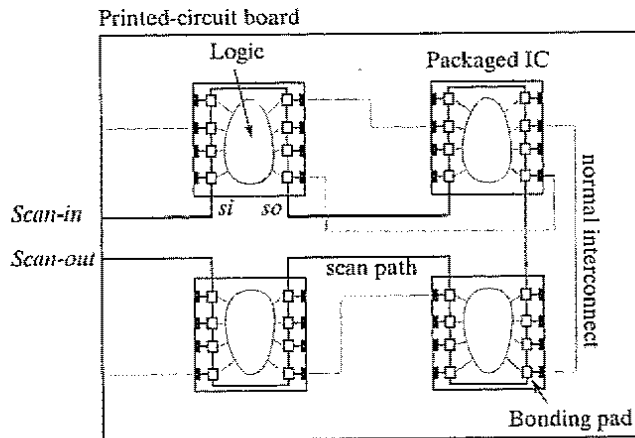


Figure H-9 The boundary-scan approach to board testing.

reduced. This problem can be addressed by extending the scan-based test approach to the component and board levels.

The resulting approach is called *boundary scan* and has been standardized to ensure compatibility between different vendors ([IEEE1149]). In essence, it connects the input-output pins of the components on a board into a serial scan chain, as shown in Figure H-9. During normal operation, the boundary-scan pads act as normal input-output devices. In test mode, vectors can be scanned in and out of the pads, providing controllability and observability at the boundary of the components (hence, the name). The test operation proceeds along similar lines as described in the previous paragraph. Various control modes allow for testing the individual components as well as the board interconnect. The overhead incurred includes slightly more complex input-output pads and an extra on-chip test controller (an FSM with 16 states). Boundary scan is now provided in most commodity components.

H.3.5 Built-in Self-Test (BIST)

An alternative and attractive approach to testability is having the circuit itself generate the test patterns instead of requiring the application of external patterns [Wang86]. Even more appealing is a technique where the circuit itself decides if the obtained results are correct. Depending upon the nature of the circuit, this might require the addition of extra circuitry for the generation and analysis of the patterns. Some of this hardware might already be available as part of the normal operation, and the size overhead of the self-test can be small.

The general format of a built-in self-test design is illustrated in Figure H-10 ([Kornegay92]). It contains a means for supplying test patterns to the device under test and a means of comparing the device's response to a known correct sequence.

There are many ways to generate stimuli. Most widely used are the *exhaustive* and the *random* approaches. In the exhaustive approach, the test length is 2^N , where N is the number of inputs to the circuit. The exhaustive nature of the test means that all detectable faults will be

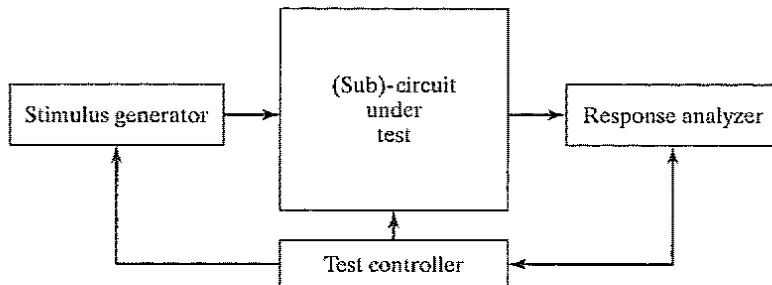


Figure H-10 General format of built-in self-test structure.

detected, given the space of the available input signals. An N -bit counter is a good example of an exhaustive pattern generator. For circuits with large values of N , the time to cycle through the complete input space might be prohibitive. An alternative approach is to use random testing that implies the application of a randomly chosen sub-set of 2^N possible input patterns. This subset should be selected so that a reasonable fault coverage is obtained. An example of a pseudorandom pattern generator is the *linear-feedback shift register* (or LFSR), which is shown in Figure H-11. It consists of a serial connection of 1-bit registers. Some of the outputs are XOR'd and fed back to the input of the shift register. An N -bit LFSR cycles through $2^N - 1$ states before repeating the sequence, which produces a seemingly random pattern. Initialization of the registers to a given seed value (different from 0 for our example circuit) determines what will be generated, subsequently.

The response analyzer could be implemented as a comparison between the generated response and the expected response stored in an on-chip memory, but this approach represents too much area overhead to be practical. A cheaper technique is to compress the responses before comparing them. Storing the compressed response of the correct circuit requires only a minimal amount of memory, especially when the compression ratio is high. The response analyzer then

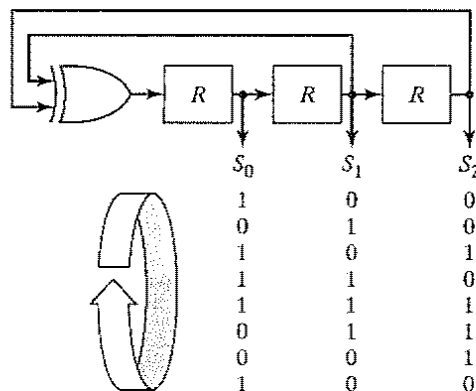


Figure H-11 Three-bit linear-feedback shift register and its generated sequence.

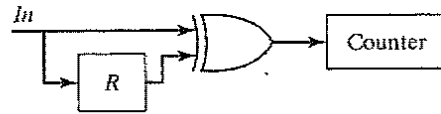
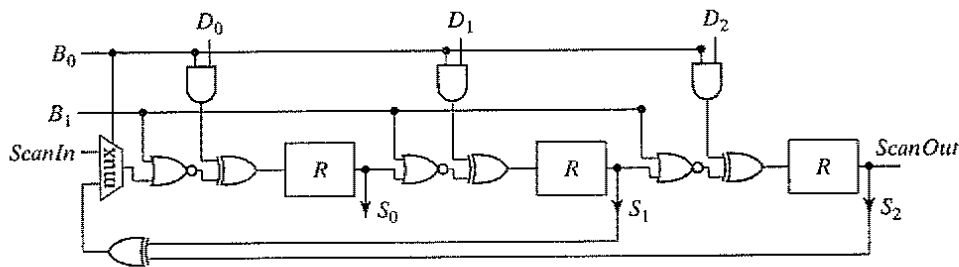


Figure H-12 Single bit-stream signature analysis.

consists of circuitry that dynamically compresses the output of the circuit under test and a comparator. The compressed output is often called the *signature* of the circuit, and the overall approach is dubbed *signature analysis*.

An example of a signature analyzer that compresses a single bit stream is shown in Figure H-12. Inspection reveals that this circuit simply counts the number of $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions in the input stream. This compression does not guarantee that the received sequence is the correct one; that is, there are many different sequences with the same number of transitions. Since the chances of this happening are slim, it may be a risk worth taking if kept within bounds.

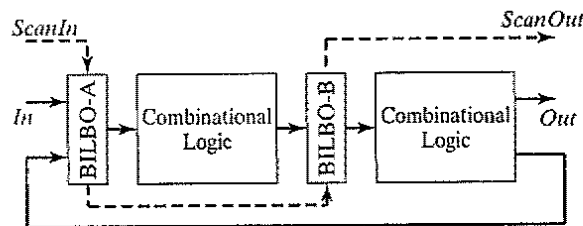
Another technique is illustrated in Figure H-13a. It represents a modification of the linear-feedback shift register and has the advantage that the same hardware can be used for both pattern generation and signature analysis. Each incoming data word is successively XOR'd with the contents of the LFSR. At the end of the test sequence, the LFSR contains the signature, or *syndrome*, of the data sequence, which can be compared with the syndrome of the correct circuit. The circuit not only implements a random-pattern generator and signature analyzer, but also can



(a) A 3-bit BILBO register

$B_0 B_1$	Operation mode
1 1	Normal
0 0	Scan
1 0	Pattern generation or signature analysis
0 1	Reset

(b) BILBO modes



(c) BILBO application

Figure H-13 Built-in logic block observation, or BILBO.

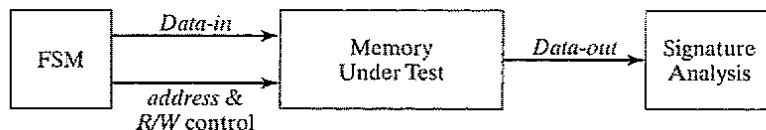


Figure H-14 Memory self-test.

be used as a normal register and scan register, depending on the values of the control signals B_0 and B_1 (Figure H-13b). This test approach, which combines all the different techniques, is known as *built-in logic block observation*, or *BILBO* [Koeneman79]. Figure H-13c illustrates the typical use of BILBO. Using the scan option, the seed is shifted into the BILBO register A while BILBO register B is initialized. Next, registers A and B are operated in the random pattern-generation and signature-analysis modes, respectively. At the end of the test sequence, the signature is read from B using the scan mode.

Finally, it is worth mentioning that self-test is extremely beneficial when testing regular structures such as memories. It is not easy to ensure that a memory, which is a sequential circuit, is fault free. The task is complicated by the fact that the data value read from or written into a cell can be influenced by the values stored in the neighboring cells because of cross coupling and other parasitic effects. Memory tests, therefore, include the reading and writing of a number of different patterns into and from the memory using alternating addressing sequences. Typical patterns can be all zeros or ones, or checkerboards of zeros and ones. Addressing schemes can include the writing of the complete memory, followed by a complete read-out or various alternating read-write sequences. With a minimal overhead compared with the size of a memory, this test approach can be built into the integrated circuit itself, as illustrated in Figure H-14. This approach significantly improves the testing time and minimizes the external control. Applying self-test is bound to become more important with the increasing complexity of integrated components and the growing popularity of embedded memories.

The advent of the systems-on-a-chip era does not make the test job any easier. A single IC may contain micro- and signal processors, multiple embedded memories, ASIC modules, FPGAs and on-chip busses and networks. Each of these modules has its own preferred way of being tested, and combining those into a coherent strategy is quite a challenge. Built-in self-test is really the only way out. A structured test-methodology for systems-on-a-chip, based on BIST is shown in Figure H-15. Each of the modules composing the system connects to the on-chip network through a "wrapper." This is a customized interface between the block and the network, supporting functions such as synchronization and communication. This wrapper can be extended to include a test support module. For instance, for an ASIC module that includes a scan chain, the test support module provides the interface to the scan chain and a buffer for the test patterns. This buffer can be directly written and read through the system bus. Similarly, a memory module can be equipped with a pattern generator and signature analysis. All of this would still not suffice if there were no general test orchestrator. Fortunately, most of these SOC's include a programmable processor which can be used at start-up time to direct the test and the verification of the other modules. Test patterns and signatures can be stored

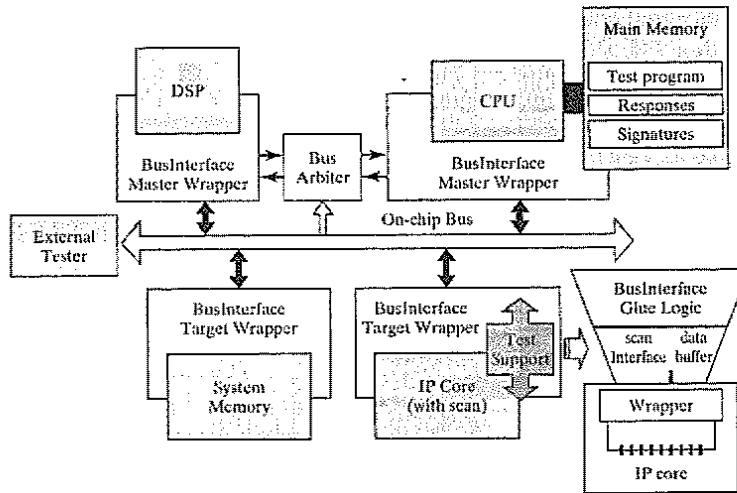


Figure H-15 System-on-a-chip test methodology.

in the main memory of the processor, and supplied to the module under test at test time. This approach has the advantage that it uses the resources that are already available on the die. Furthermore, the BIST self-test approach allows for the tests to be performed under actual clock speeds, decreasing the test time. The requirements for the external tester are minimized, reducing the tester cost. The development of a structured approach to self-test, as pictured in Figure H-15, is a topic of active research [Krstic01]. ■

H.4 Test-Pattern Generation

In the preceding sections, we have discussed how to modify a design so that test patterns can be effectively applied. What we have ignored so far is the complex task of determining what patterns should be applied so that a good fault coverage is obtained. This process was extremely problematic in the past, when the test engineer—a different person than the designer—had to construct the test vectors after the design was completed. This invariably required a substantial amount of wasteful reverse engineering that could have been avoided if testing had been considered early in the design flow. An increased sensitivity to design for testability and the emergence of automatic test-pattern generation (ATPG) has substantially changed this picture.

In this section, we delve somewhat deeper into the ATPG issue and present techniques to evaluate the quality of a test sequence. Before doing so, we must analyze the fault concept in more detail.

H.4.1 Fault Models

Manufacturing faults can be of a wide variety and manifest themselves as short circuits between signals, short circuits to the supply rails, and floating nodes. In order to evaluate the effectiveness of a test approach and the concept of a good or bad circuit, we must relate these faults to the circuit model, or, in other words, derive a *fault model*. The most popular approach is called the *stuck-at* model. Most testing tools consider only the short circuits to the supplies. These are

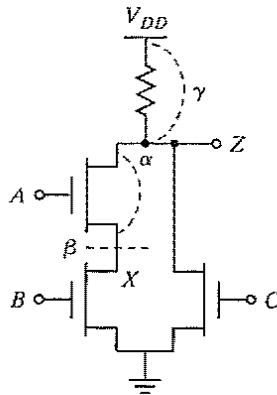


Figure H-16 Resistive-load gate, annotated with a number of stuck-at-open (β) and stuck-at-short (α , γ) faults.

called the *stuck-at-zero* (sa0) and *stuck-at-one* (sa1) faults for short circuits to *GND* and *V_{DD}*, respectively.

It can be argued that the sa0-sa1 model does not cover the complete range of faults that can occur in a state-of-the-art integrated circuit, and that *stuck-at-open* and *stuck-at-short* faults should also be introduced. However, adding these faults complicates the test pattern generation process. Moreover, a large number of these faults are covered by the sa0-sa1 model. To illustrate this observation, consider the resistive-load MOS gate of Figure H-16. All shorts to the supplies are modeled by the introduction of sa0 and sa1 faults at nodes *A*, *B*, *C*, *Z*, and *X*. The figure has been annotated with some stuck-at-open (β) and stuck-at-short faults (α , γ). It can be observed that these faults are already covered by the sa0 and sa1 faults on the various nodes. For example, fault α is covered by A_{sa1} , β is covered by A_{sa0} or B_{sa0} , while γ is equivalent to Z_{sa1} .

Even so, shorts and open-circuit faults can cause some interesting artifacts to occur in CMOS circuits that are not covered by the sa0-sa1 model and are worth mentioning. Consider the two-input NAND gate of Figure H-17, where a stuck-at-open fault α has occurred. The truth table of the faulty circuit is shown in the figure as well. For the combination ($A = 1$, $B = 0$), the output node is floating and retains its previous value, while the correct value should be a 1. Depending upon the previous excitation vector, this fault may or may not be detected. In fact, the circuit behaves as a sequential network. To detect this fault, two vectors must be applied in sequence. The first one forces the output to 0 (or $A = 1$ and $B = 1$), while the second applies the $A = 1$, $B = 0$ pattern. Also stuck-at-short faults are troublesome in CMOS circuits since they can cause dc currents to flow between the supply rails for certain input values, which produces undefined output voltages.

Even though the sa0-sa1 fault model is not perfect, its ease of use and relatively large coverage of the fault space have made it the de facto standard model. It is often supplemented with other techniques, such as functional test, *IDDQ* test—which measures the change in quiescent current of a CMOS circuit due to short circuits—and delay test. No single test model is completely foolproof, and a combination of several testing methods is often required [Bhavsar01].

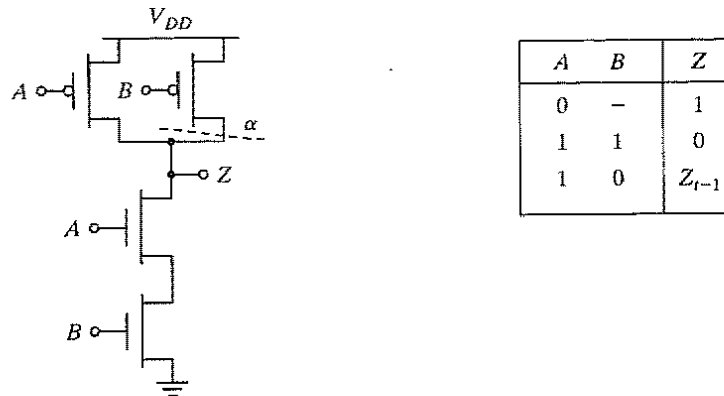


Figure H-17 Two-input complementary CMOS NAND gate and its truth table in the presence of a stuck-at-open fault.

H.4.2 Automatic Test-Pattern Generation (ATPG)

The task of the automatic test-pattern generation (ATPG) process is to determine a minimum set of excitation vectors that cover a sufficient portion of the fault set as defined by the adopted fault model. One possible approach is to start from a random set of test patterns. Fault simulation then determines how many of the potential faults are detected. With the obtained results as guidance, extra vectors can be added or removed iteratively. An alternative and potentially more attractive approach relies on the knowledge of the functionality of a Boolean network to derive a suitable test vector for a given fault. To illustrate the concept, consider the example of Figure H-18. The goal is to determine the input excitation that exposes an sa0 fault occurring at node U at the output of the network Z . The first requirement of such an excitation is that it should force the fault to occur (*controllability*, again). In this case, we look for a pattern that would set U to 1 under normal circumstances. The only option here is $A = 1$ and $B = 1$. Next, the faulty signal has to propagate to output node Z , so that it can be *observed*. This phase is called *path sensitizing*. For any change in node U to propagate, it is necessary for node X to be set to 1 and node E to 0. The (unique) test vector for U_{sa0} can now be assembled: $A = B = C = D = 1$, $E = 0$.

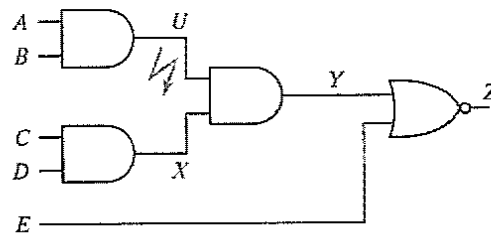


Figure H-18 Simple logic network, with sa0 fault at node U .

This example is extremely simple, and the derivation of a minimum test-vector set for more complex circuits is substantially more complex. A number of excellent approaches to address this problem have been developed. Landmark efforts in this domain are the *D* [Roth66] and PODEM algorithms [Goel81], which underlie many current ATPG tools. It suffices to say that ATPG is currently in the mainstream of design automation, and powerful tools are available from many vendors.

H.4.3 Fault Simulation

A fault simulator measures the quality of a test program. It determines the *fault coverage*, which is defined as the total number of faults detected by the test sequence divided by two times the number of nodes in the network—each node can give rise to an sa0 and sa1 fault. Naturally, the obtained coverage number is only as good as the fault model employed. In an sa0-sa1 model, some of the bridge and short faults are not covered and will not appear in the coverage statistics.

The most common approach to fault simulation is the parallel fault-simulation technique, in which the correct circuit is simulated concurrently with a number of faulty ones, each of which has a single fault injected. The results are compared, and a fault is labeled as detected for a given test vector set if the outputs diverge. This description is overly simplistic, and most simulators employ a number of techniques, such as selecting the faults with a higher chance of detection first, to expedite the simulation process. Hardware fault-simulation accelerators, based on parallel processing and providing a substantial speedup over pure software-based simulators, are available as well [Agrawal88, pp. 159–240].

H.5 To Probe Further

For an in-depth treatment of the Design-for-Testability topic, please refer to [Agrawal88] and [Abramovic91]. A great overview of the testing challenges and solutions for high-performance microprocessors can be found in [Bhavsar01].

References

- [Abramovic91] M. Abramovic, M. Breuer, and A. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, Piscataway, NJ, 1991.
- [Agrawal88] V. Agrawal and S. Seth, Eds. *Test Generation for VLSI Chips*, IEEE Computer Society Press, 1988.
- [Bhavsar01] D. Bhavsar, "Testing of High-Performance Microprocessors," in *Design of High-Performance Microprocessor Circuits*, A. Chandrakasan et al., Ed., pp. 523–44, IEEE Press, 2001.
- [Eichelberger78] E. Eichelberger and T. Williams, "A Logic Design Structure for VLSI Testability," *Journal on Design Automation of Fault-Tolerant Computing*, vol. 2, pp. 165–78, May 1978.
- [Goel81] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Computers*, vol. C-30, no. 3, pp. 26–268, June 1981.
- [IEEE1149] IEEE Standard 1149.1, "IEEE Standard Test Access Port and Boundary-Scan Architecture," IEEE Standards Board, New York.
- [Koeneman79] B. Koeneman, J. Mucha, and O. Zwiehoff, "Built-in Logic-Block Observation Techniques," in *Digest 1979 Test Conference*, pp. 37–41, October 1979.

- [Kornegay92] K. Kornegay, *Automated Testing in an Integrated System Design Environment*, Ph. D dissertation., Mem. No. UCB/ERL M92/104, Sept. 1992.
- [Krstic01] A. Krstic, W. Lai, L. Chen, K. Cheng, and S. Dey, "Embedded Software-Based Self-Testing for SOC Design," <http://www.gigascale.org/pubs/139.html>, December 2001.
- [Roth66] J. Roth, "Diagnosis of Automata Failures: A Calculus and a Method," *IBM Journal of Research and Development*, vol. 10, pp. 278–291, 1966.
- [Wang86] L. Wang and E. McCluskey, "Complete Feedback Shift-Register Design for Built-in Self Test," *Proc. ICCAD 1986*, pp. 56–59, November, 1986.
- [Weste93] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design—A Systems Perspective*, Addison-Wesley, 1993.
- [Williams83] T. Williams and K. Parker, "Design for Testability—A Survey," *Proceedings IEEE*, vol. 71, pp. 98–112, Jan. 1983.