CHAPTER

8

فصل های ۸ و ۱۱ و ۱۲ کتاب Digital Integrated Circuits, Rabaey

Implementation Strategies for Digital ICs

Semicustom and structured design methodologies

ASIC and system-on-a-chip design flows

Configurable hardware

8.1 Introduction

The dramatic increase in complexity of contemporary integrated circuits poses an enormous design challenge. Designing a multimillion-transistor circuit and ensuring that it operates correctly when the first silicon returns is a daunting task that is virtually impossible without the help of computer aids and well-established design methodologies. In fact, it has often been suggested that technology advancements might be outpacing the absorption bandwidth of the design community. This is articulated in Figure 8-1, which shows how IC complexity (in logic transistors) is growing faster than the productivity of a design engineer, creating a "design gap." One way to address this gap is to increase steadily the size of the design teams working on a single project. We observe this trend in the high-performance processor world, where teams of more than 500 people are no longer a surprise.

Obviously, this approach cannot be sustained in the long term—just imagine all the design engineers in the world working on a single design. Fortunately, about once in a decade we witness the introduction of a novel design methodology that creates a step function in design productivity, helping to bridge the gap temporarily. Looking back over the past four decades, we can identify a number of these productivity leaps. Pure custom design was the norm in the early integrated circuits of the 1970s. Since then, programmable logic arrays (PLAs), standard cells, macrocells, module compilers, gate arrays, and reconfigurable hardware have steadily helped to ease the time and cost of mapping a function onto silicon. In this chapter, we provide a description of some commonly used design implementation approaches. Due to the extensive nature of the field, we cannot be comprehensive—doing so would require a textbook of its own. Instead, we present *a user perspective* that provides a basic perception and insight into what is offered and can be expected from the different design methodologies.

The preferred approach to mapping a function onto silicon depends largely upon the function itself. Consider, for instance, the simple digital processor of Figure 8-2. Such a processor

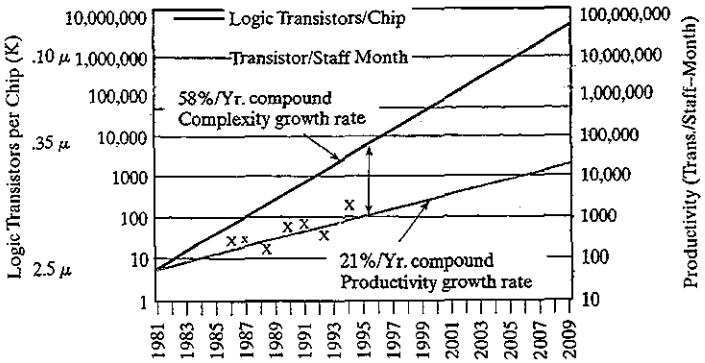


Figure 8-1 The design productivity gap. Technology (in logic transistors/chip) outpaces the design productivity (in transistors designed by a single design engineer per month). Source: SIA [SIA97].

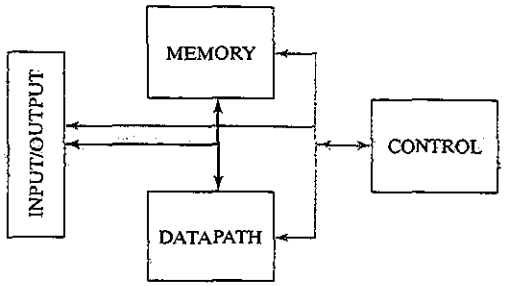


Figure 8-2 Composition of a generic digital processor. The arrows represent the possible interconnections.

could be the brain of a personal computer (PC), or the heart of a compact-disc player or cellular phone. It is composed of a number of building blocks that occur in one form or another in almost every digital processor:

- **The datapath** is the core of the processor; it is where all computations are performed. The other blocks in the processor are support units that either store the results produced by the datapath or help to determine what will happen in the next cycle. A typical datapath consists of an interconnection of basic combinational functions, such as logic (AND, OR, EXOR) or arithmetic operators (addition, multiplication, comparison, shift). Intermediate results are stored in registers. Different strategies exist for the implementation of datapaths—structured custom cells versus automated standard cells, or fixed hard-wired versus flexible field-programmable fabric. The choice of the implementation platform is mostly influenced by the trade-off between different design metrics such as area, speed, energy, design time, and reusability.
- **The control module** determines what actions happen in the processor at any given point in time. A controller can be viewed as a finite state machine (FSM). It consists of registers and logic, and thus is a sequential circuit. The logic can be implemented in different ways—either as an interconnection of basic logic gates (standard cells), or in a more structured fashion using programmable logic arrays (PLAs) and instruction memories.
- **The memory module** serves as the centralized data storage area. A broad range of different memory classes exist. The main difference between those classes is in the way data can be accessed, such as "read only" versus "read–write," sequential versus random access, or single-ported versus multiported access. Another way of differentiating between memories is related to their data-retention capabilities. Dynamic memory structures must be refreshed periodically to keep their data, while static memories keep their data as long as the power source is turned on. Finally, nonvolatile memories such as flash memories conserve the stored data even when the supply voltage is removed. A single processor might combine different memory classes. For example, random access memory can be used to store data, and read-only memory may store instructions.

- **The interconnect** network joins the different processor modules to one another, while the **input/output circuitry** connects to the outside world. For a long time, interconnections were an afterthought in the design process. Unfortunately, the wires composing the interconnect network are less than ideal and present a capacitive, resistive, and inductive load to the driving circuitry. As die sizes grow larger, the length of the interconnect wires also tends to grow, resulting in increasing values for these parasitics. Today, automated or structured design methodologies are being introduced that ease the deployment of these interconnect structures. Examples include *on-chip busses*, *mesh interconnect* structures, and even complete *networks on a chip*. Some components of the interconnect network typically are abstracted away on schematic block diagrams, such as the one shown in Figure 8-2, yet are of critical importance to the well-being of the design. These include the power- and clock-distribution networks. Early planning of these "service" networks can go a long way toward ensuring the correct operation of the integrated circuit.

The structure of Figure 8-2 may be repeated many times on a single die. Figure 8-3 shows an example of a *system on a chip*, which combines all the functions needed for the realization of a complete high-definition digital TV set. It combines two processors, memory units, specialized accelerators for functions such as MPEG (de)coding and data filtering, as well as a range of

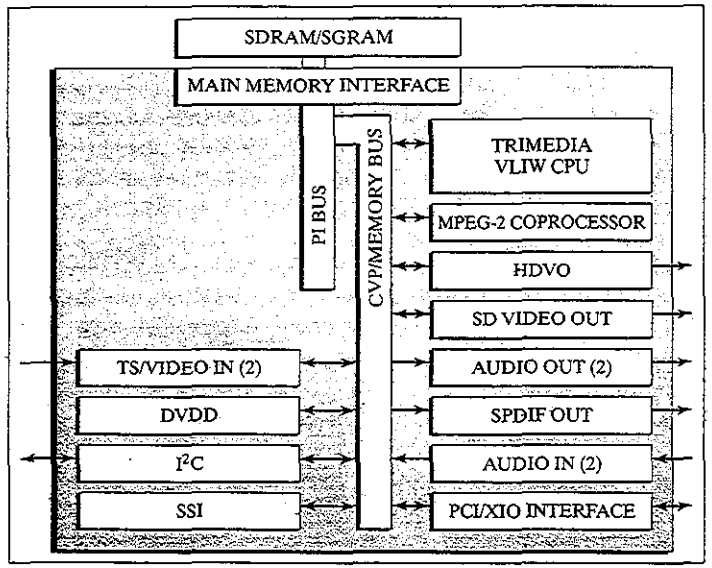


Figure 8-3 The "Nexperia" system on a chip [Philips99]. This single chip combines a general-purpose microprocessor core, a VLIW (very large instruction word) signal processor, a memory system, an MPEG coprocessor, multiple accelerator units, and input/output peripherals, as well as two system busses.

peripheral units. Other applications such as wireless transceivers or hard-disk read/write units may even include some sizable analog modules.

Choosing an effective implementation approach strongly depends upon the function of the modules under consideration. For example, memory units tend to be very regular and structured. A module compiler that stacks cells in an arraylike fashion is thus the preferred implementation approach. Controllers, on the other hand, tend to be unstructured, and other implementation approaches are desirable. The choice of the implementation strategy can have a tremendous effect on the quality of the final product. The challenge for the designer is to pick the style that meets the product specifications and constraints. What works well for one design may well be a disaster for another one.

Example 8.1 Trading Off Energy Efficiency and Flexibility

A design that embraces flexibility (or programmability) is very attractive from an application perspective. It allows for "late binding," in which the application can still be changed after the chip has gone to fabrication. Flexibility makes it possible to reuse a single design for multiple applications, or to upgrade the firmware of a component in the field, reducing the risk for the manufacturer. In contrast, a hard-wired component is totally fixed at manufacturing time and cannot be modified afterwards.

So, why not use flexible or programmable components for every possible design? As always, there is no free lunch. Flexibility comes at a price in both performance and energy efficiency. Providing programmability means adding overhead to implementation. For example, a programmable processor uses stored instructions and an instruction decoder to make a single datapath perform multiple functions. Most designers are not aware of the large cost of flexibility. The impact is illustrated in Figure 8-4, which compares the *energy*

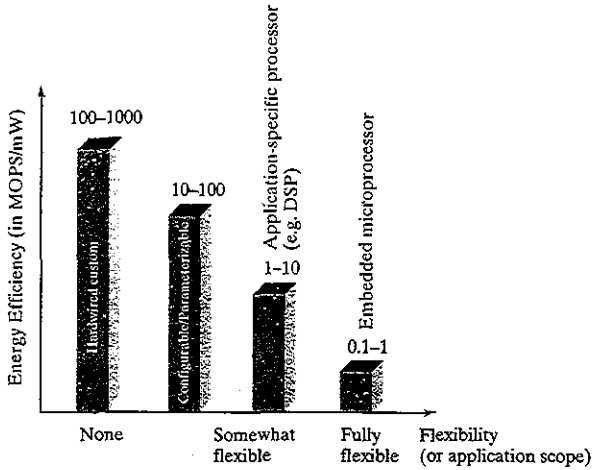


Figure 8-4 Trading off flexibility versus energy efficiency (in MOPS/mW or millions of operations per mJ of energy) for different implementation styles. The numbers were collected for a 0.25 µm CMOS process [Rabaey00].

efficiency—the number of operations that can be performed for a given amount of energy—of various implementation styles versus their *flexibility*—that is, the range of applications that can be mapped onto them. A staggering **three orders of magnitude** in variation can be observed. This clearly demonstrates that hard-wired or implementation styles with limited flexibility (such as configurable or parameterizable modules) are preferable when energy efficiency is a must.

In this and the following three chapters, we discuss, respectively, implementation techniques for random logic and controllers (this chapter), interconnect (Chapter 9), datapaths (Chapter 11), and memories (Chapter 12). Observe that the choice of the implementation approach can have a tremendous effect on the quality of the final product. Important aspects in the design of complex systems consisting of multiple blocks and thus deserving special attention are synchronization and timing (Chapter 10) and the power distribution network (Chapter 9). The distribution of clock signals and supply current has become one of the dominant problems in the design of state-of-the-art processors. A number of Design Methodology Inserts, interspersed between the chapters, address the design challenge posed by these complex components, and introduce the advanced design automation tools that are available to the designer. Inserts F, G, and H discuss design synthesis, verification, and test, respectively.

8.2 From Custom to Semicustom and Structured-Array Design Approaches

The viability of a microelectronics design depends on a number of (often) conflicting factors, such as performance in terms of speed or power consumption, cost, and production volume. For example, to be competitive in the market, a microprocessor has to excel in performance at a low cost to the customer. Achieving both goals simultaneously is only possible through large sales volumes. The high development cost associated with high-performance design is then amortized over many parts. Applications such as supercomputing and some defense applications present another scenario. With ultimate performance as the primary design goal, high-performance custom design techniques often are desirable. The production volume is small, but the cost of electronic parts is only a fraction of the overall system costs and thus not much of an issue. Finally, reducing the system size through integration, not performance, is the major objective in most consumer applications. Under these circumstances, the design cost can be reduced substantially by using advanced design-automation techniques, which compromise performance, but minimize design time. As noted in Chapter 1, the cost of a semiconductor device is the sum of two components:

- The *nonrecurring expense* (NRE), which is incurred only once for a design and includes the cost of designing the part.
- The *production cost per part*, which is a function of the process complexity, design area, and process yield.

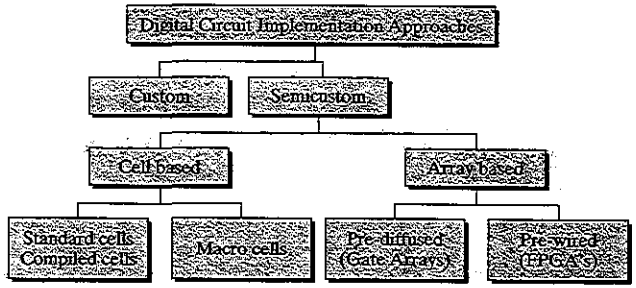


Figure 8-5 Overview of implementation approaches for digital integrated circuits (after [DeMicheli94]).

These economic considerations have spurred the development of a number of distinct implementation approaches ranging from high-performance, handcrafted design to fully programmable, medium-to-low performance designs. Figure 8-5 provides an overview of the different methodologies. In the sections that follow, we discuss first the custom design methodology, followed by the semicustom and array-based approaches.

8.3 Custom Circuit Design

When performance or design density is of primary importance, handcrafting the circuit topology and physical design seems to be the only option. Indeed, this approach was the only option in the early days of digital microelectronics, as is adequately demonstrated in the design of the Intel 4004 microprocessor (see Figure 8-5a). The labor-intensive nature of custom design translates into a high cost and a long *time to market*. Therefore, it can only be justified economically under the following conditions:

- The custom block can be reused many times (for example, as a library cell).
- The cost can be amortized over a large volume. Microprocessors and semiconductor memories are examples of applications in this class.
- Cost is not the prime design criterion, as it is in supercomputers or hypersupercomputers.

With continuous progress in the design-automation arena, the share of custom design reduces from year to year. Even in the most advanced high-performance microprocessors, such as the Intel Pentium® 4 processor (see Figure 8-6), virtually all portions are designed automatically using semicustom design approaches. Only the most performance-critical modules such as the phase locked-loops and the clock buffers are designed manually. In fact, library cell design is the only area where custom design still thrives today.

The amount of design automation in the custom-design process is minimal, yet some design tools have proven indispensable. In concert with a wide range of verification, simulation, extraction and modeling tools, layout editors, design-rule and electrical-rule checkers—as

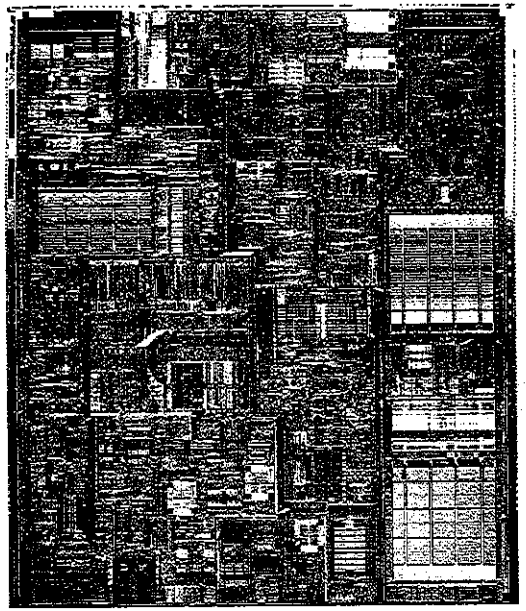


Figure 8-6 Chip microphotograph of Intel Pentium® 4 processor. It contains 42 million transistors, designed in a 0.18-μm CMOS technology. Its first generation runs at a clock speed of 1.5 GHz (Courtesy Intel Corp.).

described earlier in Design Methodology Insert A—are at the core of every custom-design environment. A excellent discussion of the opportunities and challenges of custom design can be found in [Grundman97].

8.4 Cell-Based Design Methodology

Since the custom-design approach proves to be prohibitively expensive, a wide variety of design approaches have been introduced over the years to shorten and automate the design process. This automation comes at the price of reduced integration density and/or performance. The following rule tends to hold: **the shorter the design time, the larger is the penalty incurred.** In this section, we discuss a number of design approaches that still require a full run through the manufacturing process for every new design. The *array-based design* approach discussed in the next section cuts the design time and cost even further by requiring only a limited set of extra processing steps or by eliminating processing completely.

The idea behind cell-based design is to reduce the implementation effort by *reusing* a limited library of cells. The advantage of this approach is that the cells only need to be designed and verified once for a given technology, and they can be reused many times, thus amortizing the design cost. The disadvantage is that the constrained nature of the library reduces the possibility

of fine-tuning the design. Cell-based approaches can be partitioned into a number of classes depending on the granularity of the library elements.

8.4.1 Standard Cell

The standard-cell approach standardizes the design entry level at the logic gate. A library containing a wide selection of logic gates over a range of fan-in and fan-out counts is provided. Besides the basic logic functions, such as inverter, AND/NAND, OR/NOR, XOR/XNOR, and flip-flops, a typical library also contains more complex functions, such as AND-OR-INVERT, MUX, full adder, comparator, counter, decoders, and encoders. A design is captured as a schematic containing only cells available in the library, or is generated automatically from a higher level description language. The layout is then automatically generated. This high degree of automation is made possible by placing strong restrictions on the layout options. In the standard-cell philosophy, cells are placed in rows that are separated by routing channels, as illustrated in Figure 8-7. To be effective, this requires that all cells in the library have identical heights. The width of the cell can vary to accommodate for the variation in complexity between the cells. As illustrated in the drawing, the standard-cell technique can be intermixed with other layout approaches to allow for the introduction of modules such as memories and multipliers that do not adapt easily or efficiently to the logic-cell paradigm.

An example of a design implemented in an early standard-cell design style is shown in Figure 8-8a. A substantial fraction of the area is devoted to signal routing. The minimization of the interconnect overhead is the most important goal of the standard-cell placement and routing tools. One approach to minimizing the wire length is to introduce feed-through cells (Figure 8-7) that make it possible to connect between cells in different rows without having to route around a complete row. A far more important reduction in wiring overhead is obtained by adding more

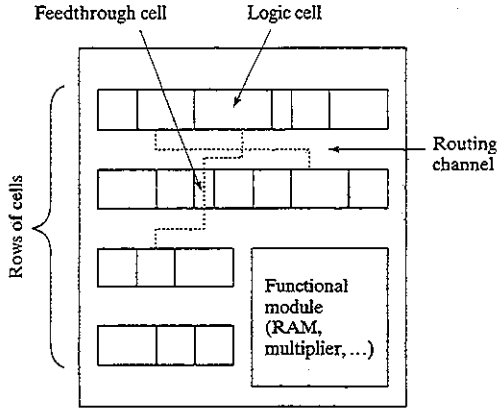


Figure 8-7 Standard-cell layout methodology.

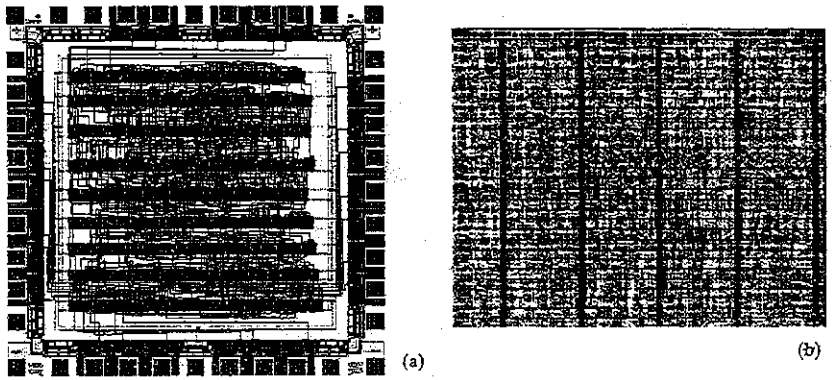


Figure 8-8 The evolution of standard-cell design. (a) Design in a three-layer metal technology. Wiring channels represent a substantial amount of the chip area. (b) Design in a seven-layer metal technology. Routing channels have virtually disappeared, and all interconnection is laid on top of the logic cells.

interconnect layers. The seven or more metal layers that are available in contemporary CMOS processes make it possible to all but eliminate the need for routing channels. Virtually all signals can be routed on top of the cells, creating a truly three-dimensional design. Figure 8-8b shows a fraction of a standard-cell design, implemented by using seven metal layers. The design achieves more than 90% density, which means that virtually all of the chip area is covered by logic cells, and that only a limited amount of the area is wasted for interconnect.

The design of a standard-cell library is a time-intensive undertaking that, fortunately, can be amortized over a large number of designs. Determining the composition of the library is a nontrivial task. A pertinent question is, Are we better off with a small library in which most cells have a limited fan-in, or is it more beneficial to have a large library with many versions of every gate (e.g., containing two-, three-, and four-input NAND gates, and different sizes for each of these gates)? Since the fan-out and load capacitance due to wiring are not known in advance, it used to be common practice to ensure that each gate had large current-driving capabilities, (i.e., employs large output transistors). While this simplifies the design procedure, it has a detrimental effect on area and power consumption. Today's libraries employ many versions of each cell, sized for different driving strengths, as well as performance and power consumption levels. It is left to the synthesis tool to select the correct cells, given speed and area requirements.

To make the library-based approach work, a detailed documentation of the cell library is an absolute necessity. The information should not only contain the layout, a description of functionality and terminal positioning, but it also must accurately characterize the delay and power consumption of the cell as a function of load capacitance and the input rise and fall times. Generating this information accounts for a large portion of the library generation effort. How to characterize logic and sequential cells is the topic of "Design Methodology Insert E."

Example 8.2 A Three-Input NAND-Gate Cell

To illustrate some of the preceding observations, the design of a three-input NAND standard-cell gate, implemented in a 0.18 μm CMOS technology, is depicted in Figure 8-9. The library actually contains five versions of the cell, supporting capacitive loads from 0.18 pF up to 0.72 pF and ranging in area from 16.4 μm^2 to 32.8 μm^2. The cell shown represents the low-performance, energy-efficient design corner, and uses high-threshold transistors to reduce leakage. The NMOS and PMOS transistors in the pull-down (-up) networks are both sized at a (W/L) ratio of approximately 8.

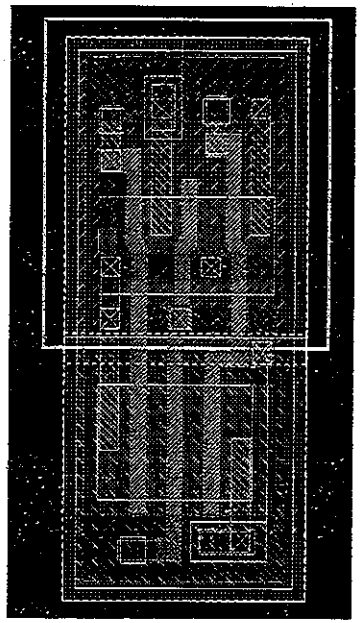


Figure 8-9 Three-input NAND standard cell (Courtesy ST Microelectronics).

Observe how the layout strategy follows the approach outlined in Figure D-2. Supply lines are distributed horizontally and shared between cells in the same row. Input signals are wired vertically using polysilicon. The input/output terminals are located throughout the cell body (as exemplified by the *pin* terminal in the layout drawing), in line with the over-the-cell wiring approach of today's standard-cell methodology.

The standard-cell approach has become immensely popular, and is used for the implementation of virtually all logic elements in today's integrated circuits. The only exceptions are when extreme high performance or low energy consumption is needed, or when the structure of the targeted function is very regular (such as a memory or a multiplier). The success of the standard-cell approach can be attributed to a number of developments, including the following:

- **The increased quality of the automatic cell placement and routing tools** in conjunction with the availability of multiple routing layers. In fact, it has been shown in a number of studies that the automated approach of today rivals if not surpasses manual design for complex, irregular logic circuits. This is a major departure from a couple of years ago, when automated layout carried a large overhead.
- **The advent of sophisticated *logic-synthesis* tools.** The logic-synthesis approach allows for the design to be entered at a high level of abstraction using Boolean equations, state machines, or register-transfer languages such as VHDL or Verilog. The synthesis tools automatically translate this specification into a gate netlist, minimizing a specific cost function such as area, delay, or power. Early synthesis tools—such as those used in the first half of the 1980s—focused mostly on two-level logic minimization. While this enabled automatic design mapping for the first time, it limited the area efficiency and the performance of the generated circuits. It is only with the arrival of *multilevel logic synthesis* in the late 1980s that automated design generation has really taken off. Today, virtually no designer uses the standard-cell approach without resorting to automatic synthesis. A more detailed description of the design synthesis process can be found in "Design Methodology Insert F" which follows this chapter.

Historical Perspective: The Programmable Logic Array

In the early days of MOS integrated circuit design, logic design and optimization was a manual and labor-intensive task. Karnaugh maps and Quine–McCluskey tables were the techniques of choice at that time. In the late 1970s, a first approach toward automating the tedious process of designing logic circuits emerged, triggered by two important developments:

- Rather than using the ad hoc approach to laying out logic circuits, a regular structured design approach was adopted called the *Programmable Logic Array* or PLA. This methodology enabled the automatic layout generation of two-level logic circuits, and, more importantly, it did so in a predictable fashion in terms of area and performance.
- The emergence of automated logic synthesis tools for two-level logic [Brayton84] made it possible to translate any possible Boolean expression into an optimized two-level (sum-of-products or product-of-sums) logic structure. Tools for the synthesis of sequential circuits followed shortly thereafter.

The idea of structured logic design gained a rapid foothold, and already in the mid-1980s it was adopted by major microprocessor design companies such as Intel and DEC. While PLAs are only sparingly used in today's semicustom logic design, the topic deserves some discussion (especially since PLAs might be poised for a come-back).

The concept is best explained with the aid of an example. Consider the following logic functions, for which we have transformed the equations into the sum-of-products format by using logic manipulations:

$$\begin{aligned} f_0 &= x_0x_1 + \overline{x_2} \\ f_1 &= x_0x_1x_2 + \overline{x_2} + \overline{x_0}x_1 \end{aligned} \tag{8.1}$$

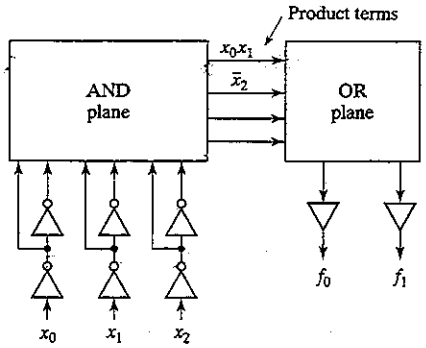


Figure 8-10 Regular two-level implementation of Boolean functions.

One important advantage of this representation is that a regular realization is easily conceived, as illustrated in Figure 8-10. A first layer of gates implements the AND operations—also called *product terms* or *minterms*—while a second layer realizes the OR functions, called the *sumterms*. Hence, a PLA is a rectangular macrocell, consisting of an array of transistors aligned to form rows in correspondence with product terms, and columns in correspondence with inputs and outputs. The input and output columns partition the array into two subarrays, called AND and OR planes, respectively.

The schematic of Figure 8-10 is not directly realizable since single-layer logic functions in CMOS are always inverting. With a few simple Boolean manipulations, Eq. (8.1) can be rewritten into a NOR–NOR format:

$$\begin{aligned} \overline{f_0} &= \overline{\overline{(\overline{x_0} + \overline{x_1})} + \overline{x_2}} \\ \overline{f_1} &= \overline{\overline{(\overline{x_0} + \overline{x_1} + \overline{x_2})} + \overline{x_2} + \overline{(x_0 + \overline{x_1})}} \end{aligned} \tag{8.2}$$

Problem 8.1 Two-Level Logic Representations

It is equally conceivable to represent Eq. (8.1) in a NAND–NAND format. In general, the NOR–NOR representation is preferred due to the prohibitively slow speed of large fan-in NAND gates. The NAND–NAND configuration is very dense, however, and thus can help to reduce power consumption. Derive the NAND–NAND representation for the example of Eq. (8.2).

Translating a set of two-level logic functions into a physical design now boils down to a "programming" task—that is, deciding where to place transistors in both the AND and the OR planes. This task is easily automated—hence, the early success of PLAs. An automatically generated PLA implementation of the logic functions described by Eq. (8.2) is shown in Figure 8-11. Unfortunately, the regular structure, while predictable,

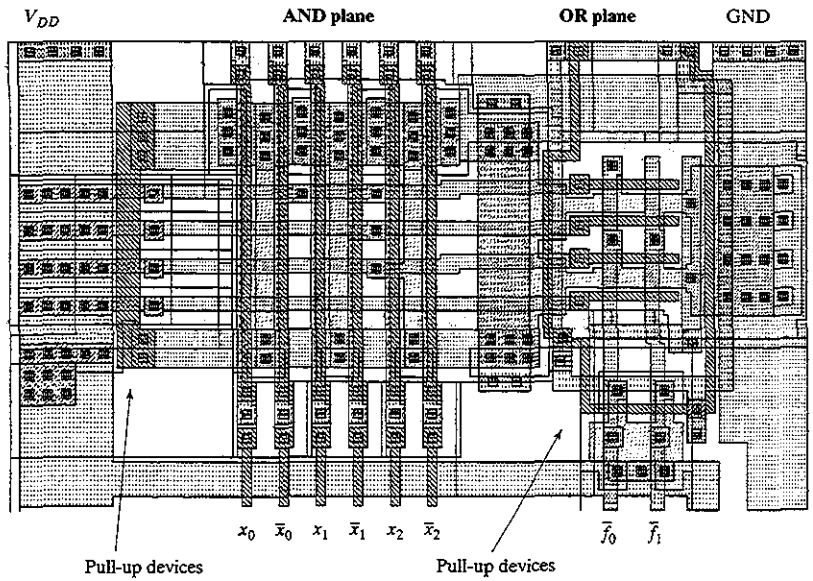


Figure 8-11 PLA layout implementing Eq. (8.2).

brings with it a lot of overhead in area and delay (as is quite visible in the layout), which was its ultimate demise in the semicustom design world. Those who are curious on how these AND and OR planes are actually implemented must wait until we get to Chapter 12, where we discuss the transistor-level implementation of PLAs. ■

8.4.2 Compiled Cells

The cost of implementing and characterizing a library of cells should not be underestimated. Today's libraries contain from several hundred to more than a thousand cells. These cells have to be redesigned with every migration to a new technology. Moreover, changes happen during the development of a single technology generation. For example, minimum metal widths or contact rules often are changed to improve yield. As a result, the complete library has to be laid out and characterized again. In addition, even an extensive library has the disadvantage of being discrete, which means that the number of design options is limited. When targeting performance or power, customized cells with optimized transistor sizes are attractive. With the increased impact of interconnect load, providing cells with adjusted driver sizes is an absolute necessity from both a performance and a power perspective [Sylvester98]—hence, the quest for automated (or compiled) cell generation.

A number of automated approaches have been devised that generate cell layouts on the fly, given the transistor netlists, but high-quality automatic cell layout has remained elusive. Earlier approaches relied on fixed topologies. Later approaches allowed for more flexibility in the transistor placement (e.g., [Hill85]). Layout densities close to what can be accomplished by a human

designer are now within reach, and a number of cell-generation tools are commercially available—for example [Cadabra01, Prolific01]:

Example 8.3 Automatic Cell Generation

The flow of a typical cell-generation process is illustrated with the example of a simple inverter (using the Abracad tool [Cadabra01]).

- The cell schematics are developed first. The Spice netlist is the starting point for the automatic layout generation. The generator examines the netlist and starts with transistor geometries. In case of a CMOS inverter, the cell contains just two transistors (see Figure 8-12a) .
- The tool proceeds along the same lines that a designer would follow. The transistors are placed in a cell architecture with predefined topology rules (Figure 8-12b). This architecture is common for all the cells in the library, including the cell height, power rails, pin placements, routing and contact styles.
- The cell is routed symbolically (Figure 8-12c).
- The routing is rearranged, and the cell is compacted to meet design rules and library preferences (Figure 8-12d).
- The final step cleans the cell of any remaining design rule errors and produces the final layout (Figure 8-12e).

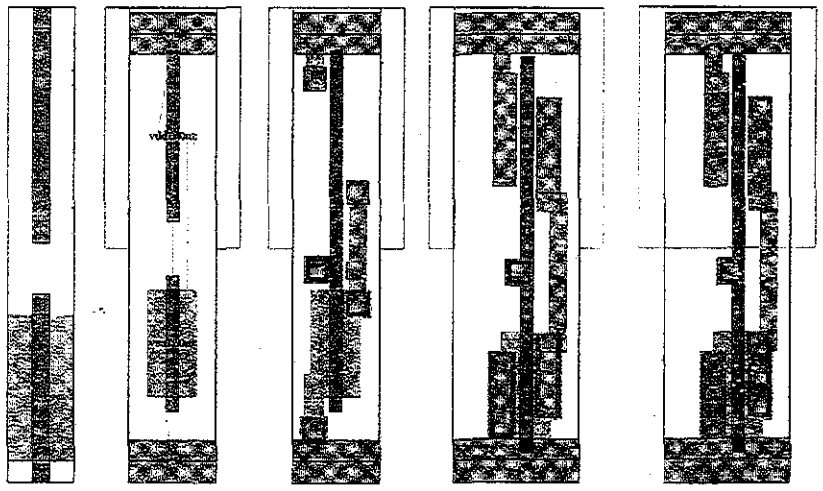


Figure 8-12 Automatic cell layout (a) initial transistor geometries, (b) placed transistors with flylines indicating intended interconnections, (c) initially routed cell, and (d) compacted cell, (e) finished cell.

8.4.3 Macrocells, Megacells and Intellectual Property

Standardizing at the logic-gate level is attractive for random logic functions, but it turns out to be inefficient for more complex structures such as multipliers, data paths, memories, and embedded microprocessors and DSPs. By capturing the specific nature of these blocks, implementations can be obtained that outperform the results of the standard ASIC design process by a wide margin. Cells that contain a complexity that surpasses what is found in a typical standard-cell library are called *macrocells* (or, sometimes, *megacells*). Two types of macrocells can be identified:

The Hard Macro represents a module with a given functionality and a predetermined physical design. The relative location of the transistors and the wiring within the module is fixed. In essence, a hard macro represents a custom design of the requested function. In some cases, the macro is parameterized, which means that versions with slightly different properties are available or can be generated. Multipliers and memories are examples: A hard multiplier macro may not only generate a 32×16 multiplier, but also an 8×8 one.

The advantage of the hard macro is that it brings with it all the good properties of custom design: dense layout, and optimized and predictable performance and power dissipation. By encapsulating the function into a macromodule, it can be reused over and over in different designs. This reuse helps to offset the initial design cost. The disadvantage of the hard macro is that it is hard to port the design to other technologies or to other manufacturers. For every new technology, a major redesign of the block is necessary. For this reason, hard macros are used less and less, and are employed mainly when the automated generation approach is far inferior or even impossible. Embedded memories and microprocessors are good examples of hard macros. They typically are provided by the IC manufacturer (who also provides the standard cell library), or the semiconductor vendor who has a particularly desirable function to offer (such as a standard microprocessor or DSP).

In the case of a macro that can be parameterized, a generator called the *module compiler* is used to create the actual physical layout. Regular structures such as PLAs, memories, and multipliers are easily constructed by abutting predesigned leaf cells in a two-dimensional array topology. All interconnections are made by abutment, and no or little extra routing is needed if the cells are designed correctly, which minimizes the parasitic capacitance. The PLA of Figure 8-11 is an example of such a configuration. The whole array can be constructed with a minimal number of cells. The generator itself is a simple software program that determines the relative positioning of the various leaf cells in the array.

Example 8.4 A Memory Macromodule

Figure 8-13 shows an example of a "hard" memory macrocell. The 256×32 SRAM block is generated by a parameterizable module generator. Besides creating the layout, the generator also provides accurate timing and power information. Modern memory generators also include an amount of redundancy to deal with defects.

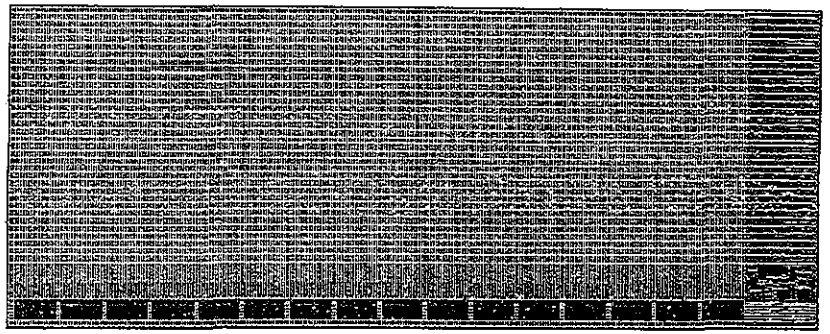


Figure 8-13 Parameterizable memory "hard" macrocell. This particular instance stores 256×32 (or 8192) bits. The decoders are located on the bottom. All eight address bits, as well as the 32 data input and output ports are placed on the right side of the cell. The total area of the memory module, implemented in a 0.18-μm CMOS technology, equals a mere 0.094 mm^2 (courtesy ST Microelectronics).

A Soft Macro represents a module with a given functionality, but without a specific physical implementation. The placement and the wiring of a soft macro may vary from instance to instance. This means that the timing data can only be determined after the final synthesis and placement and routing steps—in other words, the process is unpredictable. Yet, through intrinsic knowledge of the internal structure of the module, and by imposing precise timing and placement constraints on the physical generation process, soft macros most often succeed in offering well-defined timing guarantees. While stepping away from the advantages of the custom design process and relying on the semicustom physical design process, soft macros have the major advantage that they can be ported over a wide range of technologies and processes. This amortizes the design effort and cost over a wide set of designs.

Soft-macrocell generators come in different styles depending on the type of function they target. Virtually all of them can be classified as *structural generators*. Given the desired function and values for the requested parameters, the generator produces a netlist, which is an enumeration of the standard cells used and their interconnections. It also provides a set of timing constraints that the placement and routing tools should meet. The advantage of this approach is that the generator exploits its knowledge of the function under consideration to come up with clever structures that are more efficient than what logic synthesis would produce. For example, the design of fast and area-efficient multipliers has been the topic of decades of research.[1] The multiplier generator just incorporates the best of what the multiplier literature has to offer into an automated generation tool.

[1] Multiplier design is explained more thoroughly in Chapter 11, which discusses the design of arithmetic structures.

Example 8.5 Multiplier Macromodule

Two instances of an 8 × 8 multiplier module with different aspect ratios are shown in Figure 8-14. The modules are generated using the ModuleCompiler tool from Synopsys [ModuleCompiler01]. As can observed from the layout, a common standard-cell methodology is used to generate the physical artwork. The contribution of the macrocell generator is to translate the compact input description into an optimized connection of standard cells that meets the timing constraints. This "soft" approach has the advantage that modules with different aspect ratios can easily be generated. Also, porting between different manufacturing technologies is relatively easy.

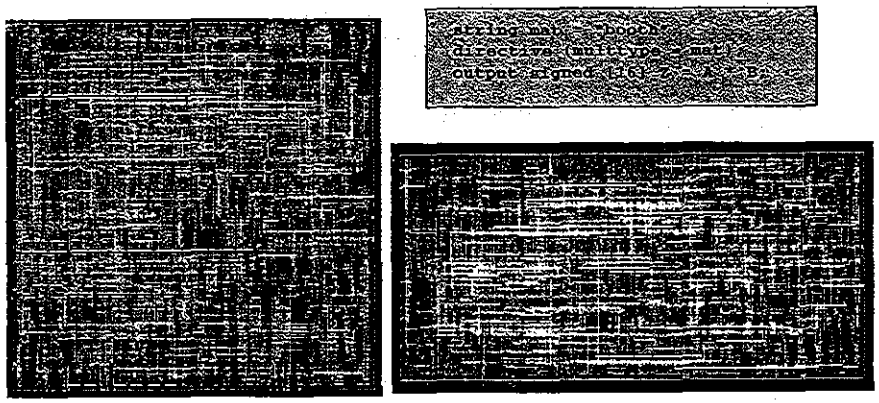


Figure 8-14 Multiplier "soft" macro modules. Both layouts implement an 8 × 8 booth multiplier, but with different aspects ratios. The compact input description to the compiler is shown in the gray box on top.

The availability of macromodules has substantially changed the semicustom design landscape in the 21st century. With the complexity of ICs going up exponentially, the idea of building every new IC from scratch becomes an uneconomic and nonplausible proposition. More and more, circuits are being built from reusable building blocks of increasing complexity and functionality. Typically, these modules are acquired from third-party vendors, who make the functions available through royalty or licensing agreements. Macromodels distributed in this style are called *intellectual property* (or IP) modules. This approach is somewhat comparable to the software world, where a large programming project typically makes intensive use of reusable software libraries. Good examples of commonly available intellectual property modules are embedded microprocessors and microcontrollers, DSP processors, bus interfaces such as PCI, and several special-purpose functional modules such as FFT and filter modules for DSP applications, error-correction coders for wireless communications, and MPEG decoding and encoding for video. Obviously, for an IP module to be useful, it has to not only deliver the hardware, but it also has to come with the appropriate software tools (such as compilers and debuggers for embedded processors), prediction models, and test benches. The latter are quite important because they represent the only means for the end user to verify that the module delivers the promised functionality and performance.

The design of a system on a chip is rapidly becoming an exercise in reuse at different levels of granularity. At the lowest level, we have the standard cell library; at a level higher, we have the functional modules such as multipliers, datapaths and memories; next, we have the embedded processors; and finally, the application-specific megacells. With more and more of the system functionality migrating onto a single die, it is not surprising to see that a typical ASIC consists of a blend of design styles and modules, embedding a number of hard or soft macrocells within a sea of standard cells.

Example 8.6 A Processor for Wireless Communications

Figure 8-15 shows an integrated circuit implementing the protocol stack for a wireless indoor communication system [Silva01]. The majority of the area is occupied by the embedded microprocessor (the Tensilica Xtensa processor [Xtensa01]) and its memory system. This processor allows for a flexible implementation of the higher levels of the protocol stack (Application/Network), and enables changes in the functionality of the chip, even after fabrication. The memory modules are generated using module compilers provided by the process vendor. The processor core itself is automatically generated from a higher level description in Verilog, and uses standard cells for its physical implementation. The advantage of using the "soft-core" approach is that the processor instruction set can be

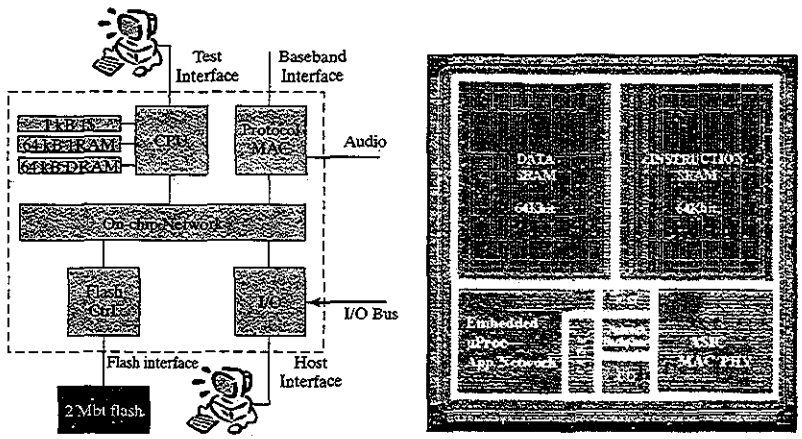


Figure 8-15 Wireless communications processor—an example of a hybrid ASIC design methodology. The processor combines an embedded microprocessor and its memory system with dedicated hardware accelerators and I/O modules. Observe also the on-chip network module [Silva01].

tailored to the application, and that the processor itself can easily be ported to different technologies and fabrication processes.

Implementing the computation-intensive parts of the protocol (MAC/PHY) on the microprocessor would require very high clock speeds and would unnecessarily increase the power dissipation of the chip. Fortunately, these functions are fixed and typically do not require a flexible implementation. Hence, they are implemented as an accelerator module in standard cells. The hard-wired implementation accomplishes the task of implementing a huge number of computations at a relatively low power level and clock frequency. The designer of a system on a chip is continuously faced with the challenge of deciding what is more desirable—after-the-fabrication flexibility versus higher performance at lower power levels. Fortunately, tools are emerging that help the designer to explore the overall design space and analyze the trade-offs in an informed fashion [Silva01]. Observe also that the chip contains a set of I/O interfaces, as well as an embedded network module, which helps to orchestrate the traffic between processor and the various accelerator and I/O modules.

The generation process of a macro module depends on the hard or soft nature of the block, as well as the level of design entry. In the following sections, we briefly discuss some commonly: used approaches.

8.4.4 Semicustom Design Flow

So far, we have defined the components that make up the cell-based design methodology. In this section, we discuss how it all comes together. Figure 8-16 details the traditional sequence of steps to design a semicustom circuit. The steps of what we call the design flow are enumerated in the figure, with a brief description of each:

1. **Design Capture** enters the design into the ASIC design system. A variety of methods can be used to do so, including schematics and block diagrams; hardware description languages (HDLs) such as VHDL, Verilog, and, more recently, C-derivatives such as SystemC; behavioral description languages followed by high-level synthesis; and imported intellectual property modules.
2. **Logic Synthesis** tools translate modules described using an HDL language into a *netlist*. Netlists of reused or generated macros can then be inserted to form the complete netlist of the design.
3. **Prelayout Simulation and Verification.** The design is checked for correctness. Performance analysis is performed based on estimated parasitics and layout parameters. If the design is found to be nonfunctional, extra iterations over the design capture or the logic synthesis are necessary.
4. **Floor Planning.** Based on estimated module sizes, the overall outlay of the chip is created. The global-power and clock-distribution networks also are conceived at that time.

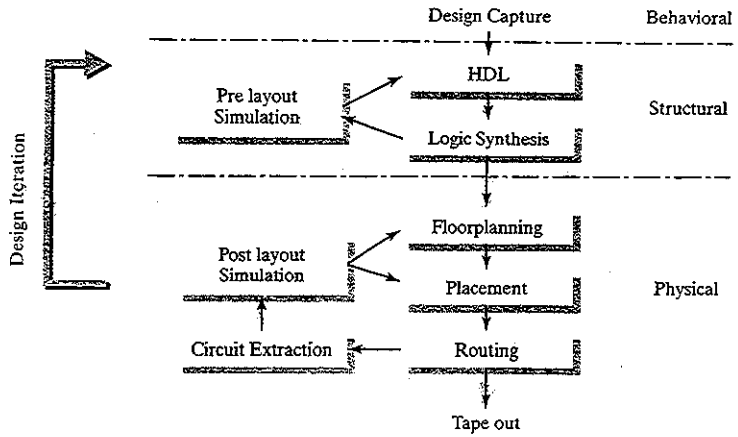
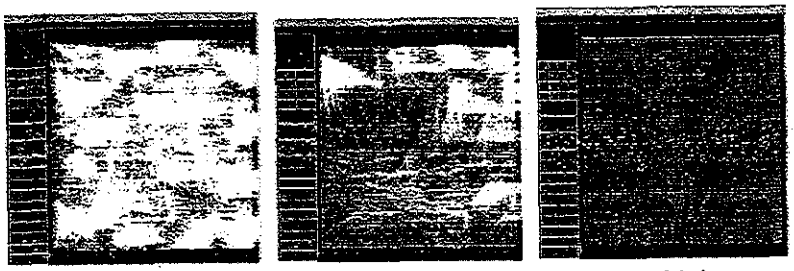


Figure 8-16 The Semicustom (or ASIC) design flow.

5. **Placement.** The precise positioning of the cells is decided.
6. **Routing.** The interconnections between the cells and blocks are wired.
7. **Extraction.** A model of the chip is generated from the actual physical layout, including the precise device sizes, devices parasitics, and the capacitance and resistance of the wires.
8. **Postlayout Simulation and Verification.** The functionality and performance of the chip is verified in the presence of the layout parasitics. If the design is found to be lacking, iterations on the floorplanning, placement, and routing might be necessary. Very often, this might not solve the problem, and another round of the structural design phase might be necessary.
9. **Tape Out.** Once the design is found to be meeting all design goals and functions, a binary file is generated containing all the information needed for mask generation. This file is then sent out to the ASIC vendor or foundry. This important moment in the life of a chip is called *tape out.*

While the design flow just described has served us well for many years, it was found to be severely lacking once technology reached the 0.25-μm CMOS boundary. With design technology proceeding into the deep submicron region, layout parasitics—especially from the interconnect—are playing an increasingly important role. The prediction models used by the logic and structural synthesis tools have a hard time providing accurate estimates for these parasitics. The chances that the generated design meets the timing constraints at the first try are thus very small (Figure 8-17a). The designer (or design team) is then forced to go through a number of costly iterations of synthesis followed by layout generation until an acceptable artwork that meets the timing constraints is obtained (Figure 8-17b and c). Each of these iterations may take several days—just routing a complex chip can take a week on the most advanced computers! The



(a) Initial design (b) Intermediate design (c) Final design

Figure 8-17 The timing closure process. The white lines indicate nets with timing violations. In each iteration of the design process, timing errors are removed by optimizing the logic, by insertion of buffers, by constraining the placement, or by streamlining the routing until an error-free design is obtained [Avanti01].

number of needed iterations continues to grow with the scaling of technology. This problem, called *timing closure*, made it obvious that new solutions and a change in design methodology were required.

The common answer is to create a tighter integration between the logical and physical design processes. If the logic synthesis tool, for example, also performs some part of the placement—or directs the placement—more precise estimates of the layout parameters can be obtained. Figure 8-18 shows an example of a design environment that merges RTL synthesis with first-order placement and routing. The resulting netlist is then fed into an optimization tool that performs the detailed placement and routing, while guaranteeing the timing constraints are met. While this approach has shown to be quite successful in reducing the number of design iterations, it throws quite a challenge at the design-tool developers. With the number of parasitic effects increasing with every round of technology scaling, the design optimization process that must take all this into account becomes exponentially complex as well. As a result, other approaches might be required as well. In the coming chapters, we will highlight "design solutions" that can help to alleviate some of these problems. An example is the use of regular and predictable structures, both at the logical and the physical level.

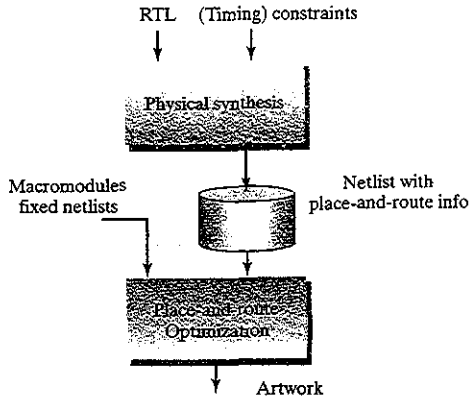


Figure 8-18 Integrated synthesis place-and-route reduces the number of iterations to reach timing closure in deep submicron.

8.5 Array-Based Implementation Approaches

While design automation can help reduce the design time, it does not address the time spent in the manufacturing process. All of the design methodologies discussed thus far require a complete run through the fabrication process. This can take from three weeks to several months, and it can substantially delay the introduction of a product. Additionally, with ever-increasing mask costs, a dedicated process run is expensive, and product economics must determine if this is a viable route.

Consequently, a number of alternative implementation approaches have been devised that do not require a complete run through the manufacturing process, or they avoid dedicated processing completely. These approaches have the advantage of having a lower NRE (nonrecurring expense) and are, therefore, more attractive for small series. This comes at the expense of lower performance, lower integration density, or higher power dissipation.

8.5.1 Prediffused (or Mask-Programmable) Arrays

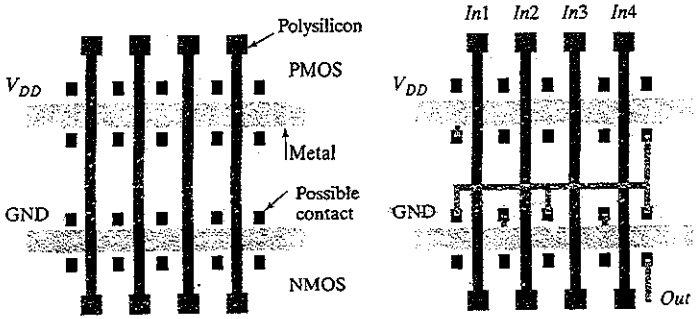
In this approach, batches of wafers containing arrays of primitive cells or transistors are manufactured by the vendors and stored. All the fabrication steps needed to make transistors are standardized and executed without regard to the final application.

To transform these uncommitted wafers into an actual design, only the desired interconnections have to be added, determining the overall function of the chip with only a few metallization steps. These layers can be designed and applied to the premanufactured wafers much more rapidly, reducing the turnaround time to a week or less.

This approach is often called the *gate-array* or the *sea-of-gates* approach, depending on the style of the prediffused wafer. To illustrate the concept, consider the gate-array primitive cell shown in Figure 8-19a. It comprises four NMOS and four PMOS transistors, polysilicon gate connections, and a power and ground rail. There are two possible contact points per diffusion area and two potential connection points for the polysilicon strips. We can turn this cell, which does not implement any logic function so far, into a real circuit by adding some extra wires on the metal layer and contact holes. This is illustrated in Figure 8-19b, where the cell is turned into a four-input NOR gate.

The original *gate-array* approach[2] places the cells in rows separated by wiring channels, as shown in Figure 8-20a. The overall look is similar to the traditional standard-cell technique. With the advent of extra metallization layers, the routing channels can be eliminated, and routing can

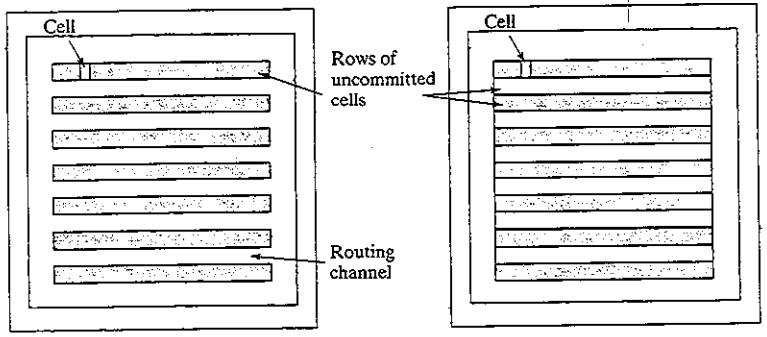
[2]This approach is often called the *channeled* gate array.



(a) Primitive gate-array cell

(b) Programmed cell, implementing a four-input NOR

Figure 8-19 An example of the gate-array approach.



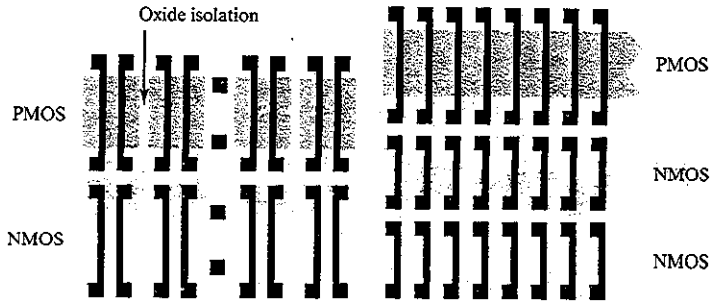
(a) Channelled

(b) Channelless (or sea of gates)

Figure 8-20 Gate-array architectures.

be performed on top of the primitive cells—occasionally leaving a cell unused. This channelless architecture, also called *sea of gates* (Figure 8-20b), yields an increased density, and makes it possible to achieve integration levels of millions of gates on a single die. Another advantage of the sea-of-gates approach is that it customizes the contact layer between metal-1 and diffusion and/or polysilicon, in contrast to the standard gate-array approach where the contacts are predefined (see Figure 8-19a). This extra flexibility leads to a further reduction in cell size.

The primary challenge when designing a gate-array (or sea-of-gates) template is to determine the composition of the primitive cell and the size of the individual transistors. A sufficient number of wiring tracks must be provided to minimize the number of cells wasted to interconnect. The cell should be chosen so that the prefabricated transistors can be utilized to a maximal extent over a wide range of designs. For example, the configuration of Figure 8-19 is well suited for the realization of four-input gates, but wastes devices when implementing two-input gates.



(a) Sea-of-gate cell using oxide-isolation between gates

(b) Cell using the gate-isolation approach and supporting multiple transistor sizes

Figure 8-21 Examples of sea-of-gates primitive cells (from [Veendrick92]).

Multiple cells are needed when implementing a flip-flop. A number of alternative cell structures are pictured in Figure 8-21 in a simplified format. In one approach, each cell contains a limited number of transistors (four to eight). The gates are isolated by means of *oxide isolation* (also called *geometry isolation*). The "dog-bone" terminations on the poly gates allow for denser routing. A second approach provides long rows of transistors, all sharing the same diffusion area. In this architecture, it is necessary to electrically turn off some devices to provide isolation between neighboring gates by tying NMOS and PMOS transistors to *GND* and V_{DD}, respectively. This technique is called *gate isolation*. This approach wastes a number of transistors to provide the isolation, but provides an overall higher transistor density.

Figure 8-22 shows the base cell for a gate-isolated gate array (from [Smith97]). The cell is one routing track wide, and contains one *p*-channel and one *n*-channel transistor. Also shown is a base cell containing all possible contact positions. There is room for 21 contacts in the vertical direction, which means that the cell has a height of 21 tracks.

It is worth observing that the cell in Figure 8-21b provides two rows of smaller NMOS transistors that can be connected in parallel if needed. Smaller transistors come in handy when implementing pass-transistor logic or memory cells. Sizing the transistors in the cells is a clear challenge. Due to the interconnect-oriented nature of the array-based design methodology, the propagation delay is generally dominated by the interconnect capacitance. This seems to favor larger device sizes that cause a larger area loss when unused. On the other hand, it is possible to construct larger transistors by putting several smaller devices in parallel.

Mapping a logic design onto an array of cells is a largely automated process, involving logic synthesis followed by placement and routing. The quality of these tools has an enormous impact on the final density and performance of a sea-of-gates implementation. Utilization factors in sea-of-gates structures are a strong function of the type of application being implemented. Utilization factors of nearly 100% can be obtained for regular structures such as memories. For

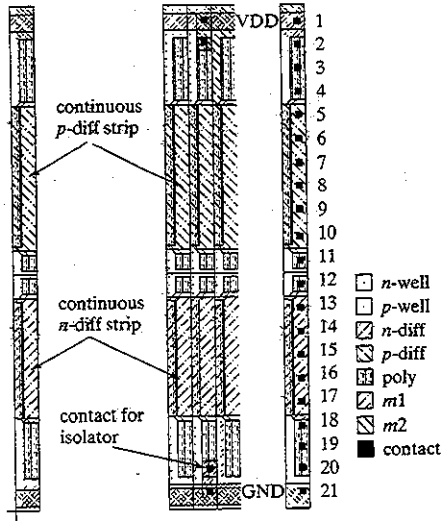


Figure 8-22 Base cell of gate-isolated gate array (from [Smith97]).

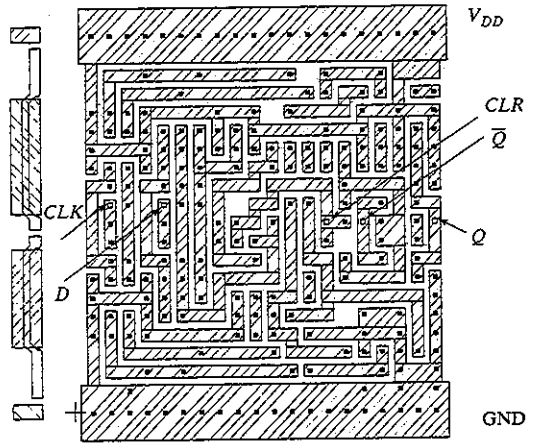


Figure 8-23 Flip-flop implemented in a gate-isolated gate-array library. The base cell is shown on the left (from [Smith97]).

other applications, utilization factors can be substantially lower (< 75%), due largely to wiring restrictions. Figure 8-23 shows an example of a flip-flop macrocell, implemented in a gate-isolated, gate-array library.

Similar to the scenarios unfolding in the standard-cell arena, designers of sea-of-gate arrays discovered that a design with a large number of gates also has large memory needs. Implementing these memory cells on top of the gate-array base-cells is possible, but not very efficient. A more efficient approach is to set aside some area for dedicated memory modules. The mixing of gate arrays with fixed macros is called the *embedded gate-array* approach. Other modules such as microprocessor and microcontrollers are also ideal candidates for embedding.

Example 8.7 Sea-of-Gates

An example of a sea-of-gates implementation is shown in Figure 8-24. The array has a maximum capacity of 300 K gates and is implemented in a 0.6-μm CMOS technology. The upper left part of the array implements a memory subsystem, which results in a regular modular layout. The rest of the array implements random logic.

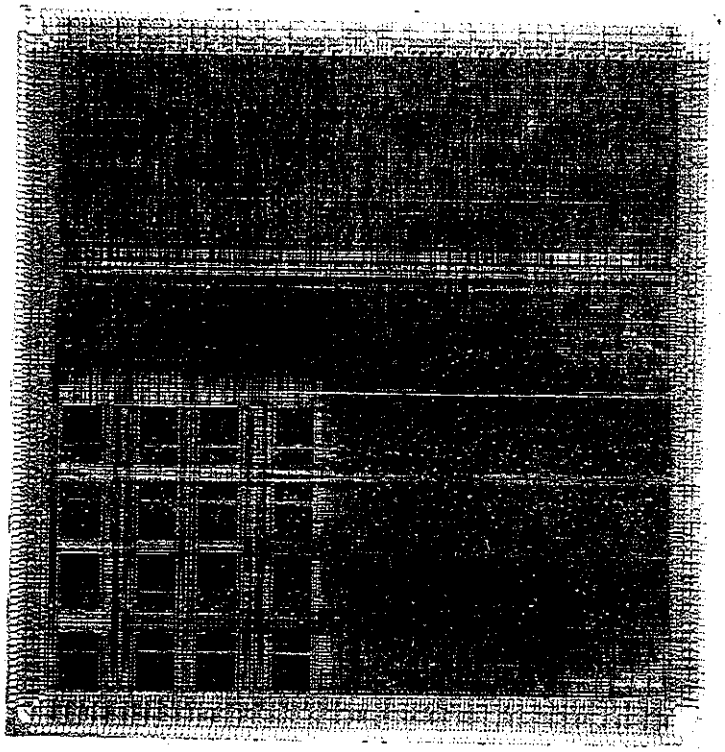


Figure 8-24 Gate-array die microphotograph (LEA300K) (Courtesy of LSI Logic.)

Design Consideration—Gate Arrays versus Standard Cells

In the 1980s and 1990s, when the majority of the chips were less than 50,000 gates, design cycles often could be measured in weeks or a few months. The two- or three-week savings in turnaround time for a gate-array design was then a significant portion of the total design cycle, more than enough to offset the additional die size. With today's deep-submicron processes and multimillion-gate complexities come longer design times, and the small reduction in turnaround time is no longer much of an issue. Furthermore, metallization has become the most time-consuming and yield-impacting part of the semiconductor manufacturing process, reducing further the advantage that gate arrays had to offer. Consequently, gate arrays have lost a lot of their luster. Another alternative for rapid prototyping—the prewired arrays discussed in the next section—has arisen, and it has taken a large portion out of the gate-array market.

Still, beware of dismissing the idea of the mask-programmable logic module as a thing of the past. A regular and fixed layout style has the advantage that load factors, wiring parasitics, and cross-coupling noise are easily and accurately estimated. This is in contrast to the standard-cell approach, where these values are ultimately only known after placement, routing, and extraction. One may consider populating sections of a large chip with a regular logic array consisting of uncommitted (prediffused) logic cells superimposed by a wiring grid. The actual programming of the module is performed by placing vias at predefined positions. As shown in Figure 8-25, the use of a via-programmable cross-point switch makes it possible to overlay a wide variety of wiring patterns on a regular repetitive wiring grid. It is the opinion of the authors that prediffused arrays have quite some life left into them.

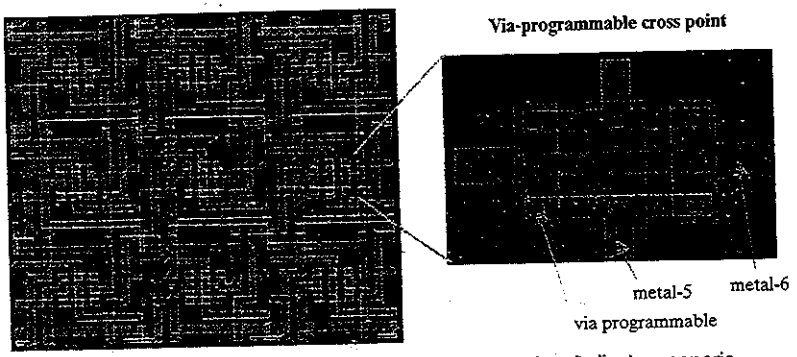


Figure 8-25 Via-programmable gate array. Vias are used to dedicate a generic wiring grid to a specific wiring pattern, resulting in predictable arrays [Pileggi02].

■

8.5.2 Prewired Arrays

While the prediffused arrays offer a fast road to implementation, it would be even more efficient if dedicated manufacturing steps could be avoided altogether. This leads to the concept of the preprocessed die that can be programmed in the field (i.e., outside the semiconductor foundry) to implement a set of given Boolean functions. Such a programmable, prewired array of cells is called a *field-programmable gate array (FPGA)*. The advantage of this approach is that the manufacturing process is completely separated from the implementation phase and can be amortized over a large number of designs. The implementation itself can be performed at the user site with

negligible turnaround time. The major drawback of this technique is a loss in performance and design density, compared with the more customized approaches.

Two main issues have to be addressed when attempting to implement a set of Boolean functions on top of a regular array of cells without requiring any processing steps:

1. How do we implement "programmable" logic—that is, logic that can committed to perform any possible Boolean function?
2. How and where do we store the *program*—also called the configuration—that dedicates the programmable array to a certain logic function?

The answer to the second question depends on the memory technology used. Since memory technology is the topic of a later chapter, we limit ourselves here to a high-level overview. In general, three different techniques can be identified:

- **The write-once or fuse-based FPGA.** The logic array is committed to a particular function by blowing "fuses" or by short-circuiting "antifuses." A fuse is a connection element that is short-circuited by default. A large current causes it to blow, and then it becomes an open circuit. The antifuse has the opposite behavior. An example of an antifuse implementation is shown in Figure 8-26 [El-Ayat89]. The advantage of the write-once approach is that the area overhead of the program memory (i.e., the fuses) is very small. But it has the important disadvantage of being *one-time programmable*. Circuit corrections or extensions are not possible, and new components are required for every design change.
- **The nonvolatile FPGA.** The program is stored in nonvolatile memory, which is memory that retains its value even when the supply voltage is turned off. Examples include EEPROM (*Electrically Erasable Programmable Read-Only Memory*) or Flash memories. Once programmed, the logic remains functional and fixed until a new programming round. The disadvantage of this approach is that nonvolatile memories require special steps in the manufacturing process, such as the deposition of ultrathin oxides. Also, high voltages

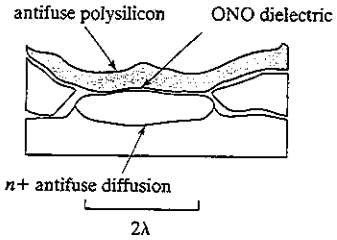


Figure 8-26 Example of antifuse. A 10-nm-thin layer (< 10 nm) of ONO (oxide–nitride–oxide) dielectric is deposited between conducting polysilicon and diffusion layers. The circuit is open by default, unless a large programming current is forced through it. This causes the dielectric to melt, and a permanent connection with fixed resistance is formed (from [Smith97]).

(> 10 V) are needed for the programming and erasure of the memory cells. Generating these high voltages and distributing them through the logic array adds extra complexity to the design.

- **The Volatile or RAM-Based FPGA.** This popular approach to programming the logic array employs volatile static RAM (random-access memory) cells for the storage of the program. Since these memories lose their stored contents when the FPGA is powered down, a reloading of the configuration from an external permanent memory is necessary every time the part is turned on. To program the component at start-up time, programming data is shifted serially into the part over a single line (or pin). For all practical purposes, one can consider the FPGA RAM cells to be configured as a giant shift register during that period. Once all memories are loaded, normal execution is started. The configuration time is proportional to the number of programmable elements. This can become excessive for today's larger FPGAs, which often feature more than one million gates. Recent parts therefore rely more and more on a parallel programming interface, allowing multiple cells to be programmed at the same time.

  In contrast to their nonvolatile counterparts, volatile FPGAs do not have special manufacturing process requirements, and can be implemented in a regular CMOS process. In addition, designers can reuse chips during prototyping. Logic can be modified and upgraded once deployed in the field—a customer can be sent a new configuration file to upgrade the chip, instead of sending a new chip. In addition, logic can be dynamically modified on the fly during execution. The latter approach is called *reconfiguration*, and it became quite popular in the late 1990s. In some sense, this brings a paradigm that was extremely successful in the world of programming (as embodied by the microprocessor) to the domain of logic design.

As for the first question, the answer is somewhat more extensive. Implementing a complex circuit in a programmable fashion requires that both the logic functions as well as the interconnect between them are realized in a configurable fashion. In the coming sections, we first discuss different ways of implementing programmable logic, followed by an overview of programmable interconnection. Finally, we detail a number of specific ways of putting the two together.

Programmable Logic

Similar to the situation in semicustom design, two fundamentally different approaches towards programmable logic are currently in vogue: array based and cell based.

Array-Based Programmable Logic Earlier we discussed how a *programmable logic array* (PLA) implements arbitrary Boolean logic functions in a regular fashion (see page 388). A similar approach can be applied to field-programmable devices as well. Consider, for example, the logic structure of Figure 8-27. A circle (o) at an intersection indicates a programmable connection—that is, an interconnect point that is either enabled or not. An inspection of the diagram reveals that it is equivalent to a PLA, where both the AND and OR planes can be programmed by selectively enabling connections. This approach allows for the implementation of arbitrary

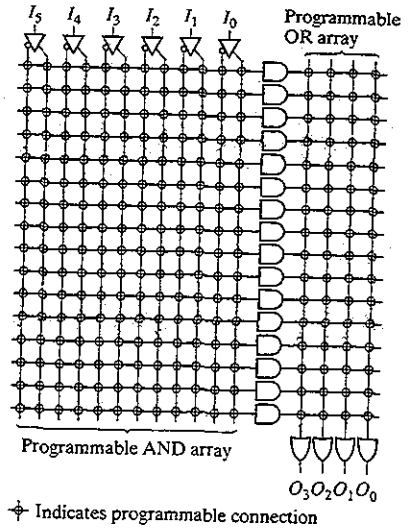


Figure 8-27 Fuse-programmable logic array (PLA).

logic functions in a two-level *sum-of-products* format. The AND plane creates the required minterms, while the OR plane takes the sum of a selected set of products to form the outputs. To include a given input variable (for instance, I_1) in a specific minterm, just close the switch at the intersection of the input signal and the minterm. Similarly, a minterm is included into an output by closing the appropriate connection in the OR plane. The functionality of PLA is restricted by the number of inputs, outputs, and minterms.

We can envision variations on this theme, some of which are represented in Figure 8-28. The dot (•) at the intersection of two lines represents a nonfusible, hard-wired link. The first structure represents the PROM architecture, in which the AND plane is fixed and enumerates all possible minterms. The second structure, called a *programmable array logic device* (PAL), is located at the other end of the spectrum, where the OR plane is fixed, and the AND plane is programmable. The PLA architecture is the most generic one for the implementation of arbitrary logic functions. The PROM and PAL structures, on the other hand, trade off flexibility for density and performance. Which structure to select depends strongly on the nature of the Boolean functions to be implemented. All these approaches are generally classified under the common term of *programmable logic devices* (or PLDs).

The single-array architecture of the PLA, PROM, and PAL structures in Figure 8-27 and Figure 8-28 becomes less attractive in the era of higher integration density. First of all, implementing very complex logic functions on a single, large array results in a loss of programming density and performance. Secondly, the arrays shown implement only combinational logic. To realize complete, sequential subdesigns, the presence of registers and/or flip-flops is an absolute requirement. These deficiencies can be addressed as follows:

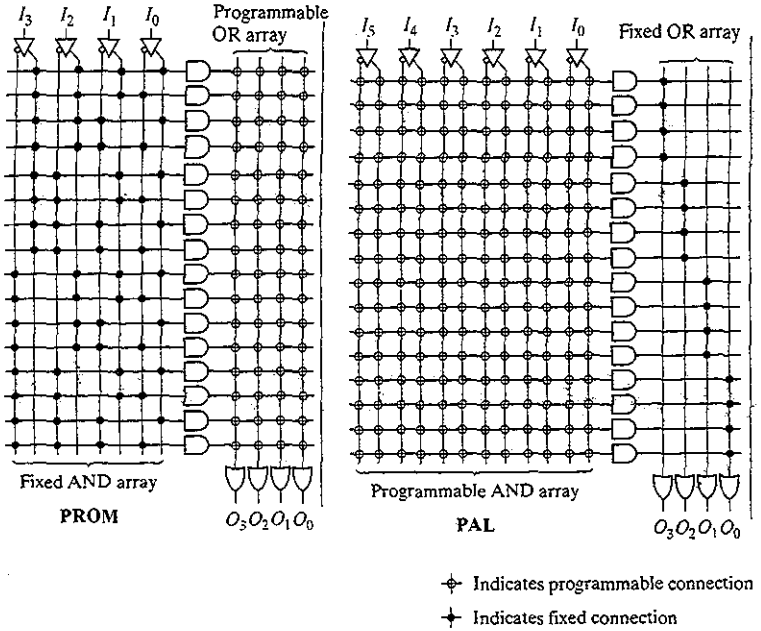


Figure 8-28 Alternative fuse-based programmable logic devices (or PLDs).

1. Partition the array into a number of smaller sections, often called macrocells.
2. Introduce flip-flops and provide a potential feedback from output signals to the inputs.

One example of how this can be accomplished is shown in Figure 8-29. The PAL consists of k macrocells, each of which can select from i inputs and features, at most, j product terms. Each macrocell contains a single register, which also is programmable—it can be configured as a *D*, *T*, *J-K*, or a clocked *S-R* flip-flop. The k output signals are fed back to the input bus, and thus form a subset of the i input signals.

The PLA approach to configurable logic has two distinct advantages:

- The structure is very regular, which makes the estimation of the parasitics quite easy, and enables accurate predictions of area, speed, and power dissipation.
- It provides an efficient implementation for logic functions that map well into a two-level logic description. Functions with a large fan-in fall into that category. Examples of such are finite-state machines used in controllers and sequencers.

On the other hand, the array structure has the disadvantage of higher overhead. Every intermediate node has a sizable capacitance, which negatively affects performance and power. This is especially true when parts of the array are underutilized—that is, if only some of the minterms are actively used.

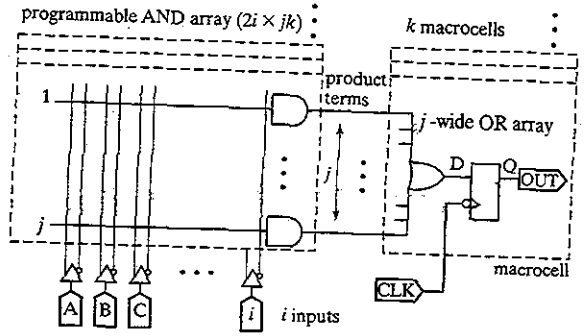


Figure 8-29 Schematic diagram of a PAL with i inputs, j minterms/macrocell and k macrocells (or outputs) [Smith97].

Example 8.8 Example of Programmed Macrocell

Figure 8-30 shows an example of how to program a PROM module. The structure is programmed to realize the logical functions used earlier during the discussion on PLAs (Eq. (8.1)):

$$f_0 = x_0x_1 + \overline{x_2}$$

$$f_1 = x_0x_1x_2 + \overline{x_2} + \overline{x_0}x_1$$

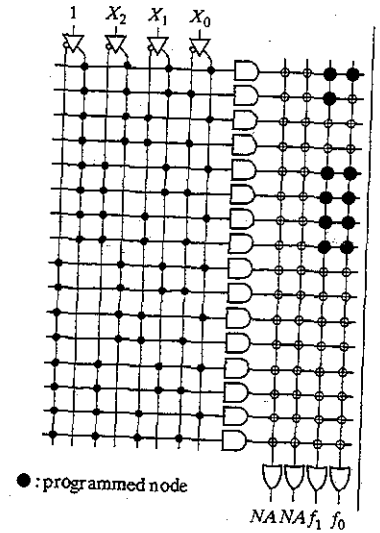


Figure 8-30 Programming a PROM.

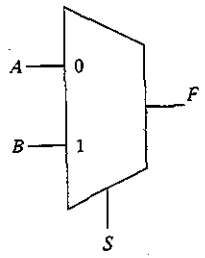
Observe that only a fraction of the array is used as the number of input (3) and output (2) variables are smaller than the dimensions of the 4 × 4 array. Unused input variables are tied either to 0 or 1. The large dots in the output planes represent programmed nodes. The reader is invited to repeat the exercise for the PLA and PAL modules presented in Figure 8-28 and Figure 8-29.

Cell-Based Programmable Logic The sum-of-products approach results in regular structures, and is very effective for logic functions that have a large fan-in such as finite-state machines. On the other hand, it performs rather poorly for logic that features a large fan-out, or that benefits from a multilevel logic implementation. (Arithmetic operations such as addition and multiplication are an example of such). Other approaches can be conceived that are more in line with the multilevel approach favored in the standard-cell and sea-of-gate approaches.

There are many ways to design a logic block that can be configured to perform a wide range of logic functions. One approach is to use *multiplexers as function generators*. Consider the two-input multiplexer of Figure 8-31, which implements the logic function F:

$$F = A \cdot \bar{S} + B \cdot S \tag{8.3}$$

By carefully choosing the connections between the variables X and Y and the input ports A, B, and S of the multiplexer, we can program it to perform ten useful logic operations on one or more of those inputs (see Figure 8-31).



(a)

Configuration			
A	**B**	**S**	**F =**
0	0	0	0
0	X	1	X
0	Y	1	Y
0	Y	X	XY
X	0	Y	$X\bar{Y}$
Y	0	X	$\bar{X}Y$
Y	1	X	X + Y
1	0	X	$\bar{X}$
1	0	Y	$\bar{Y}$
1	1	1	1

(b)

Figure 8-31 Using a two-input multiplexer (a) as a configurable logic block. By properly connecting the inputs A,B, and S to the input variables X or Y, or to 0 or 1, 10 different logic functions can be obtained (b).

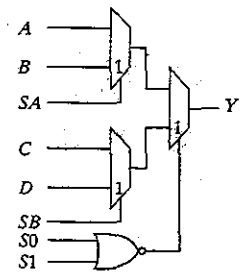


Figure 8-32 Logic cell as used in the Actel fuse-based FPGA.

A number of multiplexers can be combined to form more complex logic gates. Consider, for example, the logic cell of Figure 8-32, which is used in the Actel ACT family of FPGAs. It consists of three two-input multiplexers and a two-input NOR gate. The cell can be programmed to realize any two- and three-input logic functions, some four-input Boolean functions, and a latch.

Example 8.9 Programmable Logic Cell

It can be verified that the logic cell of Figure 8-32 acts as a two-input XOR under the programming conditions that follow. Assume the multiplexers select the bottom input signal when the control signal is high. We have the following:

$$A = 1;\ B = 0;\ C = 0;\ D = 1;\ SA = SB = In1;\ S0 = S1 = In2$$

As an exercise, determine the programming required for the two-input XNOR function. A three-input AND gate can be realized as follows:

$$A = 0;\ B = In1;\ C = 0;\ D = 0;\ SA = In2;\ SB = 0;\ S0 = S1 = In3$$

Finally, the largest function that can be realized is the four-input multiplexer. A, B, C, and D act as inputs, while SA, SB, and ($S0$ + $S1$) are control signals.

The "multiplexer-as-functional-block" approach provides configurability through programmable interconnections. The *lookup table* (LUT) method employs a vastly different strategy. To configure a fully programmable module with fan-in of i for a specific function, a two-bit large memory, called the lookup table, is programmed to capture the truth table of that function. The input variables serve as control inputs to a multiplexer, which picks the appropriate value from the memory. The idea is illustrated in Figure 8-33 for a two-input cell. To implement an EXOR function, the lookup table is loaded with the output column of the EXOR truth table, this is "0 1 1 0". For an input value of "0 0", the multiplexer selects the first value in the table ("0"), etc. With this approach, any logic function of two inputs can be realized by a simple (re)programming of the memory.

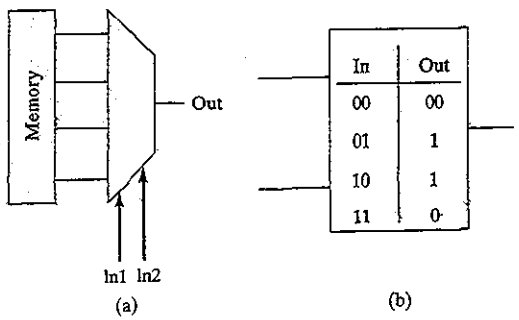


Figure 8-33 Configurable logic cell based on lookup table. (a) cell schematic; (b) programming the cell to implement an EXOR function.

As in the case of the multiplexer-based approach, more complex gates can be constructed. This is accomplished by either combining a number of LUTs, or by increasing the LUT sizes, or a combination of both. Additional functionality is provided by incorporating flip-flops.

Example 8.10 LUT-Based Programmable Logic Cell

Figure 8-34 shows the basic cell, called a *Configurable Logic Block* or CLB, used in the Xilinx 4000 FPGA series [Xilinx4000]. It combines two four-input LUTs feeding a three-

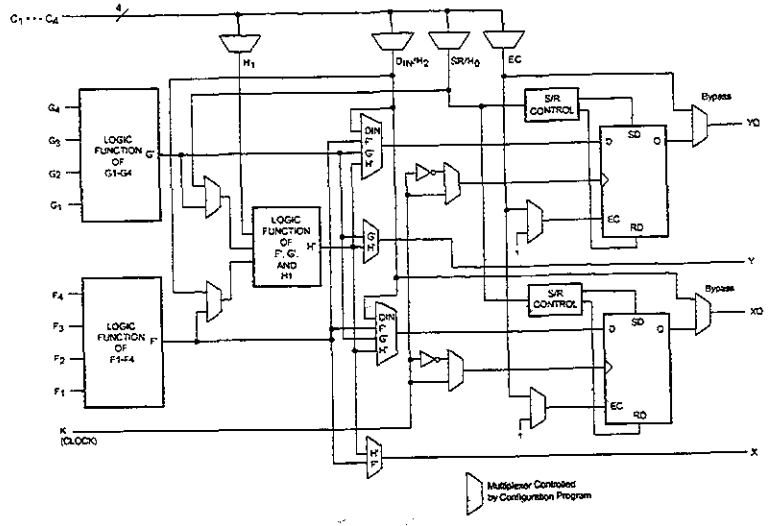


Figure 8-34 Simplified block diagram of XC4000 Series CLB (RAM and Carry-logic functions not shown) [Xilinx4000].

input LUT. The cell features two flip-flops, whose inputs can be any one of the LUT outputs *F*, *G*, or *H*, or an extra external input D_{in}, and whose outputs are available at the *XQ* and *YQ* output pins. The *X* and *Y* outputs export the outputs of the LUTs and make it possible to build more complex combinational functions. The cell has four extra inputs (*C1*...*C4*) that either can be used as inputs or as set/reset and clock-enable signals for the flip-flops.

Programmable Interconnect

So far, we have discussed in some depth how to make logic programmable. A compelling question is how to make interconnections between those gates changeable or programmable as well. To fully utilize the available logic cells, the interconnect network must be flexible and routing bottlenecks must be avoided. Speed is another prerequisite, since interconnect delay tends to dominate the performance in this style of design. At the same time, the reader should be aware that programmable interconnect comes at a substantial cost in performance in area, performance, and power. In fact, most of the power dissipation in field-programmable architectures is attributable to the interconnect network [George01].

Once again, we can differentiate between mask-programmable, one-time programmable and reprogrammable approaches. It also is worth differentiating between local cell-to-cell interconnections and global signals, such as clocks, that have to be distributed over the complete chip with low delay. In the local-area class, programmable wiring can be classified into two major groupings: array and switchbox routers.

Array-Based Programmable Wiring In this approach, wiring is grouped into routing channels, each of which contains a complete grid of horizontal and vertical wires. An interconnect wire can then be programmed into the structure by short-circuiting some of the intersections between horizontal and vertical wires (see Figure 8-35). This can be accomplished by providing a pass transistor at each of the cross points. Closing the interconnection means raising the control signal—by programming a "1" into the connected memory cell M (see Figure 8-36). This approach is prohibitive and expensive because it requires a large number of transistors and control signals. Also, the large number of transistors connected to each wire leads to a high fan-out, translating into delay and power consumption. A fuse is a more effective programmable connector. In this approach, each routing channel as a fully connected grid of horizontal and vertical interconnect wires, and a fuse is blown whenever a connection is not needed. Unfortunately, interconnect networks tend to be sparsely populated, which requires the interruption of an excessive number of switches and results in prohibitively long programming times.

To circumvent this problem, an *antifuse* can be used (as in Figure 8-26). Antifuses only need to be enabled when a connection is required in the routing channel. This represents a small fraction of the overall grid. Notice in Figure 8-35 how only two antifuses are needed to set up a connection. Be aware that this figure hides the programming circuitry. This operation is a one-time event and cannot be undone. The array-based wiring approach has thus been most successful in the write-once class of FPGAs. Circuit corrections or extensions are not possible, and new

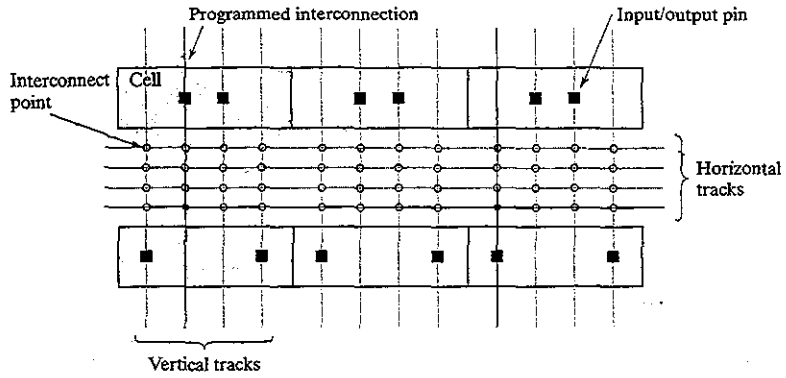


Figure 8-35 Array-based programmable wiring.

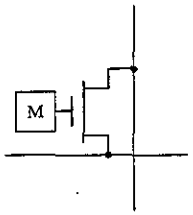


Figure 8-36 Programmable interconnect point. The memory cell controls the interconnection. A stored 0 and 1 mean an open or a closed circuit, respectively. The memory cell can be nonvolatile (EEPROM) or volatile (SRAM).

components are required for every design change. Providing true field (re)programmability requires a more efficient routing strategy.

Switch-Box-Based Programmable Wiring It's easy to imagine more efficient programmable-routing approach once we realize that the fully connected wiring grid represents major overkill. By restricting the number of routing resources and interconnect points, we can still manage to wire the desired interconnections, while drastically reducing the overhead. The disadvantage of this approach is that occasionally an interconnection cannot be routed. Most often, this can be addressed by remapping the design—for instance, by choosing another group of logic cells for a given function.

A large number of local interconnections can be accounted for by providing a mesh-like interconnection between neighboring cells. For instance, the outputs of each logic cell (LC) can be distributed to its neighbors to the north, east, south, and west. To account for interconnections between disjoint cells or to provide global interconnections, routing channels are placed between the cells containing a fixed number of uncommitted vertical and horizontal routing wires (Figure 8-37). At the junctions of the horizontal and vertical wires, RAM-programmable switching matrices (S-boxes) are

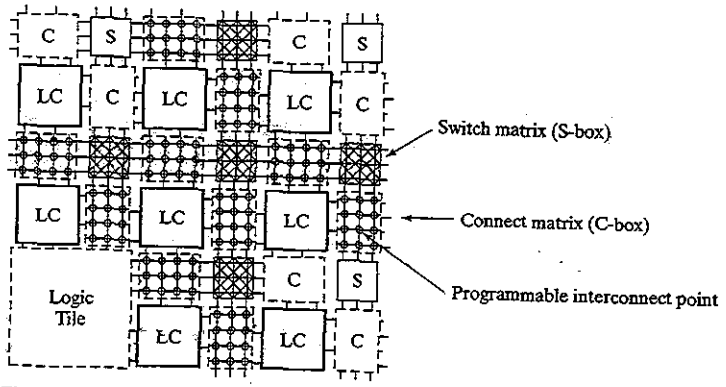


Figure 8-37 Programmable mesh-based interconnect network (Courtesy Andre Dehon and John Wawrzyniek.).

provided that direct the routing of the data. Cell inputs and outputs are connected to the global interconnect network by RAM-programmable interconnect points (C-box). Figure 8-38 provides a more detailed view, showing the transistor implementation of the switch and interconnect boxes. Be aware that the single pass-transistor implementation of the switches comes with a threshold-voltage drop. While advantageous from a power perspective, this reduced signal swing has a negative impact on the performance. Special design techniques such as zero-threshold devices, level restorers, or boosted control signals might be required.

The mesh architecture provides a flexible and scalable means for connecting a large number of components. It is quite efficient for local connections, as the number of switches traversed by a single interconnection is small and the fan-out is small. However, the mesh network does not lend itself well to global interconnections. The delay caused by the combination of the many

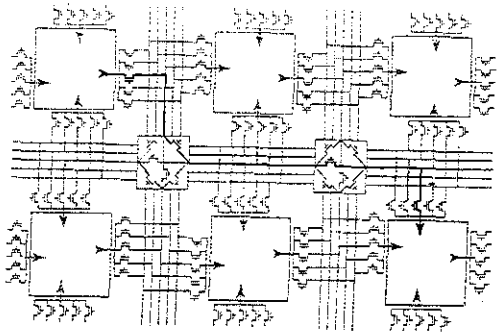


Figure 8-38 Transistor-level schematic diagram of mesh-based programmable routing network (Courtesy Andre Dehon and John Wawrzyniek.).

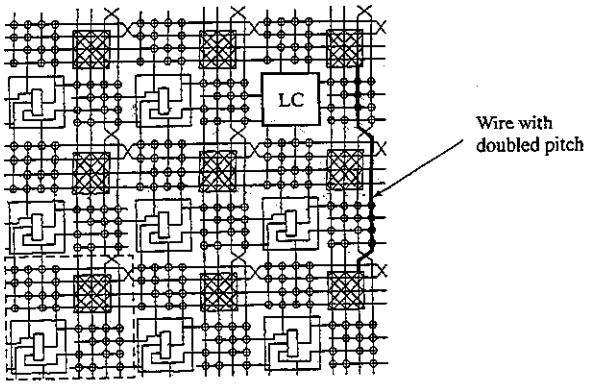


Figure 8-39 Programmable mesh-based interconnect architecture with overlaid 2 x 2 grid (Courtesy Andre DeHon and John Wawrzyniek.).

switches and the large capacitive load becomes excessive. Most mesh-based FPGA architectures therefore offer alternative wiring resources that allow for effective global wiring. One approach for accomplishing this task is shown in Figure 8-39. In addition to the standard S-box-to-S-box wiring, the network also includes wires connecting S-boxes that are two steps away from each other. Eliminating one S-box from an interconnection decreases the resistance. Similarly, we can include long wires that connect every 4^{th}, 8^{th}, or 16^{th} S-box. What we are creating, in fact, is a number of overlaying meshes with different granularity (single pitch, double pitch, etc.). Long wires are, by preference, mapped on the wiring meshes with the larger pitch.

Putting It All Together

A complete field-programmable gate array can now be assembled by joining logic-cell and interconnect approaches. Many alternative architectures can be (and have been) conceived. The most important decision to make at the start is the configuration style (write once, nonvolatile, volatile). This puts some constraints on the types of cells and interconnects that can be used. Giving a complete overview is out of the scope of this textbook, so we limit ourselves to two popular architectures, which are illustrative for the field. The interested reader can find more information in [Trimberger94], [Smith97], [Betz99], and [George01].

The Altera MAX Series [Altera01] The MAX family of devices (Figure 8-40) belongs to the class of nonvolatile FPGAs (often called EPLDs, or *Electrically Programmable Logic Devices*). It uses a PAL module, (as introduced in Figure 8-29) as the basic logic module. The module (called the *Logic Array Block* or LAB in Altera language) varies little over the members of the family: a wide programmable AND array followed by a narrow fixed OR array and programmable inversion. A LAB typically contains 16 macrocells.

The major differentiation lies in the interconnect architecture between the LABs. The smaller devices (MAX5000, MAX7000) use an array-based routing architecture. The backbone of the routing channel is formed by the outputs of all the macrocells, complemented with the direct chip inputs. These can be connected to the inputs of the LABs through programmable interconnect points. The advantage of this architecture, called the *Programmable Interconnect Array* or *PIA*, is that it is simple, and the routing delay between the blocks is totally predictable and fixed (see Figure 8-41). The disadvantage is that it does not scale very well. This is why the larger members of the series (MAX9000) have to resort to another scheme. With the number of macrocells reaching up to 560, the single-channel approach runs out of steam, and becomes slow. A mesh-based routing architecture has been opted for instead. Individual macrocells can connect to both row and column channels, which are quite wide (48 to 96 wires).

The EPLD approach delivers up to 15,000 logic gates, and typically is used when high performance is a necessity. Other architectures become desirable when more complex functions have to be implemented.

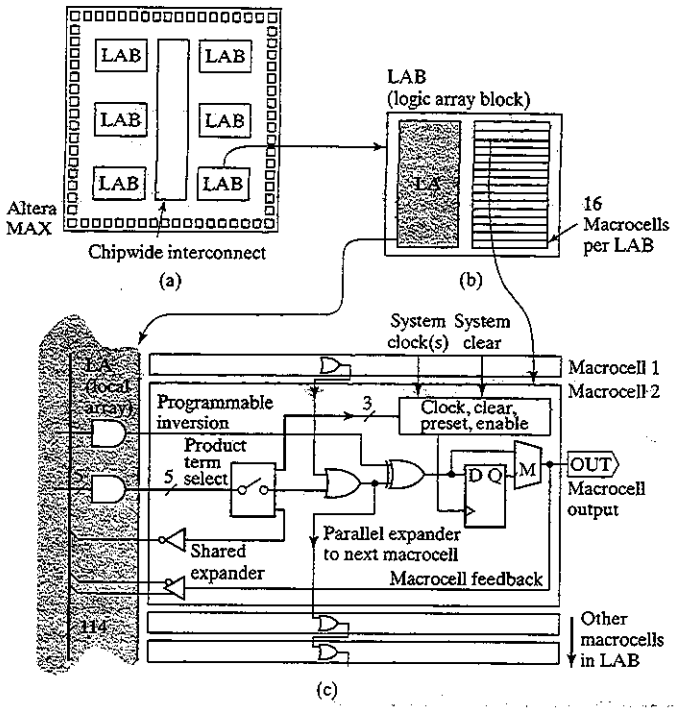


Figure 8-40 The Altera MAX Architecture. (a) Organization of logic and interconnect; (b) LAB module; (c) a MAX family macrocell. The expanders increase the number of products available by taking another pass through the logic array (from [Smith97]).

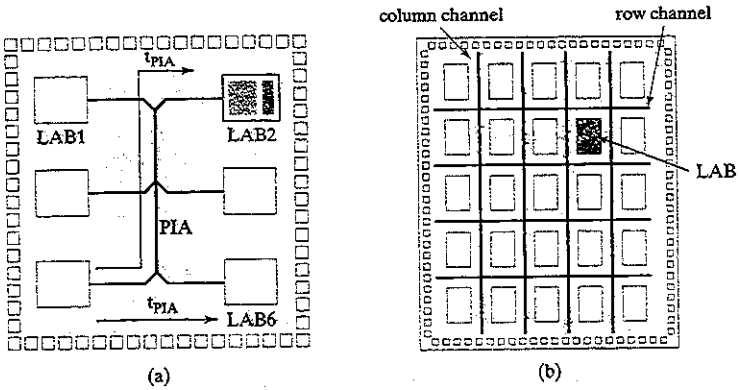


Figure 8-41 Interconnect architectures used in the Altera MAX series. (a) Array-based architecture used in MAX 3000-7000; (b) Mesh architecture of the MAX9000.

The Xilinx XC40xx Series This popular RAM-programmable device family combines the lookup table approach for the implementation of the logic cells, with a mesh-based interconnect network. The largest part in the series (XC4085) supports almost 100,000 gates using a 56×56 CLB array. The architecture of the CLB was shown in Figure 8-34. An interesting feature is that the CLB can also be configured as an array of Read/Write memory cells, using the memory lookup tables in the F' and G' blocks. Depending on the selected mode, a single CLB can be configured as either a 16×2, 32×1, or 16×1 bit array. This feature comes in handy, because it is typical for large modules of logic to need comparable amounts of storage.

The interconnect architecture is also quite rich, and combines a wide variety of wiring resources, as shown in Figure 8-42. The overlaid meshes consist of wire segments of lengths 1, 2, and 4. Some components also support direct connections, which link adjacent CLBs without using general wiring resources. Signals routed on the direct interconnect experience minimum wiring delay, as the fan-out is small. These *Directs* are especially effective in the implementation of fast arithmetic modules, which feature many critical local connections. To address global wiring, *long lines* are provided that form a grid of metal interconnect segments that run the entire length or width of the array. These are intended for high fan-out, time-critical signal nets, or nets that are distributed over long distances (such as buses). In addition, special wires are provided for the routing of the clocks.

One topic we have ignored so far in our discussion of configurable array structures is the input/output architecture. For maximum usability, it is crucial that the I/O pins of the component are flexible, and that they provide a wide range of options in terms of direction, logic levels, and drive strengths. One style of input/output block (IOB), used in the XC4000 series, is shown in Figure 8-43. It can be programmed to act as an input, output, or bidirectional port. It includes a flip-flop that can be programmed to be either edge triggered or level sensitive. The slew-rate

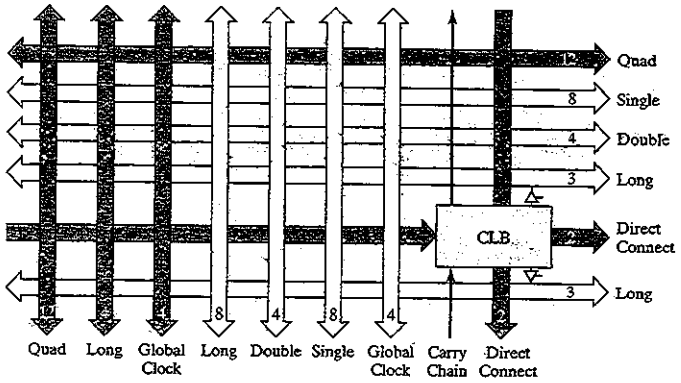


Figure 8-42 Interconnect architecture of the Xilinx XC4000 series. The numbers annotated on the diagram indicate the amount of each of the resources available.

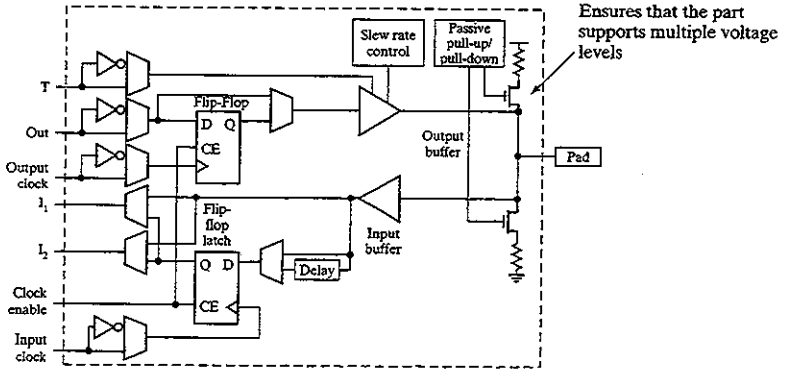


Figure 8-43 Programmable input/output Block of XC4000 series.

control provides variable drive strengths and allows for a reduction in the rise–fall time for non-critical signals.

Example 8.11 FPGA Complexity and Performance

To get an impression of what can be achieved with the volatile field-programmable components, consider the Xilinx 4025. It contains approximately 1000 CLBs organized in a 32×32 array. This translates into a maximum equivalent gate count of 25,000 gates. The chip contains 422 Kbits of RAM, used mostly for programming. A single CLB is specified to operate at 250 MHz. When taking into account the interconnect network and attempting more complex logic configurations such as adders, clock speeds between 20 and 50 MHz

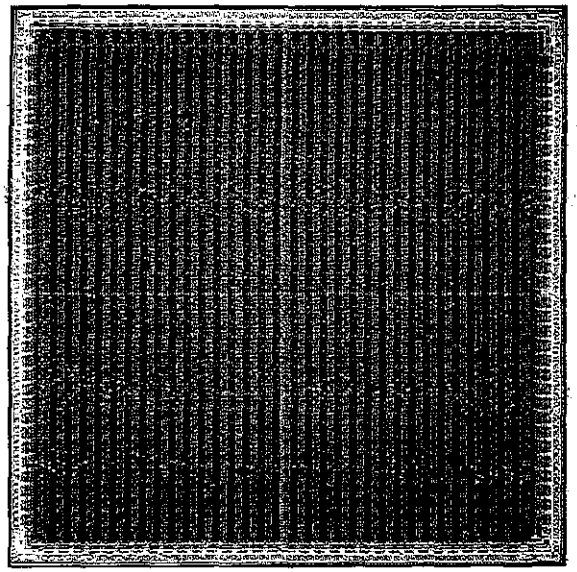


Figure 8-44 Chip microphotograph of XC4025 volatile FPGA (Courtesy of Xilinx, Inc.).

are attainable. To put the integration complexity in perspective, a 32-bit adder requires approximately 62 CLBs. A chip microphotograph of the XC4025 part is shown in Figure 8-44. The horizontal and vertical routing channels are easily recognizable.

Prewired logic arrays have rapidly claimed a significant part of the logic component market. Their arrival has effectively ended the era of logic design using discrete components represented by the TTL logic family. It is generally believed that the impact of these components is increasing with a further scaling of the technology. To make this approach successful, however, advanced software support in terms of cell placement, signal routing, and synthesis are required. Also, one should not ignore the overhead that flexibility brings with it. Programmable logic is at least 10 times less efficient in terms of energy and performance with respect to ASIC solutions. Hence, its scope has been mostly restricted to prototyping and small-volume applications so far. Yet, flexibility and reuse are alluring. Field-programmable components are bound to see a substantial growth in the years to come.

8.6 Perspective—The Implementation Platform of the Future

The designer of today's advanced systems-on-a-chip is offered a broad range of implementation choices. What approach is ultimately chosen is determined by a broad range of factors:

- performance, power and cost constraints
- design complexity
- testability
- time to market, or more precisely, time to revenue
- uncertainty of the market, or late changes in the design
- application range to be covered by the design
- prior experiences of the design team

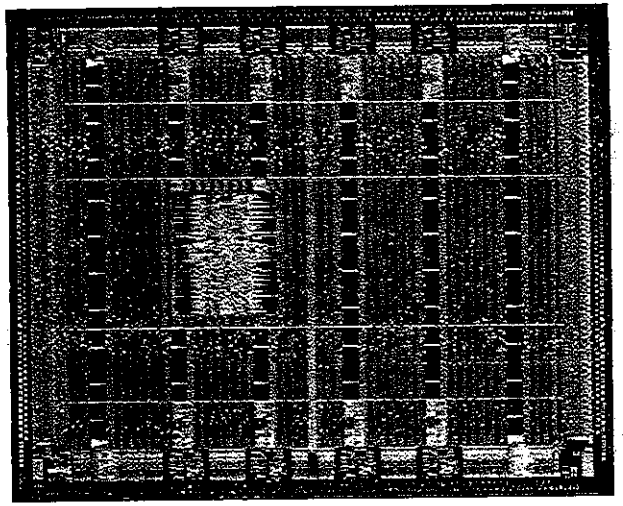
A number of these factors seem to imply a trend towards more flexible, programmable components that can be reused and that can be modified even after manufacturing. At the same time, solutions that offer the best "bang for the buck" most often end up the winners. Too much flexibility often results in ineffective and expensive solutions, which rapidly end up on the dust heap. Finding the balance between the two extremes is the ultimate challenge of the chip architects of today.

On the basis of these observations, it seems logical to assume that the implementation platform of the future will be a combination of the strategies we have discussed in this chapter, providing implementation efficiency and flexibility when and where needed. The system on a chip is becoming a combination of embedded microprocessors with their memory subsystems, DSPs, fixed ASIC-style hardware accelerators, parameterizable modules, and flexible logic implemented in FPGA style. How these components are balanced is a function of the application requirements and the intended market.

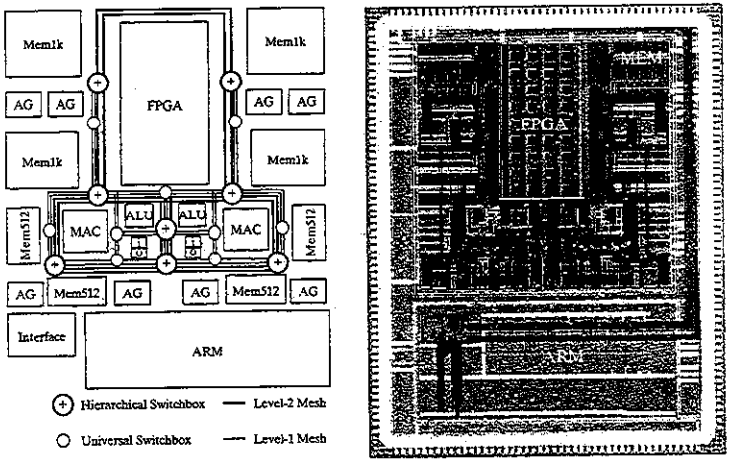
Example 8.12 Examples of Hybrid Implementation Platforms

Figure 8-45 shows two contrasting implementation platforms for wireless applications. The first device, the Virtex-II Pro from Xilinx [XilinxVirtex01] is centered around a large FPGA array. A PowerPC microprocessor is embedded in the center of the array. The processor provides an effective implementation approach for application-level functionality and system-level control. To provide higher performance for signal processing applications, an array of embedded 18 × 18 multipliers is added. These dedicated components offer a significant performance, power, and area advantage over a pure FPGA implementation of the same function. Finally, a number of very fast 3.125-Gbps transceivers are provided, allowing for high-speed serial communication off chip.

A somewhat contrasting approach is offered in the design of Figure 8-45b [Zhang00]. The center of this device is an ARM-7 embedded microprocessor, acting as the overall chip manager. Functions that need high performance and energy efficiency are offloaded to a configurable array of functional units such as multipliers, ALUs, memories, and address generators. These components can be combined dynamically into application-specific processors. The chip also provides an embedded FPGA array for functions that need bit-level granularity.



(a) The Xilinx Virtex-II Pro embeds a PowerPC microprocessor into an FPGA fabric (Courtesy Xilinx, Inc.).



(b) The Maia chip combines embedded microprocessor, configurable accelerators, and FPGA [Zhang00].

Figure 8-45 Examples of hybrid implementation platforms.

8.7 Summary

In this chapter, we have briefly scanned the complex world of design implementation strategies for digital integrated circuits. New implementation styles have rapidly emerged over the last few decades, presenting the designer with a wide variety of options. These design techniques and the accompanying tools are having a major impact on the way design is performed today, and make possible the exciting and impressive processors and application-specific circuits to which we have become so accustomed. We have touched on the following issues in this chapter:

- *Custom design*, where each transistor is individually handcrafted, offers the implementation from an area and performance perspective. This approach has become prohibitively expensive, and should be reserved for the design of the few critical modules in which extreme performance is required, or for often-reused library cells.
- The *semicustom* approach, based on the standard-cell methodology, is the workhorse of today's digital design industry. The advantage is the high degree of automation. The challenge is to deal with the impact of deep-submicron technologies.
- To deal with the increasing complexity of integrated circuits, designers increasingly rely on the availability of large *macrocells* such as memories, multipliers, and microprocessors. These modules are often provided by third-party vendors, and they have spurned a new industry focused on "*intellectual property*."
- Starting a new design for every new emerging application has become prohibitively expensive. The majority of the semiconductor market now focuses on flexible solutions that allow a single component to be used for a variety of applications, either through software programming or reconfiguration. *Configurable hardware* delays the time when the required function is actually committed to the hardware. Different approaches toward late binding also have been discussed. Delaying the binding time comes with an efficiency penalty: The more flexibility that is provided, the larger the impact on performance and power dissipation.

Undoubtedly, new design styles will come on the scene in the near future. Becoming familiar with the available options is an essential part of the learning experience of the beginning digital designer. We hope this chapter, although compressed, entices the reader to further explore the numerous possibilities. One final observation is as follows: Even with the increasing automation of the digital circuit design process, new challenges are continuously emerging—challenges that require the profound insight and intuition offered only by a human designer.

8.8 To Probe Further

The literature on design methodologies and automation for digital integrated circuits has exploded in the last few decades. Several reference works are worth mentioning:

- ASIC and FPGA design methodologies: [Smith97]
- FPGA architectures: [Trimberger94], [George01]

- System on a Chip: [Chang99]
- Design methodology and technology: [Bryant01]
- Design synthesis: [DeMicheli94]

State-of-the-art developments in the design automation domain are generally reported in the *IEEE Transactions on CAD*, the *IEEE Transactions on VLSI Systems*, and the *IEEE Design and Test Magazine*. Premier conferences are, among others, the Design Automation Conference (DAC) and the International Conference on CAD (ICCAD). The web sites of the major Electronic Design Automation Companies (Cadence, Synopsys, Mentor, etc.) provide a treasure of information as well.

References

[Altera04] Altera Device Index, *http://www.altera.com/products/devices/dev-index.html*, 2001.

[Avanti01] Saturn Efficient and Concurrent Logical and Physical-Optimization of SoC Timing, Area and Power, *http://www.synopsis.com/product/avmrg/saturn_ds.html*.

[Betz99] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, 1999.

[Brayton84] R. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.

[Bryant01] R. Bryant, T. Cheng, A. Kahng, K. Keutzer, W. Maly, R. Newton, L. Pileggi, J. Rabaey, and A. Sangiovanni-Vincentelli, "Limitations and Challenges of Computer-Aided Design Technology for CMOS VLSI," *IEEE Proceedings*, pp. 341–365, March 2001.

[Cadabra01] AbraCAD Automated Layout Creation, *http://www.cadabratech.com/?id=145products*, Cadabra Design Automation.

[Chang99] H. Chang et al., "Surviving the SOC Revolution: A Guide to Platform-Based Design," Kluwer Academic Publishers. 1999.

[DeMicheli94] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, 1994.

[El-Ayat89] K. El-Ayat, "A CMOS Electrically Configurable Gate Array," *IEEE Journal of Solid State Circuits*, vol. SC-24, no. 3, pp. 752–762, June 1989.

[George01] V. George and J. Rabaey, *Low-Energy FPGAs*, Kluwer Academic Publishers, 2001.

[Grundman97] W. Grundmann, D. Dobberpuhl, R. Allmon, and N. Rethman "Designing High-Performance CMOS Processors Using Full Custom Techniques," *Proceedings Design Automation Conference*, pp. 722–727, Anaheim, June 1997.

[Hill85] D. Hill, "S2C—A Hybrid Automatic Layout System," *Proc. ICCAD-85*, pp. 172–174, November 1985.

[ModuleCompiler01] Synopsys Module Compiler Datasheet, *http://www.synopsys.com/products/datapath/datapath.html*, Synopsys, Inc.

[Philips99] The Nexperia System Silicon Implementation Platform, *http://www.semiconductors.philips.com/platforms/nexperia/*, Philips Semiconductors.

[Pileggi02] Pileggi, Schmit et al., "*Via Patterned Gate Array*," CMU Center for Silicon System Implementation Technical Report Series, no. CSSI 02-15, April 2002.

[Prolific01] The ProGenensis Cell Compiler, *http://www.prolificinc.com/progenesis.html*, Prolific, Inc.

[Rabaey00] J. Rabaey, "Low-Power Silicon Architectures for Wireless Applications," *Proceedings ASPDAC Conference*, Yokohama, January 2000.

[Silva01] J. L. da Silva Jr., J. Shamberger, M. J. Ammer, C. Guo, S. Li, R. Shah, T. Tuan, M. Sheets, J. M. Rabaey, B. Nikolic, A. Sangiovanni-Vincentelli, P. Wright, "Design Methodology for PicoRadio Networks," *Proc. DATE Conference*, Munich, March 2000.

[Smith97] M. Smith, *Application-Specific Integrated Circuits*, Addison-Wesley, 1997.

[Sylvester98] D. Sylvester and K. Keutzer, "Getting to the Bottom of Deep Submicron," *Proc. ICCAD Conference*, pp. 203, San Jose, November 1998.

[Trimberger94] S. Trimberger, *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, 1994.

[Veendrick92] H. Veendrick, *MOS IC's: From Basics to ASICS*, Wiley-VCH, 1992.

[Xilinx4000] The Xilinx-4000 Product Series, *http://www.xilinx.com/apps/4000.htm*, Xilinx, Inc.

[XilinxVirtex01] Virtex-II Pro Platform FPGAs, *http://www.xilinx.com/xlnx/xil_prodcat_landingpage.jsp?title=Virtex-II+Pro+FPGAs*, Xilinx, Inc.

[Xtensa01] Xtensa Configurable Embedded Processor Core, *http://www.tensilica.com/technology.html*, Tensilica.

[Zhang00] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, and J. Rabaey, "A 1 V Heterogeneous Reconfigurable Processor IC for Baseband Wireless Applications," *Proc. ISSCC*, pp. 68–69, February 2000.

Exercises

For problems and exercises on design methodology, please check **http://bwrc.eecs.berkeley.edu/IcBook**.

[Jouppi84] N. Jouppi, *Timing Verification and Performance Improvement of MOS VLSI Designs*, Ph. D. diss., Stanford University, 1984.

[Ousterhout83] J. Ousterhout, "Crystal: A Timing Analyzer for nMOS VLSI Circuits," *Proc. 3rd Caltech Conf. on VLSI* (Bryant ed.), Computer Science Press, pp. 57–69, March 1983.

[PathMill] PathMill: Transistor-Level Static Timing Analysis, *http://www.synopsys.com/products/analysis/pathmill_ds.html*, Synopsys, Inc.

CHAPTER

11

Designing Arithmetic Building Blocks

Designing adders, multipliers, and shifters for performance, area, or power

Logic and system optimizations for datapath modules

Power-delay trade-offs in datapaths

11.1 Introduction

After the in-depth study of the design and optimization of the basic digital gates, it is time to test our acquired skills on a somewhat larger scale and put them in a more system-oriented perspective.

We will apply the techniques of the previous chapters to design a number of circuits often used in the datapaths of microprocessors and signal processors. More specifically, we discuss the design of a representative set of modules such as adders, multipliers, and shifters. The speed and power of these elements often dominates the overall system performance. Hence, a careful design optimization is required. It rapidly becomes obvious that the design task is not straightforward. For each module, multiple equivalent logic and circuit topologies exist, each of which has its own positives and negatives in terms of area, speed, or power.

Although far from complete, the analysis presented helps focus on the essential trade-offs that must be made in the course of the digital design process. You will see that optimization at only one design level—for instance, through transistor sizing only—leads to inferior designs. A global picture is therefore of crucial importance. Digital designers focus their attention on the gates, circuits, or transistors that have the largest impact on their goal function. The noncritical parts of the circuit can be developed routinely. We also may develop first-order performance models that foster understanding of the fundamental behavior of a module. The discussion also clarifies which computer aids can help to simplify and automate this phase of the design process.

Before analyzing the design of the arithmetic modules, a short discussion of the role of the datapath in the digital-processor picture is appropriate. This not only highlights the specific design requirements for the datapath, but also puts the rest of this book in perspective. Other processor modules—the input/output, controller, and memory modules—have different requirements and were discussed in Chapter 8. After an analysis of the area-power-delay trade-offs in the design of adders, multipliers, and shifters, we will use the same structures to illustrate some of the power-minimization approaches introduced in Chapter 6. The chapter concludes with a short perspective on datapath design and its trade-offs.

11.2 Datapaths in Digital Processor Architectures

We introduced the concept of a digital processor in Chapter 8. Its components consist of the datapath, memory, control, and input/output blocks. The datapath is the core of the processor—this is where all computations are performed. The other blocks in the processor are support units that either store the results produced by the datapath or help to determine what will happen in the next cycle. A typical datapath consists of an interconnection of basic combinational functions, such as arithmetic operators (addition, multiplication, comparison, and shift) or logic (AND, OR, and XOR). The design of the arithmetic operators is the topic of this chapter. The

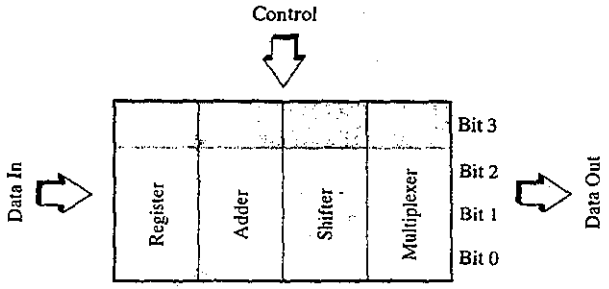


Figure 11-1 Bit-sliced datapath organization.

intended application sets constraints on the datapath design. In some cases, such as in personal computers, processing speed is everything. In most other applications, there is a maximum amount of power that is allowed to be dissipated, or there is maximum energy available for computation while maintaining the desired throughput.

Datapaths often are arranged in a *bit-sliced* organization, as shown in Figure 11-1. Instead of operating on single-bit digital signals, the data in a processor are arranged in a *word*-based fashion. Typical microprocessor datapaths are 32 or 64 bits wide, while the dedicated signal processing datapaths, such as those in DSL modems, magnetic disk drives, or compact-disc players are of arbitrary width, typically 5 to 24 bits. For instance, a 32-bit processor operates on data words that are 32 bits wide. This is reflected in the organization of the datapath. Since the same operation frequently has to be performed on each bit of the data word, the datapath consists of 32 bit slices, each operating on a single bit—hence the term *bit sliced*. Bit slices are either identical or resemble a similar structure for all bits. The datapath designer can concentrate on the design of a single slice that is repeated 32 times.

11.3 The Adder

Addition is the most commonly used arithmetic operation. It often is the speed-limiting element as well. Therefore, careful optimization of the adder is of the utmost importance. This optimization can proceed either at the logic or circuit level. Typical logic-level optimizations try to rearrange the Boolean equations so that a faster or smaller circuit is obtained. An example of such a logic optimization is the *carry lookahead adder* discussed later in the chapter. Circuit optimizations, on the other hand, manipulate transistor sizes and circuit topology to optimize the speed. Before considering both optimization processes, we provide a short summary of the basic definitions of an adder circuit (as defined in any book on logic design [e.g., Katz94]).

11.3.1 The Binary Adder: Definitions

Table 11.1 shows the truth table of a binary full adder. A and B are the adder inputs, C_i is the carry input, S is the sum output, and C_o is the carry output. The Boolean expressions for S and C_o are given in Eq. (11.1).

Table 11-1 Truth table for full adder.

A	B	C_i	S	C_o	Carry Status
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate/propagate
1	1	1	1	1	generate/propagate

$$\begin{aligned} S &= A \oplus B \oplus C_i \\ &= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i \\ C_o &= AB + BC_i + AC_i \end{aligned} \tag{11.1}$$

It is often useful from an implementation perspective to define S and C_o as functions of some intermediate signals G (generate), D (delete), and P (propagate).[1] $G = 1$ ($D = 1$) ensures that a carry bit will be *generated* (*deleted*) at C_o independent of C_i, while $P = 1$ guarantees that an incoming carry will propagate to C_o. Expressions for these signals can be derived from inspection of the truth table:

$$\begin{aligned} G &= AB \\ D &= \bar{A}\bar{B} \\ P &= A \oplus B \end{aligned} \tag{11.2}$$

We can rewrite S and C_o as functions of P and G (or D):

$$\begin{aligned} C_o(G, P) &= G + PC_i \\ S(G, P) &= P \oplus C_i \end{aligned} \tag{11.3}$$

Notice that G and P are only functions of A and B and are not dependent upon C_i. In a similar way, we can also derive expressions for $S(D, P)$ and $C_o(D, P)$.

[1] Note that the propagate signal sometimes is also defined as the OR function of the inputs A and B—this condition guarantees that the input carry propagates to the output when $A = B = 1$, too. We will provide appropriate warning whenever this definition is used.

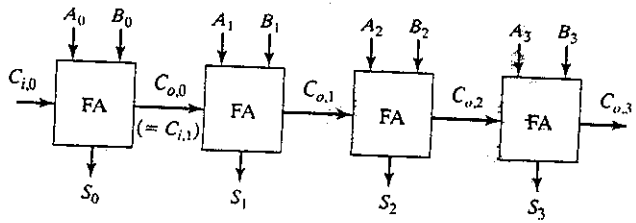


Figure 11-2 Four-bit ripple-carry adder: topology.

An N-bit adder can be constructed by cascading N full-adder (FA) circuits in series, connecting $C_{o,k-1}$ to $C_{i,k}$ for $k = 1$ to $N-1$, and the first carry-in $C_{i,0}$ to 0 (Figure 11-2). This configuration is called a *ripple-carry adder*, since the carry bit "ripples" from one stage to the other. The delay through the circuit depends upon the number of logic stages that must be traversed and is a function of the applied input signals. For some input signals, no rippling effect occurs at all, while for others, the carry has to ripple all the way from the *least significant bit (lsb)* to the *most significant bit (msb)*. The propagation delay of such a structure (also called the *critical path*) is defined as the *worst case delay over all possible input patterns*.

In the case of the ripple-carry adder, the worst case delay happens when a carry generated at the least significant bit position propagates all the way to the most significant bit position. This carry is finally consumed in the last stage to produce the sum. The delay is then proportional to the number of bits in the input words N and is approximated by

$$t_{adder} \approx (N-1)t_{carry} + t_{sum} \tag{11.4}$$

where t_{carry} and t_{sum} equal the propagation delays from C_i to C_o and S, respectively.[2]

Example 11.1 Propagation Delay of Ripple-Carry Adder

Derive the values of A_k and B_k ($k = 0 \ldots N-1$) so that the worst case delay is obtained for the ripple-carry adder.

The worst case condition requires that a carry be generated at the *lsb* position. Since the input carry of the first full adder C_{i0} is always 0, this both A_0 and B_0 must equal 1. All the other stages must be in propagate mode. Hence, either A_i or B_i must be high. Finally, we would like to physically measure the delay of a transition on the *msb* sum bit. Assuming an initial value of 0 for S_{N-1}, we must arrange a $0 \rightarrow 1$ transition. This is achieved by setting both A_{N-1} and B_{N-1} to 0 (or 1), which yields a high sum bit given the incoming carry of 1.

For example, the following values for A and B trigger the worst case delay for an 8-bit addition:

$$A\text{: } 0000001;\ B\text{: } 01111111$$

[2] Equation (11.4) assumes that both the delay from the input signals A_0 (or B_0) to $C_{o,0}$ for the *lsb*, and the C_i-to-C_o delay for all other bits equal to t_{carry}.

To set-up the worst case delay transition, all the inputs can be kept constant with A_0 undergoing a $0 \rightarrow 1$ transition.

The left-most bit represents the *msb* in this binary representation. Observe that this is only one of the many worst case patterns. This case exercises the $0 \rightarrow 1$ delay of the final sum. Derive several other cases that exercise the $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions.

Two important conclusions can be drawn from Eq. (11.4):

- The propagation delay of the ripple-carry adder is *linearly* proportional to N. This property becomes increasingly important when designing adders for the wide data paths ($N = 16...128$) that are desirable in current and future computers.
- When designing the full-adder cell for a fast ripple-carry adder, it is far more important to optimize t_{carry} than t_{sum}, since the latter has only a minor influence on the total value of t_{adder}.

Before starting an in-depth discussion on the circuit design of full-adder cells, the following additional logic property of the full adder is worth mentioning:

Inverting all inputs to a full adder results in inverted values for all outputs.

This property, also called the *inverting property*, is expressed in the pair of equations

$$\begin{aligned} \overline{S}(A, B, C_i) &= S(\overline{A}, \overline{B}, \overline{C_i}) \\ \overline{C_o}(A, B, C_i) &= C_o(\overline{A}, \overline{B}, \overline{C_i}) \end{aligned} \tag{11.5}$$

and will be extremely useful when optimizing the speed of the ripple-carry adder. It states that the circuits of Figure 11-3 are identical.

11.3.2 The Full Adder: Circuit Design Considerations

Static Adder Circuit

One way to implement the full-adder circuit is to take the logic equations of Eq. (11.1) and translate them directly into complementary CMOS circuitry. Some logic manipulations can help to reduce the transistor count. For instance, it is advantageous to share some logic between the

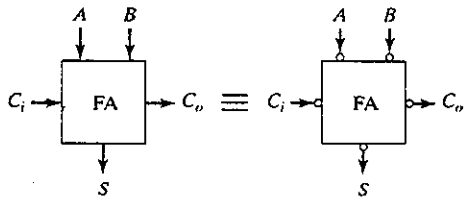


Figure 11-3 Inverting property of the full adder. The circles in the schematics represent inverters.

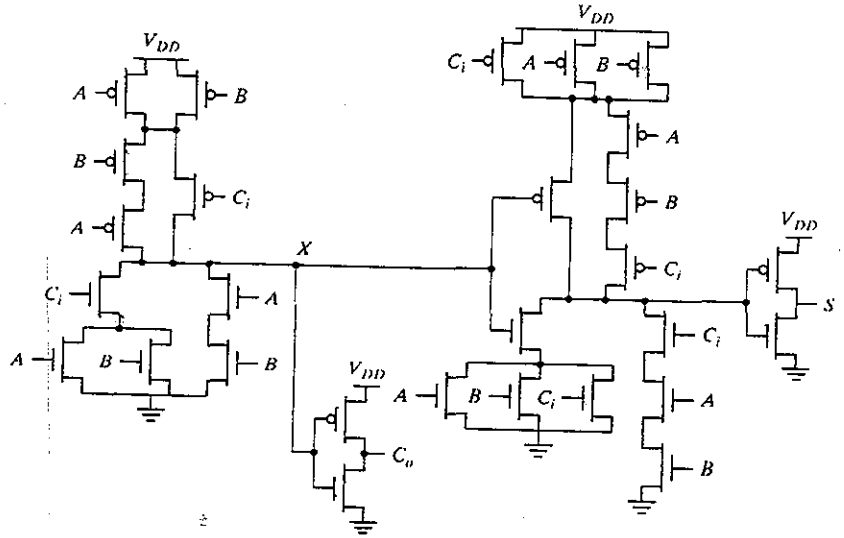


Figure 11-4 Complementary static CMOS implementation of full adder.

sum- and carry-generation subcircuits, as long as this does not slow down the carry generation, which is the most critical part, as stated previously. The following is an example of such a reorganized equation set:

$$\begin{aligned} C_o &= AB + BC_i + AC_i \\ S &= ABC_i + \overline{C_o}(A + B + C_i) \end{aligned} \tag{11.6}$$

and

The equivalence with the original equations is easily verified. The corresponding adder design, using complementary static CMOS, is shown in Figure 11-4 and requires 28 transistors. In addition to consuming a large area, this circuit is slow:

- Tall PMOS transistor stacks are present in both *carry*- and *sum*-generation circuits.
- The intrinsic load capacitance of the C_o signal is large and consists of two diffusion and six gate capacitances, plus the wiring capacitance.
- The signal propagates through two inverting stages in the carry-generation circuit. As mentioned earlier, minimizing the carry-path delay is the prime goal of the designer of high-speed adder circuits. Given the small load (fan-out) at the output of the carry chain, having two logic stages is too high a number, and leads to extra delay.
- The sum generation requires one extra logic stage, but that is not that important, since a factor appears only once in the propagation delay of the ripple-carry adder of Eq. (11.4).

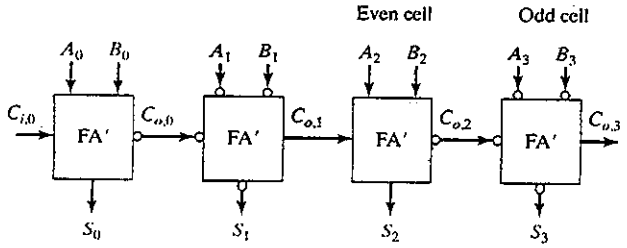


Figure 11-5 Inverter elimination in carry path. FA′ stands for a full adder without the inverter in the carry path.

Although slow, the circuit includes some smart design tricks. Notice that the first gate of the carry-generation circuit is designed with the C_i signal on the smaller PMOS stack, lowering its logical effort to 2. Also, the NMOS and PMOS transistors connected to C_i are placed as close as possible to the output of the gate. This is a direct application of a circuit-optimization technique discussed in Section 4.2—transistors on the critical path should be placed as close as possible to the output of the gate. For instance, in stage k of the adder, signals A_k and B_k are available and stable long before $C_{i,k}$ (= $C_{o,k-1}$) arrives after rippling through the previous stages. In this way, the capacitances of the internal nodes in the transistor chain are precharged or discharged in advance. On arrival of $C_{i,k}$, only the capacitance of node X has to be (dis)charged. Putting the $C_{i,k}$ transistors closer to V_{DD} and GND would require not only the (dis)charging of the capacitance of node X, but also of the internal capacitances.

The speed of this circuit can now be improved gradually by using some of the adder properties discussed in the previous section. First, the number of inverting stages in the carry path can be reduced by exploiting the inverting property—inverting all the inputs of a full-adder cell also inverts all the outputs. This rule allows us to eliminate an inverter in a carry chain, as demonstrated in Figure 11-5.

Mirror Adder Design

An improved adder circuit, also called the *mirror adder*, is shown in Figure 11-6 [Weste93]. Its operation is based on Eq. (11.3). The carry-generation circuitry is worth analyzing. First, the carry-inverting gate is eliminated, as suggested in the previous section. Secondly, the PDN and PUN networks of the gate are not dual. Instead, they form a clever implementation of the propagate/generate/delete function—when either D or G is high, $\overline{C}_o$ is set to V_{DD} or GND, respectively. When the conditions for a *Propagate* are valid (or P is 1),[3] the incoming carry is propagated (in inverted format) to $\overline{C}_o$. This results in a considerable reduction in both area and delay. The analysis of the sum circuitry is left to the reader. The following observations are worth considering:

- This full-adder cell requires only 24 transistors.

[3]The $P = A + B$ definition of the *propagate* signal is used here.

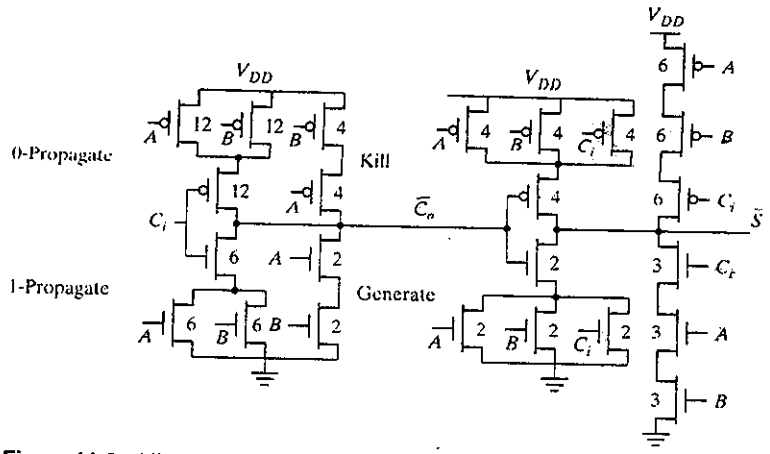


Figure 11-6 Mirror adder—circuit schematics.

- The NMOS and PMOS chains are completely symmetrical, which still yields correct operation due to *self-duality* of both the sum and carry functions. As a result, a maximum of two series transistors can be found in the carry-generation circuitry.
- The transistors connected to C_i are placed closest to the output of the gate.
- Only the transistors in the carry stage have to be optimized for speed. All transistors in the sum stage can be of minimum size. When laying out the cell, the most critical issue is the minimization of the capacitance at node $\overline{C}_o$. Shared diffusions reduce the stack node capacitances.
- In the adder cell of Figure 11-4, the inverter can be sized independently to drive the C_i input of the adder stage that follows. If the carry circuit in Figure 11-6 is symmetrically sized, each of its inputs has a logical effort of 2. This means that the optimal fan-out, sized for minimum delay, should be (4/2) = 2. However, the output of this stage drives two internal gate capacitances and six gate capacitances in the connecting adder cell. A clever solution to keep the transistor sizes the same in each stage is to increase the size of the carry stage to about three to four times the size of the sum stage. This maintains the optimal fan-out of 2. The resulting transistor sizes are annotated on Figure 11-6, where a PMOS/NMOS ratio of 2 is assumed.

Transmission-Gate-Based Adder

A full adder can be designed to use multiplexers and XORs. While this is impractical in a complementary CMOS implementation, it becomes attractive when the multiplexers and XORs are implemented as transmission gates. A full-adder implementation based on this approach is shown in Figure 11-7 and uses 24 transistors. It is based on the *propagate–generate* model, introduced in Eq. (11.3). The propagate signal, which is the XOR of inputs A and B, is used to

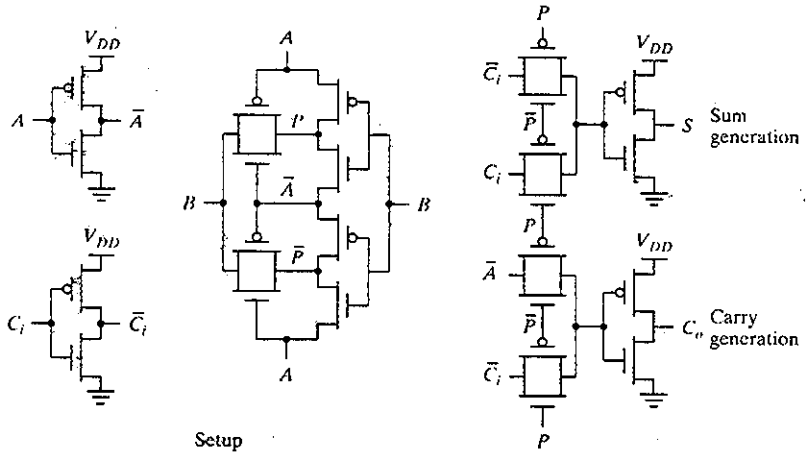


Figure 11-7 Transmission-gate-based full-adder cell with sum and carry delays of similar value (after [Weste93]).

select the true or complementary value of the input carry as the new sum output. Based on the propagate signal, the output carry is either set to the input carry, or either one of inputs A or B. One of interesting features of such an adder is that it has similar delays for both sum and carry outputs.

Manchester Carry-Chain Adder

The carry-propagation circuitry in Figure 11-7 can be simplified by adding generate and delete signals, as shown in Figure 11-8a. The propagate path is unchanged, and it passes C_i to the C_o output if the propagate signal ($A_i \oplus B_i$) is true. If the propagate condition is not satisfied, the output is either pulled low by the D_i signal or pulled up by $\overline{G}_i$. The dynamic implementation (Figure 11-8b), makes even further simplification possible. Since the transitions in a dynamic circuit are monotonic, the transmission gates can be replaced by NMOS-only pass transistors. Precharging the output eliminates the need for the kill signal (for the case in which the carry chain propagates the complementary values of the carry signals).

A *Manchester carry-chain adder* uses a cascade of pass transistors to implement the carry chain [Kilburn60]. An example, based on the dynamic circuit version introduced in Figure 11-8, is shown in Figure 11-9. During the precharge phase ($\phi = 0$), all intermediate nodes of the pass-transistor carry chain are precharged to V_{DD}. During evaluation, the A_k node is discharged when there is an incoming carry and the propagate signal P_k is high, or when the generate signal for stage k (G_k) is high.

Figure 11-10 shows an example layout of the Manchester carry chain in stick-diagram format. The datapath layout consists of three rows of cells organized in bit-sliced style: The top row

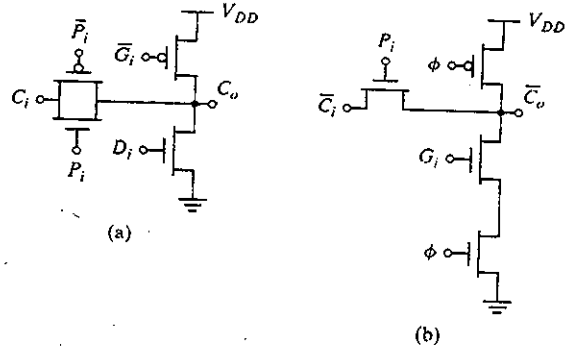


Figure 11-8 Manchester carry gates. (a) Static, using propagate, generate, and kill, (b) dynamic implementation, using only propagate and generate signals.

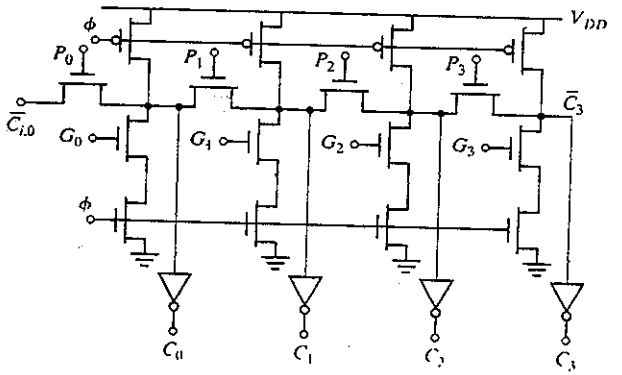


Figure 11-9 Manchester carry-chain adder in dynamic logic (four-bit section).

of cells computes the propagate and generate signals, the middle row propagates the carry from left to right, and the bottom row generates the final sums.

The worst case delay of the carry chain of the adder in Figure 11-9 is modeled by the linearized RC network of Figure 11-11. As derived in Chapter 4, the propagation delay of such a network equals

$$t_p = 0.69 \sum_{i=1}^{N} C_i \left(\sum_{j=1}^{i} R_j \right) = 0.69 \frac{N(N+1)}{2} RC \tag{11.7}$$

when all $C_i = C$ and $R_j = R$.

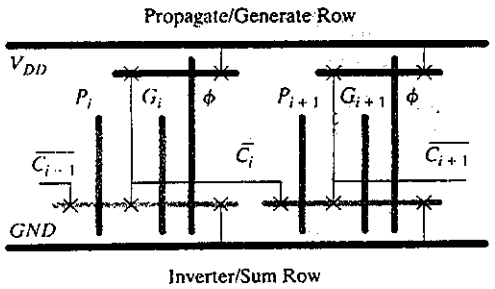


Figure 11-10 Stick diagram of two bits of a Manchester carry chain.

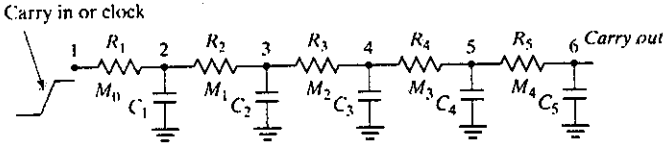


Figure 11-11 Equivalent network to determine propagation delay of a carry chain.

Example 11.2 Sizing of Manchester Carry Chain

The capacitance per node on the carry chain equals four diffusion capacitances, one inverter input capacitance, and the wiring capacitance proportional to the size of the cell. The inverter and the PMOS precharging transistor can be kept at unit size. Together with the wire capacitance, the fixed capacitance can be estimated as 15 fF (for our technology). If a unit-sized transistor with width W_0 has a resistance of 10 kΩ and a diffusion capacitance of 2 fF, then the RC time constant for a chain of transistors of width W is

$$RC = \left(6\ \text{fF} \cdot \frac{W}{W_0} + 15\ \text{fF}\right) \cdot 10\ \text{k}\Omega \cdot \frac{W_0}{W}$$

Increasing the transistor width reduces this time constant, but it also loads the gates in the previous stage. Therefore, the transistor size is limited by the input loading capacitance.

Unfortunately, the distributed RC-nature of the carry chain results in a propagation delay that is quadratic in the number of bits N. To avoid this, it is necessary to insert signal-buffering inverters. The optimum number of stages per buffer depends on the equivalent resistance of the inverter and the resistance and capacitance of the pass transistors, as was discussed in Chapter 9. In our technology, and in most other practical cases, this number is between 3 and 4. Adding the inverter makes the overall propagation delay a linear function of N, as is the case with ripple-carry adders.

11.3.3 The Binary Adder: Logic Design Considerations

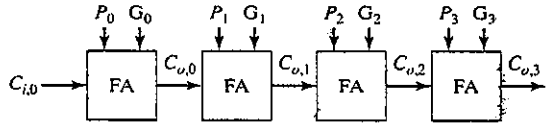
The ripple-carry adder is only practical for the implementation of additions with a relatively small word length. Most desktop computers use word lengths of 32 bits, while servers require 64; very fast computers, such as mainframes, supercomputers, or multimedia processors (e.g., the Sony PlayStation2) [Suzuoki99], require word lengths of up to 128 bits. The linear dependence of the adder speed on the number of bits makes the usage of ripple adders rather impractical. Logic optimizations are therefore necessary, resulting in adders with $t_p < O(N)$. We briefly discuss a number of those in the sections that follow. We concentrate on the circuit design implications, since most of the presented structures are well known from the traditional logic design literature.

The Carry-Bypass Adder

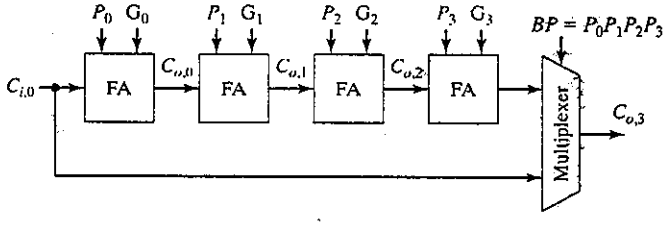
Consider the four-bit adder block of Figure 11-12a. Suppose that the values of A_k and B_k (k = 0...3) are such that all propagate signals P_k (k = 0...3) are high. An incoming carry $C_{i,0}$ = 1 propagates under those conditions through the complete adder chain and causes an outgoing carry $C_{o,3}$ = 1. In other words,

$$\text{if } (P_0P_1P_2P_3 = 1) \text{ then } C_{o,3} = C_{i,0} \quad \text{else either DELETE or GENERATE occurred} \tag{11.8}$$

This information can be used to speed up the operation of the adder, as shown in Figure 11-12b. When $BP = P_0P_1P_2P_3 = 1$, the incoming carry is forwarded immediately to the next block through the bypass transistor M_b—hence the name *carry-bypass adder* or *carry-skip adder* [Lehman62]. If this is not the case, the carry is obtained by way of the normal route.



(a) Carry propagation



(b) Adding a bypass

Figure 11-12 Carry-bypass structure—basic concept.

Example 11.3 Carry Bypass in Manchester Carry-Chain Adder

Figure 11-13 shows the possible carry-propagation paths when the full-adder circuit is implemented in Manchester-carry style. This picture demonstrates how the bypass speeds up the addition: The carry propagates either through the bypass path, or a carry is generated somewhere in the chain. In both cases, the delay is smaller than the normal ripple configuration. The area overhead incurred by adding the bypass path is small and typically ranges between 10 and 20%. However, adding the bypass path breaks the regular bit-slice structure (as was present in Figure 11-10).

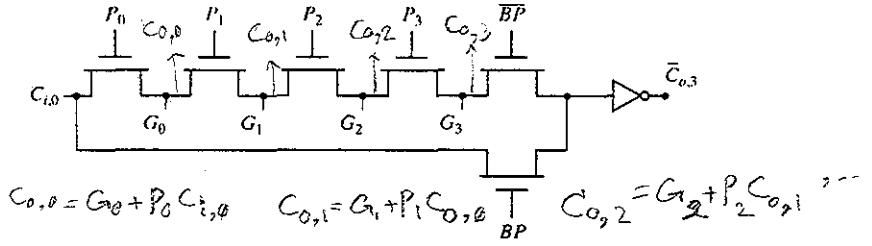


Figure 11-13 Manchester carry-chain implementation of bypass adder.

Let us now compute the delay of an N-bit adder. At first, we assume that the total adder is divided in (N/M) equal-length bypass stages, each of which contains M bits. An approximating expression for the total propagation time can be derived from Figure 11-14a and is given in Eq. (11.9). Namely,

$$t_p = t_{setup} + Mt_{carry} + \left(\frac{N}{M} - 1\right)t_{bypass} + (M-1)t_{carry} + t_{sum} \qquad (11.9)$$

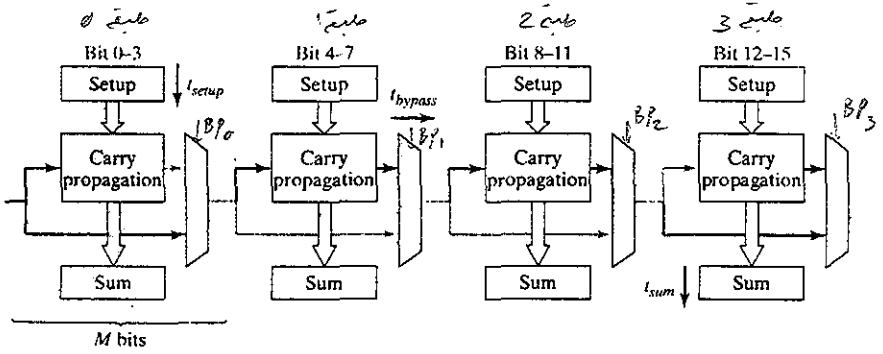


Figure 11-14 ($N = 16$) carry-bypass adder: composition. The worst case delay path is shaded in gray.

with the composing parameters defined as follows:

- t_{setup}: the fixed overhead time to create the generate and propagate signals.
- t_{carry}: the propagation delay through a single bit. The worst case carry-propagation delay through a single stage of M bits is approximately M times larger.
- t_{bypass}: the propagation delay through the bypass multiplexer of a single stage.
- t_{sum}: the time to generate the sum of the final stage.

The critical path is shaded in gray on the block diagram of Figure 11-14. From Eq. (11.9), it follows that t_p is still linear in the number of bits N, since in the worst case, the carry is generated at the first bit position, ripples through the first block, skips around $(N/M - 2)$ bypass stages, and is consumed at the last bit position without generating an output carry. The optimal number of bits per skip block is determined by technological parameters such as the extra delay of the bypass-selecting multiplexer, the buffering requirements in the carry chain, and the ratio of the delay through the ripple and the bypass paths.

Although still linear, the slope of the delay function increases in a more gradual fashion than for the ripple-carry adder, as pictured in Figure 11-15. This difference is substantial for larger adders. Notice that the ripple adder is actually faster for small values of N, for which the overhead of the extra bypass multiplexer makes the bypass structure not interesting. The cross-over point depends upon technology considerations and is normally situated between four and eight bits.

Problem 11.1 Delay of Carry-Skip Adder

Determine an input pattern that triggers the worst case delay in a 16-bit (4 × 4) carry-bypass adder. Assuming that $t_{carry} = t_{setup} = t_{skip} = t_{sum} = 1$, determine the delay and compare it with that of a normal ripple adder.

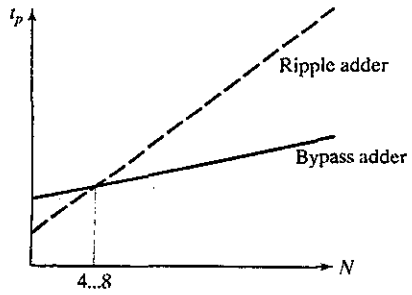


Figure 11-15 Propagation delay of ripple-carry versus carry-bypass adder.

The Linear Carry-Select Adder

In a ripple-carry adder, every full-adder cell has to wait for the incoming carry before an outgoing carry can be generated. One way to get around this linear dependency is to anticipate both possible values of the carry input and evaluate the result for both possibilities in advance. Once the real value of the incoming carry is known, the correct result is easily selected with a simple multiplexer stage. An implementation of this idea, appropriately called the *carry-select adder* [Bedrij62], is demonstrated in Figure 11-16. Consider the block of adders, which is adding bits k to $k + 3$. Instead of waiting on the arrival of the output carry of bit $k - 1$, both the 0 and 1 possibilities are analyzed. From a circuit point of view, this means that two carry paths are implemented. When $C_{o,k-1}$ finally settles, either the result of the 0 or the 1 path is selected by the multiplexer, which can be performed with a minimal delay. As is evident from Figure 11-16, the hardware overhead of the carry-select adder is restricted to an additional carry path and a multiplexer, and equals about 30% with respect to a ripple-carry structure.

A full carry-select adder is now constructed by chaining a number of equal-length adder stages, as in the carry-bypass approach (see Figure 11-17). The critical path is shaded in gray. From inspection of the circuit, we can derive a first-order model of the worst case propagation delay of the module, written as

$$t_{add} = t_{setup} + Mt_{carry} + \left(\frac{N}{M}\right)t_{mux} + t_{sum} \tag{11.10}$$

where t_{setup}, t_{sum}, and t_{mux} are fixed delays and N and M represent the total number of bits, and the number of bits per stage, respectively. t_{carry} is the delay of the carry through a single full-adder cell. The carry delay through a single block is proportional to the length of that stage or equals $M\,t_{carry}$.

The propagation delay of the adder is, again, linearly proportional to N (Eq. (11.10)). The reason for this linear behavior is that the *block-select* signal that selects between the 0 and 1 solutions still has to ripple through all stages in the worst case.

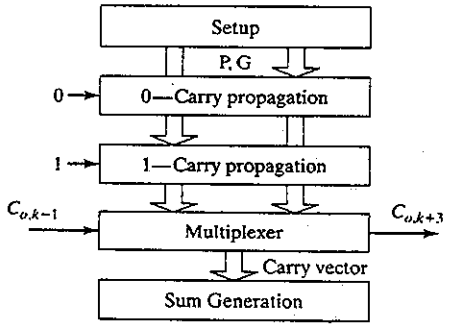


Figure 11-16 Four-bit carry-select module—topology.

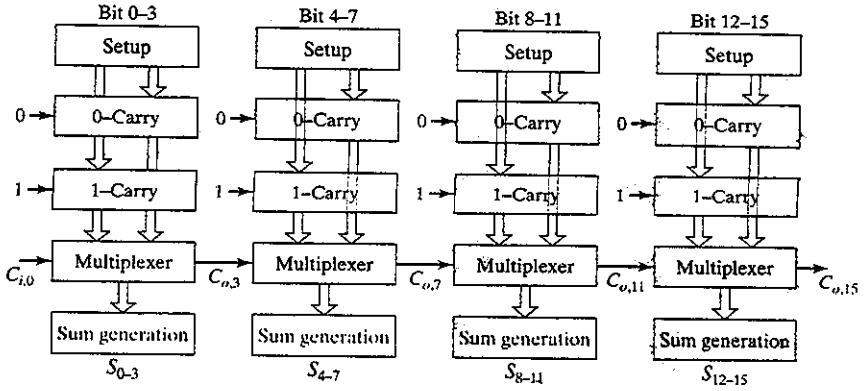


Figure 11-17 Sixteen-bit, linear carry-select adder. The critical path is shaded in gray.

Problem 11.2 Linear Carry-Select Delay

Determine the delay of a 16-bit linear carry-select adder by using unit delays for all cells. Compare the result with that of Problem 11.1. Compare various block configurations as well.

The Square-Root Carry-Select Adder

The next structure illustrates how an alert designer can make a major impact. To optimize a design, it is essential to locate the critical timing path first. Consider the case of a 16-bit linear carry-select adder. To simplify the discussion, assume that the full-adder and multiplexer cells have identical propagation delays equal to a normalized value of 1. The worst case arrival times of the signals at the different network nodes with respect to the time the input is applied are marked and annotated on Figure 11-18a. This analysis demonstrates that the critical path of the adder ripples through the multiplexer networks of the subsequent stages.

One striking opportunity is readily apparent. Consider the multiplexer gate in the last adder stage. The inputs to this multiplexer are the two carry chains of the block and the block-multiplexer signal from the previous stage. A major mismatch between the arrival times of the signals can be observed. The results of the carry chains are stable long before the multiplexer signal arrives. It makes sense to equalize the delay through both paths. This can be achieved by progressively adding more bits to the subsequent stages in the adder, requiring more time for the generation of the carry signals. For example, the first stage can add 2 bits, the second contains 3, the third has 4, and so forth, as demonstrated in Figure 11-18b. The annotated arrival times show that this adder topology is faster than the linear organization, even though an extra stage is needed. In fact, the same propagation delay is also valid for a 20-bit adder. Observe that the discrepancy in arrival times at the multiplexer nodes has been eliminated.

In effect, the simple trick of making the adder stages progressively longer results in an adder structure with sublinear delay characteristics. This is illustrated by the following analysis:

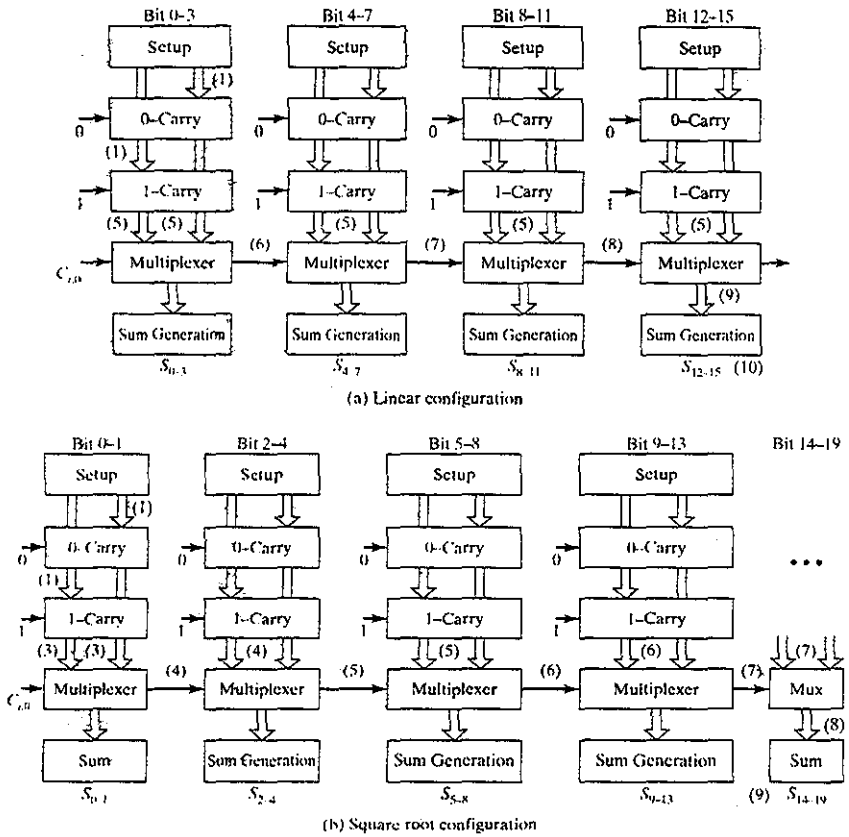


Figure 11-18 Worst case signal arrival times in carry-select adders. The signal arrival times are marked in parentheses.

Assume that an N-bit adder contains P stages, and the first stage adds M bits. An additional bit is added to each subsequent stage. The following relation then holds:

$$\begin{aligned} N &= M + (M+1) + (M+2) + (M+3) + \ldots + (M+P-1) \\ &= MP + \frac{P(P-1)}{2} = \frac{P^2}{2} + P\left(M - \frac{1}{2}\right) \end{aligned} \tag{11.11}$$

If $M << N$ (e.g., $M = 2$, and $N = 64$), the first term dominates, and Eq. (11.11) can be simplified to

$$N \approx \frac{P^2}{2} \tag{11.12}$$

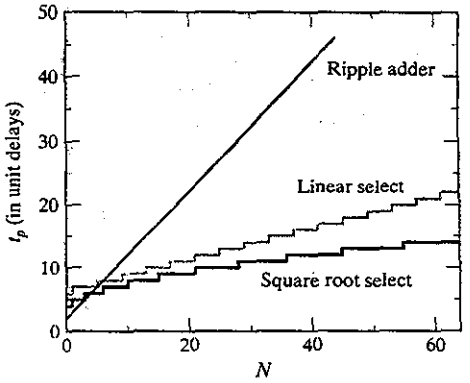


Figure 11-19 Propagation delay of square-root carry-select adder versus linear ripple and select adders. The unit delay model is used to model the cell delays.

or

$$P \approx \sqrt{2N} \tag{11.13}$$

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

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

The delay is proportional to $\sqrt{N}$ for large adders ($N >> M$), 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:

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

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

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

In a fully expanded form,

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

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. Furthermore, 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).

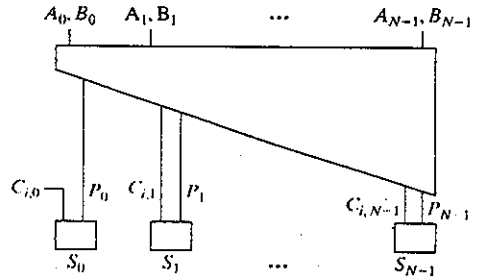


Figure 11-20 Conceptual diagram of a carry-lookahead adder.

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

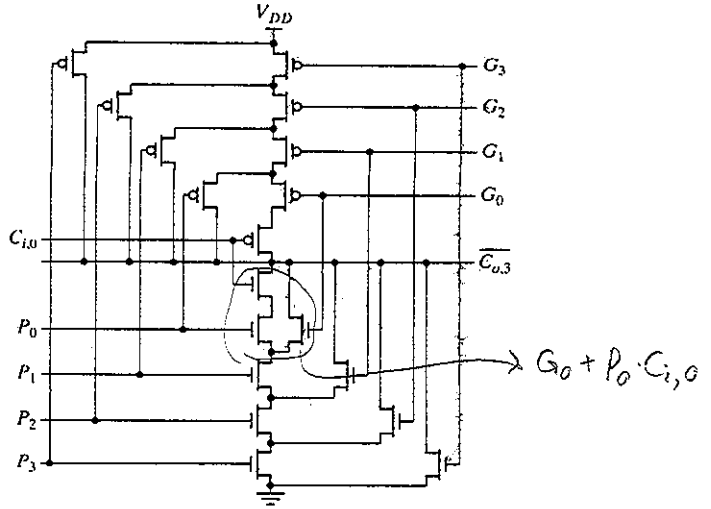


Figure 11-21 Schematic diagram of mirror implementation of four-bit lookahead adder (from [Weste85]).

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:

$$(G, P) \cdot (G', P') = (G + PG', PP') \tag{11.19}$$

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

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

The associative property allows us to rewrite this function and express $C_{o,3}$ 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 $i = 1...\log_2(N)$. 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

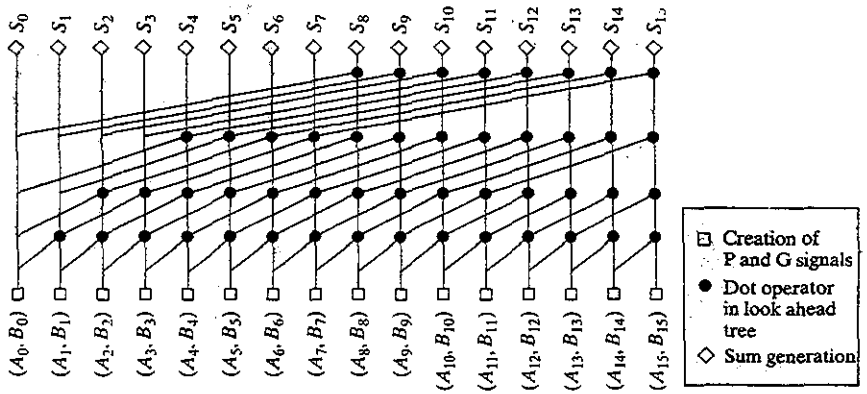


Figure 11-22: Schematic diagram for Kogge–Stone 16-bit lookahead logarithmic adder.

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

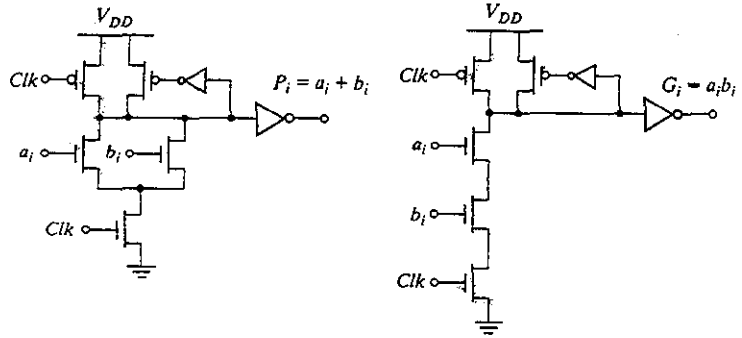


Figure 11-23 Dynamic Implementation of propagate and generate signals.

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 $S_i^0 = \overline{a_i \oplus b_i}$ and $S_i^1 = a_i \oplus b_i$. The dynamic gate of Figure 11-25 is then used to select one of these possibilities, based on the incoming carry.

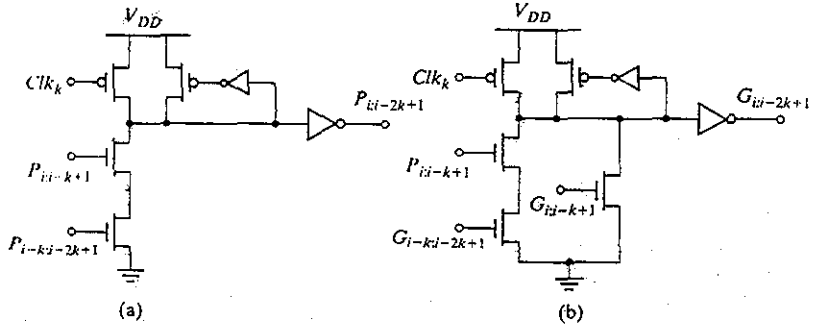


Figure 11-24 Implementation of the dot operator in dynamic logic: (a) propagate and (b) generate logic at stage *k* and bit position *i* (see Figure 11-22).

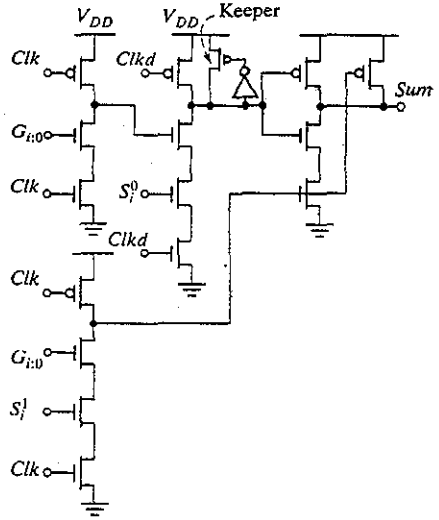


Figure 11-25 Sum select in dynamic logic.

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 (*Clkd*) 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 $2^N - 1$:

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

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

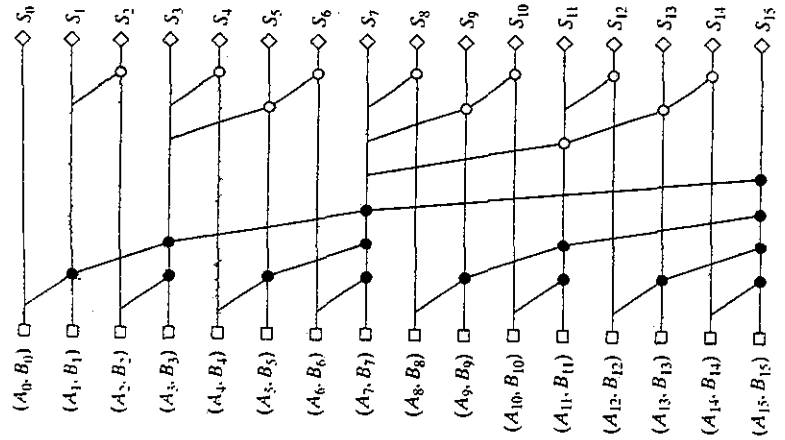


Figure 11-26 A 16-bit Brent–Kung tree. The reverse binary tree is colored gray.

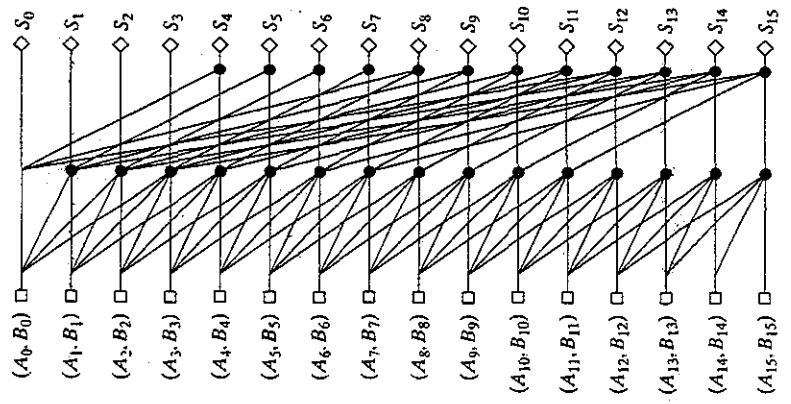


Figure 11-27 Radix-4 Kogge–Stone tree for 16-bit operands.

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

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

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

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

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

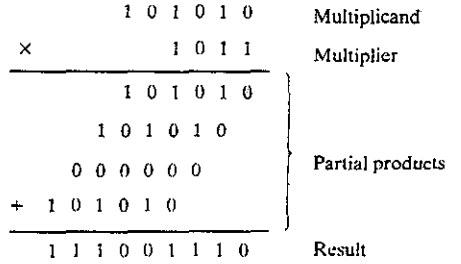


Figure 11-28 Binary multiplication—an example.

resulting structure is called an *array multiplier* and combines the following three functions: *partial-product generation*, *partial-product accumulation*, and *final addition*.

Problem 11.6 Multiplication by a Constant

It is often necessary to implement multiplication of an operand by a constant. Describe how you would proceed with this task.

11.4.2 Partial-Product Generation

Partial products result from the logical AND of multiplicand X with a multiplier bit Y_i (see Figure 11-29). Each row in the partial-product array is either a copy of the multiplicand or a row of zeroes. Careful optimization of the partial-product generation can lead to some substantial delay and area reductions. Note that in most cases the partial-product array has many zero rows that have no impact on the result and thus represent a waste of effort when added. In the case of a multiplier consisting of all ones, all the partial products exist, while in the case of all zeros, there is none. This observation allows us to reduce the number of generated partial products by half.

Assume, for example, an eight-bit multiplier of the form 01111110, which produces six nonzero partial-product rows. One can substantially reduce the number of nonzero rows by recoding this number ($2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2$) into a different format. The reader can verify that the form $10000\bar{1}0$, with $\bar{1}$ a shorthand notation for -1, represents the same number. Using

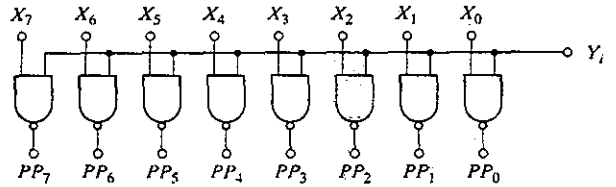


Figure 11-29 Partial-product generation logic.

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:

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

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 ×), 00 (0 ×), 00 (0 ×), $\bar{1}0$ (−2 ×), 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

[5] This 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.

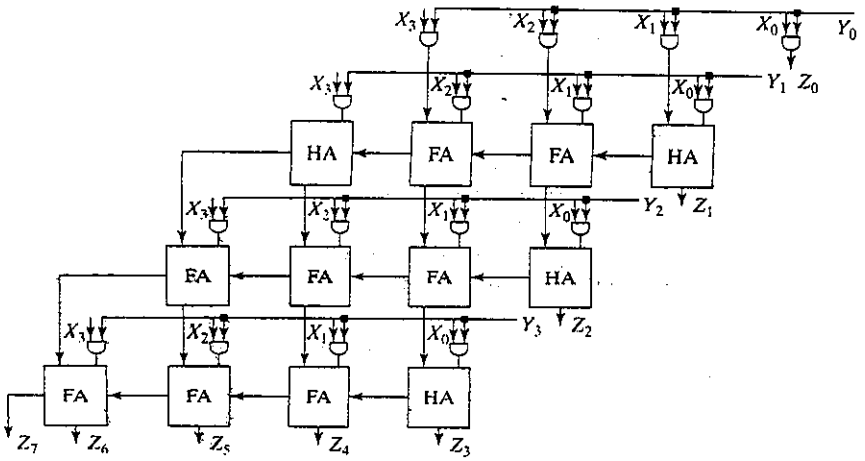


Figure 11-30 A 4 × 4 bit-array multiplier for unsigned numbers—composition. HA stands for a half adder, or an adder cell with only two inputs. The hardware for the generation and addition of one partial product is shaded in gray.

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

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

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}

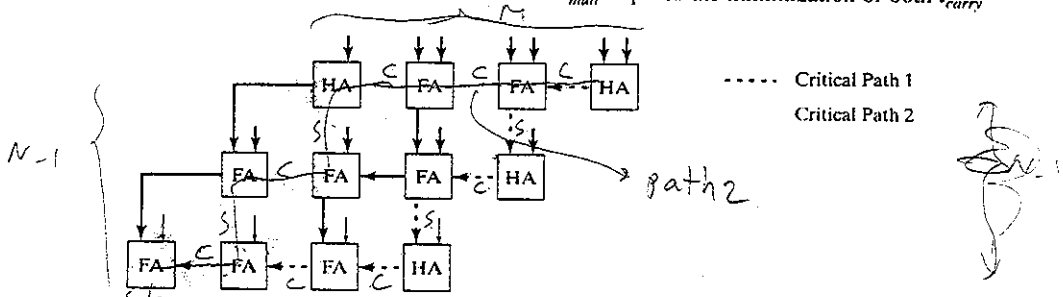


Figure 11-31 Ripple-carry based 4 × 4 multiplier (simplified diagram). Two of the possible critical paths are highlighted.

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 $t_{add} = t_{carry}$.

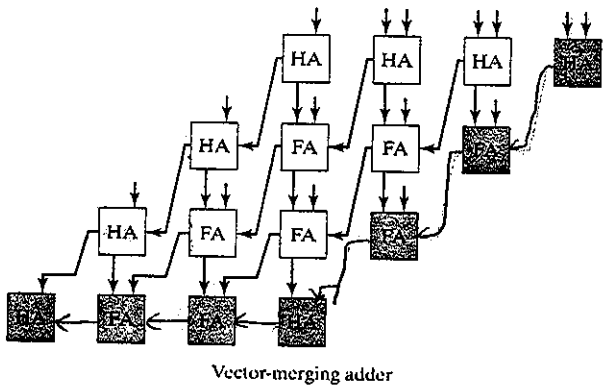


Figure 11-32 A 4 × 4 carry-save multiplier. The critical path is highlighted in gray.

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.

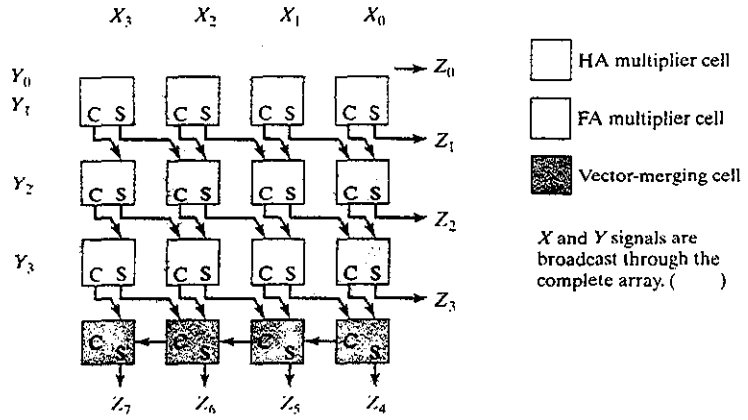


Figure 11-33 Rectangular floorplan of carry-save multiplier. Different cells are differentiated by shades of gray. *X* and *Y* signals are AND'ed before being added. The leftmost column of cells is redundant and can be eliminated.

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.

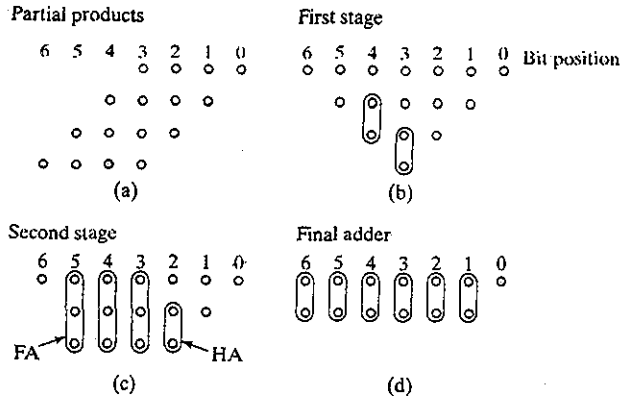


Figure 11-34 Transforming a partial-product tree (a) into a Wallace tree (b,c,d), using an iterative covering process. The example shown is for a four-bit operand.

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

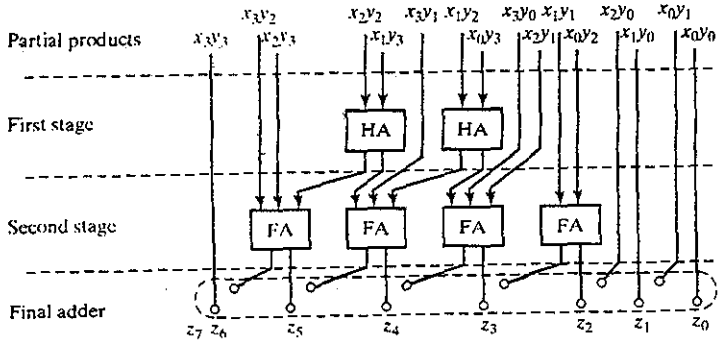


Figure 11-35 Wallace tree for four-bit multiplier.

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, scalers, 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

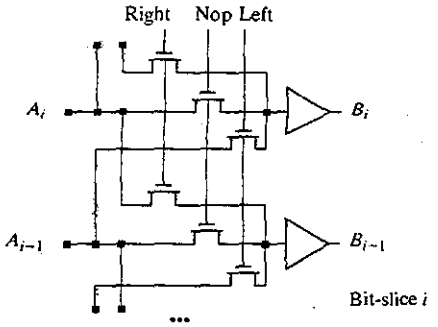


Figure 11-36 One–bit (left–right) programmable shifter. The data passes undisturbed under the nop condition.

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.

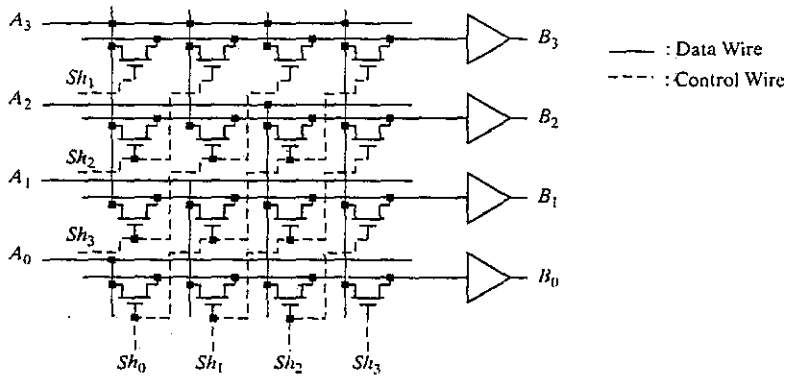


Figure 11-37 Barrel shifter with a programmable shift width from zero to three bits to the right. The structure supports automatic repetition of the sign bit (A_3), also called *sign-bit extension*.

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

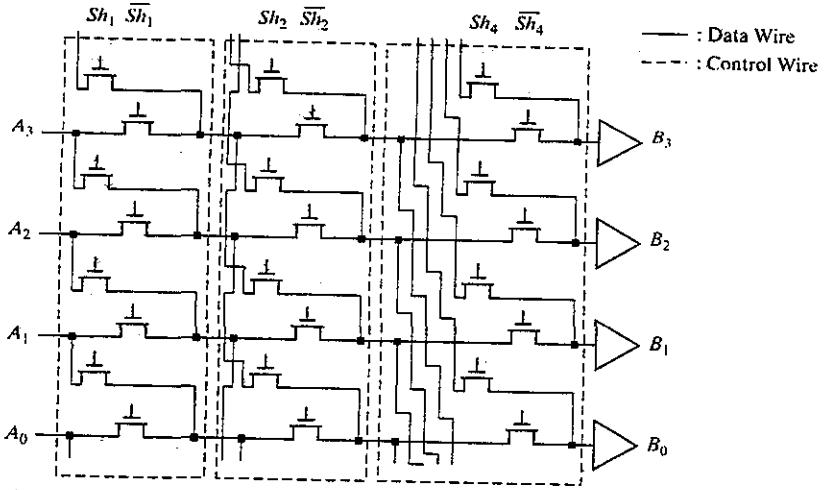


Figure 11-38 Logarithmic shifter with maximal shift width of seven bits to the right. (Only the four least significant bits are shown.)

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 $A \geq B$.

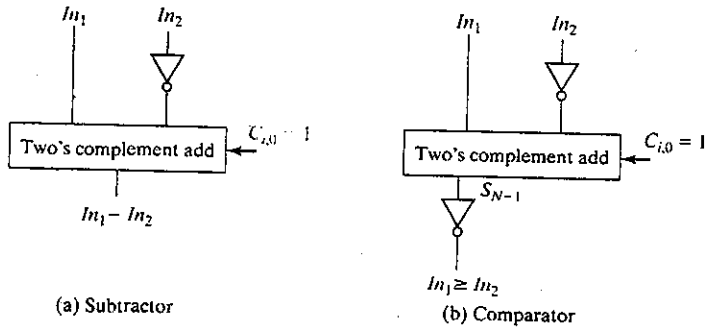


Figure 11-39 Arithmetic structures derived from a full adder.

Problem 11.10 Comparator

Derive a logic diagram for a comparator that implements the following logic functions: ≥, =, and ≤.

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.

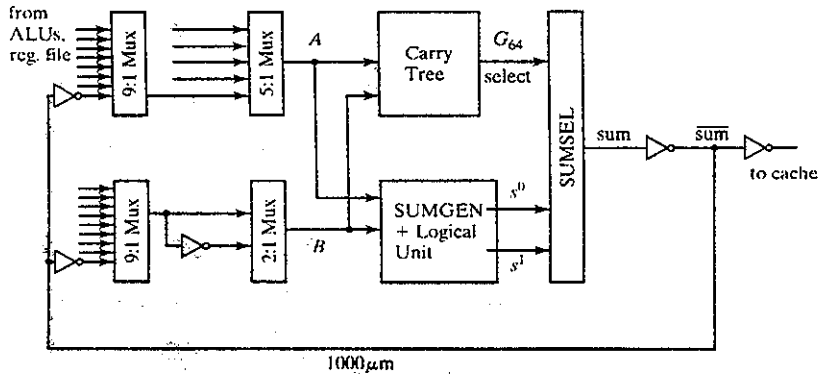


Figure 11-40 A 64-bit arithmetic-logic unit.

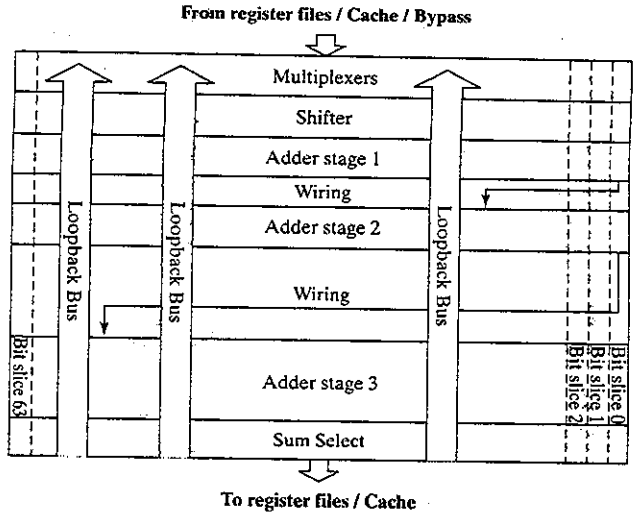


Figure 11-41 Floor plan of a 64-bit ALU.

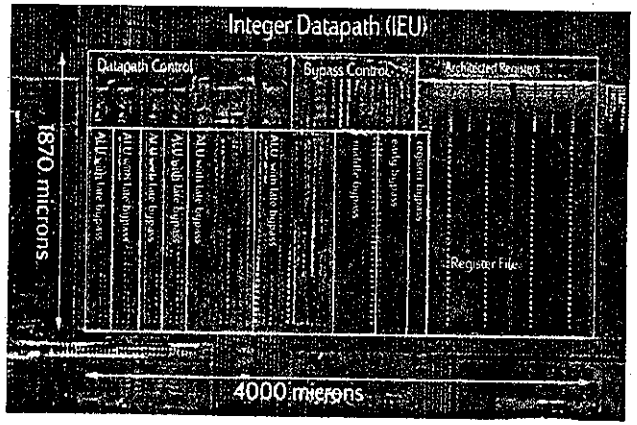


Figure 11-42 Microphotograph of the Itanium processor, showing bit-sliced design of six ALUs together with bypasses, register file, and control logic. (Courtesy of Intel.)

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

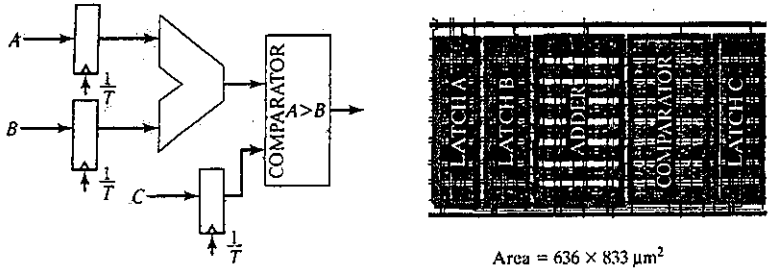


Figure 11-43 A simple datapath with corresponding layout.

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

$$P_{ref} = C_{ref} V_{ref}^2 f_{ref} \tag{11.26}$$

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

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

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 important 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.

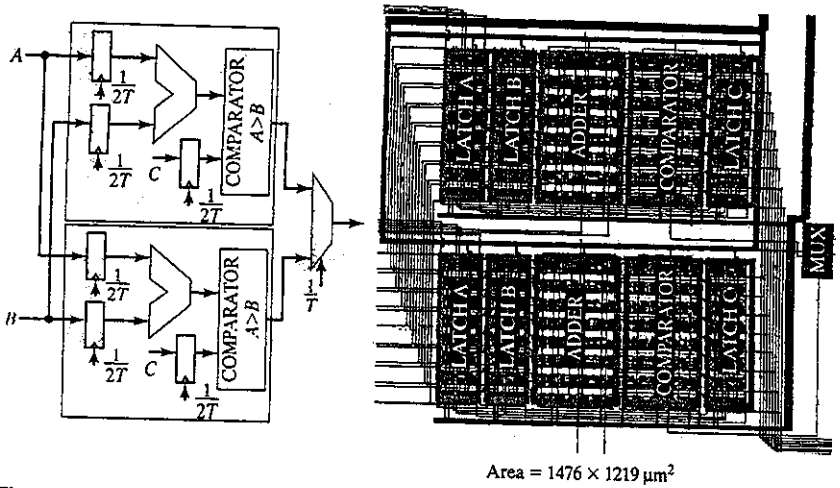


Figure 11-44 Parallel implementation of the simple datapath.

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 traightforward, 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, $V_{DDL}-V_{Tn}$, 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

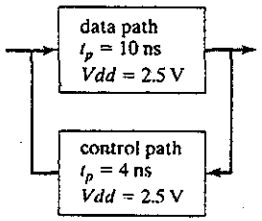


Figure 11-45 Design with diverging critical-path lengths.

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

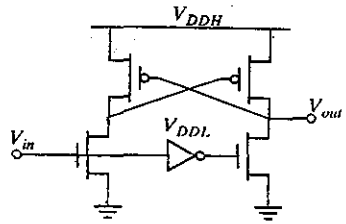


Figure 11-46 Level converter.

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

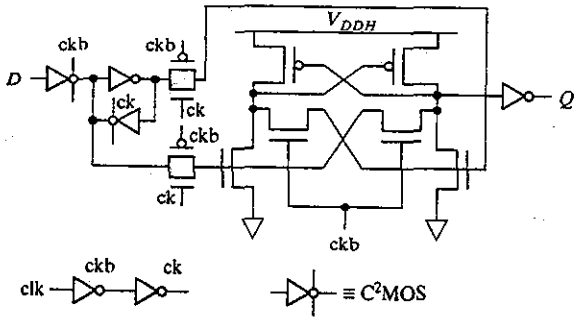


Figure 11-47 Level-converting register. Shaded gates are supplied from V_{DDL} [Usami95].

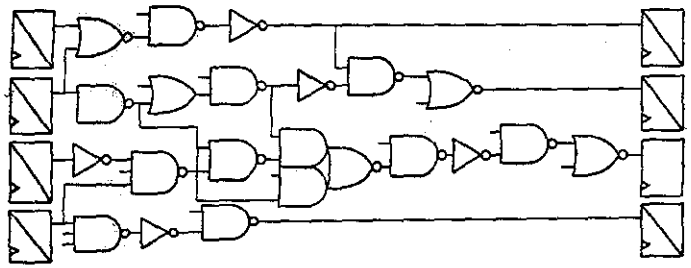


Figure 11-48 Dual-supply design using clustered voltage scaling. Each path starts with a high supply and switches to a low supply (gray logic gates) if there is a delay slack. Level conversion is performed in the registers.

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, Augsburger02].

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**

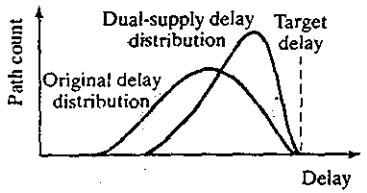


Figure 11-49 Typical path-delay distribution before and after applying the second supply.

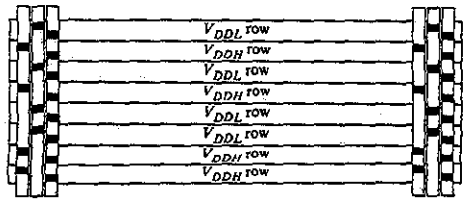


Figure 11-50 Layout of a dual-supply logic block where all cells at different supplies are placed in different rows.

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%.

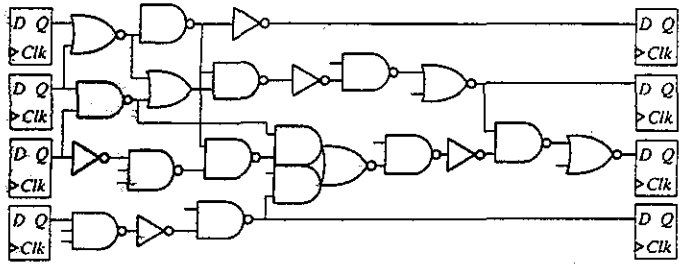


Figure 11-51 Use of dual-threshold gates. Gray-shaded gates use low thresholds and are employed only in critical paths.

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:

$$s_i^2 = s_{i+1}s_{i-1} \tag{11.28}$$

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:

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

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.**

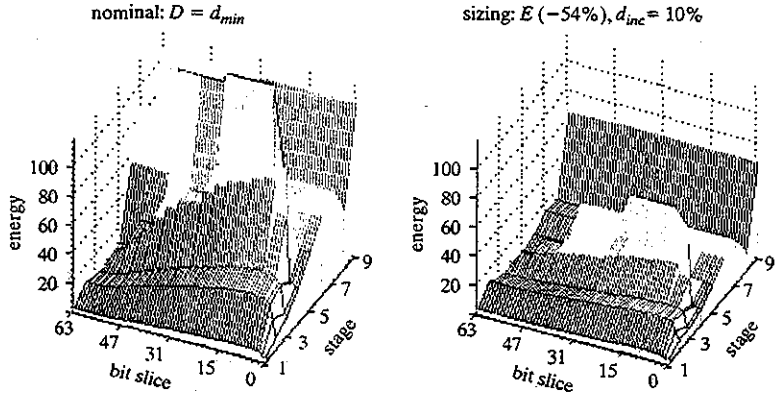


Figure 11-52 Energy profiles of a 64-bit adder: (a) sized for minimum delay, (b) sized for 10% delay increment and minimized energy.

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

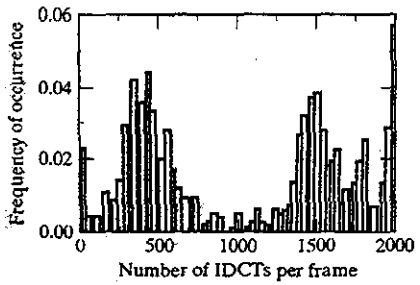


Figure 11-55 Typical IDCT histogram for MPEG decoding.

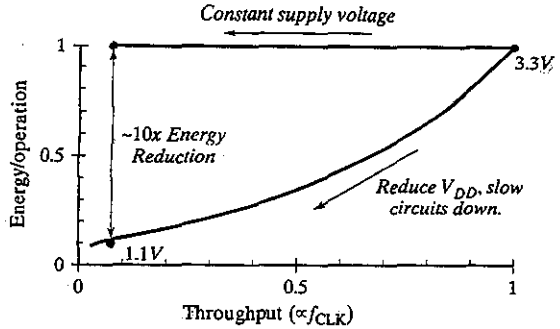


Figure 11-56 Energy/operation versus throughput (1/T) for constant and variable supply voltage operation.

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,

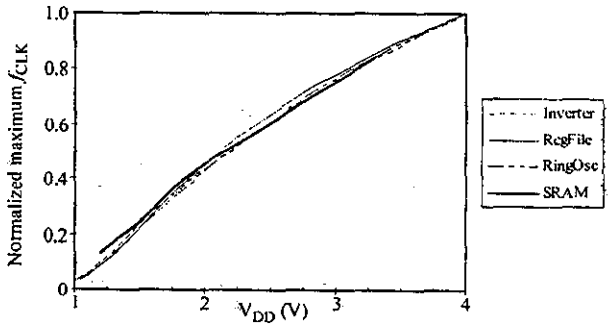
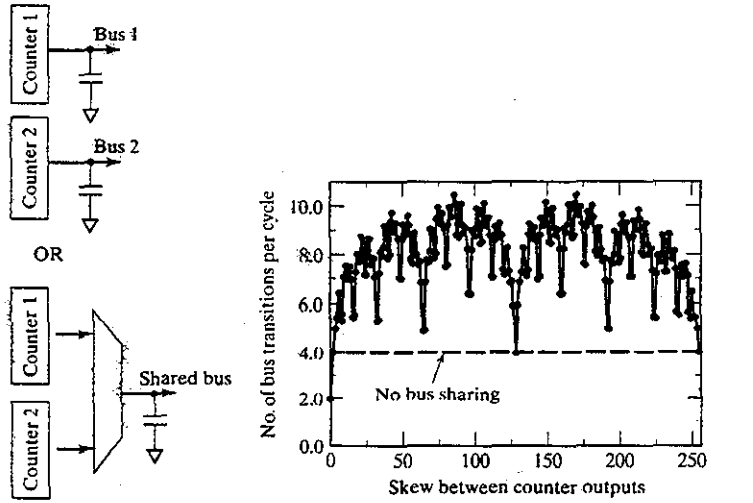


Figure 11-57 Delay of representative CMOS functions as a function of supply voltage for a 0.6-μm CMOS technology.

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.



(a) Counter architectures (b) Number of transitions as a function of counter skew

Figure 11-53 Multiplexing increases the switching activity.

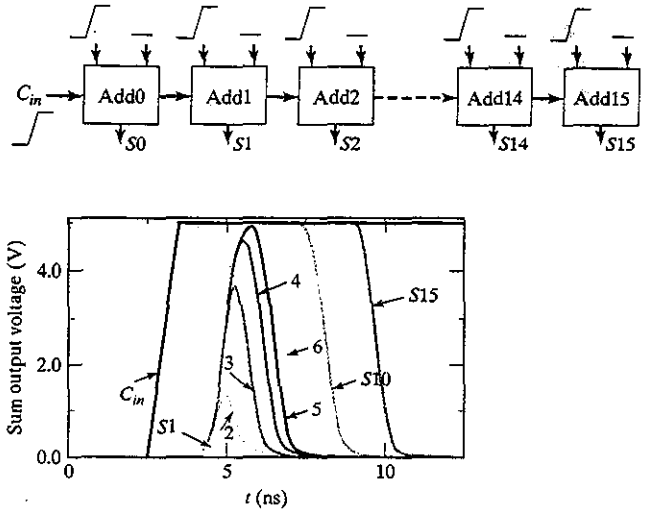


Figure 11-54 Glitching in the sum bits of a 16-bit ripple-carry adder.

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.

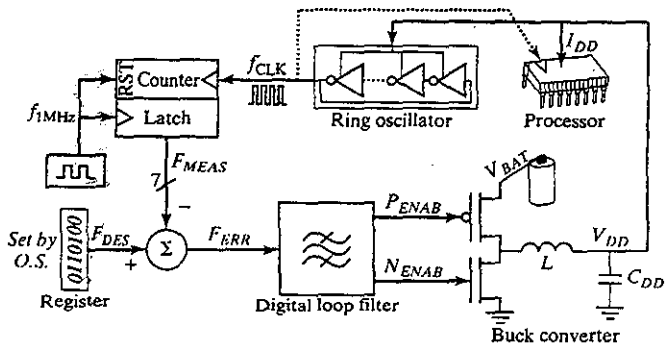


Figure 11-58 Block diagram of a dynamic voltage-scaled system.

- 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 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

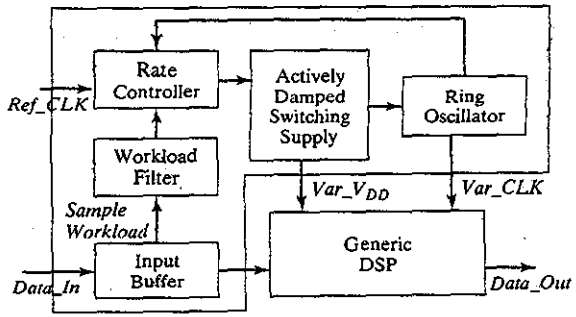


Figure 11-59 Using a queue to determine the workload in a signal processor [Gutnik97].

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

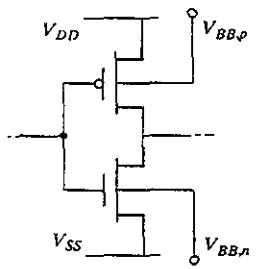


Figure 11-60 An inverter with body terminals used for threshold control.

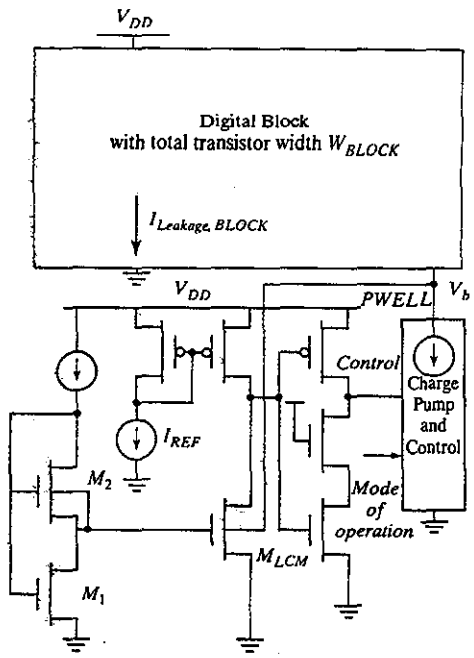


Figure 11-61 Variable threshold control scheme implemented with a leakage current monitor [Kuroda96].

a digital module, is shown in Figure 11-61 [Kuroda96 and Kuroda01]. It consists of a leakage control monitor and a substrate bias charge pump, added to the digital block of interest.

The leakage current monitor is crucial for implementing this scheme. Transistors M_1 and M_2 are biased to operate in the subthreshold region. When an NMOS transistor is in the subthreshold, its current is given by

$$I_D = I_S/W_0 \cdot W \cdot 10^{V_{GS}/S} \tag{11.30}$$

where S is the subthreshold slope. The output of the bias generator V_b equals

$$V_b = S \cdot \log(W_2/W_1) \tag{11.31}$$

From the total transistor width of all transistors in the block that is to be monitored, W_{BLOCK}, the total current scaling factor can be found:

$$\frac{I_{LCM}}{I_{BLOCK}} = \frac{W_{LCM}}{W_{BLOCK}} \cdot 10^{\frac{V_b}{S}} = \frac{W_{LCM}}{W_{BLOCK}} \cdot \frac{W_2}{W_1} \tag{11.32}$$

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

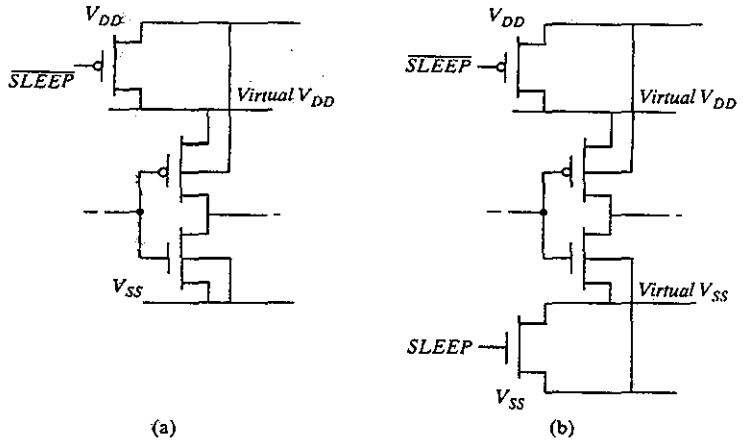


Figure 11-62 Sleep transistors used (a) only on the supply rail, (b) on both supply and ground.

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:

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

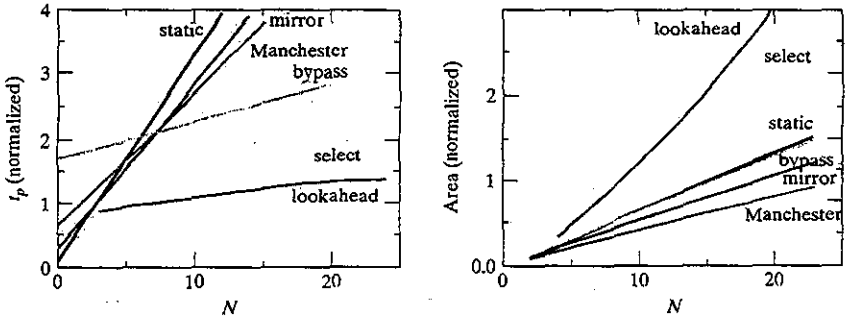


Figure 11-63 Area and propagation delay of various adder structures as a function of the number of bits *N*. Based on results from [Vermassen86].

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, Ed., *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-μm 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, Ed., "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. Eshragian, *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.**

CHAPTER

12

Designing Memory and Array Structures

Memory classification and architecture

Data-storage cells for read-only, nonvolatile, and read–write memories

Peripheral circuits—sense amplifiers, decoders, drivers, and timing generators

Power consumption and reliability issues in memory design

12.1 Introduction

A large portion of the silicon area of many contemporary digital designs is dedicated to the storage of data values and program instructions. More than half of the transistors in today's high-performance microprocessors are devoted to cache memories, and this ratio is expected to further increase. The situation is even more dramatic at the system level. High-performance workstations and computers contain several Gbytes of semiconductor memory, a number that is continuously rising. With the introduction of semiconductor audio (MP3) and video players (MPEG4), the demand for nonvolatile storage has skyrocketed. Obviously, dense data-storage circuitry is and will be one of the primary concerns of a digital circuit or system designer. The total market share for semiconductor memories is expected to be over $45 billion in 2003, which is twice as large as it was in 1998.

In Chapter 7, we introduced means of storing Boolean values based on either positive feedback or capacitive storage. While semiconductor memories are built on the same concepts, the simple use of a register cell as a means for mass storage leads to excessive area requirements. Memory cells are therefore combined into large arrays, which minimizes the overhead caused by peripheral circuitry and increases the storage density. The sheer size and complexity of these array structures introduces a variety of design problems, some of which are discussed in this chapter.

We first introduce the basic memory architectures and their essential building blocks. Next, we analyze the different memory cells and their properties. The cell structure and topology is mainly driven by the available technology, and is somewhat out of the control of the digital designer. On the other hand, the peripheral circuitry has a tremendous impact on the robustness, performance, and power consumption of the memory unit. Therefore, a careful analysis of the options and considerations of the periphery design is appropriate. Reliability and power dissipation are two other large concerns of the semiconductor memory designer, and they are discussed in separate sections.

An interesting aspect of Chapter 12 is that it applies a large number of the circuit techniques introduced in the earlier chapters. In a sense, one can consider memory design as a case study of high-performance, high-density, low-power circuit design. This becomes quite clear from the two case studies that conclude the chapter.

12.1.1 Memory Classification

Electronic memories come in many different formats and styles. The type of memory unit that is preferable for a given application is a function of the required memory size, the time it takes to access the stored data, the access patterns, the application, and the system requirements.

Size Depending upon the level of abstraction, different means are used to express the size of a memory unit. The *circuit* designer tends to define the size of the memory in terms of *bits* that are equivalent to the number of individual cells (flip-flops or registers) needed to store the data. The *chip* designer expresses the memory size in *bytes* (groups of 8 or 9 bits) or its multiples—kilobytes (Kbyte), megabytes (Mbyte), gigabytes (Gbyte), and ultimately terabytes (Tbyte). The *system* designer likes to quote the storage requirement in terms of *words*, which represent a basic computational entity. For instance, a group of 32 bits represents a word in a computer that operates on 32-bit data.

Timing Parameters The timing properties of a memory are illustrated in Figure 12-1. The time it takes to retrieve (*read*) from the memory is called the *read-access time*, which is equal to the delay between the read request and the moment the data is available at the output. This time is different from the *write-access time*, which is the time elapsed between a write request and the final writing of the input data into the memory. Finally, another important parameter is the (read or write) *cycle time* of the memory, which is the minimum time required between successive reads or writes. This time is normally greater than the access time for reasons that become apparent later in the chapter. Read and write cycles do not necessarily have the same length, but their lengths are considered equal for simplicity of system design.

Function Semiconductor memories are most often classified on the basis of memory functionality, access patterns, and the nature of the storage mechanism. A distinction is made between *read-only* (ROM) and *read–write* (RWM) memories. The RWM structures have the advantage of

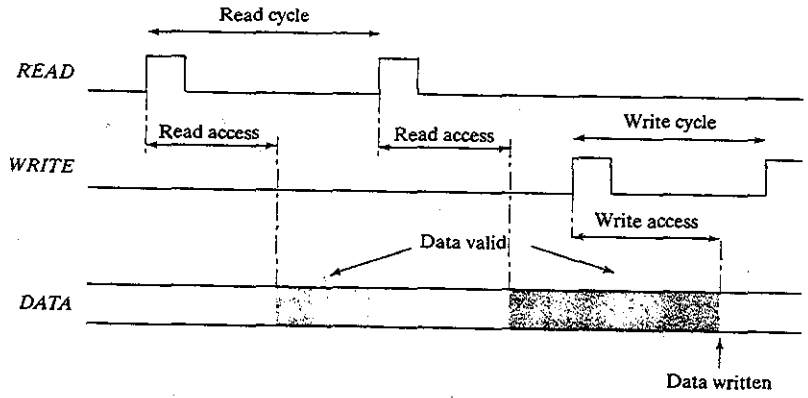


Figure 12-1 Memory-timing definitions.

offering both read and write functionality with comparable access times and are the most flexible memories. Data are stored either in flip-flops or as a charge on a capacitor. As in the classification introduced in the discussion on sequential circuitry, these memory cells are called *static* and *dynamic*, respectively. The former retain their data as long as the supply voltage is retained, while the latter need periodic refreshing to compensate for the charge loss caused by leakage. Since RWM memories use active circuitry to store the information, they belong to the class called *volatile* memories, in which the data is lost when the supply voltage is turned off.

Read-only memories, on the other hand, encode the information into the circuit topology—for example, by adding or removing transistors. Since this topology is hard wired, the data cannot be modified; it can only be read. Furthermore, ROM structures belong to the class of the *nonvolatile* memories. Disconnection of the supply voltage does not result in a loss of the stored data.

The most recent entry in the field are memory modules that can be classified as nonvolatile, yet offer both read and write functionality. Typically, their write operation takes substantially more time than the read. We call them *nonvolatile read-write* (NVRWM) memories. Members of this family are the *EPROM* (erasable programmable read-only memory), *E^2PROM* (electrically erasable programmable read-only memory), and *flash* memories. The emergence of novel, cheap, and dense nonvolatile technologies over the last decade has made this approach to storage the fastest growing in the memory arena.

Access Pattern A second memory classification is based on the order in which data can be accessed. Most memories belong to the *random-access* class, which means memory locations can be read or written in a random order. One would expect memories of this class to be called *RAM* modules (random-access memory). For historical reasons, this name has been reserved for the random-access RWM memories, probably because the RAM acronym is more easily pronounced than the awkward RWM. Be aware that most ROM or NVRWM units also provide random access, but the acronym RAM should not be used for them.

Some memory types restrict the order of access, which results in either faster access times, smaller area, or a memory with a special functionality. Examples of such are the *serial memories*: the FIFO (*first-in first-out*), LIFO (*last-in first-out*, most often used as a stack), and the *shift register*. *Video memories* are an important member of this class. In video processing, data is acquired and outputted serially, and random access is not required. *Contents-addressable memories* (CAM) represent another important class of nonrandom access memories. Instead of using an address to locate the data, a CAM (also called an *associative memory*), uses a word of data itself as input in a query-style format. When the input data matches a data word stored in the memory array, a MATCH flag is raised. The MATCH signal remains low if no data stored in the memory corresponds to the input word. Associative memories are an important component of the cache architecture of many microprocessors.

An overview of the memory classes, as introduced earlier, is given in Figure 12-2. Implementations for each of the mentioned memory structures are discussed in subsequent sections. It

RWM		NVRWM	ROM
Random Access	**Non-Random Access**	EPROM E^2PROM FLASH	Mask-programmed programmable (PROM)
SRAM DRAM	FIFO LIFO Shift register CAM		

Figure 12-2 Semiconductor memory classification.

will be demonstrated how the nature of the memory affects not only the choice of the basic storage cell, but also the composition of the peripheral units.

Input/Output Architecture A final classification of semiconductor memories is based on the *number of data input and output ports*. While a majority of the memory units presents only a single port that is shared between input and output, memories with higher bandwidth requirements often have multiple input and output ports—and thus are called *multiport memories*. Examples of the latter are the register files used in RISC (reduced instruction set computer) microprocessors. Adding more ports tends to complicate the design of the storage cell.

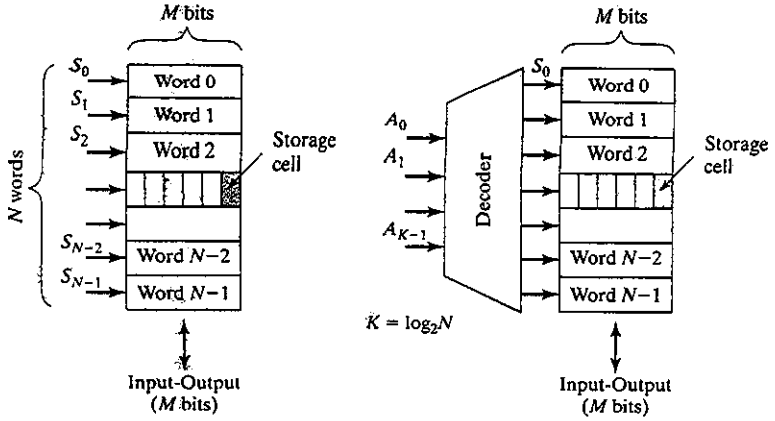
Application Before the end of the century, most large-size memories were packaged as stand-alone ICs. With the advent of the system-on-a-chip and the integration of multiple functions on a single die, an ever larger fraction of memory is now integrated on the same die as the logic functionality. Memories of this type are called *embedded*. The colocation of these diverse functions has a severe impact on the memory design—not only on the overall memory architecture, but also on the its underlying technology and circuit techniques.

When massive amounts of storage are needed (multiples of Gigabytes and more), semiconductor memories tend to become too expensive. More cost-effective technologies such as magnetic and optical disk should be used. While these provide extensive storage capabilities at a low cost per bit, they tend to be either slow or provide limited access patterns. For instance, a magnetic tape generally allows for serial access only. For these reasons, such memories do not communicate directly with the computing processor, but are interfaced through a number of faster semiconductor memories. They are called *secondary* or *tertiary* memories and are beyond the scope of this textbook.

12.1.2 Memory Architectures and Building Blocks

When implementing an N-word memory where each word is M bits wide,[1] the most intuitive approach is to stack the subsequent memory words in a linear fashion, as shown in Figure 12-3a. One word at a time is selected for reading or writing with the aid of a *select* bit (S_0 to S_{N-1}), if we

[1]The length of a word varies between 1 and 128 bits. In commercial memory chips, the word length typically equals 1, 4, or 8 bits.



(a) Intuitive architecture for $N \times M$ memory

(b) A decoder reduces the number of address bits

Figure 12-3 Architectures for N-word memory (where each word is M bits).

assume that this module is a single-port memory. In other words, only one signal S_i can be high at any time. For simplicity, let us temporarily assume that each storage cell is a D flip-flop and that the select signal is used to activate (clock) the cell. While this approach is relatively simple and works well for very small memories, one runs into a number of problems when trying to use it for larger memories.

Assume that we would like to implement a memory that holds 1 million ($N = 10^6$) 8-bit ($M = 8$) words. The reader should be aware that 1 million is a simplification of the actual memory size, since memory dimensions always come in powers of two. In this particular case, the actual number of words equals $2^{20} = 1024 \times 1024 = 1{,}048{,}576$. For ease of use, it is common practice to denote such a memory as 1 Mword unit.

When implementing this structure using the strategy of Figure 12-3a, we quickly realize that 1 million select signals are needed—one for every word. Since these signals are normally provided from off-chip or from another part of the chip, this translates into insurmountable wiring and/or packaging problems. A *decoder* is inserted to reduce the number of select signals (Figure 12-3b). A memory word is selected by providing a binary encoded *address word* (A_0 to A_{K-1}). The decoder translates this address into $N = 2^K$ select lines, only one of which is active at a time. This approach reduces the number of external address lines from 1 million to 20 ($\log_2 2^{20}$) in our example, which virtually eliminates the wiring and packaging problems. The decoder is typically designed so that its dimensions are matched to the size of the storage cell and the connections between the two, in particular the S signals in Figure 12-3b, do not produce any area overhead. The value of this approach can be appreciated by interpreting Figure 12-3b as a physical floor plan of the memory module. By performing the pitch matching between decoder and memory core, the S wires can be very short, and no large routing channel is required.

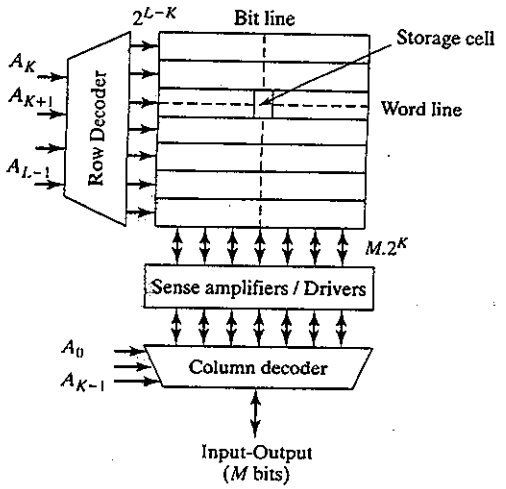


Figure 12-4 Array-structured memory organization.

While this resolves the select problem, it does not address the issue of the memory aspect ratio. Evaluation of the dimensions of the storage array of our token example shows that its height is approximately 128,000 times larger than its width ($2^{20}/2^3$), assuming the shape of the basic storage cell is approximately square which is almost always the case. Obviously, this results in a design that cannot be implemented. Besides the bizarre shape factor, the resulting design is also extremely slow. The vertical wires connecting the storage cells to the input/outputs become excessively long. Remember that the delay of an interconnect line increases at least linearly with its length.

To address this problem, memory arrays are organized so that the vertical and horizontal dimensions are of the same order of magnitude; thus, the aspect ratio approaches unity. Multiple words are stored in a single row and are selected simultaneously. To route the correct word to the input/output terminals, an extra piece of circuitry called the *column decoder* is needed. The concept is illustrated in Figure 12-4. The address word is partitioned into a *column address* (A_0 to A_{K-1}) and a *row address* (A_K to A_{L-1}). The row address enables one row of the memory for *R/W*, while the column address picks one particular word from the selected row.

Example 12.1 Memory Organization

An alternative choice would be to organize the memory core of our example as an array of 4000 by 2000 cells (to be more precise, 4096 × 2048), which approaches a square aspect ratio. Each of the 4000 rows stores 256 8-bit words. This results in a row address of 12 bits, while the column address measures 8 bits. It can be verified that the total address space still equals 20 bits.

Figure 12-4 introduces commonly used terminology. The horizontal select line that enables a single row of cells is called the *word line*, while the wire that connects the cells in a single column to the input/output circuitry is named the *bit line*.

The area of large memory modules is dominated by the size of the memory core. Thus, it is crucial to keep the size of the basic storage cell as small as possible. We could use one of the register cells introduced in Chapter 7 to implement a *R/W* memory. Such a cell easily requires more than 10 transistors per bit, and employing it in a large memory results in excessive area requirements. Semiconductor memory cells therefore reduce the cell area by trading off some desired properties of digital circuits, such as noise margin, logic swing, input/output isolation, fan-out, or speed. While a degradation of some of those properties is allowable within the confined domain of the memory core where noise levels can be tightly controlled, this is not acceptable when interfacing with the external or surrounding circuitry. The desired digital signal properties must be recovered with the aid of peripheral circuitry.

For example, it is common to reduce the voltage swing on the bit lines to a value substantially below the supply voltage. This reduces both the propagation delay and the power consumption. A careful control of the cross talk and other disturbances is possible within the memory array, ensuring that sufficient noise margin is obtained even for these small signal swings. Interfacing to the external world, on the other hand, requires an amplification of the internal swing to a full rail-to-rail amplitude. This is achieved by the *sense amplifiers* shown in Figure 12-4. The design of those peripheral circuits is discussed in Section 12.3. Relaxation of bounds on a number of the coveted digital properties makes it possible to reduce the transistor count of a single memory cell to between one and six transistors!

The architecture of Figure 12-4 works well for memories up to a range of 64 Kbits to 256 Kbits. Larger memories start to suffer from a serious speed degradation as the length, capacitance, and resistance of the word and bit lines become excessively large. Larger memories have consequently gone one step further and added one extra dimension to the address space, as illustrated in Figure 12-5.

The memory is partitioned into P smaller blocks. The composition of each of the individual blocks is identical to one of Figure 12-4. A word is selected on the basis of the row and column addresses that are broadcast to all the blocks. An extra address word called the *block address*, selects one of the P blocks to be read or written. This approach has a dual advantage.

1. The length of the local word and bit lines—that is, the length of the lines within the blocks—is kept within bounds, resulting in faster access times.
2. The block address can be used to activate only the addressed block. Nonactive blocks are put in power-saving mode with sense amplifiers and row and column decoders disabled. This results in a substantial power saving that is desirable, since power dissipation is a major concern in very large memories.

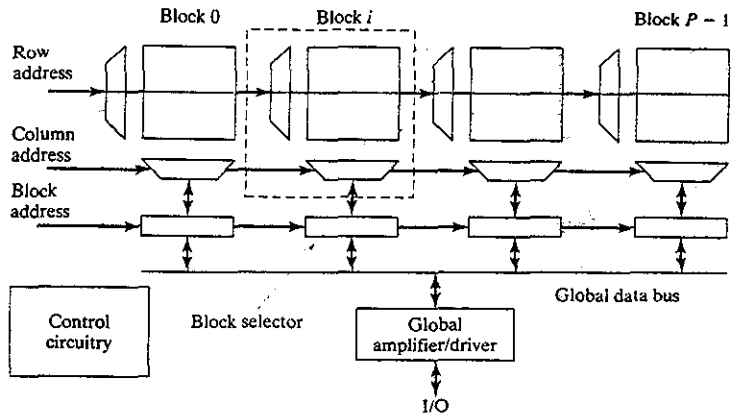


Figure 12-5 Hierarchical memory architecture. The block selector enables a single memory block at a time.

Example 12.2 Hierarchical Memory Architecture

As an example, a 4-Mbit SRAM can be designed [Hirose90] as a composition of 32 blocks, each of which contains 128 Kbits (Figure 12-6). Each block is structured as an

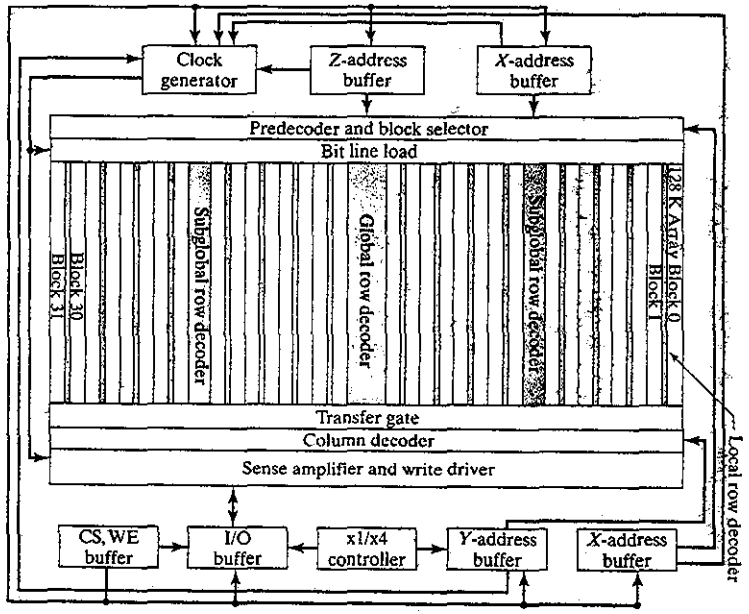


Figure 12-6 Block diagram of 4 Mbit memory (from [Hirose90]).

array with 1024 rows and 128 columns. The row address (X), column address (Y), and block address (Z) are 10, 7, and 5 bits wide, respectively.

Multiple variants of the proposed architecture are possible. Variations include the positioning of the sense amplifiers, the partitioning of the word and bit lines, and the styles of the decoders used. The underlying concept is the same—it is advantageous to partition a large memory into smaller subdivisions to combat the delay associated with extra long lines. The gains in performance and power consumption easily outweigh the overhead incurred by the partitioning.

The nature of *serial and contents-addressable memories* naturally leads to variations in the architecture and composition of the memory. Figure 12-7 shows an example of a 512-word CAM memory, which supports three modes of operation: read, write, and match. The read and write modes access and manipulate data in the CAM array in the same way as in an ordinary memory. The match mode is unique to associate memories. The *comparand* block is filled with the data pattern to match, and the *mask* word indicates which bits are significant. For example, to find all the words in the CAM array that have the pattern 0×123 in the most significant bits, we would fill the comparand with 0×12300000 and the mask with $0 \times$ FFF00000. All 512 rows of the CAM array then simultaneously compare the 12 most significant bits of the comparand with the data contained in that row. Every row that matches the pattern is passed to the validity block. Since we do not care about rows that contain invalid data (which typically happens when the array is not full), only the valid rows that match are passed to the *priority encoder*. In the event that two or more rows match the pattern, the address of the row in the CAM array is used to break the tie. To do this, the priority encoder considers all 512 match lines from the CAM array, selects the one with the highest address, and encodes it in binary.[2] Since there are 512 rows in

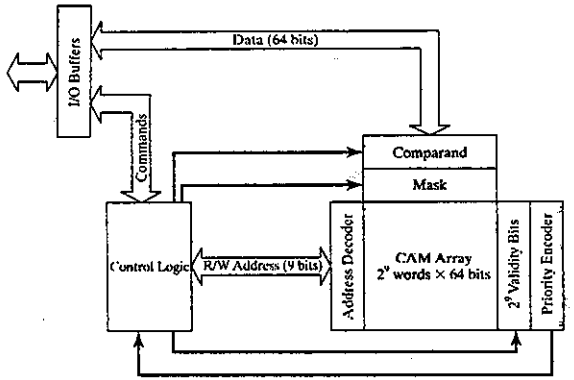


Figure 12-7 Architecture of 512-word contents-addressable memory.

[2]The highest address typically corresponds to the latest entry in the cache, and this the most desirable match.

the CAM array, 9 bits are required to indicate the highest row that matched. One additional 'match found' bit is provided, since it is possible that none of the rows matches the pattern.

Finally, a component of memory design that is often overlooked is the *input/output interface and control circuitry*. The nature of the I/O interface has an enormous impact on the global memory control and timing. This statement is illustrated by comparing the input–output behavior of typical DRAM and SRAM components with the associated timing structure.

Since the early days, DRAM designers have opted for a multiplexed addressing scheme. In this model, the lower and upper halves of the address words are presented sequentially on a single address bus. This approach reduces the number of package pins and has survived through the subsequent memory generations.DRAMs are generally produced in higher volumes. Lowering the pin count reduces cost and size at the expense of performance. The presence of a new address word is asserted by raising a number of strobe signals. (See Figure 12-8a.) Raising the *RAS* (row-access strobe) signal asserts that the *msb* part of the address is present on the address bus, and that the word-decoding process can be initiated. The *lsb* part of the address is applied next, and the *CAS* (column-access strobe) signal is asserted. To ensure correct memory operation, a careful timing of the *RAS–CAS* interval is necessary. In fact, the *RAS* and *CAS* signals act as clock inputs to the memory module, and are used to synchronize memory events, such as decoding, memory core access, and sensing.

The SRAM designers, on the other hand, have chosen a self-timed approach, as in Figure 12-8b. The complete address word is presented at once, and circuitry is provided to automatically detect any transitions on that bus. No external timing signals are needed. All internal timing events, such as the enabling of the decoders and sense amplifiers, are derived from the internally generated transition signal. This approach has the advantage that the cycle time of the SRAM is close or equal to its access time, while this is definitely not the case for the DRAM. The increased overall performance of compute systems that use DRAM as storage requires DRAM speeds to evolve at approximately the same pace. Several new approaches to improve the performance of the DRAM for read operations have been introduced. Examples are *Synchronous DRAM* (SDRAM) and *Rambus DRAM* (RDRAM). The main novelty in these new architectures, which are discussed later in the chapter, is not in the memory core, but in how they communicate with the outside world.

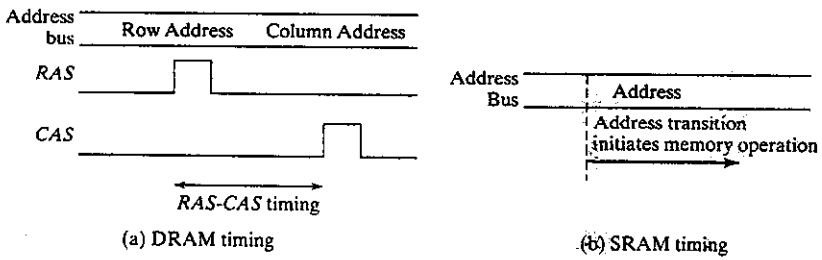


Figure 12-8 Input/output interface of DRAM and SRAM memories and their impact on memory control.

Designing the control and timing circuitry so that the memory is functional over a wide range of manufacturing tolerances and operating temperatures is a demanding task that requires extensive simulation and design optimization. It is an integral, but often overlooked, part of the memory-design process and has a major impact on both memory reliability and performance.

12.2 The Memory Core

This section concentrates on the design of the memory core and its composing cells for a variety of semiconductor-memory types. While the most compelling issue in designing large memories is to keep the size of the cell as small as possible, this should be done so that other important design-quality measures such as speed and reliability are not fatally affected. In sequence, we discuss *read-only*, *nonvolatile*, and *read–write* memory cores. This section is concluded with a short discussion of the associative memory cell.

12.2.1 Read-Only Memories

While the idea of a memory that can only be read and never altered might seem odd at first, but a second glance reveals a large number of potential applications. Programs for processors with fixed applications such as washing machines, calculators, and game machines, once developed and debugged, need only reading. Fixing the contents at manufacturing time leads to small and fast implementations.

ROM Cells—An Overview

The fact that the contents of a ROM cell are permanently fixed considerably simplifies its design. The cell should be designed so that a 0 or 1 is presented to the bit line upon activation of its word line. Figure 12-9 shows several ways to accomplish this.

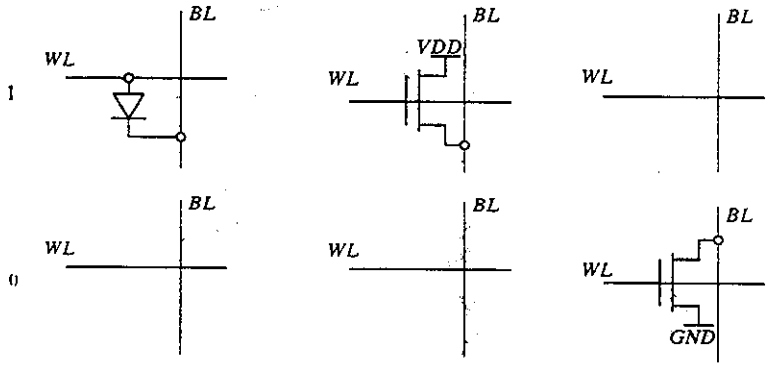


Figure 12-9 Different approaches for implementing 1 and 0 ROM cells.

Consider first the simplest cell, which is the diode-based ROM cell shown in Figure 12-9a. Assume that the bit line *BL* is resistively clamped to ground—that is, *BL* is pulled low through a resistor connected to ground lacking any other excitations or inputs. This is exactly what happens in the 0 cell. Since no physical connection between the word line *WL* and *BL* exists, the value on *BL* is low, independent of the value of *WL*. On the other hand, when a high voltage V_{WL} is applied to the word line of the 1 cell, the diode is enabled, and the word line is pulled up to $V_{WL} - V_{D(on)}$, resulting in a 1 on the bit line. In summary, the presence or absence of a diode between *WL* and *BL* differentiates between ROM cells storing a 1 or a 0, respectively.

The disadvantage of the diode cell is that it does not isolate the bit line from the word line. All current required to charge the bit line capacitance, which can be quite high for large memories, has to be provided through the word line and its drivers; therefore, this approach only works for small memories. A better approach is to use an active device in the cell, as proposed in Figure 12-9b. The diode is replaced by the gate-source connection of an NMOS transistor, whose drain is connected to the supply voltage. The operation is identical to that of the diode cell with one major difference: All output-driving current is provided by the MOS transistor in the cell. The word-line driver is only responsible for charging and discharging the word-line capacitance.

The improved isolation comes at the penalty of a more complex cell and a larger area. The latter is caused primarily by the extra supply contact. This contact must be provided in every cell, so that the supply rail must be distributed throughout the array. An example of a 4 × 4 array is shown in Figure 12-10. Notice how the overhead of the supply lines is reduced by sharing them between neighboring cells. This requires the *mirroring* of the odd cells around the horizontal axis, an approach that is extensively used in memory cores of all styles.

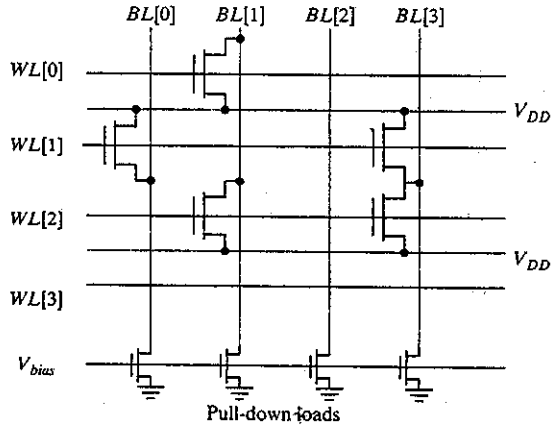


Figure 12-10 A 4 × 4 OR ROM cell array.

Problem 12.1 ROM Array

Determine the values of the data stored at addresses 0, 1, 2, and 3 in the ROM of Figure 12-10.

An alternative implementation is offered by the MOS cell of Figure 12-9c. To be operational, this cell requires the bit line to be resistively clamped to the supply voltage, or equivalently, the default value at the output must equal 1. The absence of a transistor between *WL* and *BL* thus means that a 1 is stored. The 0 cell is realized by providing an MOS device between bit line and ground. Applying a high voltage on the word line turns on the device, which in turn pulls down the bit line to GND. An example of a 4×4 MOS ROM array is shown in Figure 12-11. A PMOS load is used to pull up the bit lines in case none of the attached NMOS devices is enabled.

Problem 12.2 MOS NOR ROM Memory Array

Determine the values of the data stored at addresses 0, 1, 2, and 3 in the ROM of Figure 12-11.

Programming the ROM Memory

You may have noticed in the last example that the combination of a bit line, PMOS pull-up, and NMOS pull-downs constitutes nothing other than **a pseudo-NMOS NOR gate with the word lines as inputs**. An $N \times M$ ROM memory can be considered as a combination of M NOR gates with at most N inputs (for a fully populated column). It is therefore called a NOR ROM.[3] Under

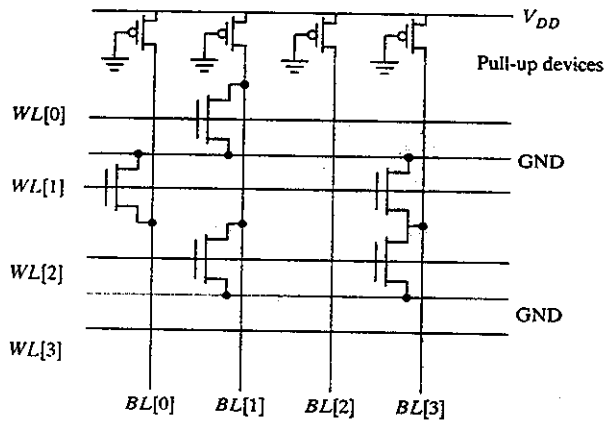


Figure 12-11 A 4×4 MOS NOR ROM.

[3]Similarly, the memory structure in Figure 12-10 implements an OR function, and hence is called an OR ROM.

normal operating conditions, only one of the word lines goes high, and, at most, one of the pull-down devices is turned on. This raises some interesting issues regarding the sizing of both the cell and pull-up transistors:

- To keep the cell size and the bit line capacitance small, the pull-down device should be kept as close as possible to minimum size.
- On the other hand, the resistance of the pull-up device must be larger than the pull-down resistance to ensure an adequate low level. In Chapter 6, we derived a factor of at least 4 for pseudo-NMOS gates. This large resistance has a detrimental effect on the low-to-high transitions on the bit lines. The bit line capacitance consists of the contributions of all connected devices and can be in the pF range for larger memories.

This is where memory and logic design differ. For the sake of density and performance, it is possible to relax some of the quality standards imposed on digital gates. In the NOR ROM, we can trade off noise margin for performance by letting the V_{OL} of the bit line stand at a higher voltage (e.g., 1 to 1.5 V for a 2.5-V supply). The pull-up device can now be widened, improving the low-to-high transition. The reduced noise margin is tolerable within the memory core, where the noise conditions and signal interferences can be carefully controlled. Going to the external world requires a restoration of the full voltage swing. This is accomplished by the peripheral devices—in this case, the sense amplifier. For instance, feeding the bit line into a complementary CMOS inverter with an appropriately adjusted switching threshold restores the full signal swing.

Figure 12-12 shows two possible layouts of the 4×4 NOR ROM array of Figure 12-11. The arrays are constructed by repeating the same cell in both the horizontal and vertical directions, mirroring the odd cells around the horizontal axis in order to share the *GND* wire. The difference between two artifacts lies in the way they are programmed. In the structure of Figure 12-12a, the memory is written (personalized) by selectively adding transistors when needed. This is accomplished with the aid of only the diffusion layer (the ACTIVE mask in the fabrication process). In the second approach (Figure 12-12b), the memory is programmed by the selective addition of metal-to-diffusion contacts. The presence of a metal contact to the bit line creates a 0 cell, while its absence indicates a 1 cell. Observe that only one mask layer, the CONTACT, is used to program the memory in this case.

A comparison of the ACTIVE and CONTACT implementations, based on identical design rules, reveals that the former results in an area savings of approximately 15%. On the other hand, the CONTACT programming strategy has the advantage that the contact layer is a later step in the manufacturing process. This delays the actual programming of the memory in the process cycle. Wafers can be prefabricated up to the CONTACT mask and stockpiled. The remaining fabrication steps can be executed quickly once a specific program is defined, reducing the turnaround time between order and delivery. In multilayer processes, programming is increasingly done in one of the VIA masks. Which programming approach is ultimately chosen depends on the dominant design metric—size/performance versus turnaround time.

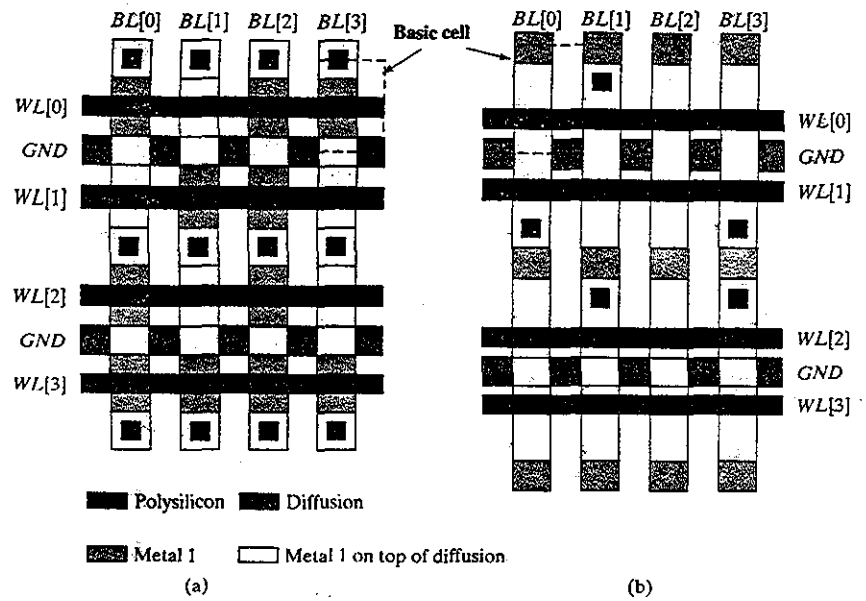


Figure 12-12 Possible layout of a 4 × 4 MOS NOR ROM. The bit lines are implemented in Metal 1 and are routed on top of the cell diffusion. *GND* lines are distributed horizontally in diffusion. The memory is programmed using (a) the ACTIVE layer, resulting in a cell size of 9.5 λ × 7 λ; (b) the CONTACT layer, yielding a cell of 11 λ × 7 λ.

Also, you may have observed that diffusion is used to route the *GND* signal. In general, this is an absolute "no-no." Yet, it is common practice in memories for the sake of increased density. A metal bypass with regularly spaced straps keeps the voltage drop over the ground wire within strict bounds.

It is important to note that the transistor occupies only a small ratio of the total cell size, which measures around 70 λ^2. It is actually possible to increase its size over the minimum dimensions without affecting the cell size. A transistor with a 4/2 aspect was chosen in the examples of Figure 12-12. A large part of the cell is devoted to the bit line contact and ground connection. One way to avoid this overhead is to adopt a different memory organization. Figure 12-13 shows a 4 × 4 ROM array based on a NAND configuration. All transistors constituting a column are connected in series. A basic property of a NAND gate is that all transistors in the pull-down chain must be on to produce a low value. The word lines must be operated in reverse-logic mode to make this memory function. All word lines are high by default with the exception of the selected row, which is set to 0. Transistors on nonselected rows are thus turned on. Now suppose that no transistor is present on the intersection between

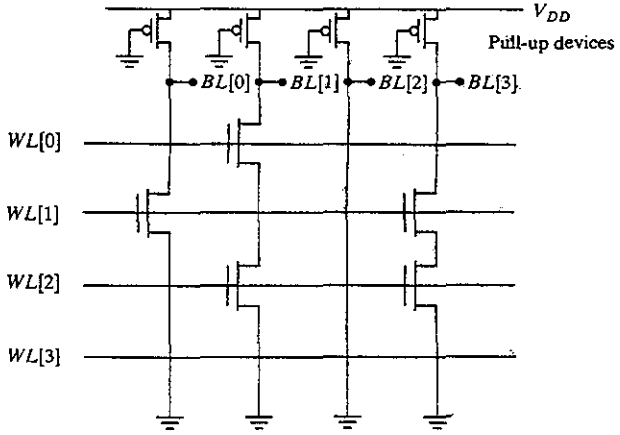


Figure 12-13 4 × 4 MOS NAND ROM.

the row and column of interest. Since all other transistors on the series chain are selected, the output is pulled low, and the stored value equals 0. On the other hand, a transistor present at the intersection is turned off when the associated word line is brought low. This results in a high output, which is equivalent to reading a 1.

Problem 12.3 MOS NAND ROM

Determine the values of the data stored at addresses 0, 1, 2, and 3 in the ROM of Figure 12-13.

The main advantage of the NAND structure is that the basic cell only consists of a transistor (or a lack of a transistor) and that no connection to any of the supply voltages is needed. This reduces the cell size substantially, as illustrated in the layouts of Figure 12-14. Again, different programming approaches can be employed. A first option is to use the METAL-1 layer to selectively short-circuit the transistor (a). The resulting cell reduces the cell size approximately 15% below the smallest NOR ROM cell. Even more impressive area reductions can be obtained if an extra implant step is available to program the memory. A threshold-lowering implant using *n*-type impurities turns the device into a depletion transistor, which is always on, regardless of the applied word-line voltage. It thus is equivalent to a shorted circuit. The resulting cell area of 30 λ^2 is more than two times smaller than the equivalent NOR ROM cell. This comes at a price, however. In the next section, we demonstrate that the NAND configuration results in a considerable loss in performance and is only useful for small memory arrays. Also, the extra implant step represents an additional process step and generally is not available to a designer.

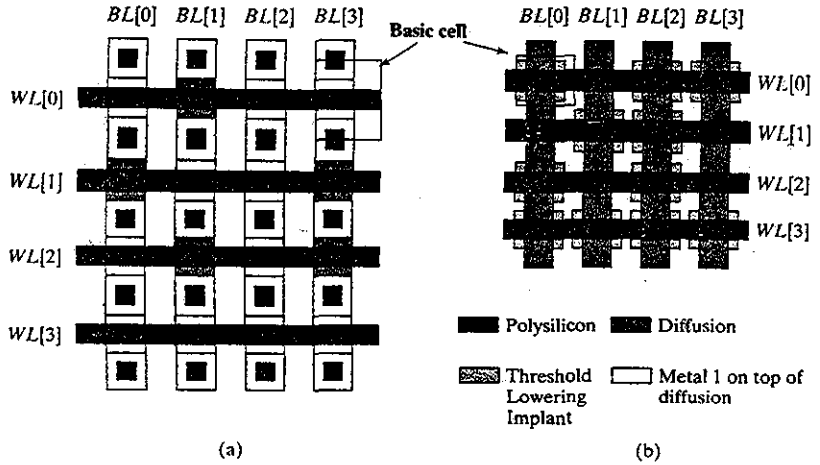


Figure 12-14 4 × 4 MOS NAND ROM (a) using metal-1 layer programming (cell size: 8 λ × 7 λ; (b) using threshold-lowering implants (cell size: 5 λ × 6 λ, assuming that the implant has to extend 1 λ in all directions from the gate).

Example 12.3 Voltage Swings in NOR and NAND ROMs

Assuming that the layouts of Figure 12-12 and Figure 12-14 are implemented using our standard 0.25–μm CMOS technology, determine the size of the PMOS pull-up device so that the worst case value of V_{OL} is never higher than 1.5 V (for a 2.5-V supply voltage). This translates into a bit line swing of 1 V. Determine the values for an 8 × 8 and a 512 × 512 array.

1. NOR ROM

Since, at most, one transistor can be on at a time, the value of V_{OL} is not a function of the array size nor the programming of the array. The low output voltage is computed by analyzing a pseudo-NMOS inverter with a single pull-down device of a (4/2) size. This circuit was analyzed in detail in Chapter 6 for a different operation region. It can be determined by inspection that both PMOS and NMOS are in velocity saturation for the aforementioned bias conditions:

$$\frac{(W/L)_p}{(W/L)_n} = \frac{k'_n[(V_{DD}-V_{Tn})V_{DSATn}-V_{DSATn}^2/2](1+\lambda_n V_{OL})}{-k'_p[(-V_{DD}-V_{Tp})V_{DSATp}-V_{DSATp}^2/2](1+\lambda_p(V_{OL}-V_{DD}))}$$

Solving for V_{DD} = 2.5 V and V_{OL} = 1.5 V leads to a PMOS/NMOS ratio of 2.62, or a required size for the PMOS device of $(W/L)_p = 5.24$.

2. NAND ROM

Due to the series chaining, the value of V_{OL} is a function of both the size of the memory (number of rows) and the programming. The worst case occurs when all bits in a column are set to 1, which means N transistors are connected in series in the pull-down network. Making the simplifying assumption that the N transistors can be replaced by an N-times longer device, the values of $(W/L)_p$ can be derived for the (8 × 8) and (512 × 512) cases. Observe that the design of Figure 12-14b uses minimum-size devices of (3/2):

$$(8 \times 8)\text{: } (W/L)_p = 0.49$$

$$(512 \times 512)\text{: } (W/L)_p = 0.0077$$

While the first case still produces acceptable results for the PMOS device, the second one would require an extremely long pull-up transistor, which is unacceptable. For this reason, NAND ROMs are rarely used for arrays with more than 8 or 16 rows.

ROM Transient Performance

The transient response of a memory array is defined as the time it takes from the time a word line switches until the point where the bit line has traversed a certain voltage swing ΔV. Since the bit line normally feeds into a sense amplifier, it is not necessary for it to traverse its complete swing. A voltage drop (or rise) of ΔV is sufficient to make the sense amplifier react. Typical values of ΔV range around 0.5 V.

One important difference between the analysis of the propagation delay of a logic gate and that of a memory array is that most of the delay is attributable to the interconnect parasitics. An accurate modeling of these parasitics is therefore of prime importance. This is illustrated in the example that follows, which extracts the parasitic resistance and capacitance of the word and bit lines of both the NOR and NAND ROM arrays introduced earlier. We cover this example quite extensively because the same approach also holds for other memory styles, such as SRAM and DRAM.

Example 12.4 Word- and Bit Line Parasitics

In this example, we first derive an equivalent model of the memory arrays of Figure 12-12b and Figure 12-14b, respectively. Taking into account all transistors in the array simultaneously quickly leads to intractable equations and models, especially for larger memories. Simplification and abstraction is the obvious approach to be followed. We consider only the case of the (512 × 512) array.

1. NOR ROM

Figure 12-15 shows a model that is appropriate for the analysis of the word- and bit line delay of the NOR ROM. The word line is best modeled as a distributed RC line, since it is implemented in polysilicon with a relatively high sheet resistance. The use of a silicided polysilicon is definitely advisable. The bit line, on the other hand, is implemented in

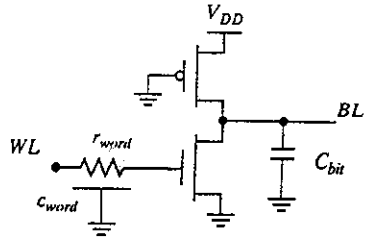


Figure 12-15 Equivalent transient model for the NOR ROM.

aluminum, and the resistance of the line only comes into play for very long lines. It is reasonable to assume for this example that a purely capacitive model is adequate and that all capacitive loads connected to the wire can be lumped into a single element.

Word-line parasitics (for the memory array of Figure 12-12b) are as follows:

Resistance/cell: $(7/2) \times 5\ \Omega/\text{sq} = 17.5\ \Omega$ (using the data of Table 4-5 on page 145)

Wire capacitance/cell: $(3\ \lambda \times 2\ \lambda)\ (0.125)^2\ 0.088 + 2 \times (3\ \lambda \times 0.125) \times 0.054 = 0.049$ fF (from Table 4-2 on page 143)

Gate capacitance/cell: $(4\ \lambda \times 2\ \lambda)\ (0.125)^2\ 6 = 0.75$ fF.

The bit line parasitics are as follows:

Resistance/cell: $(11/4) \times 0.1\ \Omega/\text{sq} = 0.275\ \Omega$ (which is negligible)

Wire capacitance/cell: $(11\ \lambda \times 4\ \lambda)\ (0.125)^2\ 0.041 + 2 \times (11\ \lambda \times 0.125) \times 0.047 = 0.09$ fF

Drain capacitance/cell:

$((5\ \lambda \times 4\ \lambda)\ (0.125)^2 \times 2 \times 0.56 + 14\ \lambda \times 0.125 \times 0.28 \times 0.6 + 4\ \lambda \times 0.125 \times 0.31 = 0.8$ fF

The latter term deserves an explanation. The *drain capacitance* contributed by every cell connected to the bit line consists of the bottom junction, side wall, and overlap capacitances. It may be assumed that all cells, besides the one being switched, are in the off state, which explains why only the overlap part of the gate capacitance is taken into account. It is assumed further that the bit line swings between 1.5 V and 2.5 V. Under these conditions, the K_{eq} factor evaluates to 0.56 and 0.6 for area and side wall, respectively. For all other capacitance values, please refer to Table 3-5 on page 112.

2. NAND ROM

As in the approach taken for the NOR ROM, we can derive an equivalent model for the analysis of the delay of the NAND structure. While the word-line model is identical, modeling the bit line behavior is more complex due to the long chain of series-connected transistors. The worst case behavior occurs when the transistor at the bottom of the chain is switched and the column is completely populated with transistors. A model approximating the behavior for that case is shown in Figure 12-16. Each of the series transistors (which are normally in the on mode) is modeled as a resistance–capacitance combination. The entire chain can be modeled as a distributed *rc*-network for large memories.

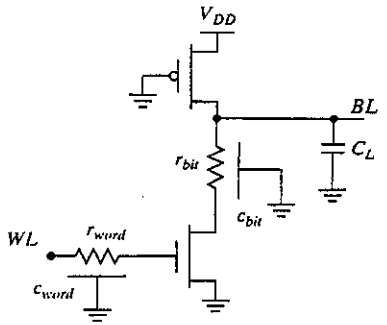


Figure 12-16 Equivalent model for word and bit line of NAND ROM.

Word-line parasitics (for the memory array of Figure 12-14) are as follows:

Resistance/cell: $(6/2) \times 5\ \Omega/\text{sq} = 15\ \Omega$

Wire capacitance/cell: $(3\ \lambda \times 2\ \lambda)\ (0.125)^2\ 0.088 + 2 \times (3\ \lambda \times 0.125) \times 0.054 = 0.049$ fF

Gate Capacitance/cell: $(3\ \lambda \times 2\ \lambda)\ (0.125)^2\ 6 = 0.56$ fF

The bit line parasitics are as follows:

Resistance/cell: $13 / 1.5 = 8.7\ \text{k}\Omega$

Wire capacitance/cell: Included in diffusion capacitance

Finally, the source/drain capacitance/cell is given by

$((3\ \lambda \times 3\ \lambda)\ (0.125)^2 \times 2 \times 0.56 + 2 \times 3\ \lambda \times 0.125 \times 0.28 \times 0.6 + (3\ \lambda \times 2\ \lambda)\ (0.125)^2 \times 6 = 0.85$ fF

The source/drain capacitance must include the gate-source and gate-drain capacitances, which means the complete gate capacitance. This is in contrast to the NOR case, in which only the drain-source overlap capacitance was included.

Determining the average transistor resistance is more complex. In fact, the resistance varies from device to device depending on the position in the chain, caused by differing gate-source voltages and body effects. In this first-order analysis, we simply use the equivalent resistance derived in Chapter 3. This approximation yielded reasonable results in the 4-input NAND-gate example of Chapter 6 (Example 6.4).

One might wonder why we bother with models at all, if simulations could readily produce more accurate results. One has to be aware that the memory modules can contain thousands to millions of transistors, making repeated simulations prohibitively slow. Models also help us to understand the behavior of the memory more readily. Computer simulations do not tell where the bottlenecks are located and how to address them.

Using the computed data and the equivalent model, an estimated value of the propagation delay of the memory core and its components can now be derived.

Example 12.5 Propagation delay of a 512 × 512 NOR ROM

1. The word-line delay of the distributed *rc*-line containing M cells can be approximated using the expressions derived in Chapter 4:

$$t_{word} = 0.38\,(r_{word} \times c_{word})\,M^2 = 0.38\,(17.5\ \Omega \times (0.049 + 0.75)\ \text{fF})\ 512^2 = 1.4\ \text{ns}$$

2. For the bit line, its response time depends upon the transition direction. Assuming a (0.5/0.25) pull-down device and a (1.3125/0.25) pull-up transistor, as derived in Example 12.3, we can compute the propagation delay using the familiar techniques:

$$C_{bit} = 512 \times (0.8 + 0.09)\ \text{fF} = 0.46\ \text{pF}$$

$$t_{HL} = 0.69\,(13\ \text{k}\Omega / 2 \parallel 31\ \text{k}\Omega / 5.25)\ 0.46\ \text{pF} = 0.98\ \text{ns}$$

The low-to-high response time can be computed using a similar approach:

$$t_{LH} = 0.69\,(31\ \text{k}\Omega / 5.25)\ 0.46\ \text{pF} = 1.87\ \text{ns}$$

Inspection of the preceding results shows that the word-line delay dominates. The former is almost completely due to the large resistance of the polysilicon wire. Some of the techniques to reduce the delay of distributed *rc*-lines, as introduced in Chapter 9, come in handy here. Driving the address line from both sides and using metal bypass lines (often called *global word lines*) go a long way towards addressing the word-line delay problem (Figure 9-18). Yet, the most effective approach is to carefully partition the memory into sub-blocks of adequate size that balance word- and bit line delay. Partitioning also helps to reduce the energy consumption attributed to driving and switching the word lines.

If necessary, the bit line delay can be reduced as well. The approach most often used is to further reduce the voltage swing on the bit line and to let the sense amplifier restore the output signal to the full swing. Voltage swings around 0.5 V are quite common.

Example 12.6 Propagation Delay of NAND ROM

Using techniques similar to the ones used in Example 12.5, we determine the word-line and bit line delay of the 512 × 512 NAND ROM.

1. The word-line delay is quite similar to that of the NOR case:

$$t_{word} = 0.38\,(r_{word} \times c_{word})\,M^2 = 0.38\,(15\ \Omega \times (0.049 + 0.56)\ \text{fF})\ 512^2 = 1.3\ \text{ns}$$

2. For the bit line delay, the worst case occurs when the complete column is populated with 0s except one, and the bottommost transistor is turned on. The distributed model offers a fair approximation of the delay (although this ignores the impact of the pull-up transistor):

$$t_{HL} = 0.38 \times 8.7\ \text{k}\Omega \times 0.85\ \text{fF} \times 511^2 = 0.73\ \mu\text{s}$$

The worst case for the low-to-high transition occurs when the bottommost transistor is turned off. Using the Elmore delay approach, we find that

$$t_{LH} = 0.69\ (31\ \text{k}\Omega / 0.0077)\ (511 \times 0.85\ \text{fF}) = 1.2\ \mu\text{s}$$

These delays are clearly unacceptable in most cases. Partitioning the memory into smaller modules is the only plausible option.

Power Consumption and Precharged Memory Arrays

The proposed NAND and NOR structures inherit all the disadvantages of the pseudo-NMOS gate discussed in Chapter 6:

1. **Ratioed logic.** The V_{OL} is determined by the ratio of the pull-up and pull-down devices. This can result in unacceptable transistor ratios, as demonstrated earlier in the examples.
2. **Static power consumption.** A static current path exists between the supply rails when the output is low. This may cause severe power dissipation problems.

Example 12.7 Static Power Dissipation of NOR ROM

Consider the case of the (512 × 512) NOR ROM. It is reasonable to assume that, on the average, 50% of the outputs are low. The standby current for the design of Example 12.4 equals approximately 0.21 mA (for an output voltage of 1.5 V). This translates into a total static dissipation of (512/2) × 0.21 mA × 2.5 V = 0.14 W, which is consumed even when nothing happens. Obviously, this is far from desirable.

To address these two issues, one can fall back on the same practices used in designing digital gates. One approach would be to use fully complimentary NAND or NOR gates. The larger number of transistors and the connection to both supply rails makes this approach unattractive from an area perspective. A better approach is to use precharged logic, as shown in Figure 12-17. This approach eliminates the static dissipation as well as the ratioed logic requirements, while keeping the cell complexity the same. Since the logic structure of both the NAND and NOR ROM is simple and is only one level deep, it is possible to ensure that all pull-down paths are off during precharging. This allows us to eliminate the enabling NMOS transistor at the bottom of the pull-down network, keeping the cell simple.

The dynamic architecture enables independent control of the pull-up and pull-down timing. For instance, the PMOS precharge device can be made as large as necessary. Be aware that this transistor loads the clock driver, which might become increasingly hard to design.

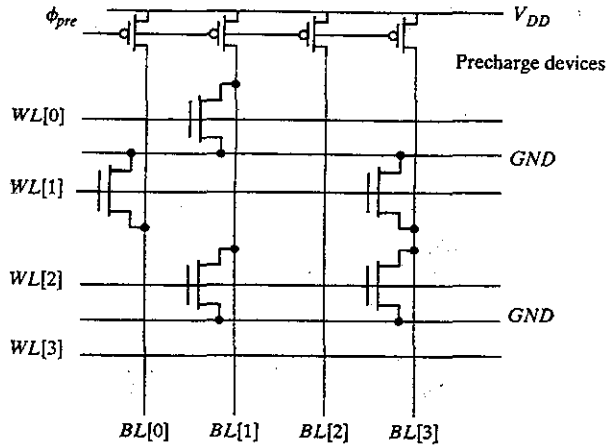


Figure 12-17 Precharged (4 x 4) MOS NOR ROM.

Problem 12.4 Precharged NAND ROM

While precharging works great for NOR ROMs, some severe problems emerge when it is applied to NAND ROMs. Explain why.

The excellent properties of the precharged approach have made it the memory structure of choice. Virtually all large memories currently designed, including NVRWM and RAMs, use dynamic precharging.

ROM Memories: A User Perspective

The reader should be aware that most of the static, dynamic, and power problems raised in the preceding sections are general in nature and apply to other memory architectures as well. Before addressing some of these structures, it is worth discussing the classification of ROM modules and programming approaches. The first class of ROM modules comprises the so-called *application-specific* ROMs, where the memory module is part of a larger custom design and programmed for that particular application only. Under these circumstances, the designer has all degrees of freedom and can use any mask layer (or combination thereof) to program the device.

A second class is formed by the *commodity* ROM chips, where a vendor mass-produces memory modules that are later customized according to customer specifications. Under these circumstances, it is essential that the number of process steps involved in programming be minimal and that they can be performed as a last phase of the manufacturing process. In this way, large amounts of unprogrammed dies can be preprocessed. This *mask-programmable* approach preferably uses the contact (or a metal) mask to personalize or program the memory, as was shown in some of the examples. The programming of a ROM module involves the manufacturer,

which introduces an unwelcome delay in product development. It has consequently become increasingly unpopular. A variant of this approach is emerging in the system-on-a-chip arena, where the majority of the chip is preprocessed. Only a minor fraction of the die is mask programmed, preferably using one of the upper metal layers. This could be used, for instance, to program the microcontroller, embedded on the chip, for a variety of applications.

A more desirable approach is for the client to program the memory at his own facility. One technology that offers such capability is the PROM (Programmable ROM) structure that allows the customer to program the memory one time; hence, it is called a WRITE ONCE device. This is most often accomplished by introducing *fuses* (implemented in nichrome, polysilicon, or other conductors) in the memory cell. During the programming phase, some of these fuses are blown by applying a high current, which disables the connected transistor.

While PROMs have the advantage of being "customer programmable," the single write phase makes them unattractive. For instance, a single error in the programming process or application makes the device unusable. This explains the current preference for devices that can be programmed several times (albeit slowly). The next section explains how this can be achieved.

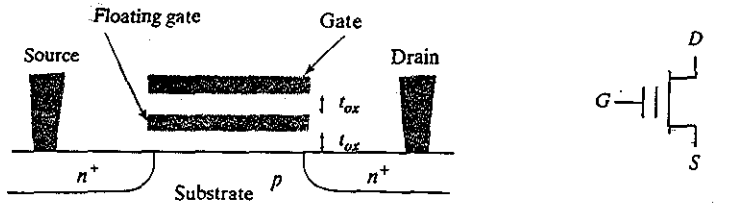
12.2.2 Nonvolatile Read–Write Memories

The architecture of the NVRW memories is virtually identical to the ROM structure. The memory core consists of an array of transistors placed on a word-line/bit line grid. The memory is programmed by selectively disabling or enabling some of those devices. In a ROM, this is accomplished by mask-level alterations. In an NVRW memory, a modified transistor that permits its threshold to be altered electrically is used instead. This modified threshold is retained indefinitely (or at least over a long lifetime) even when the supply voltage is turned off. To reprogram the memory, the programmed values must be *erased*, after which a new programming round can be started. The method of erasing is the main differentiating factor between the various classes of reprogrammable nonvolatile memories. The programming of the memory is typically an order of magnitude slower than the reading operation.

We start this section with a description of the floating-gate transistor, which is the device at the heart of the majority of the reprogrammable memories. The rest of the section is devoted to a number of alterations of the device, mainly with respect to the erase procedure. These modifications are at the source of the different NVRWM families. The section is concluded with a discussion of some emerging nonvolatile memories.

The Floating-Gate Transistor

Over the years, various attempts have been made to create a device with electrically alterable characteristics and enough reliability to support a multitude of write cycles. For example, the MNOS (metal nitride oxide semiconductor) transistor held promise, but has been unsuccessful until now. In this device, threshold-modifying electrons are trapped in a Si_3N_4 layer deposited on top of the gate SiO_2 [Chang77]. A more accepted solution is offered by the floating-gate transistor shown in Figure 12-18, which forms the core of virtually every NVRW memory built today. The structure is similar to a traditional MOS device, except that an extra polysilicon strip is



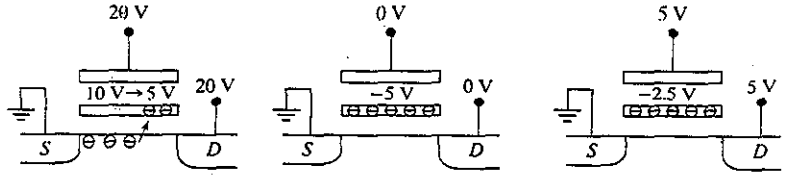
(a) Device cross section (b) Schematic symbol

Figure 12-18 Floating-gate transistor (FAMOS).

inserted between the gate and channel. This strip is not connected to anything and is called a *floating gate*. The most obvious impact of inserting this extra gate is to double the gate oxide thickness t_{ox}, which results in a reduced device transconductance as well as an increased threshold voltage. Both these properties are not particularly desirable. From other points of view, this device acts as a normal transistor.

More important, this device has the interesting property that its threshold voltage is programmable. Applying a high voltage (above 10 V) between the source and gate-drain terminals creates a high electric field and causes avalanche injection to occur. Electrons acquire sufficient energy to become "hot" and traverse through the first oxide insulator, so that they get trapped on the floating gate. This phenomenon can occur with oxides as thick as 100 nm, which makes it relatively easy to fabricate the device. In reference to the programming mechanism, the floating-gate transistor is often called a *floating-gate avalanche-injection MOS* (or FAMOS) [Frohman74].

The trapping of electrons on the floating gate effectively drops the voltage on that gate. (See Figure 12-19a.) This process is self-limiting—the negative charge accumulated on the floating gate reduces the electrical field over the oxide so that ultimately it becomes incapable of accelerating any more hot electrons. Removing the voltage leaves the induced negative charge in place, and results in a negative voltage on the intermediate gate (Figure 12-19b). From a device point of view, this translates into an effective increase in threshold voltage. To turn on the device, a higher voltage is needed to overcome the effect of the induced negative charge (Figure 12-19c). Typically, the resulting threshold voltage is around 7 V; thus, a 5-V



(a) Avalanche injection (b) Removing programming voltage leaves charge trapped (c) Programming results in higher V_T

Figure 12-19 Programming the floating-gate transistor.

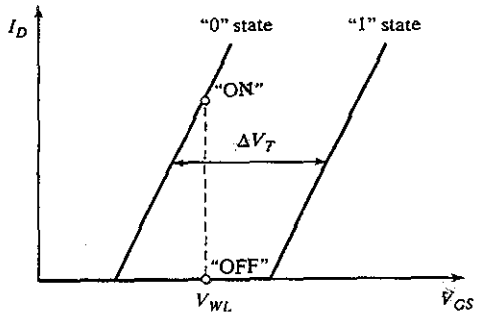


Figure 12-20 I–V curve shift caused by hot-electron programming. Applying a word-line voltage V_{WL} results in either current to flow ("0" state), or not ("1" state).

gate-to-source voltage is not sufficient to turn on the transistor, and the device is effectively disabled. The charge injected onto the floating gate effectively shifts the I–V curves of the transistor, as shown in Figure 12-20. Curve A is for the "0" state, while curve B stands for the transistor with the V_T shift, representing the "1" state. It can be established that the value of the V_T shift is expressed as

$$\Delta V_T = (-\Delta Q_{FG})/C_{FC} \tag{12.1}$$

with C_{FC} the capacitance between external gate contact and floating gate, and ΔQ_{FG} the charge injected onto the floating gate.

Since the floating gate is surrounded by SiO_2, which is an excellent insulator, the trapped charge can be stored for many years, even when the supply voltage is removed, creating a nonvolatile storage mechanism. One of the major concerns of the floating-gate approach is the need for high programming voltages. By tailoring the impurity profiles, technologists have been able to reduce the required voltage from the original 25 V to approximately 12.5 V in today's memories.

Virtually all nonvolatile memories are currently based on the floating-gate approach. Different classes can be identified, based on the erasure mechanism.

Erasable-Programmable Read-Only Memory (EPROM)

An EPROM is erased by shining ultraviolet light on the cells through a transparent window in the package. The UV radiation renders the oxide slightly conductive by the direct generation of electron-hole pairs in the material. The erasure process is slow and can take from seconds to several minutes, depending on the intensity of the UV source. Programming takes several (5–10) μs/word. Another problem with this approach is the *limited endurance*—the number of erase/program cycles is generally limited to a maximum of one thousand, mainly as a result of the UV erasing procedure. Reliability is also an issue. The device thresholds might vary with repeated programming cycles. Most EPROM memories therefore contain on-chip circuitry to control the value of the thresholds to within a specified range during programming. Finally, the injection

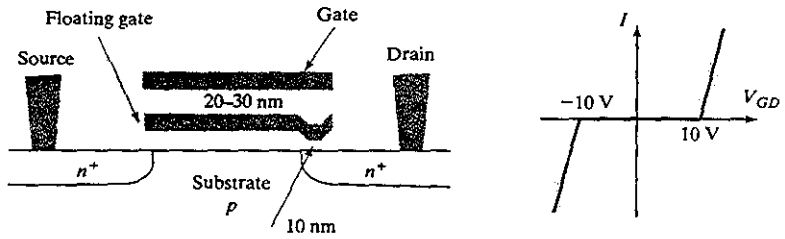
always entails a large channel current, as high as 0.5 mA at a control gate voltage of 12.5 V. This causes high power dissipation during programming.

On the other hand, the EPROM cell is extremely simple and dense, making it possible to fabricate large memories at a low cost. EPROMs were therefore attractive in applications that do not require regular reprogramming. Due to the cost and reliability issues, EPROMs have fallen out of favor and have been replaced by Flash memories.

Electrically Erasable Programmable Read-Only Memory (EEPROM or E^{2}PROM)

The major disadvantage of the EPROM approach is that the erasure procedure has to occur "off system." This means the memory must be removed from the board and placed in an EPROM programmer for programming. The EEPROM approach avoids this labor-intensive and annoying procedure by using another mechanism to inject or remove charges from a floating gate—namely, *tunneling*. A modified floating-gate device called the FLOTOX (floating-gate tunneling oxide) transistor is used as a programmable device that supports an electrical-erasure procedure [Johnson80]. A cross section of the FLOTOX structure is shown in Figure 12-21a. It resembles the FAMOS device, except that a portion of the dielectric separating the floating gate from the channel and drain is reduced in thickness to about 10 nm or less. When a voltage of approximately 10 V (equivalent to an electrical field of around 10^9 V/m) is applied over the thin insulator, electrons travel to and from the floating gate through a mechanism called *Fowler–Nordheim tunneling* [Snow67]. The I–V characteristic of the tunneling junction is plotted in Figure 12-21b.

The main advantage of this programming approach is that it is reversible; that is, erasing is simply achieved by reversing the voltage applied during the writing process. Injecting electrons onto the floating gate raises the threshold, while the reverse operation lowers the V_T. This bidirectionality, however, introduces a threshold-control problem: Removing too much charge from the floating gate results in a depletion device that cannot be turned off by the standard word-line signals. Notice that the resulting threshold voltage depends on the initial charge on the gate, as well as the applied programming voltages. It also is a strong function of the oxide thickness, which is subject to nonnegligible variations over the die. To remedy this problem, an extra transistor connected in series with the floating-gate transistor is added to the EEPROM cell to remedy this problem. This transistor acts as the access device during the read operation, while



(a) FLOTOX transistor (b) Fowler–Nordheim I–V characteristic

Figure 12-21 FLOTOX transistor, programmable by using Fowler–Nordheim tunneling.

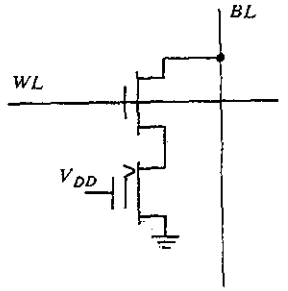


Figure 12-22 EEPROM cell as configured during a read operation. When programmed, the threshold of the FLOTOX device is higher than V_{DD}, effectively disabling it. If not, it acts as a closed switch.

the FLOTOX transistor performs the storage function. (See Figure 12-22.) This is in contrast to the EPROM cell, where the FAMOS transistor acts as both the programming and access device.

The EEPROM cell with its two transistors is larger than its EPROM counterpart. This area penalty is further aggravated by the fact that the FLOTOX device is intrinsically larger than the FAMOS transistor due to the extra area of the tunneling oxide. Additionally, the fabrication of the very thin oxide is a challenging and costly manufacturing step. EEPROM components thus pack less bits at a higher cost than EPROMs. On the positive side of the balance, EEPROMs offer a higher versatility. They also tend to last longer, as they can support up to 10^5 erase/write cycles.[4] Repeated programming causes a drift in the threshold voltages due to permanently trapped charges in the SiO_2. This finally leads to malfunction or the inability to reprogram the device.

Flash Electrically Erasable Programmable Read-Only Memory (Flash)

The concept of Flash EEPROMs was introduced in 1984 and has rapidly evolved into the most popular nonvolatile memory architecture. It combines the density of the EPROM with the versatility of the EEPROM structures, with cost and functionality ranging somewhere between the two.

Technically, the Flash EEPROM is a combination of the EPROM and EEPROM approaches. Most Flash EEPROM devices use the avalanche hot-electron-injection approach to program the devices. Erasure is performed using Fowler–Nordheim tunneling, as for EEPROM cells. The main difference is that erasure is performed in bulk for the complete chip, or for a subsection of the memory. While this represents a reduction in flexibility, it has the advantage that the extra access transistor of the EEPROM cell can be eliminated. Erasing the complete memory core at once makes it possible to carefully monitor of the device characteristics during erasure, guaranteeing that the unprogrammed transistor acts as an enhancement device. The monitoring control hardware on the memory chip regularly checks the value of the threshold during erasure.

[4] Thin oxides are not the only way to realize electron tunneling. Other approaches include the use of textured surfaces to locally enhance the surface field [Masuoka91].

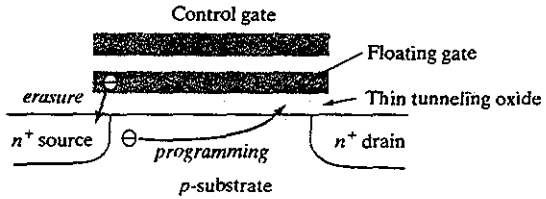


Figure 12-23 ETOX device as used in Flash EEPROM memories.

and adjusts the erasure time dynamically. This approach is only practical when erasing large chunks of memory at a time; hence the flash concept. The simpler cell structure results in a substantial reduction in cell size and an increased integration density.

For instance, Figure 12-23 shows the ETOX Flash cell introduced by Intel [Pashley89]. This is only one of the many existing alternatives. It resembles a FAMOS gate, except that a very thin tunneling gate oxide is utilized (10 nm). Different areas of the gate oxide are used for programming and erasure. Programming is performed by applying a high voltage (12 V) on gate and drain terminals for a grounded source, while erasure occurs with the gate grounded and the source at 12 V.

Figure 12-23 illustrates how this cell can be incorporated into a NOR ROM structure. A programming cycle starts with an *erase operation* (a): A 0 V gate voltage is applied, combined with a high voltage (12 V) at the source. Electrons, if any, at the floating gate are ejected to the source by tunneling. All cells are erased simultaneously. The different initial values of the cell threshold voltages, as well as variations in the oxide thickness, may cause variations in the threshold voltage at the end of the erase operation. This is remedied in two ways: (1) Before applying the erase pulse, all the cells in the array are programmed so that all the thresholds start at approximately the same value; (2) After that, an erase pulse of controlled width is applied. Subsequently, the whole array is read to ensure to check whether or not the cells have been erased. If not, another erase pulse is applied, followed by a read cycle. The algorithm is applied until all cells have threshold voltages that are below the required level. Typical erasing times are between 100 ms to 1 s. For the *write (programming) operation* (b), a high-voltage pulse is applied to the gate of the selected device. If a "1" is applied to the drain at that time, hot electrons are generated and injected onto the floating gate, raising the threshold (effectively turning it into an always-off device). If not, the floating gate remains in the previous state of no electrons, corresponding to a "0" state. To obtain the necessary threshold shift of 3 to 3.5 V, a pulse with typical values in the 1–10-μs range must be applied. The *read operation* (c) proceeds as in any NOR ROM structure. To select a cell, its word line is raised to 5 V, causing a conditional discharge of the bit line.

The NOR architecture leads to fast random read access times. At the same time, erasure and programming times are slow due to the need for precise control of the thresholds. These properties make this style of Flash memory attractive for applications such as program-code storage. Other applications, such as video or audio file storage, do not need fast random access, but are better served by large storage density (reducing the memory cost), fast erasure and

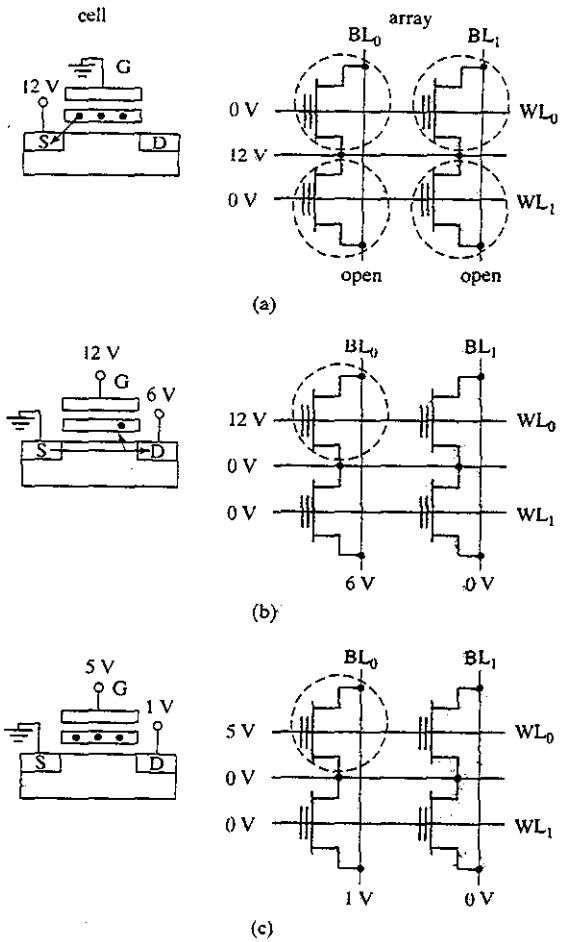


Figure 12-24 Basic operations in a NOR Flash memory [from Itoh01]. (a) Erase; (b) write; (c) read.

programming, and fast serial access. These requirements are more readily provided by the NAND ROM architecture described earlier. A range of Flash memory manufacturers have thus opted for this topology. The basic module consists of 8 to 16 floating-gate transistors connected in series, as shown in Figure 12-25. This chain is connected to the bit line and the source line (GND) with the aid of two select transistors. By eliminating all contacts between word lines, the resulting cell size is approximately 40% smaller than the NOR cell. The memory shown uses Fowler–Nordheim tunneling for both programming and erasure. During erasure, all cells in the

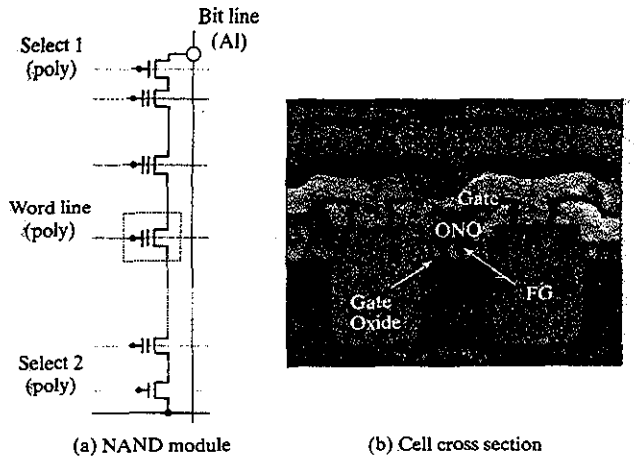


Figure 12-25 NAND-based Flash Module [Nakamura02]. The second gate uses a high-dielectric material (ONO or Oxide–Nitride–Oxide) to increase the C_{FG} capacitance, and hence the impact of charge injection on *VT*.

module are programmed to become depletion devices (this is, transistors with a negative threshold). This is accomplished by applying 20 V to the bit and source lines, and 0 V to the word lines. During programming, the selection transistors are set such that the bit line is connected and the source line isolated. To write a "1", the bit line is grounded and a high voltage (20 V) is applied to the word line of interest. Electrons tunnel into the floating gate, causing the threshold to increase. For a "0", the cell threshold is left unperturbed (by keeping the bit line high). The read operation proceeds just like in the NAND ROM, with both select transistors enabled. The use of tunneling as the primary mechanism reduces the current requirements, and allows for the parallel programming of many modules while keeping power consumption under control. The programming time per byte is as fast as 200–400 ns. By processing a complete module at a time, fast and reliable erasure can be obtained, leading to fast erasure of around 100 ms for a complete chip.

Many other architectures for flash memories have been defined, trading off read performance, erasure and programming speed, and density. All are based on the principles defined earlier. For a more detailed description, we refer the reader to the excellent overview in [Itoh01]. One issue deserves special attention. All nonvolatile memories need high voltages (> 10 V) for programming and erasure. Some of these supplies can be generated on chip using charge pumps, especially if the current requirements are low, such as is the case for Fowler–Nordheim tunneling. Hot-electron injection, on the other hand, draws a large current (> 0.5 mA), and an external supply is a necessity, adding to the cost and the complexity of the system. The search is definitely on for nonvolatile memories that do not need high voltages and external supplies.

New Trends in Nonvolatile Memories

All nonvolatile memories discussed so far are based on the floating-gate transistor concept. Over the years, a number of alternative approaches have been proposed. While most of those have languished and not made it to the commercial market, some novel cell structures are currently catching on, and may make a big impact in the foreseeable future. The two most prominent among those are the *FRAM* (or Ferroelectric RAM) and the *MRAM* (Magnetoresistive RAM). However, calling the floating-gate device at the end of its cycle is premature. Innovative approaches such as multilevel cells are creating opportunities for ever denser nonvolatile storage. We will briefly discuss the basic concepts of each of these exciting developments.

Multilevel Nonvolatile Memories All memory cells considered so far are bistable, which means that they store either a 0 or a 1. The memory density could be increased substantially if more than two states could be stored in a single cell. The problem with this approach is that it requires a substantial improvement in the signal-to-noise ratio of the memory cell. By storing k bits in a memory cell, the available signal is reduced by $1/(2^k - 1)$. Fortunately, the flash cell has the advantage of providing internal gain. Also, the capability of V_T adjustment for each cell can help to overcome the signal-to-noise issue. Commercial memories, storing two bits per cell, have been demonstrated [Baur95]. Yet, with the continuing reduction of the supply voltages, it is highly unlikely that multilevel memories are destined for a lucrative future.

FRAM Ferroelectric RAM has been "almost there" for quite some time, mainly based on optimistic expectations. After a lot of false starts, it seems like FRAM is finally close to fulfilling its promises. In contrast to the "programmable transistor" used in current NVRWMs, the FRAM is based on a "programmable capacitor." The dielectric material used in a ferroelectric capacitor belongs to a class of materials called *Perovskite crystals*. These crystals polarize when an electric field is applied across them. This polarization is maintained when the electrical field is removed. To depolarize the dielectric, an electrical field of the opposite direction has to be applied.

A memory cell built on this concept closely resembles the single-transistor DRAM circuit, to be discussed in a following section. During a write operation, the capacitor cell is polarized one way or the other. To read the contents, an electrical field is applied. Depending upon the state of the capacitor, a current is generated, which can be detected by a sense amplifier.

The advantage of the FRAM is a very high storage density, leading to cell sizes smaller than DRAMs, while offering nonvolatility at the same time. The number of read/write cycles for FRAMs is also orders of magnitude higher than EEPROMs and Flash memories. Its low power consumption makes it very attractive for applications such as smart cards and RF tags. The main challenge still is to generate large memories reliably using this technology.

MRAM *Magnetoresistive random access memory* is a method of storing data bits using magnetic charges instead of the electrical charges used by DRAM. A metal is called *magnetoresistive* if it shows a slight change in electrical resistance when placed in a magnetic field. The concept is quite similar to the magnetic core memories used in the mainframe computers in the

early days of electronics. The main differences are the scale (millions times smaller) and the materials used. Development of MRAM has followed two scientific schools: (1) spin electronics, the science behind giant magnetoresistive heads used in disk drives and (2) tunneling magnetic resistance, or TMR, which is expected to be the basis of future MRAM. Researchers at IBM demonstrated a 1 Kbit MRAM memory in 2000. Each cell consists of a magnetic tunnel junction and a FET, for a total area per cell of 3 μm^2 in a 0.25-μm CMOS process [Scheuerlein00]. Read and write times in the range of 10 ns were obtained. While it is obviously too early to predict the future of this particular device, it seems apparent that new materials and approaches are bound to change the field of nonvolatile storage as well as semiconductor memories in general.

Nonvolatile Read–Write Memories—Summary

Some numbers are useful to put the different nonvolatile technologies in perspective. Table 12-1 summarizes the current essential data for nonvolatile memories. The table confirms that the flexibility of the EEPROM structure comes at the expense of density and performance. EPROMs and Flash EEPROM devices are comparable in both density and speed. The versatility of the latter explains its explosive growth in a short time span.

A large number of design considerations raised in the section on read-only memories are valid for the NVRWMs as well. In addition, the (E)EPROM structures must cope with the extra complexity of the programming and erasure circuitry. Remember that all the proposed structures require the availability of high-voltage signals (12–20 V) on word and bit lines during the programming, while standard 3- or 5-V signals are used on the same wires during the read mode. The generation and distribution of those signals requires some interesting circuit design, which, unfortunately, is beyond the scope of this text.

Table 12-1 Comparison between nonvolatile memories ([Itoh01]).
V_{DD} = 3.3 or 5 V; V_{PP} = 12 or 12.5 V.

	Cell—Nr. of Transistors	Cell Area (ratio wrt EPROM)	Mechanism: Erase	Mechanism: Write	External Power Supply: Write	External Power Supply: Read	Program/Erase Cycles
MASK ROM	1 T (NAND)	0.35–5	—	—	—	V_{DD}	0
EPROM	1 T	1	UV Exposure	Hot electrons	V_{PP}	V_{DD}	~100
EEPROM	2 T	3–5	FN Tunneling	FN Tunneling	V_{PP} (int)	V_{DD}	10^4–10^5
Flash Memory	1 T	1–2	FN Tunneling	Hot electrons	V_{PP}	V_{DD}	10^4–10^5
			FN Tunneling	FN Tunneling	V_{PP} (int)	V_{DD}	10^4–10^5

12.2.3 Read–Write Memories (RAM)

Providing a memory cell with roughly equal read and write performance requires a more complex cell structure. While the contents of the ROM and NVRWM memories are ingrained in the cell topology or programmed into the device characteristics, storage in RAM memories is based on either positive feedback or capacitive charge, similar to the ideas introduced in Chapter 6. These circuits would be perfectly suitable as R/W memory cells, but they tend to consume too much area. In this section, we introduce a number of simplifications that trade off area for either performance or electrical reliability. They are labeled as either SRAMs or DRAMs, depending on the storage concept used.

Static Random-Access Memory (SRAM)

The generic SRAM cell is introduced in Figure 12-26. It turns out to be quite similar to the static SR latch, shown in Figure 7-21. It requires six transistors per bit. Access to the cell is enabled by the word line, which replaces the clock and controls the two pass transistors M_5 and M_6, shared between the read and write operation. In contrast to the ROM cells, two bit lines transferring both the stored signal and its inverse are required. Although providing both polarities is not a necessity, doing so improves the noise margins during both read and write operations, as will become apparent in the subsequent analysis.

Problem 12.5 CMOS SRAM Cell

Does the SRAM cell presented in Figure 12-26 consume standby power? Explain. Draw an equivalent pseudo-NMOS implementation. How about the standby power in that case?

Operation of SRAM Cell The SRAM cell should be sized as small as possible to achieve high memory densities. Reliable operation of the cell, however, imposes some sizing constraints.

To understand the operation of the memory cell, let us consider the read and write operations in sequence. While doing so, we also derive the transistor-sizing constraints.

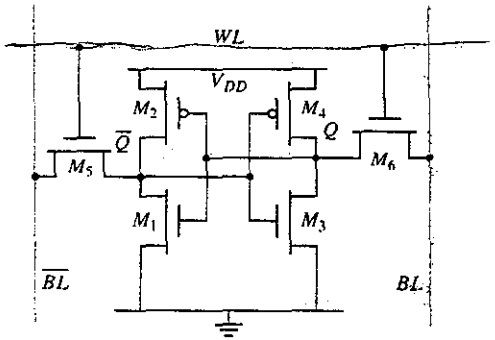


Figure 12-26 Six-transistor CMOS SRAM cell.

Example 12.8 CMOS SRAM—Read Operation

Assume that a 1 is stored at Q. We further assume that both bit lines are precharged to 2.5 V before the read operation is initiated. The read cycle is started by asserting the word line, enabling both pass transistors M_5 and M_6 after the initial word-line delay. During a correct read operation, the values stored in Q and $\overline{Q}$ are transferred to the bit lines by leaving BL at its precharge value and by discharging $\overline{BL}$ through M_1–M_5. A careful sizing of the transistors is necessary to avoid accidentally writing a 1 into the cell. This type of malfunction is frequently called a *read upset*.

This is illustrated in Figure 12-27. Consider the $\overline{BL}$ side of the cell. The bit line capacitance for larger memories is in the pF range. Consequently, the value of $\overline{BL}$ stays at the precharged value V_{DD} upon enabling of the read operation ($WL \rightarrow 1$). This series combination of two NMOS transistors pulls down the $\overline{BL}$ towards ground. For a small-sized cell, we would like to have these transistor sized as close to minimum as possible, which would result in a very slow discharge of the large bit line capacitance. As the difference between BL and $\overline{BL}$ builds up, the sense amplifier is activated to accelerate the reading process.

Initially, upon the rise of the WL, the intermediate node between these two NMOS transistors, $\overline{Q}$, is pulled up toward the precharge value of $\overline{BL}$. This voltage rise of $\overline{Q}$ must stay low enough not to cause a substantial current through the M_3–M_4 inverter, which in the worst case could flip the cell. It is necessary to keep the resistance of transistor M_5 larger than that of M_1 to prevent this from happening.

The boundary constraints on the device sizes can be derived by solving the current equation at the maximum allowed value of the voltage ripple ΔV. We ignore the body effect on transistor M_5 for simplicity and write

$$k_{n,M5}\left((V_{DD}-\Delta V-V_{Tn})V_{DSATn}-\frac{V_{DSATn}^2}{2}\right)=k_{n,M1}\left((V_{DD}-V_{Tn})\Delta V-\frac{\Delta V^2}{2}\right) \qquad (12.2)$$

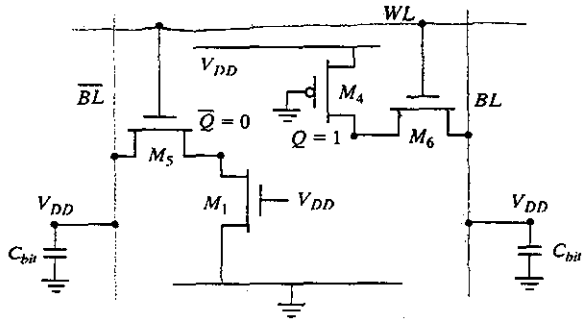


Figure 12-27 Simplified model of CMOS SRAM cell during read ($Q = 1$, $V_{precharge} = V_{DD}$).

which simplifies to

$$\Delta V=\frac{V_{DSATn}+CR(V_{DD}-V_{Tn})-\sqrt{V_{DSATn}^2(1+CR)+CR^2(V_{DD}-V_{Tn})^2}}{CR} \qquad (12.3)$$

where CR is called the cell ratio and is defined as

$$CR=\frac{W_1/L_1}{W_5/L_5} \qquad (12.4)$$

The value of voltage rise ΔV as a function of CR for our 0.25-μm technology is plotted in Figure 12-28. To keep the node voltage from rising above the transistor threshold (of about 0.4 V), the cell ratio must be greater than 1.2. For large memory arrays, it is desirable to keep the cell size minimal while maintaining read stability. If the transistor M_1 is minimum sized, the access pass transistor M_5 has to be made weaker by increasing its length. This is undesirable, because it adds to the load of the bit line. A preferred solution is to minimize the size of the pass transistor, and increase the width of the NMOS pull-down M_5 to meet the stability constraint. This slightly increases the minimum size of the cell. The designer must perform careful simulations to guarantee cell stability across all process corners [Preston01].

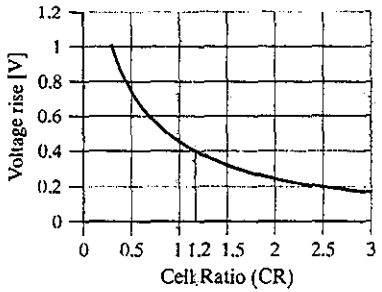


Figure 12-28 Voltage rise inside the cell upon read versus cell ratio (ratio of M_1/M_5). The voltage inside the cell does not rise above the threshold for $CR > 1.2$

The preceding analysis presents the worst case. The second bit line BL clamps Q to V_{DD}, which makes the inadvertent toggling of the cross-coupled inverter pair difficult. This demonstrates one of the major advantages of the dual bit line architecture.

Beyond adjusting the size of the cell transistors, the erroneous toggling can be prevented by precharging the bit lines to another value, such as $V_{DD}/2$. This effectively makes it impossible for Q to reach the switching threshold of the connecting inverter. Precharging to the midpoint of the voltage range has some performance benefits as well, since it limits the voltage swing on the bit lines.

Example 12.9 CMOS SRAM Write Operation

In this example, we derive the device constraints necessary to ensure a correct write operation. Assume that a 1 is stored in the cell (or $Q = 1$). A 0 is written in the cell by setting $\overline{BL}$ to 1 and BL to 0, which is identical to applying a reset pulse to an SR latch. This causes the flip- flop to change state if the devices are sized properly.

During the initiation of a write, the schematic of the SRAM cell can be simplified to the model of Figure 12-29. It is reasonable to assume that the gates of transistors M_1 and M_4 stay at V_{DD} and GND, respectively, as long as the switching has not commenced. While this condition is violated once the flip-flop starts toggling, the simplified model is more than accurate for hand-analysis purposes.

Note that $\overline{Q}$ side of the cell cannot be pulled high enough to ensure the writing of 1. The sizing constraint, imposed by the read stability, ensures that this voltage is kept below 0.4 V. Therefore, the new value of the cell has to be written through transistor M_6.

A reliable writing of the cell is ensured if we can pull node Q low enough—this is, below the threshold value of the transistor M_1.[5] The conditions for this to occur can be derived by writing out the *dc* current equations at the desired threshold point, as follows:

$$k_{n,M6}\left((V_{DD}-V_{Tn})V_Q-\frac{V_Q^2}{2}\right)=k_{p,M4}\left((V_{DD}-|V_{Tp}|)V_{DSATp}-\frac{V_{DSATp}^2}{2}\right) \tag{12.5}$$

Solving for V_Q leads to

$$V_Q = V_{DD}-V_{Tn}-\sqrt{(V_{DD}-V_{Tn})^2-2\frac{\mu_p}{\mu_n}PR\left((V_{DD}-|V_{Tp}|)V_{DSATp}-\frac{V_{DSATp}^2}{2}\right)}, \tag{12.6}$$

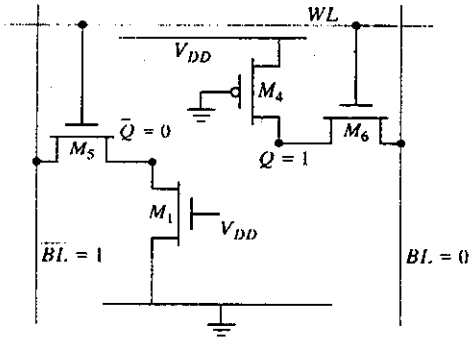


Figure 12-29 Simplified model of CMOS SRAM cell during write ($Q = 1$).

[5] In principle, it is sufficient to pull Q below the switching threshold of the inverter formed by M_1 and M_2 to ensure the initiation of the switching action. For noise margin purposes, it is safer to require that Q is pulled below the threshold of M_1.

where the *pull-up ratio of the cell, PR*, is defined as the size ratio between the PMOS pull-up and the NMOS pass transistor:

$$PR = \frac{W_4/L_4}{W_6/L_6}. \tag{12.7}$$

The dependence of V_Q on PR for a 0.25-μm process is plotted in Figure 12-30. The lower PR, the lower the value of V_Q. If we wish to pull the node below V_{Tn}, the pull-up ratio has to be below 1.8.

This constraint is met, by a large margin, when using a minimum-sized devices for both the PMOS pull-up M_4 and NMOS access transistor M_6. However, a designer must assure that the writeability constraint is met under all process corners. The worst case presents a process with strong PMOS devices, weak NMOS devices, and the memory operated at a higher than nominal supply voltage [Preston01].

Our initial assumption was that the transistors M_1 and M_2 do not participate in the writing process. This is not completely true in practice. As soon as one side of the cell starts switching, the other side eventually follows, engaging the positive feedback.

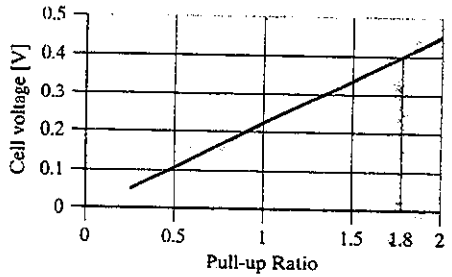


Figure 12-30 Voltage written into the cell versus pull-up ratio (size ratio between the PMOS pull-up and access transistor). *PR* should be less than 1.8.

Performance of SRAM Cell When analyzing the transient behavior of the SRAM cell, one realizes that the read operation is the critical one. It requires the (dis)charging of the large bit linebit line capacitance through the stack of two small transistors of the selected cell. For instance, $C_{\overline{BL}}$ has to be discharged through the series combination of M_5 and M_1 in the example of Figure 12-27. The write time is dominated by the propagation delay of the cross-coupled inverter pair, as the drivers that force BL and $\overline{BL}$ to the desired values can be large. To accelerate the read time, SRAMs use sense amplifiers. As the difference in voltage between BL and $\overline{BL}$ builds up, the sense amplifier is activated, and it quickly discharges one of the bit lines.

Improved MOS SRAM Cells The six-transistor SRAM cell, while simple and reliable, consumes a substantial area. Besides the devices, it requires the signal routing and connections to two bit lines, a word line, and both supply rails. Placing the two PMOS transistors in the N-well

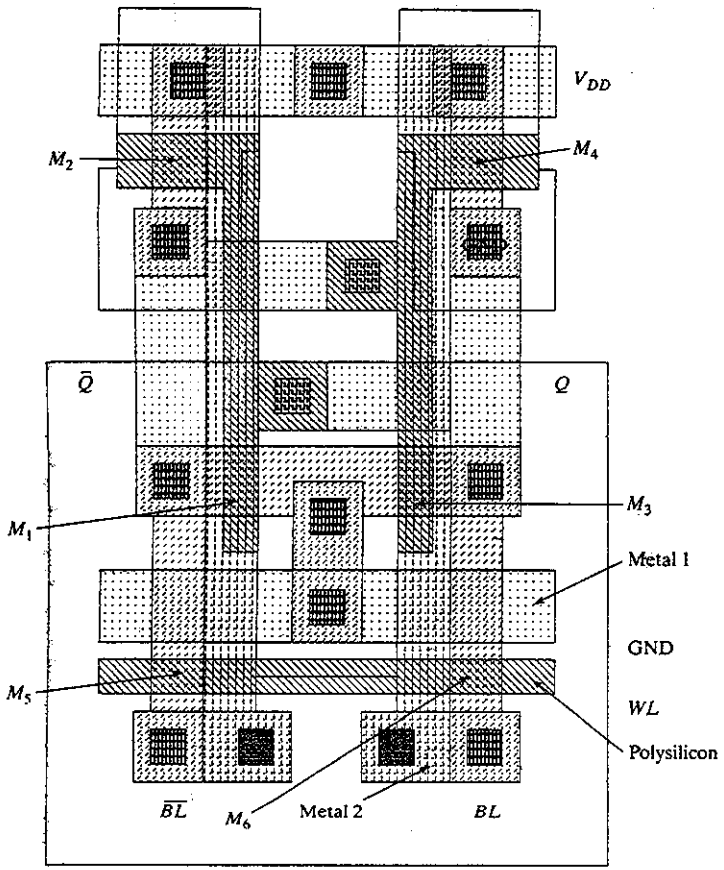


Figure 12-31 Layout of six-transistor CMOS SRAM memory cell.

significantly contributes to the area. Figure 12-31 shows a possible layout for such a cell. Its dimensions are dominated by the wiring and interlayer contacts (11.5 of them—the top and bottom ones only count for one-half, since they are shared with the neighboring cells), as well as the minimum spacing requirements for the well.

Designers of large memory arrays have therefore proposed other cell structures that are not only based on revised transistor topologies, but also on the presence of special devices and a more complex technology.

Consider the cell schematic of Figure 12-32, called the *resistive load* SRAM cell (also known as the four-transistor SRAM cell). The special feature of this cell is that the cross-coupled CMOS inverter pair is replaced by a pair of resistive-load NMOS inverters. The PMOS

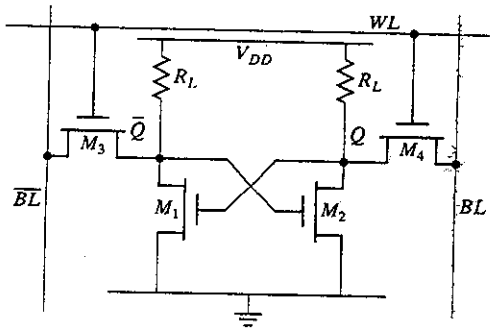


Figure 12-32 Resistive-load SRAM cell.

transistors are replaced by resistors, and the wiring is simplified. This reduces the SRAM cell size by approximately one-third, as illustrated in Table 12-2 for the example of a 1-Mbit SRAM.

From the SRAM cell writeability analysis, we found that there exists only a constraint on lower limit of the pull-up resistance. Since the bit lines are externally precharged, the cell is not involved in the pull-up process, and there is no penalty for having very large pull-ups, as opposed to conventional logic. Therefore, an advanced SRAM process technology includes special resistors that are made as large as possible to minimize static power consumption.

Keeping the static power dissipation per cell as low as possible is a prime design priority in SRAM cells. Consider a 1-Mbit SRAM memory operating at 2.5V and using a 10 kΩ resistor as the inverter load. With each cell sinking 0.25 mA in static current, a total standby dissipation of 250 W can be recorded! Therefore, the only obvious choice is to make the load resistance as large as possible. A very large, yet compact, resistor can be manufactured by using an undoped polysilicon, which has a sheet resistance of several TΩ/sq (Tera = 10^{12}!). The only additional constraint on the pull-up resistors is to maintain the state of the cell, that is to compensate for the leakage currents that typically range around 10^{-15} A/cell [Takada91]. The low leakage in recent

Table 12-2 Comparison of CMOS SRAM cells used in 1-Mbit memory (from [Takada91])

	Complementary CMOS	Resistive Load	TFT Cell
Number of transistors	6	4	4 (+2 TFT)
Cell size	58.2 μm² (0.7-μm rule)	40.8 μm² (0.7-μm rule)	41.1 μm² (0.8-μm rule)
Standby current (per cell)	10^{-15} A	10^{-12} A	10^{-13} A

technologies is achieved by using the devices with higher thresholds. The resistor current should be at least two orders of magnitude larger to accomplish that goal, or $I_{load} > 10^{-13}$ A. This puts an upper limit on the resistor value.

The realization that the pull-up devices are only needed for charge-loss compensation has resulted in a revised version of the six-transistor memory cell of Figure 12-26. Instead of using traditional, expensive PMOS devices, the pull-up transistors are realized as parasitic devices deposited on top of the cell structure using a thin-film technology. These PMOS *thin-film transistors* (TFTs) have inferior properties with respect to normal devices and are characterized by a current of approximately 10^{-8} A and 10^{-13} in the *ON* and *OFF* modes respectively for a 5 V gate-source voltage [Sasaki90, Ootani90]. The complimentary nature of the cell results in an increased cell reliability with less sensitivity to leakage and soft errors, yet at a lower standby current compared to the resistive load cell.

Note that a high-resistivity polysilicon and thin-film transistors are additional features that are not available in a standard logic process. Therefore, embedded SRAM cells, such as those used in microprocessor caches, stick to the conventional 6T cell of Figure 12-26.

Dynamic Random-Access Memory (DRAM)

While discussing the resistive-load SRAM cell, we noted that the only function of the load resistors is to replenish the charge lost by leakage. One option is to eliminate these loads completely and compensate for the charge loss by periodically rewriting the cell contents. This *refresh* operation, which consists of a read of the cell contents followed by a write operation, should occur often enough that the contents of the memory cells are never corrupted by the leakage. Typically, refresh should occur every 1 to 4 ms. For larger memories, the reduction in cell complexity more than compensates for the added system complexity imposed by the refresh requirement. These memories are called *dynamic*, since the underlying concept of these cells is based on charge storage on a capacitor.

Three-Transistor Dynamic Memory Cell The first kind of dynamic cell is obtained by eliminating the load resistors in the schematic of Figure 12-33. The four-transistor cell can be further simplified by observing that the cell stores both the data value and its complement; hence, it contains redundancy. Eliminating one more device (e.g., M_3) removes this redundancy and results in the three-transistor (3T) cell of Figure 12-33 [Regitz70]. This cell formed the core of the first popular MOS semiconductor memories such as the first 1-Kbit memory from Intel [Hoff70]. While replaced by more area-efficient cells in the very large memories of today, it is still the cell of choice in many memories embedded in application-specific integrated circuits. This can be attributed to its relative simplicity in both design and operation.

The cell is written to by placing the appropriate data value on *BL1* and asserting the *write-word line* (*WWL*). The data is retained as charge on capacitance C_S once *WWL* is lowered. When reading the cell, the *read-word line* (*RWL*) is raised. The storage transistor M_2 is either on or off depending upon the stored value. The bit line *BL2* is either clamped to V_{DD} with the aid of a load device, for example, a grounded PMOS or saturated NMOS transistor, or is precharged to either

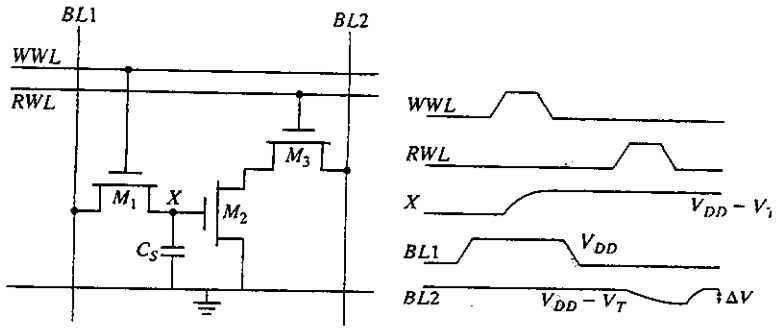


Figure 12-33 Three-transistor dynamic memory cell and the signal waveforms during read and write.

V_{DD} or $V_{DD} - V_T$. The former approach necessitates careful transistor sizing and causes static power consumption. Therefore, the precharged approach is generally preferable. The series connection of M_2 and M_3 pulls *BL2* low when a 1 is stored. *BL2* remains high in the opposite case. Notice that the cell is inverting; that is, the inverse value of the stored signal is sensed on the bit line. The most common approach to refreshing the cell is to read the stored data, put its inverse on *BL1*, and assert *WWL* in consecutive order.

The cell complexity is substantially reduced with respect to the static cell. This is illustrated by the example layout of Figure 12-34. The total area of the cell is 576 λ^2, compared to the 1092 λ^2 of the SRAM cell of Figure 12-31. These numbers do not take into account the potential area reduction obtained by sharing with neighboring cells. The area reduction is mainly due to the elimination of contacts and devices.

Further simplifications in the cell structure are possible at the expense of a more complex circuit operation. For instance, bit lines *BL1* and *BL2* can be merged into a single wire. The read and write cycles can proceed as before. The read–sense–write refresh cycle must be altered considerably, since the data value read from the cell is the complement of the stored value. This requires the bit line to be driven to both values in a single cycle. Another option is to merge the *RWL* and the *WWL* lines. Once again, this does not significantly change the cell operation. A read operation is automatically accompanied by a refresh of the cell contents. A careful control of the word-line voltage is necessary to prevent a writing of the cell before the actual value is read during refresh.

Finally, the following interesting properties of the 3T cell are worth mentioning:

1. In contrast to the SRAM cell, no constraints exist on the *device ratios*. This is a common property of dynamic circuits. The choice of device sizes is solely based on performance and reliability considerations. Observe that this statement is not valid when a static bit line load approach is employed.

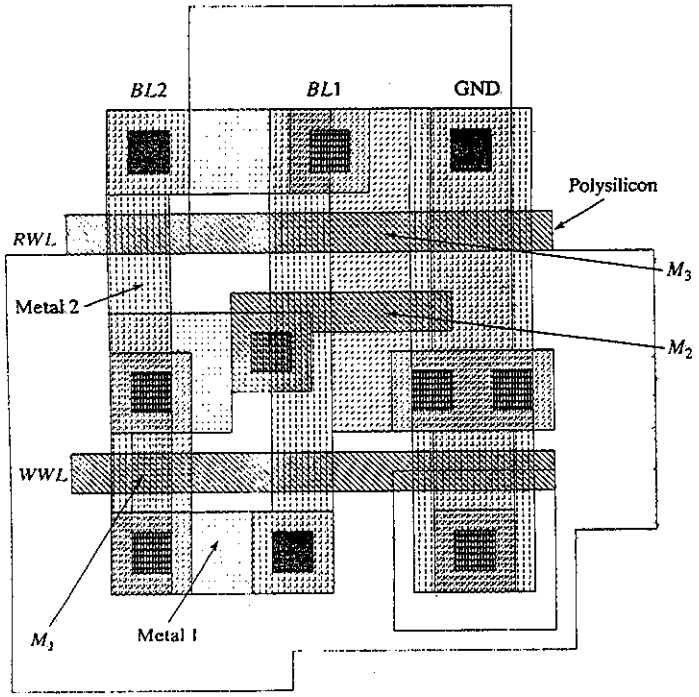


Figure 12-34 Example layout of three-transistor dynamic memory cell.

2. In contrast to other DRAM cells, reading the 3T cell contents is *nondestructive;* that is, the data value stored in the cell is not affected by a read.
3. No special process steps are needed. The storage capacitance is nothing more than the gate capacitance of the readout device. This is in contrast with the other DRAM cells, discussed next, and makes the 3T cell attractive for embedded memory applications.
4. The value stored on the storage node X when writing a 1 equals $V_{WWL} - V_{Tn}$. This threshold loss reduces the current flowing through M_2 during a read operation and increases the read access time. To prevent this, some designs *bootstrap* the word-line voltage, or in other words, raise V_{WWL} to a value higher than V_{DD}.

One-Transistor Dynamic Memory Cell Another dramatic reduction in cell complexity can be obtained by a further sacrifice in some of the cell properties. The resulting structure, called the one-transistor DRAM cell (1T), is undoubtedly the most pervasive dynamic DRAM cell in commercial memory design.[6] A schematic is shown in Figure 12-35 [Dennard68]. Its basic operational concepts are extremely simple. During a write cycle, the data value is placed on the bit line *BL*, and the word line *WL* is raised. Depending on the data value, the cell capacitance is either charged or discharged. Before a read operation is performed, the bit line is precharged to a voltage V_{PRE}. Upon asserting the word line, a charge redistribution takes place between the bit line and storage capacitance. This results in a voltage change on the bit line, the direction of which determines the value of the data stored. The magnitude of the swing is given by the expression

[6] A DRAM cell containing only two transistors can also be conceived. It offers no substantial advantages over either the 3T or 1T cell and is, therefore, only rarely used.

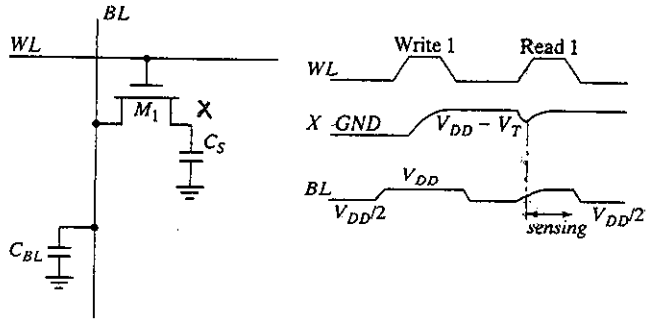


Figure 12-35 One-transistor dynamic RAM cell and the corresponding signal waveforms during read and write.

$$\Delta V = V_{BL} - V_{PRE} = (V_{BIT} - V_{PRE})\frac{C_S}{C_S + C_{BL}} \tag{12.8}$$

where C_{BL} is the bit line capacitance, V_{BL} the potential of the bit line after the charge redistribution, and V_{BIT} the initial voltage over the cell capacitance C_S. As the cell capacitance is normally one or two orders of magnitude smaller than the bit line capacitance, this voltage change is very small, typically around 250 mV for state-of-the-art memories [Itoh90]. The ratio $C_S/(C_S + C_{BL})$ is called the *charge-transfer ratio* and ranges between 1% and 10%.

Amplification of ΔV to the full voltage swing is necessary if functionality is to be achieved. This observation marks a first major difference between the 1T and 3T, as well as other, DRAM cells.

1. A 1T DRAM requires the *presence of a sense amplifier* for each bit line to be functional. This is a result of the charge-redistribution-based readout. The read operation of all cells discussed previously relies on current sinking. A sense amplifier is only needed to speed up the readout, not for functionality considerations. It is also worth noticing that the DRAM memory cells are *single ended* in contrast to the SRAM cells, which present both the data value and its complement on the bit lines. This complicates the design of the sense amplifier, as will be discussed in the section on periphery.

Example 12.10 1T DRAM Readout

Assume a bit line capacitance of 1pF in correspondence to the numbers derived earlier in the chapter, and a bit line precharge voltage of 1.25 V. The voltage over the cell capacitance C_S (of 50 fF) equals 1.9 V and 0 V for a 1 and 0, respectively. This translates into a charge-transfer efficiency of 4.8% and the following voltage swings on the bit line during a read operation:

$$\Delta V(0) = -1.25\ \text{V} \times \frac{50\ \text{fF}}{50\ \text{fF} + 1\ \text{fF}} = -60\ \text{mV}$$

$$\Delta V(1) = 31\ \text{mV}$$

Other important differences are also worth enumerating:

2. The readout of the 1T DRAM cell is *destructive*. This means that the amount of charge stored in the cell is modified during the read operation. After a successful read operation, the original value must be restored. Read and refresh operations are therefore intrinsically intertwined in a 1T DRAM. Typically, the output of the sense amplifier is imposed onto the bit line during the readout. Keeping *WL* high ensures that the cell charge is restored during that period. This is illustrated in Figure 12-36, which plots a typical bit line voltage waveform during readout.
3. Unlike the 3T cell that relies on charge storage on a gate capacitance, the 1T cell requires the presence of an extra capacitance that must be explicitly included in the design. For reliability, the charge-transfer ratio is kept large, with the minimum value of the capacitance ranging around 30 fF. Fitting that large of a capacitance in as small an area as possible is one of the key challenges in DRAM designs. Some of the most popular ways to do so are briefly summarized in a following section.
4. Observe that when writing a 1 into the cell, a threshold voltage is lost, which reduces the available charge. This charge loss can be circumvented by bootstrapping the word lines to a value higher than V_{DD}. This is a common practice in state-of-the-art memory design.

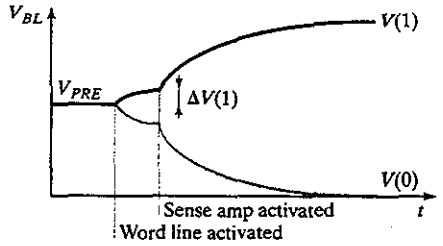


Figure 12-36 Bit line voltage waveform during read operation (for 1 and 0 data values).

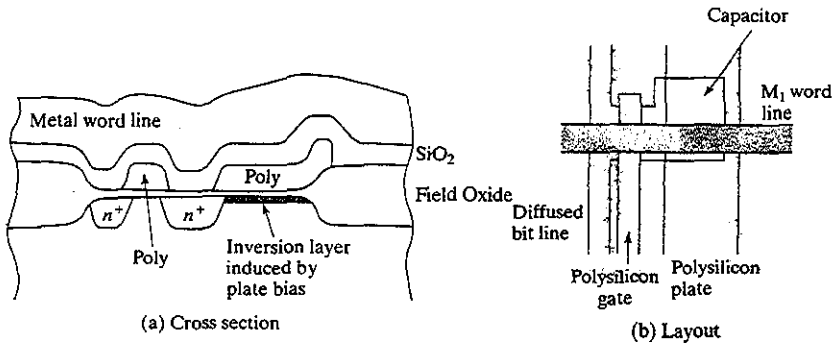
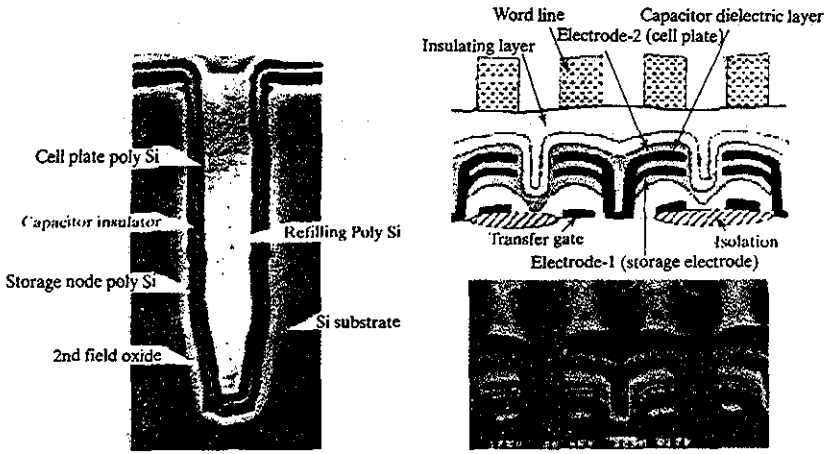


Figure 12-37 1T DRAM cell using a polysilicon-diffusion capacitance as storage node (from [Dillinger88]). The contact between word line and polysilicon gate is accomplished in the neighboring cell.

Figure 12-37 presents a first approach toward designing a 1T DRAM cell. The main advantage of this design is that it can be realized in a generic CMOS technology. The storage node in this cell is composed of the gate capacitance, sandwiched between a polysilicon plate and an inversion layer, induced into the substrate by applying a positive voltage bias on the polysilicon plate. When writing a 0 in the cell, the potential well of the storage node is filled with electrons, and the capacitor is charged. If a 1 is written in the cell (with a high voltage on the bit line) the electrons are removed from the induced inversion layer, and the surface area is depleted. The voltage over the capacitor is reduced. Observe that this represents the inverse of the scenario described before: The capacitor is charged for a 0 and discharged when storing a 1.

Implementing denser cells requires modifications in the manufacturing process. A first change is to add a second polysilicon layer, which serves as the second plate of the capacitor, with the first polysilicon layer forming the other plate. In the quest for ever-denser cells, DRAM technology as used in the 16-Mbit DRAMs and beyond, has focused on three-dimensional structures, where the storage capacitance is either implemented vertically in the substrate or on top of the access transistor [Lu89]. Cross sections of some of the most advanced cells are shown in Figure 12-38. The first cell shows the cross section of a *trench-capacitor* cell. In this structure, a vertical trench of up to 5 μm deep is etched into the substrate. The sidewalls and bottom of the trench are used for the capacitor electrode, which results in a large plate surface that occupies only a small die area. Figure 12-38b shows the cross section of a *stacked-capacitor cell* (STC), where the capacitance is superimposed on top of the access transistor and bit lines. Up to four polysilicon layers are employed to realize "fin-type" capacitors, reducing the effective cell area. Using these approaches, the cell area in a 64-Mbit DRAM ranges between 1.5 and 2.0 μm^2, yielding a storage capacitance between 20 and 30 fF in a 0.4 μm technology! Obviously, these size reductions are not free. The production and manufacturing of those esoteric devices to obtain a reasonable yield has become an increasingly difficult and expensive undertaking. More



(a) Trench-capacitor cell (from [Mano87]) (b) Stacked-capacitor cell (from [Taguchi91])

Figure 12-38 Advanced 1T DRAM memory cells.

so than physics or implementation impediments, economics will determine whether there is life beyond 256-Mbit or 1-Gbit semiconductor memories.

12.2.4 Contents-Addressable or Associative Memory (CAM)

The concept of the content-addressable memory was described earlier in the chapter. A CAM is a special type of memory device that stores data, but also has the ability to compare all the stored data in parallel with incoming data in an efficient manner. Figure 12-39 shows a possible implementation of a CAM array. The cell combines a traditional 6T RAM storage cell (*M4–M9*) with additional circuitry to perform a 1-bit digital comparison (*M1–M3*). When the cell is to be written, complementary data is forced onto the bit lines, while the word line is enabled as in a standard SRAM cell. In the compare mode, the stored data (S and its complement $\overline{S}$) are compared to in the incoming data, which is provided on the complementary bit lines (*Bit* and $\overline{Bit}$). The *Match* line is tied to all the CAM cells in a given row, and is initially precharged to V_{DD}. If S and *Bit* match, the internal node *int* is discharged, and *M1* is turned off, keeping the match line high. However, if the stored and incoming bit are different, *int* is charged to $V_{DD} - V_T$ causing the match line to discharge. For example, if $Bit = V_{DD}$ and $S = 0$, *int* charges up through *M3*. It is easily verified that the circuit performs nothing other than an XNOR function (or comparator).

It is important to note that the pull-down device in the comparator is connected to each of the CAM cells in a row in a wired-OR fashion. That is, even if only one of the bits in a given row mismatches, the match line is pulled low. For a memory with N rows, most rows (mismatches) will be pulled low in a given cycle. Clearly, not an enticing perspective from a power dissipation viewpoint. CAMs typically are not very power efficient! It is possible to

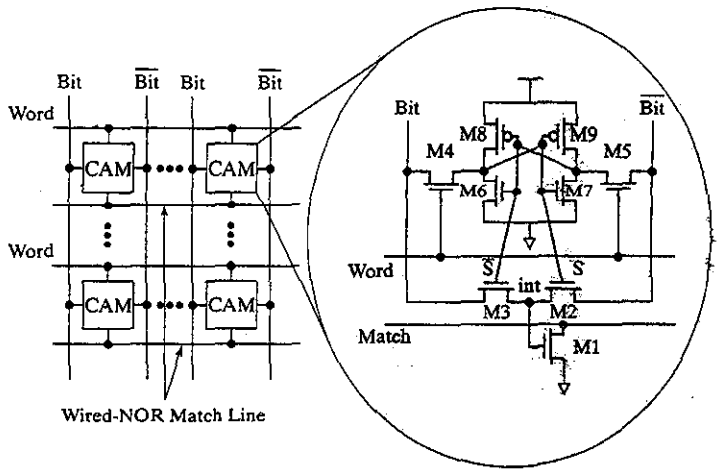


Figure 12-39 9-Transistor CAM cell.

re-arrange the logic such that only the match line switches, but this comes at a significant degradation in performance.

Example 12.11 Usage of Associate Memory in Cache Applications

The application of the CAM cell to a fully-associative *cache memory* is shown in Figure 12-40. A cache memory is a small, but fast memory that stores a small fraction of the overall memory required in a digital system. It works on the principle of spatial and temporal data locality. For example, if a data location is accessed at some point in time in a program, there is a high probability that it will be accessed again in the near future. The cache is placed between the processor and the large and dense main (DRAM) memory. It helps to defray the cost (in terms of time and power) of getting the data from the large memory. The cache memory array consists of two parts, the CAM array that stores addresses and a regular SRAM array that stores data. When the processor needs to write data, the address is written into the CAM and the data into the SRAM array. When a new address and data needs to be written and the cache is full, one entry must be displaced. This is done through a cache replacement policy. For instance, the least-accessed data word is a good candidate for displacement. In the read mode, the address of the data requested is presented to the CAM array, and parallel search takes places. A match indicates that the data is indeed available in the cache. The match signal acts as the word-line enable for the SRAM array which ultimately provides the data. In the case of a mismatch (this is, all match lines are low), the data must be obtained from the external slow system memory, and a cache miss has occurred.

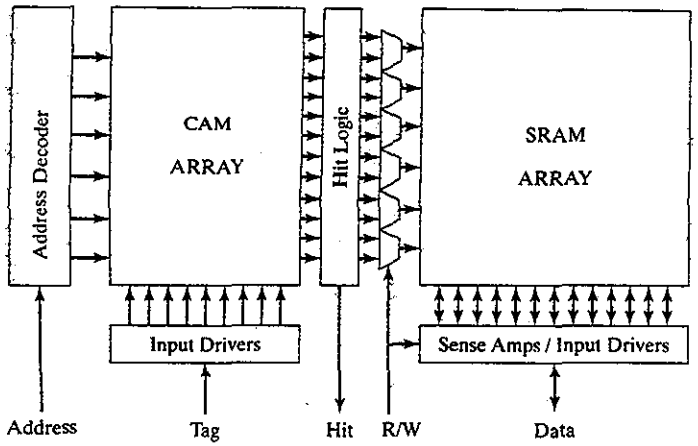


Figure 12-40 Application of CAM cell: high-performance on-chip cache memory.

Interested readers can find a lot more information about caches in the many textbooks covering computer architecture (for example, [Hennessy02]).

12.3 Memory Peripheral Circuitry*

Since the memory core trades performance and reliability for reduced area, memory design relies exceedingly on the peripheral circuitry to recover both speed and electrical integrity. While the design of the core is dominated by technological considerations and is largely beyond the scope of the circuit designer, it is in the design of the periphery where a good designer can make an important difference. In this section, we discuss the address decoders, I/O drivers/buffers, sense amplifiers, and memory timing and control.

12.3.1 The Address Decoders

Whenever a memory allows for random address-based access, address decoders must be present. The design of these decoders has a substantial impact on the speed and power consumption of the memory. In Section 12.1.2, we introduced two classes of decoders—the row encoders, whose task it is to enable one memory row out of 2^M, and the column and block decoders, which can be described as 2^K-input multiplexers, where M and K are the widths of the respective fields in the address word. While conceiving these decoders, it is important to keep the complete memory floorplan in perspective. These units are tightly coupled to the memory core, so that a geometry matching between the cell dimensions of decoders and the core is a must (*pitch matching*). Failing to do so would lead to a dramatic wiring overhead with its associated delay and power dissipation. Examples of pitch-matched decoders and memory arrays are shown in the case studies at the end of this chapter.

Row Decoders

A 1-out-of-2^M decoder is nothing less than a collection of 2^M complex, M-input, logic gates. Consider an 8-bit address decoder. Each of the outputs WL_i is a logic function of the 8 input address signals (A_0 to A_7). For example, the rows with addresses 0 and 127 are enabled by the following logic functions:

$$\begin{aligned} WL_0 &= \bar{A}_0\bar{A}_1\bar{A}_2\bar{A}_3\bar{A}_4\bar{A}_5\bar{A}_6\bar{A}_7 \\ WL_{127} &= \bar{A}_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 \end{aligned} \tag{12.9}$$

This function can be implemented in two stages, using a single 8-input NAND gate and an inverter. For a single-stage implementation, it can be transformed into a wide NOR using De Morgan's rules:

$$\begin{aligned} WL_0 &= \overline{A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6 + A_7} \\ WL_{127} &= \overline{A_0 + \bar{A}_1 + \bar{A}_2 + \bar{A}_3 + \bar{A}_4 + \bar{A}_5 + \bar{A}_6 + \bar{A}_7} \end{aligned} \tag{12.10}$$

In essence, to implement this logic function, an 8-input NOR gate is needed per row. This poses several imposing challenges. First of all, the layout of the wide-NOR gate must fit within the word-line pitch. Secondly, the large fan-in of the gate has a negative impact on performance. The propagation delay of the decoder is a matter of prime importance, as it adds directly to both read- and write-access times. In addition, this NOR gate has to drive the large load presented by the word line, while not overloading the input addresses. Finally, the power dissipation of the decoder has to be kept in check. In the following paragraphs, we discuss static and dynamic implementation options.

Static Decoder Design Implementing a wide-NOR function in complementary CMOS is impractical. One possible solution is to go back to a pseudo-NMOS design style, which allows for an efficient implementation of wide NORs. Power-dissipation concerns make this approach not very attractive in today's overconstrained design world.

Fortunately, some of the principles introduced in Chapter 6 come to the rescue. Splitting a complex gate into two or more logic layers most often produces both a faster and a cheaper implementation. This decomposition concept makes it possible to build fast and area-efficient decoders in complementary CMOS, and is used effectively in most memories today. Segments of the address are decoded in a first logic layer called the *predecoder*. A second layer of logic gates then produces the final word-line signals.

Consider the case of an 8-input NAND decoder. The expression for WL_0 can be regrouped in the following way:

$$\begin{aligned} WL_0 &= \overline{\bar{A}_0\bar{A}_1\bar{A}_2\bar{A}_3\bar{A}_4\bar{A}_5\bar{A}_6\bar{A}_7} \\ &= \overline{(\overline{A_0 + A_1})(\overline{A_2 + A_3})(\overline{A_4 + A_5})(\overline{A_6 + A_7})} \end{aligned} \tag{12.11}$$

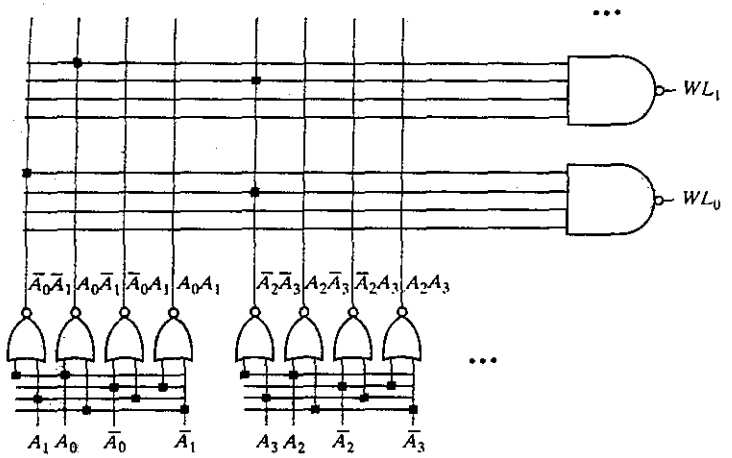


Figure 12-41 A NAND decoder using 2-input predecoders.

For this particular case, the address is partitioned into sections of 2 bits that are decoded in advance. The resulting signals are then combined using 4-input NAND gates to produce the fully decoded array of word-line signals. The resulting structure is pictured in Figure 12-41.

The use of a predecoder is advantageous in many ways:

- It reduces the number of transistors required. Assuming that the predecoder is implemented in complementary static CMOS, the number of active devices in the 8-input decoder equals $(256 \times 8) + (4 \times 4 \times 4) = 2{,}112$, which is 52% of a single stage decoder, which would require 4,096 transistors.
- As the number of inputs to the NAND gates is halved, the propagation delay is reduced by approximately a factor of 4. Remember the squared dependency between delay and fan-in.

Still, in this design, a 4-input NAND gate is driving the word line, which presents a large load. The best driver for large capacitances is an inverter—hence, the output of the NAND should be buffered. To bring the decoder design closer to optimal, the rules of the logical effort can be directly applied. It was concluded in Chapter 6 that, when driving very large loads, it is beneficial to have even more stages of logic. By performing more logic transformations similar to Eq. (12.11), the decoder can be broken into additional levels of logic, each of which consists of 2-input NANDs, 2-input NORs, or inverters.

Example 12.12 Decoder Design

Minimizing the delay of a memory decoder is a well-defined problem. The total capacitance of the word lines can be calculated, while the maximum capacitance that loads the

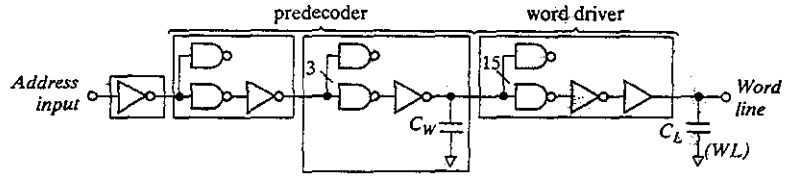


Figure 12-42 Logic path of an eight-bit decoder.

input address lines is constrained by the design. This specifies the effective fan-out of the decoder path, F. A number of logic structures can be derived that implement an 8-input AND function, one of which is shown in Figure 12-42. The chosen structure consists of inverters and 2-input NANDs. The input addresses are buffered first, and groups of 4 bits are predecoded. These predecoded signals are routed vertically, and connected to the final level of decoders. Since the decoders in the final level consist of just 2-input NANDs, they fit in the word line pitch.

The logical effort of this path equals $G = (4/3)^3 = 2.4$, and the intrinsic delay is $P = 12$. The total branching B equals 128 ($= 2 \times 4 \times 16$) as indicated in Figure 12-42. By specifying the effective fan-out, F, a minimum delay and optimal gate sizing can be determined.

In this analysis, we neglected the wire capacitance C_W, between the predecoder and the final decoders, which is substantial in larger memories. This becomes clear when inspecting complete memory architectures, such as the one shown in Figure 12-6. By including this fixed capacitive load in the optimization, the problem becomes much more difficult, but solvable [Amrutur01].

A final optimization in performance is enabled by taking into account that the word lines are normally held low. Hence, we should only optimize the rising transition. The logical effort of all the gates in the path can be reduced by skewing the gate sizing to favor only one transition.

All large decoders are realized using at least a two-layer implementation. This predecoder–final decoder configuration has another advantage. Adding a select signal to each of the predecoders makes it possible to disable the decoder when the memory block in question is not selected. This results in important power savings.

Dynamic Decoders Since only one transition determines the decoder speed, it is interesting to evaluate other circuit implementations. One option, dismissed earlier, is the use the pseudo-NMOS logic. Dynamic logic offers a better alternative. A first solution is presented in Figure 12-43, where the transistor diagram and the conceptual layout of a 2-to-4 decoder is depicted. Notice that this structure is geometrically identical to the NOR ROM array, differing only in the data patterns.

In a similar fashion, we can also implement the decoder as a NAND array, effectively realizing the inverse of the functions of Eq. (12.9). In this case, all the outputs of the array are high

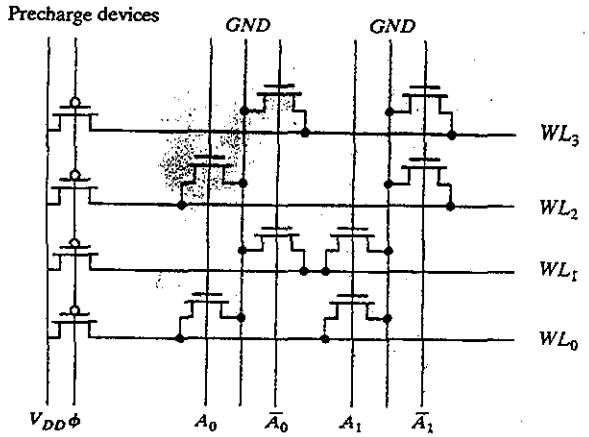


Figure 12-43 Dynamic 2-to-4 NOR decoder.

by default with the exception of the selected row, which is low. This "active low" signaling is in correspondence with the word-line requirements of the NAND ROM, as discussed in Section 12.2.1 Observe that the interface between decoder and memory often includes a buffer/driver that can be made inverting whenever needed. A 2-to-4 decoder in NAND configuration is shown in Figure 12-44.

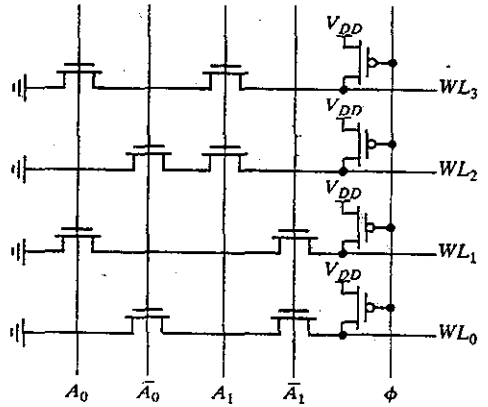


Figure 12-44 A 2-to-4 MOS dynamic NAND decoder. This implementation assumes that all address signals are low during precharge. An alternative approach is to provide evaluate transistors at the bottom of each transistor chain.

All the performance and density considerations raised in the discussion of NAND and NOR ROM arrays are valid. NOR decoders are substantially faster, but they consume more area than their NAND counterparts and dramatically more power. This is clear from the following observation: Only a single word line is being pulled down after the precharge in a NAND decoder, while only a single wire stays high in the NOR case. Similarly to the static decoder design, larger decoders are built using a multilayer approach.

Column and Block Decoders

Column decoders should match the bit line pitch of the memory array. The functionality of a column and block decoder is best described as a 2^K-input multiplexer, where K stands for the size of the address word. For read–write arrays, these multiplexers can be either separate or shared between read and write operations. During the read operation, they have to provide the discharge path from the precharged bit lines to the sense amplifier. When performing a write operation to a memory array, they have to be able to drive the bit line low to write a 0 in the memory cell.

Two implementations of this multiplexing function are in general use. Which one to choose depends upon area, performance, and architectural considerations.

One implementation is based on the CMOS pass-transistor multiplexer introduced in Chapter 6 (Figure 6-46). The control signals of the pass transistors are generated using a K-to-2^K predecoder, realized along the lines described in the previous section. The schematic of a 4-to-1 column decoder, using only NMOS transistors is shown in Figure 12-45. Complementary transmission gates must be used when these multiplexers are shared between the read and write operations to be able to provide a full swing in both directions. The main advantage of this approach is its speed. Only a single pass transistor is inserted in the signal path, which introduces only a minimal extra resistance. The column decoding is one of the last actions to be performed in the read sequence, so that the predecoding can be executed in parallel with other operations, such as the memory access and sensing, and can be performed as soon as the column address is

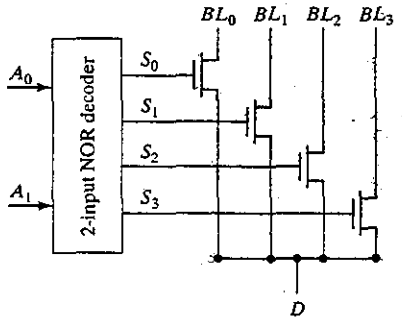


Figure 12-45 Four-input pass-transistor-based column decoder using a NOR predecoder.

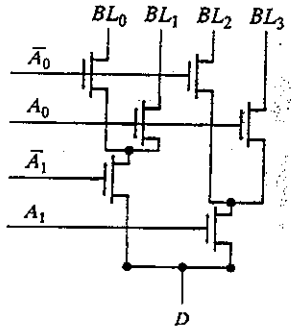


Figure 12-46 A 4-to-1 tree-based column decoder.

available. Consequently, its propagation delay does not add to the overall memory access time. Slower implementations such as NAND decoders might even be acceptable. The disadvantage of the structure is its large transistor count. $(K + 1)2^K + 2^K$ devices are needed for a 2^K-input decoder. For instance, a 1024-to-1 column decoder requires 12,288 transistors. One should also realize that the capacitance and thus the transient response at node D is proportional to the number of inputs of the multiplexer.

A more efficient implementation is offered by a *tree decoder* that uses a binary reduction scheme, as shown in Figure 12-46. Notice that no predecoder is required. The number of devices is drastically reduced, as is shown in the following equation (for a 2^K-input decoder):

$$N_{tree} = 2^K + 2^{K-1} + \ldots + 4 + 2 = 2 \times (2^K - 1) \tag{12.12}$$

This means that a 1024-to-1 decoder requires only 2046 active devices, a reduction by a factor of 6! On the negative side, a chain of K series-connected pass transistors is inserted in the signal path. Because the delay increases quadratically with the number of sections, the tree approach becomes prohibitively slow for large decoders. This can be remedied by inserting intermediate buffers. A progressive sizing of the transistors is another option, with the transistor size increasing from bottom to top. A final option is to combine the pass-transistor and tree-based approaches. A fraction of the address word is predecoded (for instance, the *msb* side), while the remaining bits are tree decoded. This can reduce both the transistor count and the propagation delay.

Example 12.13 Column Decoders

Consider a 1024-to-1 decoder. Predecoding 5 bits results in the following transistor tally:

$$N_{dec} = N_{pre} + N_{pass} + N_{tree} = 6.$$
$$2^5 + 2^{10} + 2(2^5 - 1) = 1278!$$

The number of series-connected pass transistors is reduced to six.

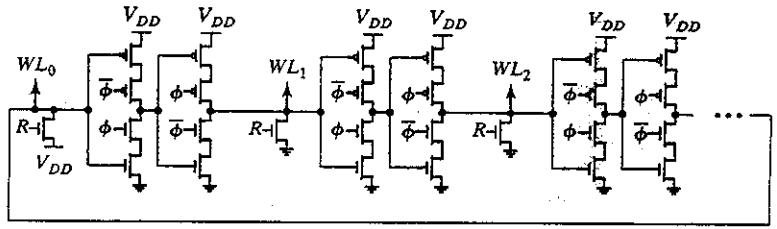


Figure 12-47 Decoder for circular shift register. The *R* signal resets the pointer to the first position.

Decoders for Non-Random-Access Memories

Memories that are not of the random-access class do not need a full-fledged decoder. In a serial-access memory, such as a video-line memory, the decoder degenerates into an M-bit shift register, with M the number of rows. Only one of the bits is high at a time and is called a *pointer*. The pointer moves to the next position every time an access is performed. An example of such a degenerated decoder implemented using a C^2MOS D-FF is shown in Figure 12-47. Similar approaches can be devised for other memory classes such as FIFOs.

12.3.2 Sense Amplifiers

Sense amplifiers play a major role in the functionality, performance and reliability of memory circuits. In particular, they perform the following functions:

- *Amplification*—In certain memory structures such as the 1T DRAM, amplification is required for proper functionality since the typical circuit swing is limited to 100 millivolts [Itoh01]. In other memories, it allows resolving data with small bit-line swings, enabling reduced power dissipation and delay.
- *Delay Reduction*—The amplifier compensates for the restricted fan-out driving capability of the memory cell by accelerating the bit line transition, or by detecting and amplifying small transitions on the bit line to large signal output swings.
- *Power reduction*—Reducing the signal swing on the bit lines can eliminate a substantial part of the power dissipation related to charging and discharging the bit lines.
- *Signal restoration*—Because the read and refresh functions are intrinsically linked in 1T DRAMs, it is necessary to drive the bit lines to the full signal range after sensing.

The topology of the sense amplifier is a strong function of the type of memory device, the voltage levels, and the overall memory architecture. Sense amplifiers are analog circuits by nature, and an in-depth analysis requires substantial analog expertise. Therefore, only a brief introduction to the design of such devices is given here. For an elaborate discussion on the design of amplifiers, the reader is referred to textbooks such as [Gray01][Sedra87].

Differential Voltage Sensing Amplifiers

A differential amplifier takes small-signal differential inputs (i.e., the bit-line voltages), and amplifies them to a large-signal single-ended output. It is generally known that a differential approach presents numerous advantages over its single-ended counterpart—one of the most important being the *common-mode rejection*. That is, such an amplifier rejects noise that is equally injected to both inputs. This is especially attractive in memories where the exact value of the bit line signal varies from die to die and even for different locations on a single die. In other words, the absolute value of a 1 or 0 signal is not exactly known and might vary over quite a large range. The picture is further complicated by the presence of multiple noise sources, such as switching spikes on the supply voltages and capacitive cross talk between word and bit lines. The impact of those noise signals can be substantial, especially when we realize that the amplitude of the signal to be sensed is generally small. The effectiveness of a differential amplifier is characterized by its ability to reject the common noise and amplify the true difference between the signals. The signals common to both inputs are suppressed at the output of the amplifier by a ratio called the *common-mode rejection ratio* (CMRR). Similarly, spikes on the power supply are suppressed by a ratio called the *power-supply rejection ratio* (PSRR). Differential sensing is therefore considered the technique of choice.

Unfortunately, the differential approach is only directly applicable to SRAM memories, since these are only the memory cells that offer a true differential output. Figure 12-48 shows the most basic differential sense amplifier. Amplification is accomplished with a single stage, based on the current mirroring concept. The input signals (*bit* and $\overline{bit}$) are heavily loaded and driven by the SRAM memory cell. The swing on those lines is small as the small memory cell drives a large capacitive load. The inputs are fed to the differential input devices (M_1 and M_2), and transistors M_3 and M_4 act as an active current mirror load. The amplifier is conditioned by the sense amplifier enable signal, *SE*. Initially, the inputs are precharged and equalized to a common value, while *SE* is low disabling the sensing circuit. Once the read operation is initiated, one of

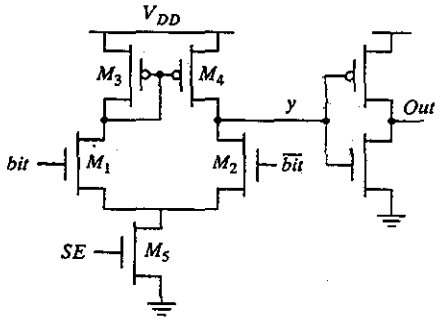


Figure 12-48 Basic differential sense amplifier circuit.

the bit lines drops. *SE* is enabled when a sufficient differential signal has been established, and the amplifier evaluates.

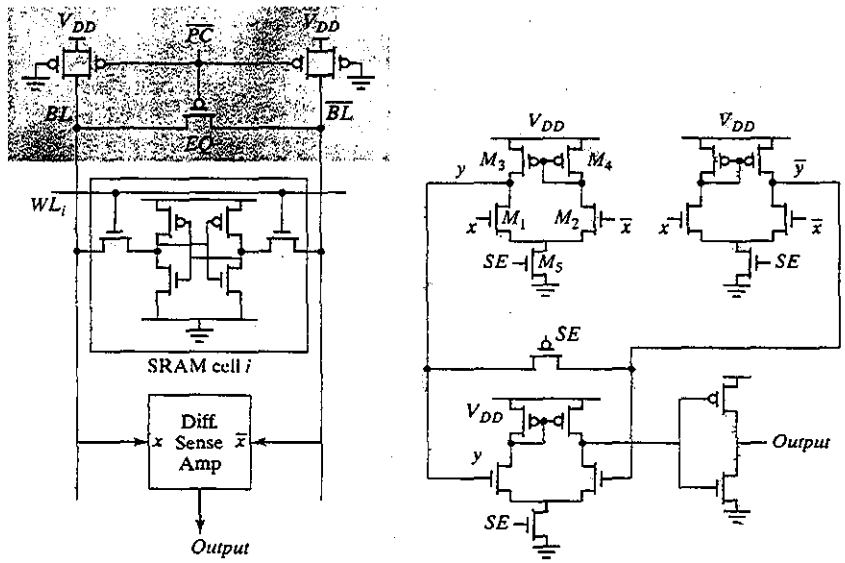
The gain of such the differential-to-single ended amplifier is given by

$$A_{sense} = -g_{m1}(r_{o2} \| r_{o4}) \tag{12.13}$$

where g_{m1} is the transconductance of the input transistors, and r_o the small-signal device resistance of the transistor. The r_o of the MOS transistor is very high in the saturation region of the MOSFET. The transconductance of the input devices can be increased by either widening the devices, or by increasing the bias current. The latter also reduces the output resistance of M_2, which limits the usefulness of this approach. A gain of around 100 can be achieved. However, the gain of sense amplifiers typically is set to around 10. The main goal of the sense amplifier is the rapid production of an output signal. Gain is hence secondary to response time. Multiple stages are required to achieve the desired full-swing signal.

Figure 12-49 shows a fully differential two-stage sensing approach along with the SRAM bit column structure. The bit lines are connected to the inputs x and $\bar{x}$ of the two-stage differential amplifier. A read cycle proceeds as follows.

1. In the first step, the bit lines are precharged to V_{DD} by pulling $\overline{PC}$ low. Simultaneously, the *EQ*-PMOS transistor is turned on, ensuring that the initial voltages on both bit lines are



(a) SRAM sensing scheme (b) two-stage differential amplifier

Figure 12-49 Differential sensing as applied to an SRAM memory column.

identical. This operation, called *equalization*, is necessary to prevent the sense amplifier from making erroneous excursions when turned on. In practice, every differential signal in the memory is equalized before performing a read. Equalization is critical when the bit lines are precharged through NMOS pull-ups, since the precharge value can differ due to the variations in the device threshold.

2. The read operation is started by disabling the precharge and equalization devices and enabling one of the word lines. One of the bit lines is pulled low by the selected memory cell. Notice that a grounded PMOS load, placed in parallel with the precharge transistor, limits the bit line swing and speeding up the next precharge cycle.
3. Once a sufficient signal is built up (typically around 0.5 V), the sense amplifier is turned on by raising *SE*. The differential input signal on the bit lines is amplified by the two stage amplifier and eventually a rail to rail output is produced at the output of the inverter.

The two-stage sense-amplifier circuit is shown in Figure 12-49b. While these circuits use PMOS loads and NMOS input devices, the dual configurations with PMOS input devices and NMOS loads are also regularly used, depending upon biasing conditions. Note that by pulsing the *SE* control signal to be active for short evaluation periods, static power in the amplifier can be reduced. It should also be noted that a single sense amplifier is shared between multiple columns by inserting the column decoder pass transistors between the memory cells and the amplifier (the input levels of the amplifier are reduced by a device threshold in this case). This results in area savings and power reduction.

The sense amplifier presented earlier decouples the inputs and outputs. That is, the bit-line swing is determined by the SRAM cells and the static PMOS load. A radically different sensing approach is offered by the circuit of Figure 12-50, where a CMOS cross-coupled inverter pair is used as a sense amplifier. A CMOS inverter exhibits a high gain when positioned in its transient region, as was established in Chapter 5. To act as a sense amplifier, the flip-flop is initialized in its metastable point by equalizing the bit lines. A voltage difference is built over the bit lines in the course of the read process. Once a large enough voltage gap is created, the sense amplifier is enabled by raising *SE*. Depending upon the input, the cross-coupled pair traverses to one of its stable operation points. The transition is swift as a result of the positive feedback.

While the flip-flop sense amplifier is simple and fast, it has the property that inputs and outputs are merged, so that a full rail-to-rail transition is enforced on the bit lines. This is exactly what is needed for a 1T DRAM, where a restoration of the signal levels on the bit lines is necessary for the refresh of the cell contents. The cross-coupled cell is, therefore, almost universally used in DRAM designs. How to turn a single-ended memory structure such as the DRAM cell into a differential one is discussed in the next section.

Single-Ended Sensing

While differential sensing is by far the preferred approach, memory cells used in ROMs, E(E)PROMs and DRAMs are inherently single ended. A first option to address this problem is to resort to single-ended amplification. Since the bit lines are typically precharged, an asymmetrically biased inverter, as introduced in Figure 9-35, is a good candidate for such. An interesting variant, called the *charge-redistribution amplifier*, is often used in small memory structures. (See Figure 12-51.) The basic idea is to exploit the imbalance between a large capacitance C_{large} and a much smaller component C_{small}. The two capacitors are isolated by the pass transistor M_1 [Heller75].

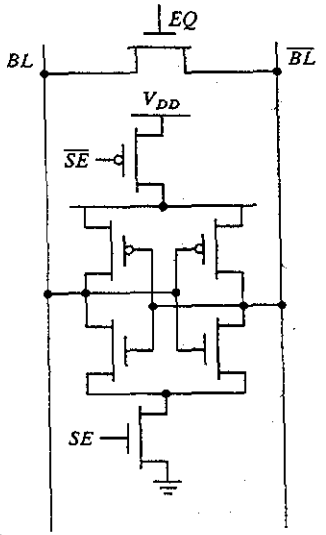
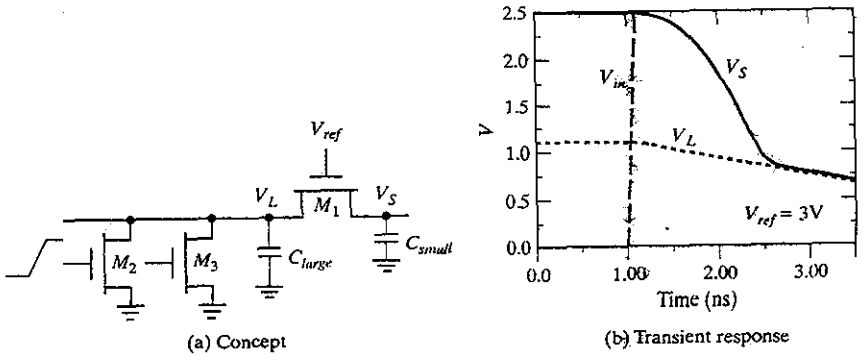


Figure 12-50 Cross-coupled CMOS inverter latch used as sense amplifier.

The initial voltages on nodes L and S (V_{L0} and V_{S0}) are precharged to $V_{ref} - V_{Tn}$ and V_{DD} by connecting node S to the supply voltage. Because of the voltage drop over M_1, V_L only



(a) Concept (b) Transient response

Figure 12-51 Charge-redistribution amplifier.

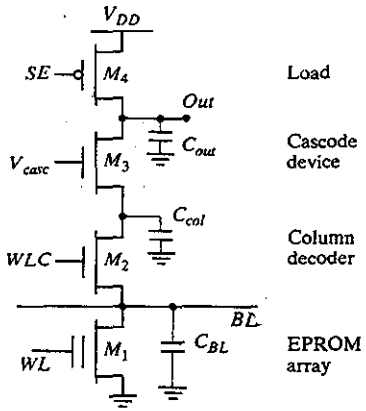


Figure 12-52 Charge-redistribution amplifier as used in EPROM memory.

precharges to $V_{ref} - V_{Tn}$. When one of the pull-down devices (e.g., M_2) turns on, node L with its large capacitance slowly discharges. As long as $V_L \geq V_{ref} - V_{Tn}$ transistor M_1 is off, and V_S remains constant. Once V_L drops below the trigger voltage $(V_{ref} - V_{Tn})$, M_1 turns on. A charge redistribution is initiated, and nodes L and S equalize. This can happen very fast due to the small capacitance on the latter node. A small voltage variation on node L translates into a large voltage drop on node S, as is illustrated in the simulated transient response of Figure 12-51b. The circuit thus acts as an amplifier. The resulting signal can be fed into an inverter with a switching threshold larger than $V_{ref} - V_{Tn}$ to produce a rail-to-rail swing.

The schematics of the charge-redistribution amplifier, as it is used in a memory, are presented in Figure 12-52. This structure has been very popular in EPROM memories. The disadvantage of the charge-redistribution amplifier is that it operates with a very small noise margin. A small variation of node L due to noise or leakage may cause an erroneous discharge of S. (See Figure 12-51.) Careful design and analysis is therefore necessary.

Single-to-Differential Conversion

Larger memories (>1 MBit) that are exceedingly prone to noise disturbances resort to translating the single-ended sensing problem into a differential one. The basic concept behind the single-to-differential conversion is demonstrated in Figure 12-53. A differential sense amplifier is connected to a single-ended bit line on one side and a reference voltage, positioned between the 0 and 1 levels, at the other end. Depending on the value of BL, the amplifier toggles in one or the other direction. Creating a good reference source is not as easy as it sounds, since the voltage levels tend to vary from die to die or even over a single die. The reference source must therefore track those variations. A popular way of doing so is illustrated in Figure 12-54 for the case of a 1T DRAM. The memory array is divided into two halves, with the differential amplifier placed in the middle. On each side, a column of so-called *dummy cells* is added. These are 1T memory cells that are similar to the others, but whose sole purpose is to serve as reference. This approach is often called the *open bit line architecture*.

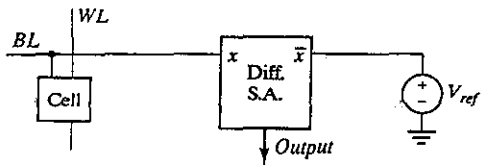


Figure 12-53 Single-to-differential conversion.

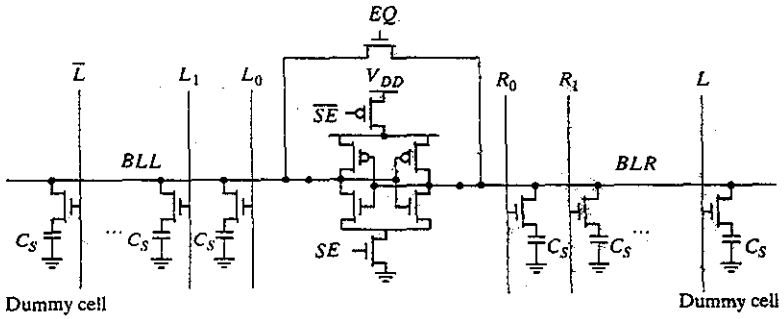


Figure 12-54 Open bit line architecture with dummy cells.

When the EQ signal is raised, both the bit lines BLL and BLR are precharged to $V_{DD}/2$. Enabling L and $\bar{L}$ at the same time ensures that the dummy cells are charged to $V_{DD}/2$. One of the word lines is enabled during the read cycle. Assume that a cell in the left half of the array is selected by raising WL_0, which causes a voltage change on BLL. The appropriate voltage reference is created by simultaneously selecting the dummy cell in the other memory half by raising L. Under the assumption that the left and right memory sides are perfectly matched, the resulting voltage on BLR resides between the 0 and 1 levels and causes the sense latch to toggle. Notice that maintaining perfect symmetry is important. Raising word lines WL_0 and L simultaneously turns the capacitive coupling between bit and word lines into a common-mode signal that is effectively eliminated by the sense amplifier. Observe that dividing the bit lines into two halves effectively reduces the bit line capacitance. This doubles the charge-transfer ratio and improves the signal-to-noise ratio.

Example 12.14 Sensing in 1T DRAM

The read-operation of a 1T DRAM implemented by using the open bit-line architecture, and the latch sense amplifier is simulated by using SPICE. The storage capacitance is set to 50 fF, while the bit line capacitance equals 0.5 pF. This is equivalent to a charge-transfer ratio of 9%. Assuming that the bit line is precharged to 1.25 V, charge redistribution results in a voltage drop of 110 mV for a 0 signal. Due to the threshold loss over the pass

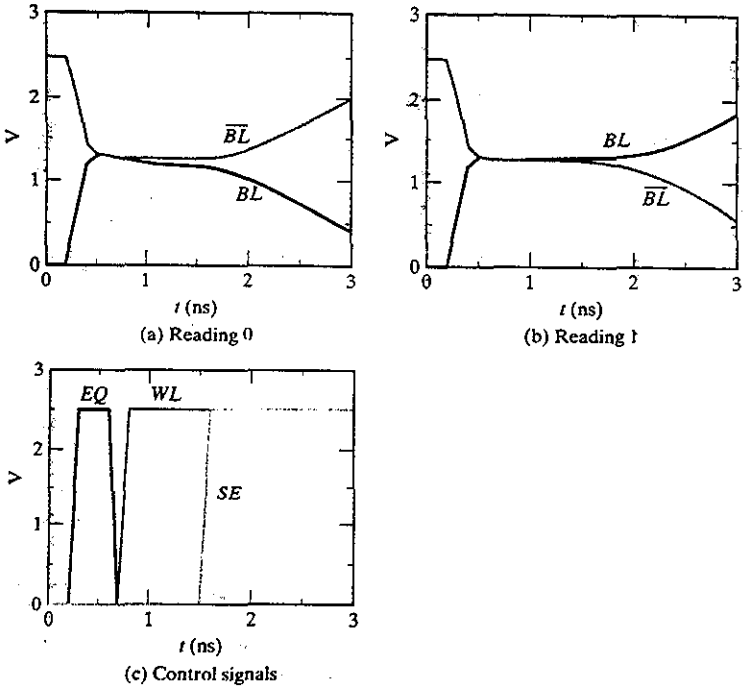


Figure 12-55 Simulation of the read process in an open bit line architecture with dummy cell. The dummy cell is connected to $\overline{BL}$ (C_S = 50 fF; C_{BL} = 0.5 pF).

transistor, the cell voltage associated with a 1 equals 1.9 V, which translates to a voltage increase of only 60 mV on the bit line after the enabling of the word line. These values are confirmed by the simulation, which shows values of 110 mV and 45 mV respectively, before the sense amplifier is turned on (Figure 12-55).

12.3.3 Voltage References

Most memories require some form of on-chip voltage generation. The operation of a sophisticated memory requires a number of voltage references and supply levels, including the following:

- *Boosted-Word-line Voltage*—In a conventional 1T DRAM cell, using an NMOS pass transistor, the maximum voltage level that can be written onto a cell equals $V_{DD} - V_T$, which negatively impacts the reliability of the memory. By raising the word-line voltage above V_{DD} (more specifically, to $V_{DD} + V_{Tn}$), a full-scale signal can be written. This "boosted-word-line" approach typically uses a charge pump to generate the elevated voltages.

- *Half-V_{DD}*: DRAM bit lines are precharged to $V_{DD}/2$, as discussed previously. This voltage must be generated on chip.
- *Reduced Internal Supply*: Most memory circuits operate at a lower power supply than the external supply. DRAM (and other memories) use internal voltage regulators to generate the required voltages, while being compliant with standard interface voltages.
- *Negative Substrate Bias*: An effective means to control the threshold voltages within a memory is to apply negative substrate biasing, augmented with a control loop. This approach has been used all recent generation of DRAM memories.

The design of voltage references falls under the category of analog circuit design. A short review of a couple of reference circuits is given next.

Voltage Down Converters

Voltage down converters are used to create low internal supplies, allowing the interface circuits to operate at higher voltages. Consequently, internal circuits can exploit aggressive voltage-scaling techniques using state-of-the-art scaled technologies, while remaining compatible with the outside world. The reduction of supply is in fact necessary to avoid breakdown in the deep-submicron devices. Level converters, discussed previously in Chapter 11, act as interfaces between the internal core and the external circuits. Regulators also serve to set a stable internal voltage while accepting a broad range on unregulated input voltages in battery-operated systems. This accommodates the fact that the battery voltage varies as a function of time.

Figure 12-56 shows the basic structure of a voltage down converter (also called a *linear regulator*). It is based on the operational amplifier that was described in the previous section. The circuit uses a large PMOS output driver transistor to drive the load of the memory circuit (an NMOS output device may also be used). The circuit uses negative feedback to set the output voltage V_{DL} to the reference voltage.

The converter must offer a voltage that is immune to variations in operating conditions. Slow variations, such as temperature changes, can be compensated by the feedback loop. Note that this is a feedback circuit, which can be unstable if improperly designed. In particular, the load current drawn by load varies wildly over time, and the converter must be designed to accommodate these wide variations.

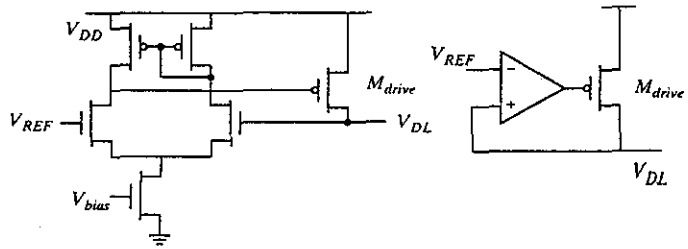


Figure 12-56 A voltage regulator and its equivalent representation.

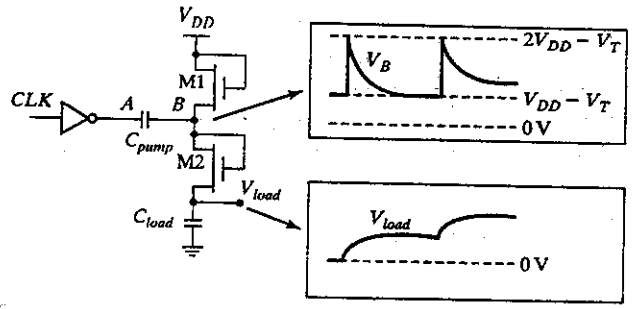


Figure 12-57 Simple charge pump and its signal waveforms.

Charge Pumps

Techniques such as word-line boosting and well biasing often need voltage sources that exceed the supply voltage, but do not draw much current. A charge pump is the ideal generator for this task. The concept is best explained with the simple circuit of Figure 12-57. Transistors M1 and M2 are connected in diode style. Assume initially that the clock *CLK* is high. During this phase, node *A* is at GND and node *B* at $V_{DD} - V_T$. The charge stored in the capacitor is given by

$$Q = C_{pump}(V_{DD} - V_T) \tag{12.14}$$

During the second phase, *CLK* goes low, raising node *A* to V_{DD}. Node *B* rises in concert, effectively shutting off M1. When *B* is one threshold above V_{load}, M2 starts to conduct and charge is transferred to C_{load}. During consecutive clock cycles, the pump continues to deliver charge to V_{load} until the maximum voltage of 2 $(V_{DD} - V_T)$ is reached at the output. The amount of current that can be drawn from the generator is primarily determined by the capacitor's size and the clock frequency. The efficiency of generator, which measures how much current is wasted during every pump cycle, is between 30 and 50%. Charge pumps should hence only be used for generators that draw little current. The circuit presented here is quite simple and not that effective. A wide range of more complex charge pumps have been devised for larger voltage ranges and improved efficiency.

Voltage Reference

An accurate and stable voltage reference is an important component of the voltage down converter. The reference voltage is assumed to be relatively constant over power supply and temperature variations. Figure 12-58 shows an example of a V_T reference generator. The bottom devices (M_3 and M_4) act as a current mirror to force the same current through the drain of M_1 and the resistor R_1. By making the device M_1 large and keeping the current small enough, the source-to-gate voltage for M_1 can be made approximately equal to $|V_{Tp}|$, as can be derived from the equation

$$|V_{GS,M1}| = |V_{Tp}| + \sqrt{\frac{2I_{M1}}{|k_{p,M1}|}} \tag{12.15}$$

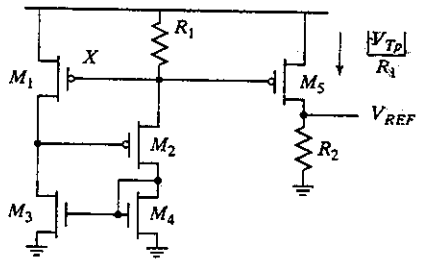


Figure 12-58 A simple V_T reference generator.

Also the currents going through the resistor and the drain current of M_1 both equal $|V_{Tp}|/R_1$. Note that M_2 acts as a biasing transistor. Since devices M_1 and M_5 experience the same gate-to-source voltage, the drain current of M_1 is mirrored to M_5. The reference voltage is thus given by

$$V_{REF} = |V_{Tp}| \cdot \frac{R_2}{R_1} \tag{12.16}$$

Variations in threshold voltage can be compensated for by laser trimming of the resistors. V_{REF} also exhibits good temperature stability. By choosing the appropriate materials for the implementation of R_1 and R_2, the temperature dependence of V_{TP} can be cancelled by that of R_2/R_1. This circuit was used in a 16-Mb DRAM [Hidaka92], and displayed excellent stability: $\Delta V_{REF}/\Delta T = 0.15$ mV/°C, and $\Delta V_{REF}/\Delta V_{DD} = 10$ mV/V.

The preceding discussion is by no means complete. A more elaborate discussion of references is unfortunately beyond the scope of this text. The reader is referred to [Itoh01] and [Gray01] for a detailed treatment.

12.3.4 Drivers/Buffers

The length of word and bit lines increases with increasing memory sizes. Even though some of the associated performance degradation can be alleviated by partitioning the memory array, a large portion of the read and write access time can be attributed to the wire delays. A major part of the memory-periphery area is therefore allocated to the drivers, in particular the address buffers and the I/O drivers. The design of cascaded buffers was discussed at length in Chapter 5. All the issues raised there are valid here as well.

12.3.5 Timing and Control

The foregoing discussions present a picture of the memory module as a complex entity whose operation is governed by a well-defined sequence of actions such as address latching, word-line decoding, bit line precharging and equalization, sense-amplifier enabling, and output driving. To be operational, it is essential that this sequence be adhered to under all operational circumstances and over a range of device and technology parameters. A careful timing of the different events is necessary if maximum performance is to be achieved. Although the timing and control

circuitry only occupies a minimal amount of area, its design is an integral and defining part of the memory design process. It requires careful optimization, and the execution of extensive and repetitive SPICE simulations over a range of operating conditions.

Over time, a number of different memory-timing approaches have emerged that can be largely classified as *clocked* and *self-timed*. A typical example of each class is discussed to illustrate the differences.

DRAM—A Clocked Approach

Since the early days, DRAM memories have opted for a multiplexed addressing scheme where the row address and column address are presented in sequence on the same address bus to save package pins. This approach has survived many memory generations and is still in use today, although it is getting increasingly awkward and in disparity with system-level requirements.

In the multiplexed addressing scheme, the user must provide two main control signals—*RAS* (row-address strobe) and *CAS* (column-address strobe)—that indicate the presence of the row and column addresses, respectively. (See Figure 12-8.) Another control signal (*W*) indicates if the intended operation is a read or a write. These signals can be interpreted as external clock signals, and are used to time the internal memory events. Similar to the synchronous clocking approach, the *RAS* and *CAS* signals must be sufficiently separated so that all the ensued operations have come to completion. Figure 12-59 shows a simplified timing diagram of a (1 × 4)-Mbit DRAM memory and some of the imposed timing constraints. The full specification involves more than 20 timing parameters, all of which have to be held within precise bounds to ensure proper operation. Obviously, generating the correct timing signals for these memories is a nontrivial task!

All the internal timing signals such as the *EQ*, *PC*, and *SE* signals are derivatives of the external control signals. To maximize performance, some of those signals have to fall within very precise intervals. For instance, the *SE* signal cannot be applied too early after word-line decoding is started, since this increases the sensitivity to noise. On the other hand, postponing it for too long slows down the memory. The separation of these events is a function of the word-line delay that might vary with temperature and process variations. It is a common practice to adjust the timing interval with operating parameters by including a model of the word line (*dummy word line*) in the timing-generation circuitry. This "delay module" tracks the delay of the actual word line accurately, making the memory more robust while preserving performance. It is thus fair to state that DRAMs combine a synchronous approach at the global level with self-timed techniques for the generation of some of the local signals.

Synchronous DRAM Memories

One of the main challenges of the DRAM memory is the large access time and the low data through-put. With the ever-increasing speed of microprocessors and their SRAM caches, the performance gap between processor core and DRAM main memory is getting larger. This gap is becoming one of the main challenges in high-performance system design. To address this problem, high-speed DRAM synchronous memories such as SDRAM (Synchronous DRAM)

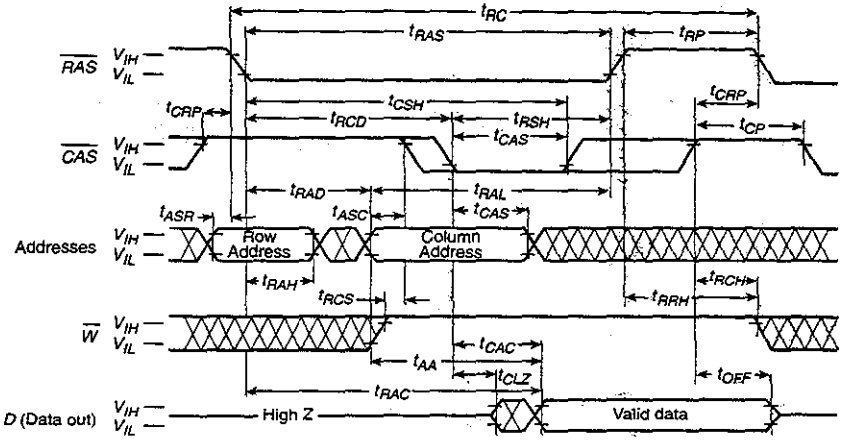


Figure 12-59 Read-cycle timing diagram for 4 M × 1 DRAM memory [Mot91]. The minimum length of the read cycle t_{RC} equals 110 ns. The spacing of the falling edges of *RAS* and *CAS* signals must range between 20 and 40 ns. A total of 24 timing constraints can be observed.

[Yoo95] and RDRAM (Rambus DRAM) [Crisp97] have been introduced. These memories present a major departure from the RAC/CAS timing model of traditional memories. While keeping the core of the memory relatively unchanged, synchronous DRAMs exploit the fact that core is highly parallel—in other words, for every given read and write cycle, a large number of bits can be read/written at the same time, albeit at a relatively low speed. Bringing these bits in or out of the memory chip can be accomplished with a high-speed synchronous interface, operating at a speed close to the processor clock frequency. This comes at the penalty of extra latches and buffers at the interface into the memory core, as well as high-speed circuitry to support the high-data rate I/O interface. The main contribution these memories offer is a redefinition of the interface into the memory. As long as data is written in large, sequential blocks, this approach has proven to be very effective.

Consider RDRAM as an example. Input/output data is transferred serially on a narrow bus, taking several clock cycles. The bus is operated at a very-high speed, however, and uses efficient packet protocols. Thus, a large amount of data can be transferred in a short period. Multiple memory chips can be connected to the bus, called the *Rambus channel*. Figure 12-60 shows a schematic diagram of the input/output circuitry of an RDRAM memory.

SRAM—A Self-Timed Approach

In contrast to DRAMs, SRAM memories have historically been designed from a globally static perspective. A memory operation is triggered by an event on the address bus or *R/W* signal (Figure 12-8b). No extra timing or control signals are needed. Such an approach seems to imply a fully static implementation for all circuitry such as decoders and sense amplifiers. A change in

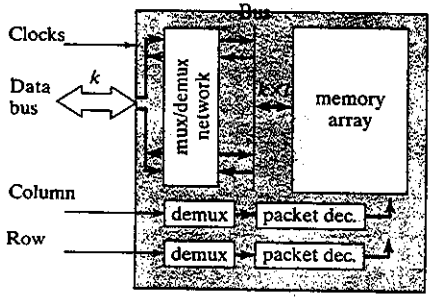


Figure 12-60 Schematic block diagram of RDRAM memory.

one of the input signals (data or address bus, *R/W*) then simply ripples through the subsequent layers of circuitry.

Our previous discussions made it apparent that such a fully static approach is not viable for larger memories for both area and power-dissipation considerations. SRAM memories, therefore, use a clever approach called *address-transition detection* (*ATD*) to automatically generate the internal signals such as *PC* and *SE* upon detection of a change in the external environment.

The *ATD* circuit plays an essential role in the architecture of SRAM and PROM modules. It acts as the source of most timing signals and is an integral part of the critical timing path. Speed is thus of utmost importance. A possible implementation of an *ATD* is shown in Figure 12-61. It consists of a number of transition-triggered one-shots (see Figure 7-50)—one per input bit—connected in a pseudo-NMOS NOR configuration. A transition on any of the input signals causes *ATD* to go low for a period t_d. The resulting pulse acts as the main timing reference for the rest of the memory, which results in a huge fan-out. Adequate buffering is thus recommended. A more elaborate view on SRAM timing can be found in the case studies at the end of this chapter.

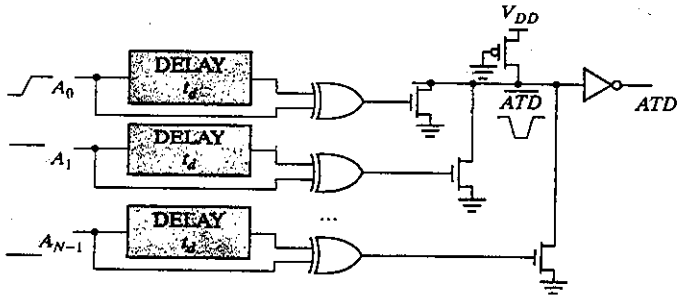


Figure 12-61 Address-transition detection circuitry. The delay lines are typically implemented as an inverter chain.

12.4 Memory Reliability and Yield*

Memories, both SRAM and DRAM, are operating under low signal-to-noise conditions. Maximizing the signals while minimizing the noise contributions is essential to achieve stable memory operation. Another problem plaguing memory design is the low yield due to structural and intermittent defects. This section discusses the nature of some of those problems, as well as potential solutions.

12.4.1 Signal-to-Noise Ratio

A tremendous effort is being made to produce memory cells that generate as large a signal as possible per unit area. Notwithstanding this effort, the produced signal quality decreases gradually with an increase in density, as is illustrated in Figure 12-62. For instance, the DRAM cell capacitance has degraded from approximately 70 fF for the 16 K memory to below 30 fF for the current generations. Simultaneously, the voltage levels have decreased for both power consumption and reliability purposes. Voltages at or below 1 V are the norm for newly designed memories, the main reason being the limited voltage stress that can be endured by the very thin oxides used in cells. Consequently, the signal charge stored on the capacitor has dropped with a factor of 5 from the 64 Kb to the 64 Mb generation. The fact that this drop is only that much, despite a cell area reduction of 100 over the same period of time, is solely due to various cell innovations.

At the same time, the increased integration density raises the noise level, due to the intersignal coupling. While the problem of the word-line-to-bit-line coupling capacitance was already an issue in the early 1980s, closer line spacing has brought the bit line–to–bit line coupling into the limelight. Contributing also to the problem are the higher speed requirements that result in an increasing switching noise for every new generation, and α-particle-

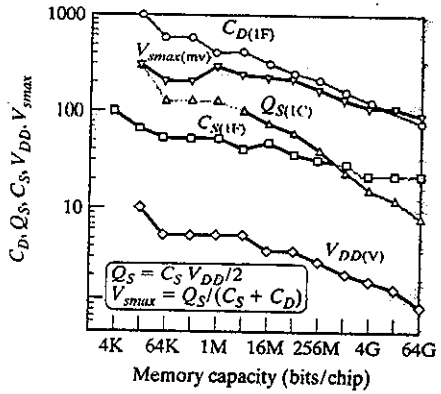


Figure 12-62 Trends in sensing parameters for DRAM memories (from [Itoh01]). C_D, V_{smax}, Q_s, and C_s stand for the bit line capacitance, sense signal, cell charge, and cell capacitance, respectively.

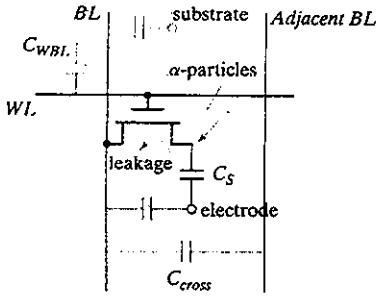


Figure 12-63 Noise sources in 1 T DRAM.

induced soft errors that can change the state of a high-impedance node at random and present an additional noise source. The latter problem was traditionally confined to dynamic memories, but now extends to static RAMs as well because the impedance of the SRAM storage nodes is rapidly increasing. Figure 12-63 enumerates the noise sources that impact the operation of the 1T DRAM cell, most of which are discussed in detail in the subsequent paragraphs.

Word-Line-to-Bit-line Coupling

Consider the open bit-line memory configuration of Figure 12-64. Word line WL_0 gets selected, which causes an amount of charge to be injected into the left bit line. We assume that the dummy cell on the right side, selected by WL_D, does not cause any charge redistribution, since it is precharged to $V_{DD}/2$. The presence of a coupling capacitance C_{WBL} between word line and bit line causes a charge redistribution to occur, whose amplitude approximately equals $\Delta_{WL} \times C_{WBL} / (C_{WBL} + C_{BL})$, where Δ_{WL} is the voltage swing on the word line and C_{BL} the bit line capacitance. If both sides of the memory array were completely symmetrical, the injected bit line noise (Δ_{BL}) would be identical on both sides and would appear as a common-mode signal to the sense amplifier. Unfortunately, this is not the case, because both coupling and bit line capacitance can vary substantially over the array.

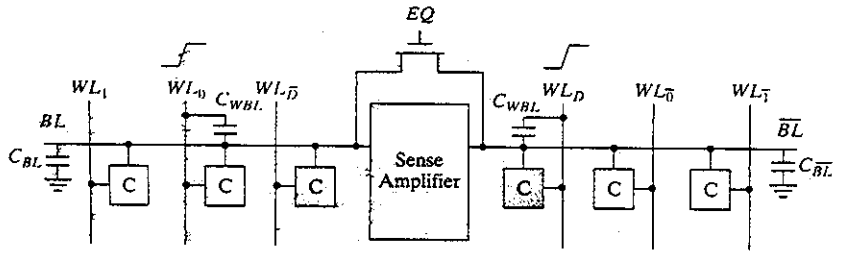


Figure 12-64 Word-line-to-bit-line coupling in open bit line architecture.

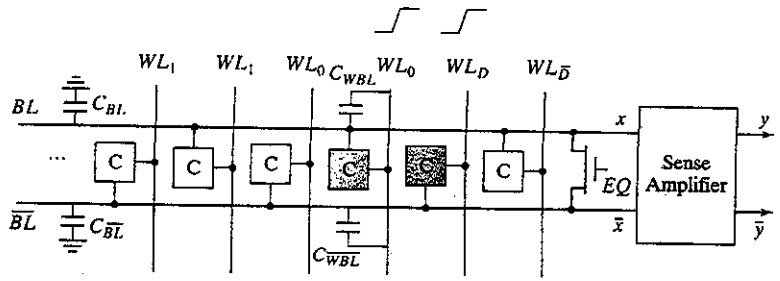


Figure 12-65 Folded bit line architecture for 1T DRAM.

This problem can be addressed by employing the *folded bit-line architecture* shown in Figure 12-65. Placing the sense amplifier at the end of the array and having BL and $\overline{BL}$ routed next to each other ensures a much closer matching between parasitic and bit line capacitances. The word-line signals (WL_0 and WL_D) cross both bit lines. The cross-coupling noise, hence, appears as a common-mode signal even if the voltage waveforms on WL_0 and WL_D differ considerably. While the folded bit line architecture has an obvious advantage in terms of noise suppression, it tends to increase the length of the bit line and consequently its capacitance. This overhead can be kept to a minimum by a clever interleaving of the cells.

The scenario just presented covers only a part of the total picture. A more complex case of word-line-to-bit-line coupling can be described as follows: During a read operation, a bit line BL undergoes a voltage transition ΔV. This voltage transition gets coupled to the nonselected word lines that are in precharged mode and thus represent high-impedance nodes. This noise signal, in turn, gets coupled back to other bit lines. The resulting cross talk depends on the data patterns stored in memory. Suppose that the majority of the bit lines are reading a 0. This causes a negative transition on the nonselected word lines, which in turn pulls down the bit lines. This is especially harmful for the (minority of) bit lines that are reading a 1 value. The fact that this noise signal is pattern dependent makes it particularly hard to detect. Testing a memory for functionality requires the application of a wide range of different data patterns. The folded bit line architecture helps to suppress this type of noise.

Bit-Line-to-Bit-Line Coupling

The impact of interwire cross coupling increases with reducing dimensions, as detailed in Chapter 9. This is especially true in the memory array, where the noise-sensitive bit lines run side by side for long distances. The art is to turn the noise signals into a common-mode signal for the sense amplifier. An ingenious way of doing so is represented by the *transposed (or twisted) bit-line architecture* illustrated in Figure 12-66. The first figure represents the straightforward implementation. Both BL and $\overline{BL}$ are coupled to the adjoining column (or row) by the capacitance C_{cross}. In the worst case, the signal swing observed at the sense amplifier can be reduced by

$$\Delta V_{cross} = 2\frac{C_{cross}}{C_{cross} + C_{BL}} V_{swing} \tag{12.17}$$

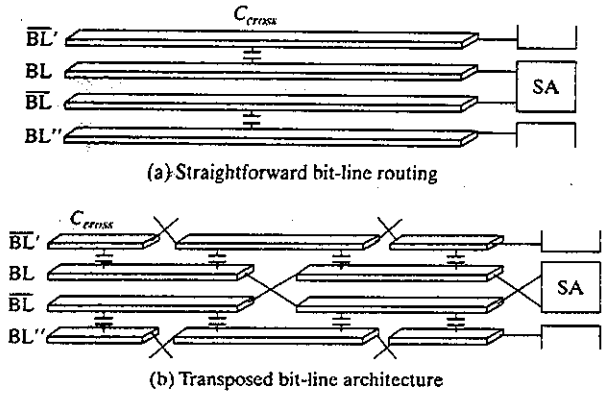


Figure 12-66 bit line-to-bit line coupling.

where V_{swing} is the signal swing on the bit lines. Up to one-fourth of the already weak signal can be lost due to this interference ([Itoh90]).

The *transposed bit line architecture* (Figure 12-66b) eliminates this source of disturbance by dividing the bit lines into segments that are connected in a cross-coupled fashion. This modification presents the interference signal introduced by a neighboring bit line in approximately the same way to both *BL* and $\overline{BL}$, turning it into a common-mode signal. An alternative approach is to use the capacitor-plate layer as an extra shielding between the data lines.

Leakage

Charge leakage is rapidly becoming one of the dominant noise sources in memories. This is no longer true only for dynamic memories, but also for static memories that increasingly feature high-impedance nodes. There are two leakage mechanisms that affect the storage node of a non-selected cell in a 1T DRAM: *pn*-junction leakage to the substrate, and subthreshold current to the bit line. This leakage causes a gradual degradation in the stored signal charge. The minimum bound on the signal-to-noise ratio sets an upper limit on the refresh time t_{REFmax}. Refresh operations consume power and represent a pure overhead from a systems point of view since they reduce the achievable memory bandwidth. Architectural modifications can help reduce the refresh overhead. A typical refresh procedure sequentially toggles each word line, refreshing all the cells connected to that line simultaneously. It is thus advantageous to keep the number of rows in the memory to a minimum. Dividing the memory into multiple blocks furthermore allows for the simultaneous refreshing of multiple rows. Even with these architectural improvements, leakage suppression is becoming more important with every new generation of memories.

Junction leakage can be improved in two ways: (1) by improving the quality of the fabrication process, for instance by changing the doping profile, and (2) by decreasing the ambient junction temperature as junction leakage is a strong function of the temperature. Low-power

techniques reducing heat dissipation and the use of packages with low thermal resistance have proven to be quite effective. Yet, it is projected that the junction leakage should be smaller than 3.5 aA at 25C for a 64-Gb memory!

Keeping the subthreshold leakage at bay is an even larger challenge. The only reasonable solution is to increase the threshold of the access transistor, this when all the other voltages are being reduced! Maintaining sufficient drive current and performance under these conditions is hard. The only plausible solution is to scale down the refresh period more aggressively!

Critical Charge

Early memory designers were puzzled by the occurrence of *soft errors*, that is, nonrecurrent and nonpermanent errors, in DRAMs that could not be explained by either supply-voltage noise, leakage, or cross coupling. The impact of soft errors on system design is enormous. If no adequate protection against them is found, they can cause a computer system to crash in a non-reconstructible way making the task of the system debugger virtually hopeless. In a landmark paper, May and Woods identified *alpha particles* as the main culprits [May79].

Alpha particles are He^{2+} nuclei (two protons, two neutrons) emitted from radioactive elements during decay. Traces of such elements are unavoidably present in the device packaging materials. With an emitted energy of 8 to 9 MeV, alpha particles can travel up to 10 μm deep into silicon. While doing so, they interact strongly with the crystalline structure, generating roughly 2×10^6 electron-hole pairs in the substrate. The soft error problem occurs when the trajectory of one of these particles strikes the storage node of a memory cell. Consider the cell of Figure 12-67. When a 1 is stored in the cell, the potential well is empty. Electrons and holes generated by a striking particle diffuse through the substrate. Electrons that reach the edge of the depletion region before recombining are swept into the storage node by the electrical field. If enough electrons are collected, the stored value can change into a 0. The *collection efficiency* measures the percentage of electrons that make it into the cell.

Recently, it has been established that neutrons originating from cosmic rays are another important source of soft errors. Research has now indicated that the neutron-induced soft error rate (SER) in CMOS SRAMs is of the same order as the alpha-induced SER. Even more, the soft error rate of CMOS latch circuits seems to be dominated by neutrons [Tosaka97].

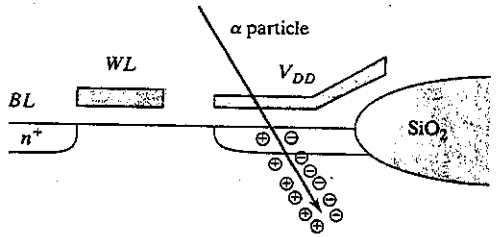


Figure 12-67 Alpha-particle induced soft errors.

The occurrence of soft errors can be reduced by keeping the cell charge larger than a *critical charge* Q_C. This puts a lower bound on the storage capacitance and cell voltage. As an example, a capacitance of 50 fF charge to a potential of 3.5 V holds 1.1×10^6 electrons. A single alpha particle striking such a cell with a collection efficiency of 55% can erase the complete charge. This is one of the main reasons why the cell capacitance of even the densest memories is kept higher than 30 fF. Another approach is to reduce the collection efficiency, which has been observed to be inversely proportional to the diagonal length of the depletion region of the cell. The value of the critical charge thus reduces with technology scaling! For instance, the critical charge equals 30 fC for a diagonal length of 2.5 μm which corresponds to a 64-Mbit memory. Chip coating and purification of materials are indispensable in reducing the number of alpha particles. For instance, a memory die can furthermore be covered with polymide to protect against alpha radiation.[7]

While the occurrence of soft errors can be kept to an absolute minimum by careful design, total elimination is hard to achieve. Free neutrons, originating from cosmic rays hitting the atmosphere, form another source of soft errors. The interaction of one of these neutrons interacting with a silicon nucleus—which is an unlikely, but possible, event—has a major probability of upsetting the stored data, as neutrons generate about 10 times as many charges as alpha particles.

The occasional occurrence of an error is thus hard to avoid, but it should not necessarily be fatal. System-level techniques such as error correction can help detect and correct most failures, and are becoming more and more indispensable in the memories of the future. This is briefly discussed in the next section.

12.4.2 Memory Yield

With increasing die size and integration density, a reduction in yield is to be expected, notwithstanding the improvements in the manufacturing process. Causes for malfunctioning of a part can be both material defects and process variations. Memory designers use two approaches to combat low yields and to reduce the cost of these complex components: redundancy and error correction. The latter technique has the advantage that it also addresses the occasional occurrence of a soft error.

Redundancy

Memories have the advantage of being extremely regular structures. Providing *redundant hardware* is easily accomplished. Defective bit lines in a memory array can be replaced by redundant ones, and the same holds for word lines. (See Figure 12-68.) When a defective column is detected during testing of the memory part, it is replaced by a spare one by programming the fuse bank connected to the column decoder. A typical way of doing so is to blow the fuses using a programming laser or pulsed current. Laser programming has a minimal impact on memory performance and occupies small chip area. It does require special equipment and increased

[7] It is worth mentioning that soft errors are also a problem in state-of-the-art SRAMs that are relying more heavily on capacitive storage on high-impedance nodes.

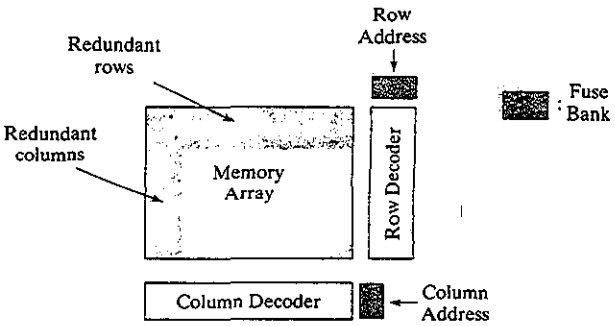


Figure 12-68 Redundancy in memory array increases the yield.

wafer handling time, however. The pulsed current approach can be executed by a standard tester, but bears larger overhead. A similar approach is followed for the defective word lines. Whenever a failing word line is addressed, the word redundancy system enables a redundant word line. In modern memories, as many as over 100 defective elements can be replaced by spare ones for an additional overhead of less than 5%. Even embedded SRAM memories, to be used in systems-on-a-chip, now come with redundancy.

Error Correction

Redundancy helps correct faults that affect a large section of the memory such as defective bit lines or word lines. It is ineffective when dealing with scattered point errors such as local errors caused by material defects. Achieving a reasonable fault coverage requires too much redundancy under these circumstances and results in a large area overhead. A better approach to address these faults is to use *error correction*. The idea behind this scheme is to use redundancy in the data representation so that an erroneous bit(s) can be detected and even corrected. Adding a parity bit to a data word, for instance, provides a way of detecting (but not correcting) an error. While a full discussion of error correction techniques would lead us astray, a simple example is used to illustrate the basic concept.

Example 12.15 Error Correction Using Hamming Codes

Consider a 4-bit number (B_i), encoded with 3 check bits (P_i),

$$P_1 P_2 B_3 P_4 B_5 B_6 B_7$$

with the P_i chosen such that

$$P_1 \oplus B_3 \oplus B_5 \oplus B_7 = 0$$
$$P_2 \oplus B_3 \oplus B_6 \oplus B_7 = 0$$
$$P_4 \oplus B_5 \oplus B_6 \oplus B_7 = 0$$

Suppose now that an error occurs in B_3. This causes the first two expressions to evaluate to 1 while the last one remains at 0. Binary encoded, this means that bit 011 or 3 is in error. This information is sufficient to correct the fault. In general, single error correction requires that

$$2^k \geq m + k + 1 \tag{12.18}$$

where m and k are the number of data and check bits, respectively. To perform single error correction on 64 data bits requires 7 check bits, resulting in a total word length of 71.

An important observation is that error correction not only combats technology-related faults, but is also an effective way of dealing with soft errors and time-variant faults. For instance, error correction is very effective in dealing with threshold variations in EEPROMs.

Error correction and redundancy address different angles of the memory yield problem. To cover all the bases, a combination of both is needed. This is convincingly illustrated in Figure 12-69, which plots the yield percentage for a 16-Mbit DRAM when the yield improvement techniques are used independently or combined [Kalter90].

One other issue is worth elaborating on in this discussion on memory reliability and yield. With the size and performance of memories ever increasing, testing a memory for defects takes ever more time on ever-more expensive testers. The situation is even more astute for embedded memories, as the memory ports may not be directly accessible through the I/O pins of the chip. A common approach to deal with this problem is to have the memory test itself. In this built-in self-test (BIST) approach, a small controller is added to the memory that generates input patterns, and verifies if the responses of the memory are correct. The BIST approach is very attractive: it eliminates the cost of the expensive tester, it can be performed at speed, and can be executed every time the chip is started up. More information on BIST can be found in Design Methodology Insert H at the end of this chapter.

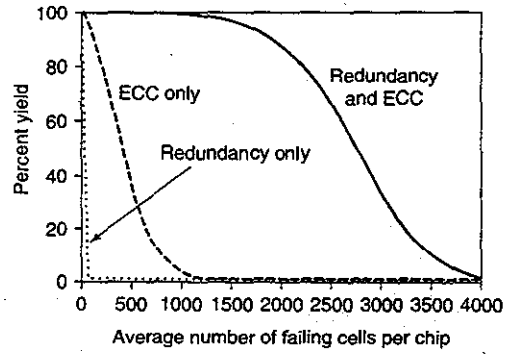


Figure 12-69 Yield curves for 16-Mbit DRAM using ECC and bit line redundancy [Kalter90].

12.5 Power Dissipation in Memories*

As in most of the arena of digital design, reduction of the power dissipation in memories is becoming of premier importance. Memory designers have been remarkably good in keeping dissipation in check, even while increasing the memory capacity with 6 orders of magnitude over the last 30 years. Yet, the challenges are mounting. Portable applications are lowering the bar on how much power memory may consume. At the same time, technology scaling with its reduction in supply and threshold voltages and its deterioration of the off-current of the transistor causes the standby power of the memory to rise. The introduction of innovative techniques is truly a necessity.

12.5.1 Sources of Power Dissipation in Memories

The power consumption in a memory chip can be attributed to three major sources: the memory cell array, the decoders (row, column, block), and the periphery. A unified active power equation for a modern CMOS memory of m columns and n rows is approximately given by [Itoh01] (see Figure 12-70). For a normal read cycle:

$$\begin{aligned} P &= V_{DD} I_{DD} \\ I_{DD} &= I_{array} + I_{decode} + I_{periphery} \\ &= [m i_{act} + m(n-1) i_{hld}] + [(n+m) C_{DE} V_{int} f] + [C_{PT} V_{int} f + I_{DCP}] \end{aligned} \tag{12.19}$$

The equation uses the following design parameters:

- i_{act}—the effective current of the selected (or active) cells
- i_{hld}—the data retention current of the inactive cells

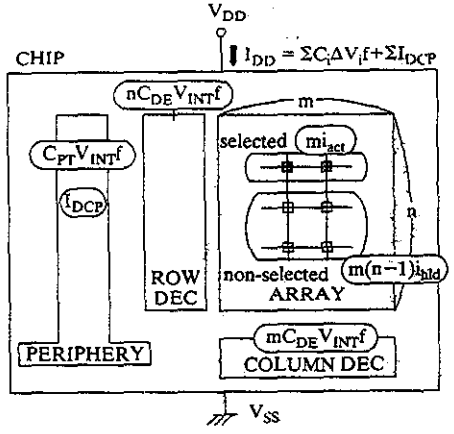


Figure 12-70 Sources of power dissipation in semiconductor memory [Itoh01].

- C_{DE}—the output node capacitance of each decoder
- C_{PT}—the total capacitance of the CMOS logic and periphery circuits
- V_{int}—the internal supply voltage
- I_{DCP}—the static (or quasi-static) current of the periphery. The major sources for this current are the sense amplifiers and the column circuitry. Other sources are the on-chip voltage generators.
- f—the operating frequency

As should be expected, the power dissipation is proportional to the size of the memory (n,m). Dividing the memory into subarrays, and keeping n and m small is essential to keep power within bounds. This approach is obviously only effective if the standby dissipation of inactive memory modules is substantially lower.

In general, the power dissipation of the memory is dominated by the array. The active power dissipation of the peripheral circuits is small compared with the other components. Its standby power can be high however, requiring that circuits such as sense amplifiers are turned off when not in action. The decoder charging current is also negligibly small in modern RAMs, especially if care is taken that only one out of the n or m nodes is charged at every cycle.

Reduction of power dissipation in memories is worth a chapter on its own. We limit ourselves to an enumeration of some techniques that are generally applicable, and refer the interested reader to [Itoh01] for more details and depth.

12.5.2 Partitioning of the Memory

A proper division of the memory in submodules goes a long way in confining active power dissipation to limited areas of the overall array. Memory units that are not in use should consume only the power necessary for data retention. Memory partitioning is accomplished by reducing m (the number of cells on a word line), and/or n (the number of cells on a bit line). By dividing the word line into several sub-word-lines that are enabled only when addressed, the overall switched capacitance per access is reduced. In some sense, this scheme is nothing more than a multistage hierarchical row decoder. This approach is quite popular in SRAM memories, as illustrated in the 4-Mbit SRAM memory discussed in the case studies of the next section.

In a similar way, partitioning of the bit line reduces the capacitance switched at every read/write operation. An approach that is often used in DRAM memories is the partially activated bit line. The bit line is partitioned in multiple sections (typically 2 or more). All these sections share a common sense amplifier, column decoder, and I/O module. This approach has helped to reduce CBL from more than 1 pF for the 16-KBit DRAM generation to approximately 200 fF for the 64-MBit DRAM.

12.5.3 Addressing the Active Power Dissipation

Similar to what we learned for logic circuits, reducing the voltage levels is one of the most effective techniques to reduce power dissipation in memories. In contrast to logic, however, voltage

scaling might run out of steam earlier. Data retention and reliability issues make the scaling of the voltages to the deep sub-1-V level quite a challenge. In addition, careful control of the capacitance and switching activity, as well as minimization of the on time of the peripheral components is essential.

SRAM Active Power Reduction

To obtain a fast read, the voltage swing on the bit line is made as small as possible—typically between 0.1 V and 0.3 V. The resulting signal is transmitted to the sense amplifier for restoration. Since the signal is developed as a result of the ratio operation of the bit line load and the cell transistor, a current flows through the bit line as long as the word line is activated (Δt). Limiting Δt and the bit line swing helps to keep the active dissipation of SRAMs low. Signal-to-noise issues ultimately determine how small the bit line swing and the on time of the sense amplifier can be made. Self-timing can be a big help in determining exactly how long to keep the peripheral circuitry on.

The situation is worse for the write operation, since BL and $\overline{BL}$ have to make a full excursion. Reduction of the core voltage is the only remedy for this (in addition, of course, to the bit line partitioning approach described earlier). Ultimately, the reduction of the core voltage is limited by the mismatch between the paired MOS transistors in the SRAM cell. Even when designed to be completely identical, process and device variations cause the MOS transistors in the cell to be different. The ever-increasing V_T threshold mismatch is the most important culprit. It is caused by implant nonuniformities, channel length and width variations, and even random microscopic fluctuations of dopant atoms in very small devices. Even if process engineers manage to keep the threshold variation constant, its relative importance is rapidly increasing in light of decreasing supply and threshold voltages. The mismatch between the transistors causes the cell to become asymmetrical, biasing it toward either the 1 or 0 state. This makes the cell substantially less reliable in the presence of noise and during read operations. Stringent control of the MOSFET characteristics, either at process time or at run time using techniques such as body biasing, is essential in the low voltage operation mode. If this cannot be accomplished, extra redundancy and error correction is needed.

Reducing the supply voltage also impacts the memory access time. Lowering the device thresholds is only an option if the resulting increase in leakage current is dealt with effectively, especially in the nonactive cells (see *reduction of data-retention power*).

DRAM Active Power Reduction

The destructive readout process of a DRAM cell necessitates successive operations of readout, amplification and restoration for the selected cells. Consequently, the bit lines are charged and discharged over the full voltage swing (ΔV_{BL}) for every read operation. Care should thus be given to reduce the bit line dissipation charge $mC_{BL}\Delta V_{BL}$, since it dominates the active power. Reducing C_{BL} (the bit line capacitance) is advantageous from both a power and a signal-to-noise perspective (Eq. (12.8)), but is not simple given the trend toward larger memories. Reducing ΔV_{BL}, while very beneficial from a power perspective, negatively impacts the signal-to-noise

ratio. This is apparent from Eq. (12.8), which shows that the charge stored in the cell—and hence the readout signal—is directly proportional to the voltage swing applied during the write or the refresh cycle. Voltage reduction thus has to be accompanied by either an increase in the size of the storage capacitor and/or a noise reduction. A number of techniques have proven to be quite effective:

- **Half-V_{DD} precharge**—Precharging the bit lines to $V_{DD}/2$ helps to reduce the active power dissipation in DRAM memories by a factor of almost 2. After amplification and restoration of the readout voltages on the bit lines, precharging is accomplished by a simple shorting of the two bit lines. Assuming that the two bit lines are balanced, this results in an exact voltage of $V_{DD}/2$.
- **Boosted word line**—Raising the value of the word line above V_{DD} during a write operation eliminates the threshold drop over the access transistor, yielding a substantial increase in stored charge.
- **Increased capacitor area or value**—Vertical capacitors such as those used in trench and stacked cells are very effective in increasing the capacitance value, albeit at a major cost in processing and manufacturing. In addition, keeping the "ground" plate of the storage capacitor at $V_{DD}/2$ reduces the maximum voltage over C_s, making it possible to use thinner oxides.
- **Increasing the cell size**—Ultimately, ultra-low-voltage DRAM memory operation might require a sacrifice in area efficiency, especially for memories that are embedded in a system-on-a-chip. At that juncture, however, it becomes worthwhile exploring whether cells that provide gain, such as the 3T DRAM, offer a better alternative.

12.5.4 Data-Retention Dissipation

Data Retention in SRAMs

In principle, an SRAM array should not have any static power dissipation (i_{ldd}). Yet, the leakage current of the cell transistors is becoming a major source of retention current. While this current historically has been kept low, a considerable increase has been observed in recent embedded memories due to subthreshold leakage. This is illustrated in Figure 12-71, which plots the standby current of the same 8-kBit embedded memory, implemented in a 0.18-μm and a 0.13-μm CMOS process. An increase of the static current with a factor of almost 7 for the same supply voltage can be observed.

Techniques to reduce the retention current of SRAM memories are thus necessary, including the following:

- **Turning off unused memory blocks.** Memory functions such as caches do not fully use the available capacity for most of the time. Disconnecting unused blocks from the supply rails using high-threshold switches reduces their leakage to very low values. Obviously, the data stored in the memory is lost in this approach.

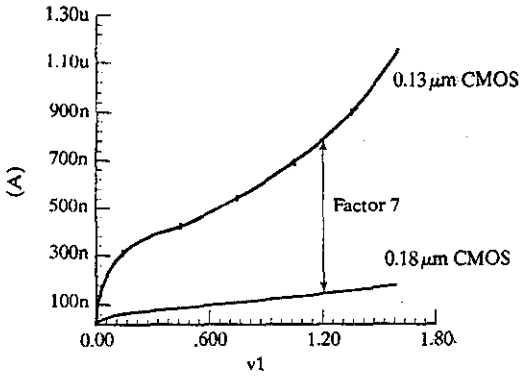


Figure 12-71 Leakage current in embedded eight-kbit SRAM memory as a function of voltage.

- **Increasing the thresholds by using body biasing.** Negative bias of the nonactive cells increases the thresholds of the devices and reduces the leakage current.
- **Inserting extra resistance in the leakage path.** When data retention is necessary, the insertion of a low-threshold switch in the leakage path provides a means to reduce leakage current while keeping the data intact (see Figure 12-72a). While the low-threshold device leaks on its own, which is sufficient to maintain the state in the memory. At the same time, the voltage drop over the switch introduces a "stacking effect" in the memory cells connected to it: A reduction of V_{GS} combined with a negative V_{BS} results in a substantial drop in leakage current.
- **Lowering the supply voltage.** Figure 12-71 indicates that the leakage current is a strong function of V_{DD}. An effective means to reduce leakage during standby is to lower the

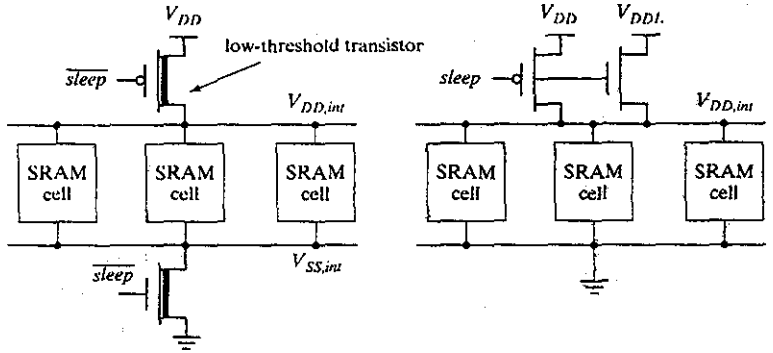


Figure 12-72 Standby leakage suppression techniques for SRAM memories: (a) inserting extra resistance; (b) lowering the supply voltage. (V_{DDL} can be as low as 100 mV.)

supply rails to a value that keeps the leakage within bounds, while ensuring data retention (see Figure 12-72b). The lower bound of the data retention voltage—the voltage that still maintains the stored value—is determined by device variations. A supply voltage as low as 100 mV (excluding noise margin) is sufficient to maintain retention in a standard 0.13-μm CMOS process. The combined effect of reduced supply voltage and leakage current is a powerful way of addressing standby power dissipation in SRAM memories.

DRAM Retention Power

Standby power in DRAMs is attributed to leakage, just as in SRAM memories. This is where the similarity ends, however. To combat leakage and loss of signal, DRAMs have to be refreshed continuously when in data-retention mode. The refresh operation is performed by reading the m cells connected to a word line and restoring them. This operation is repeated for each of the n word lines in sequence. The standby power is therefore proportional to the bit line dissipation charge, defined earlier, and the *refresh frequency*. The latter is a strong function of the leakage rate. Keeping the junction temperatures low is an effective means to keeping leakage within bounds.

Yet, the shrinking of the cell sizes and the charge stored in the cell, combined with a reduction in voltages, is forcing the refresh frequency upward, and causing the standby power of DRAMs to increase. This is reflected in Figure 12-73, which plots the evolution of the active and standby currents for different DRAM generations. The data retention current is overtaking the active current at the 1-Gbit DRAM generation and will be the dominant source of power dissipation unless some leakage suppression techniques are introduced.

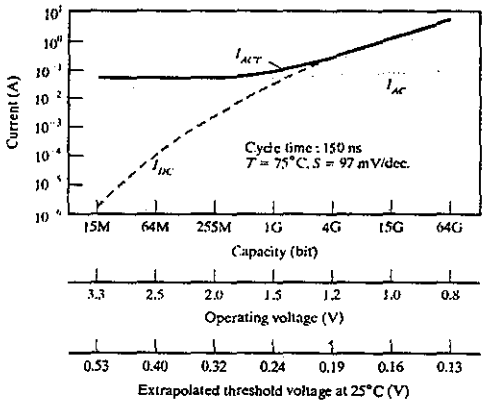


Figure 12-73 Estimated current distribution for consecutive DRAM generations (from [Itoh01]).

The secret to leakage minimization in DRAM memories is V_T control. This can be accomplished at design time (the fixed V_T approach), or dynamically (the variable V_T technique). One option to reduce leakage through the access transistor in the DRAM cell is to turn off the device hard by applying a negative voltage ($-\Delta V_{WL}$) to the word line of nonactive cells. Alternatively, we can raise the bit line voltage of unused cells by a certain amount ($+\Delta V_{BL}$). This approach, called the *boosted sense ground*, results in a negative gate-source voltage and a slight increase in the threshold voltage.

The variable V_T techniques raise the thresholds of the access transistors in inactive cells through well biasing. Control can be performed at different levels of granularity (chip, module, and individual transistor) with increasing levels of overhead. Careful attention should be paid to the generation of the well-voltages, as this can lead to instabilities.

12.5.5 Summary

SRAM memories have a distinctive advantage over DRAM from a power perspective. Yet, the increasing impact of leakage combined with reduced supply voltages might reduce this gap to a certain extent. When standby power is a dominant concern, nonvolatile memories offer a viable and attractive alternative.

12.6 Case Studies in Memory Design

While the previous sections have introduced the individual components of a semiconductor memory, it is worthwhile to study a number of examples to understand how it all comes together. We will use two case studies to achieve that goal—the programmable logic array and a 4-Mbit SRAM memory.

12.6.1 The Programmable Logic Array (PLA)

In our discussion on design implementation strategies in Chapter 8, we introduced the concept of the Programmable Logic Array (or PLA). The PLA became popular in the early 1980s as a regular approach for the implementation of random logic. (See "Historical Perspective: The Programmable Logic Array" on page 388.)

Now follows the reason why the issue of two-level logic implementation is raised in this chapter on memories. When discussing the ROM memory core, we realized that the ROM array is nothing more than a collection of large fan-in NOR (NAND) gates implemented in a very regular format. Similarly, the address decoder is also a collection of NOR (NAND) gates. The same approach can thus be used to implement any two-level logic function—hence the PLA. The only difference between a ROM and PLA is that the decoder (AND-plane) of the former enumerates all possible minterms (m inputs yield 2^m minterms), while the AND-plane of the latter only realizes a limited set of minterms, as dictated by the logic equations. Topologically, both structures are identical.

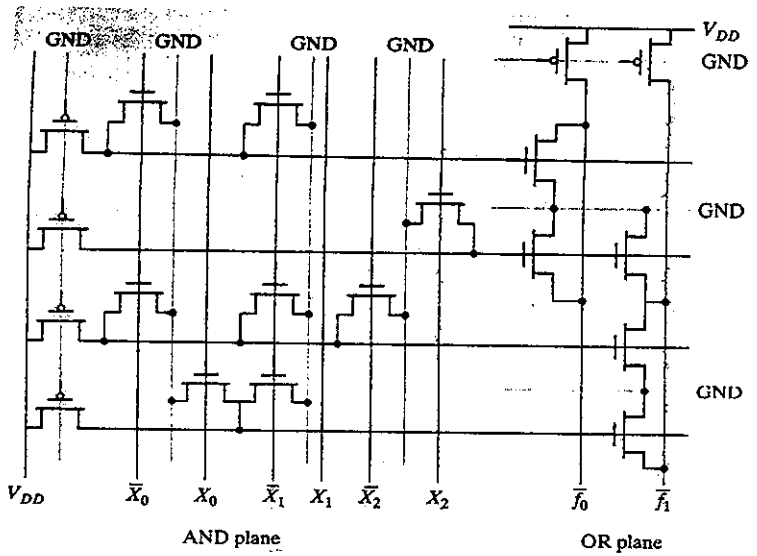


Figure 12-74 Pseudo-NMOS PLA.

A NOR–NOR implementation of a PLA using the pseudo-NMOS circuit style is shown in Figure 12-74. Although compact and fast, its power dissipation makes this realization unattractive for larger PLAs where a dynamic approach is better. As we know, a direct cascade of the dynamic planes is out of the question. Solutions could be to introduce an inverter between the planes in the DOMINO style, or implement the OR-plane with PMOS transistors and use predischarging in the *np*-CMOS fashion. The former solution causes pitch-matching problems, while the latter slows down the structure if minimum-size PMOS devices are used.

We will take this opportunity to examine the memory-timing issue more thoroughly. Figure 12-75 uses a more complex clocking scheme to resolve the plane-connection problem [Weste93]. Assume that the pull-up transistors of the AND plane and the OR plane are clocked by signals ϕ_{AND} and ϕ_{OR}, respectively. These signals are defined so that a clock cycle starts with the precharging of both planes. After a time interval long enough to ensure that precharging of the AND plane has ended, the AND plane starts to evaluate by raising ϕ_{AND}, while the OR plane remains disabled (see Figure 12-76a). Once the AND plane outputs are valid, it is safe to enable the OR plane by raising ϕ_{OR}.

The timing of these events is precarious, since it depends upon the PLA size and programming and technology variations. Although the clock signals can be provided externally, self-timing is the recommended approach to achieve maximum performance at the minimum risk. How to derive the appropriate timing signals from a single clock ϕ is shown in Figure 12-76b. The

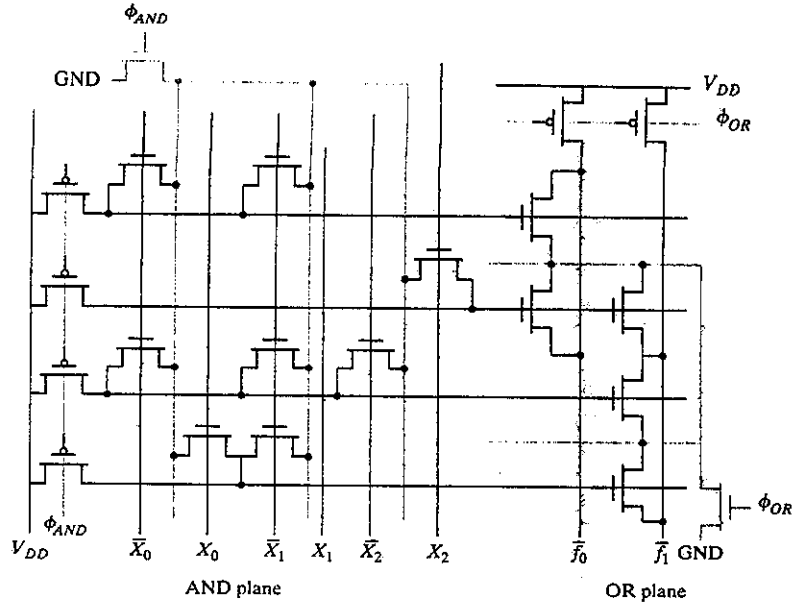


Figure 12-75 Dynamic implementation of PLA.

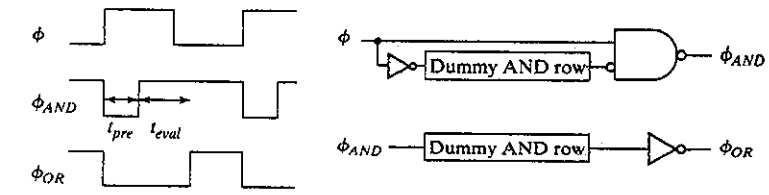


Figure 12-76 Generation of clock signals for self-timed dynamic PLA. t_{pre} and t_{eval} represent the worst case precharge and evaluation times of an AND row, respectively. These are obtained with the aid of dummy AND rows, which are fully populated; all transistors in the NOR network except one are turned off. This represents the worst case discharge scenario.

clock signal ϕ_{AND} is derived from ϕ using a monostable one-shot. The delay element consists of a dummy AND row with maximum loading—a fully populated row that provides the worst case estimate of the required precharge time. ϕ_{OR} is derived in a similar way by using a dummy AND row clocked by ϕ_{AND}. This provides a worst case estimate of the discharge time, including some extra safety margin. Although it requires additional logic, this approach ensures reliable operation of the array under a range of conditions.

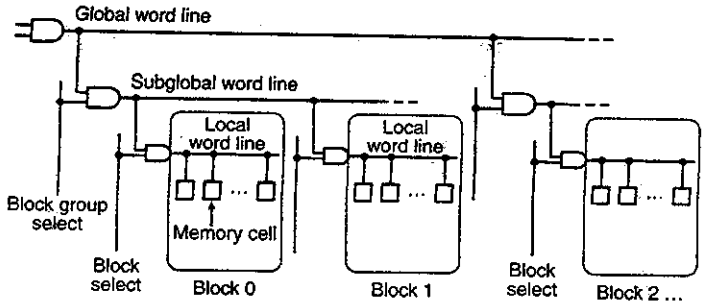


Figure 12-77 Hierarchical word-line selection scheme.

12.6.2 A 4-Mbit SRAM

In Figure 12-6, we have shown the block diagram of a 4-Mbit SRAM memory [Hirose90]. This memory has an access time of 20 ns for a single supply voltage of 3.3 V and is fabricated in a quadruple polysilicon, double metal 0.6-μm CMOS technology. The memory is organized as 32 blocks, each block counting 1024 rows and 128 columns. The row address (*X*), column address (*Y*), and block address (Z) are 10, 7, and 5 bits wide, respectively.

To provide fast and low-power row decoding, the memory uses an interesting approach called the *hierarchical word-decoding* scheme, shown in Figure 12-77. Instead of broadcasting the decoded *X* address to all blocks in polysilicon, it is distributed in metal and called the *global word line*. The local word line is confined to a single block and is only activated when that particular block is selected by using the block address. To further improve the performance and power consumption, an extra word-line hierarchy level called the *subglobal word line* is introduced. This approach results in a row-decoding delay of only 7 ns.

The bit line peripheral circuitry is presented in Figure 12-78. The memory cell is a four-transistor resistive-load cell occupying 19 μm^2. The load resistance equals 10 TΩ. A triggering of the *ATD* pulse causes the precharging and the equalizing of the bit lines of the selected block with the aid of the *BEQ* control signal. Notice that the bit line load is a composition of static and dynamic devices. Lowering *BEQ* starts the read process. One of the word lines is enabled, and the appropriate bit lines start to discharge. These are connected to the sense amplifiers after passing through the first layer of the column decoder (*CD*). Only 16 sense amplifiers are needed per block of 128 columns in this scheme. The amplifier consists of two stages (see Figure 12-78b). The first is a cross-coupled stage that provides a minimal gain, but also acts as a level shifter. This permits the second amplifier, which is of the current-mirror type, to operate at the maximum-gain point. A push–pull output stage drives the highly capacitive data lines that lead to the tristable input/output drivers. The approximative signal waveforms are displayed in Figure 12-78c. These demonstrate the subtlety of the timing and clocking strategy necessary to operate a large memory at high speed. To write into the memory, the appropriate value and its complement are imposed on the *I/O* and $\overline{I/O}$ lines, after which the appropriate row, column, and block addresses are enabled. It is worth

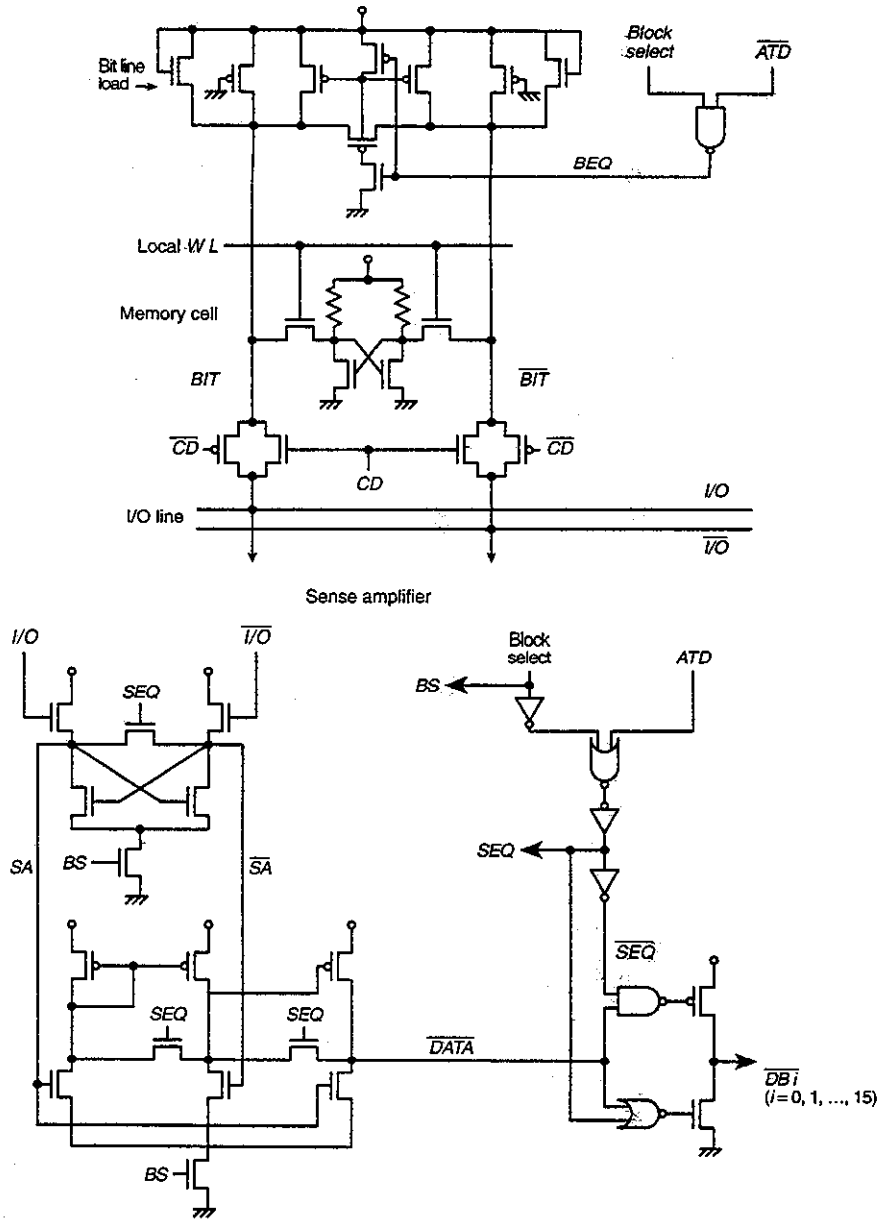


Figure 12-78 The bit line peripheral circuitry and the associated signal waveforms.

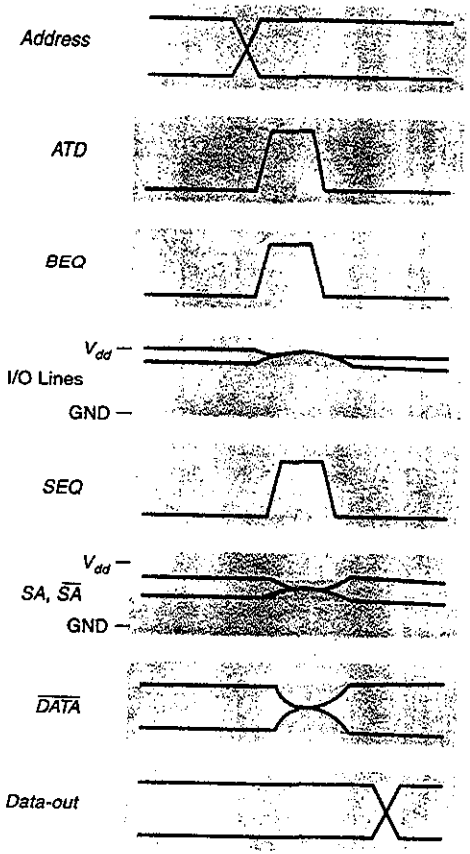


Figure 12-78 *(cont.)*

mentioning that the memory consumes only 70 mA when operated at 40 MHz. The standby current equals 1.5 μA.

12.6.3 A 1-Gbit NAND Flash Memory

The block diagram of a 1-Gb NAND Flash memory [Nakamura02] is shown in Figure 12-79. The memory is structured as two blocks of 500 Mb each. A NAND organization in the style of Figure 12-25 with 32 bits/block was chosen as the basic memory module, mainly for cell size considerations. Each bit line connects to 1024 of these modules. A block of 500 Mb combines 16,895 of these bit lines in parallel. The word lines are driven from both sides of the block. The page size of the memory, this is the number of bits that can be read/written in one cycle, equals 2 KByte. The large size of the page enables the high programming rate of 10 MByte/s. To further

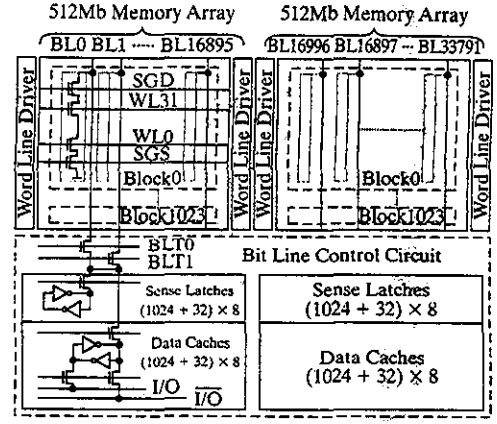


Figure 12-79 Chip architecture of 1-Gb Flash memory.

speed up the programming, an extra "cache" cell is provided per bit line, which allows for new data to be read into the memory, while the previous data is being written/verified.

Figure 12-80 illustrates the erasure/writing process of the Flash memory. During erasure, all bits in a single block are programmed to become a depletion device. During write, the threshold of selective devices is raised by using a programming/verify cycle. It takes at least four of these cycles for all of the thresholds to be above 0.8 V, which is the necessary requirement for the device to act as a correct switch (for the nominal word-line voltages). The distribution of the thresholds after the four cycles is plotted in Figure 12-80b.

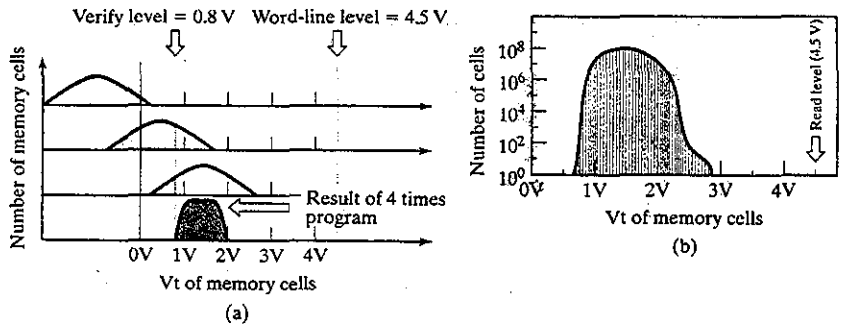
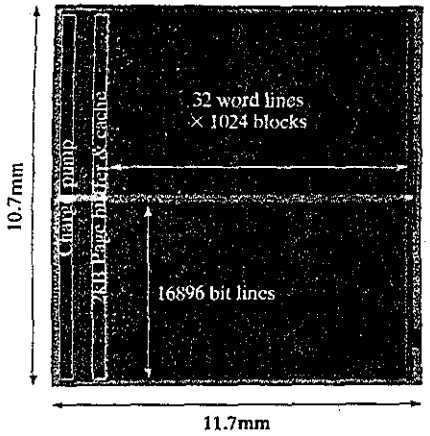


Figure 12-80 Evolution of cell threshold during writing process (a) and final threshold distribution (b) after four programming cycles.



Characteristics	
Cell Size	0.077 μm^2
Chip Size	125.2 mm^2 in 0.13 μm CMOS
Organization	2112 × 8b × 64 page × 1kb block
Power Supply	2.7-3.6 V
Cycle Time	50 ns
Read Time	25 μs
Program Time	200 μs
Erase Time	2 ms/block

Figure 12-81 Die microphotograph and chip characteristics of 1-Gb Flash memory [Nakamura02].

The chip microphotograph of the memory module and an enumeration of its characteristics are shown in Figure 12-81. The memory is implemented in a 0.13-μm CMOS triple well process with 1 poly, 1 polycide, 1 W, and 2 Al layers.

12.7 Perspective: Semiconductor Memory Trends and Evolutions

To conclude this chapter, some evolutionary trends are worth discussing. The first semiconductor memories introduced in the 1960s used bipolar technology and could store no more than 100 bits. A major breakthrough was achieved in the early 1970s when the first MOS DRAM memory was introduced with a capacity of 1 KBit ([Hoff70]). This was the beginning of a dramatic evolution. Currently, the 1-Gbit memory is here. This means that in 30 years, the amount of memory that can be integrated on a single die has increased by six orders of magnitude! It is fair to say that the number of memory cells that can be integrated on a single die quadruples approximately every three years (which generally corresponds to a memory generation). An overview of the trends in capacity for various memory styles is shown in Figure 12-82.

It is interesting to understand what gave rise to these spectacular improvements. Figure 12-83 plots the evolution of the cell sizes over the subsequent generations. Both DRAM and SRAM cells have been miniaturized at a pace of about one-fiftieth every 10 years (or a factor of somewhat higher than 3 every 3 years). In addition, die size has increased with a factor of 1.4 every generation. Combining the two factors yields the increase in density by a factor of around 4 between generations. One would be inclined to attribute the reduction in cell size solely to technology scaling, but this is only partially true. For instance, the L_{eff} of the transistors used in the 4-K SRAM memory equaled 3 μm and has decreased from generation to generation so that the 64-

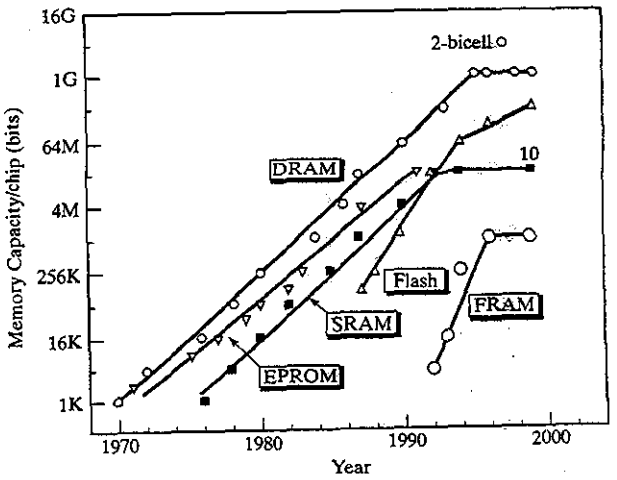


Figure 12-82 Trends in the memory capacity of VLSI memories [Itoh01].

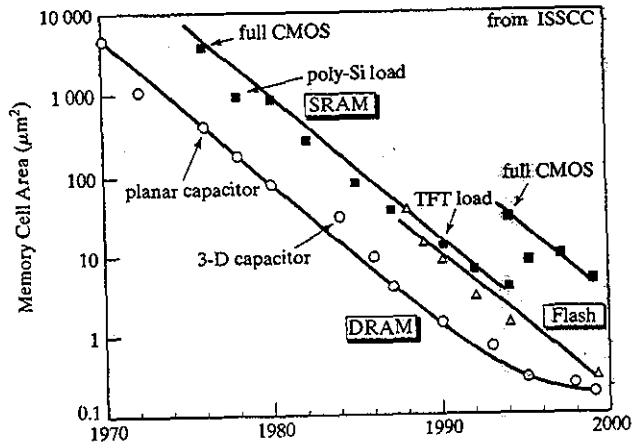


Figure 12-83 Trends in memory cell area for VLSI memories [Itoh01].

M SRAM uses transistors with an effective length of 0.25 μm. This suggests that scaling reduces the cell size by a factor of 1.5 × 1.5 = 2.25 between generations, which is not sufficient to account for the observed trends. Additional reductions are obtained by an ever-increasing sophistication in memory cell technology. Important events in this category include the introduction of the polysilicon gate, denser isolation using LOCOS and U-groove, polysilicon load

resistors for SRAM cells, and three-dimensional DRAM cells using trench or stacked capacitors. The continuous introduction of such novelties is essential if the current trends in integration are to be prolonged.

This might not be trivial, as some major challenges and roadblocks have become apparent in recent years. This is already visible in Figure 12-82, which shows a distinctive saturation in the capacity of SRAM and DRAM memories at the end of the 1990s. In fact, Flash memory is about to pull even with DRAM as we speak.This saturation is mainly due to the extreme challenge and cost of further cell miniaturization. This is not the only cause, however, and other considerations should be taken into account. We briefly enumerate a number of issues that impact the further integration of semiconductor memories.

- High-density memories need very specialized technologies that are extremely differentiated from logic technologies. Their development is becoming more and more of a financial hazard.
- Maintaining correct operation of SRAM and DRAM memories in the presence of reduced supply and threshold voltages is very hard. Effects of leakage and soft errors are posing a challenge that has been unanswered so far. Important innovations in the memory cell ate necessary. Novel approaches such as nonvolatile FRAMs and MRAMs might offer an alternative.
- As stated in an earlier section, power consumption is becoming a limiting factor on how much memory can be integrated on a single die.
- But most importantly, a shift in business climate has emerged. The unprecedented hunger for more memory was mostly driven by the computer industry. With the shift of the center of mass of the semiconductor industry toward consumer, multimedia, and communications applications, different needs in terms of memory have arisen. For instance, mobile semiconductor video and audio players feed the need for nonvolatile memory. Also, the size and cost constraints of these applications make integrated system-on-a-chip solutions desirable. Most of the memory should be embedded on the die, together with the processor and logic components. This drives the needs for memory technologies that are compatible with standard logic processes.

12.8 Summary

In this chapter, we have discussed the design of semiconductor memories in extensive detail. While the art of designing memories is vastly different from the traditional logic world, it is obvious that many of the problems addressed by memory designers are similar to those encountered by logic designers. Moreover, with the advent of Systems-on-a-Chip, embedded memories are becoming a necessary component of virtually any complex design. Finally, due to the fact that memory design operates at the frontier of technological ability, such as the smallest device dimensions, many of the problems and solutions considered in contemporary memory design might very well surface in logic design at a later time. As an example, power dissipation is one

of the major challenges facing high-density memory designers. Low-voltage operation around 1.0 V or below is one of the solutions being employed in state-of-the-art memories. Other approaches include self-timing and local power-down. Most of these techniques can be applied to the logic design world as well.

In conclusion, the memory-design problem can be summarized as follows:

- The *memory architecture* has a major impact on the ease of use of the memory, its reliability and yield, its performance, and, power consumption. Memories are organized as arrays of cells. An individual cell is addressed by a block, column, and row address.
- The *memory cell* should be designed so that a maximum signal is obtained in a minimum area. While the cell design is mostly dominated by technological considerations, a clever circuit design can help maximize the signal value and transient response. The cell topologies have not varied much over the last decades. Most of the improvement in density results from technology scaling and advanced manufacturing processes. We discussed cells for read-only, nonvolatile, and read–write memories.
- The *peripheral circuitry* is essential to operate the memory in a reliable way and with a reasonable performance, given the weak signaling characteristics of the cells. Decoders, sense amplifiers, and I/O buffers are an integral part of every memory design. The discussion presented in this chapter only touched the surface. Besides the mentioned functions, other circuitry in the periphery performs functions such as bootstrapping, voltage generation in EEPROMs, and voltage regulation ([Itoh01]). Although interesting and challenging, the discussion of these circuits would have lead us too far. Suffice it to say that the design of memory periphery is still the playground of the hard-core digital circuit designer.
- A memory must operate correctly over a variety of operating and manufacturing conditions. To increase the integration density, memory designers give up on signal-to-noise ratio. This makes the design vulnerable to a whole range of noise signals, which are normally less of an issue in logic design. Identifying the potential sources of malfunction and providing an appropriate model is the first requirement when addressing *memory reliability*. Circuit precautions to deal with potential malfunctions include redundancy and error correction.

12.9 To Probe Further

A vast amount of literature on memory design is presented in the leading circuit design journals. Most ground-breaking innovations in memory design are reported in either the *Journal of Solid State Circuits* (November issue in particular), the ISSCC conference proceedings, and the VLSI circuit symposium.Yet, until recently, not many textbooks were available on this topic. A number of interesting tutorial papers have been published over the years, some of which are included in the reference list that follows. Fortunately, a couple of great books on semiconductor memory have finally been published. [Itoh01] and [Keeth01] provide a wealth of background information for the memory aficionado.

References

[Amrutur01] B.S. Amrutur, M.A. Horowitz, "Fast Low-Power Decoders for RAMs," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 10, pp. 1506–1515, October 2001.

[Baur95] M. Baur et al., "A Multilevel-Cell 32-Mb Flash Memory," *ISSCC Digest of Technical Papers*, pp. 13–33, February 1995.

[Chang77] J. Chang, "Theory of MNOS Memory Transistor," *IEEE Trans. Electron Devices*, ED-24, pp. 511–518, 1977.

[Crisp97] R. Crisp, "Direct Rambus Technology: The New Main Memory Standard," *IEEE Micro*, pp. 18–28, November/December 1997.

[Dennard68] R. Dennard, "Field-effect transistor memory," U.S. Patent 3 387 286, June 1968.

[Dillinger88] T. Dillinger, *VLSI Engineering*, Chapter 12, Prentice Hall, 1988.

[Frohman74] D. Frohman, "FAMOS—A New Semiconductor Charge Storage Device," *Solid State Electronics*, vol. 17, pp. 517–529, 1974.

[Gray01] P. Gray, P. Hurst, S. Lewis, and R. Meyer, *Analysis and Design of Analog Integrated Circuits*, 4th ed., John Wiley and Sons, 2001.

[Heller75] L. Heller et al., "High-Sensitivity Charge-Transfer Sense Amplifier," *Proceedings ISSCC Conf.*, pp. 112–113, 1975.

[Hennessy02] J. Hennessy, D. Patterson, and D. Goldberg, *Computer Architecture—A Quantitavie Approach*, 2d ed, Morgan Kaufman Publishers, 2002.

[Hidaka92] H. Hidaka et al., "A 34-ns 16-Mb DRAM with Controllable Voltage-Down Converter," *IEEE Journal of Solid State Circuits*, vol. 27, no. 7, pp. 1020–1027, July 1992.

[Hirose90] T. Hirose et al., "A 20-ns 4-Mb CMOS SRAM with Hierarchical Word Decoding Architecture," *IEEE Journal of Solid State Circuits*, vol. 25, no. 5, pp. 1068–1074, October 1990.

[Hoff70] E. Hoff, "Silicon-Gate Dynamic MOS Crams 1024 Bits on a Chip," *Electronics*, pp. 68–73, August 3, 1970.

[Itoh90] K. Itoh, "Trends in Megabit DRAM Circuit Design," *IEEE Journal of Solid State Circuits*, vol. 25, no. 3, pp. 778–798, June 1990.

[Itoh01] K. Itoh, *VLSI Memory Chip Design*, Springer-Verlag, 2001.

[Johnson80] W. Johnson et al., "A 16 Kb Electrically Erasable Nonvolatile Memory," *ISSCC Digest of Technical Papers*, pp. 152–153, February 1980.

[Kalter90] H. Kalter et al., "A 50-ns 16-Mb DRAM with a 10-ns Data Rate and On-Chip ECC," *IEEE Journal of Solid State Circuits*, vol. 25, no. 5, pp. 1118–1128, October 1990.

[Keeth01] B. Keeth and R. Baker, *DRAM Circuit Design—A Tutorial*, IEEE Press, 2001.

[Lu89] N. Lu, "Advanced Cell Structures for Dynamic RAMs," *IEEE Circuits and Devices Magazine*, pp. 27–36, January 1989.

[Mano87] T. Mano et al., "Circuit Technologies for 16-Mbit DRAMs," *ISSCC Digest of Technical Papers*, pp. 22–23, February 1987.

[Masuoka91] F. Masuoka et al., "Reviews and Prospects of Non-Volatile Semiconductor Memories," *IEICE Transactions*, vol. E74, no. 4, pp. 868–874, April 1991.

[May79] T. May and M. Woods, "Alpha-Particle-Induced Soft Errors in Dynamic Memories," *IEEE Transactions on Electron Devices*, ED-26, no. 1, pp 2–9, January 1979.

[Mot91] Motorola, "Memory Device Data," Specification Data Book, 1991.

[Nakamura02] H. Nakamura et al., "A 125-mm2 1-Gb NAND Flash Memory with 10-Mb/s Program Troughput," *ISSCC Digest of Technical Papers*, pp. 106-107, February 2002.

[Ootani90] T. Ootani et al., "A 4-Mb CMOS SRAM with PMOS Thin-Film-Transistor Load Cell," *IEEE Journal of Solid State Circuits*, vol. 25, no. 5, pp. 1082–1092, October 1990.

[Pashley89] R. Pashley and S. Lai, "Flash Memories: The Best of Two Worlds," *IEEE Spectrum*, pp. 30–33, December 1989.

[Preston01] R.P. Preston, "Register Files and Caches," in A. Chandrakasan, W.J. Bowhill and F. Fox (eds.), *Design of High-Performance Microprocessor Circuits*, Piscataway, NJ: IEEE Press, 2001.

[Regitz70] W. Regitz and J. Karp, "A Three-Transistor Cell, 1,024-bit 500 ns MOS RAM," *ISSCC Digest of Technical Papers*, pp 42–43, 1970.

[Sasaki90] K. Sasaki et al., "A 23-ns 4-Mb CMOS SRAM with 0.2-mA Standby Current," *IEEE Journal of Solid State Circuits*, vol. 25, no. 5, pp. 1075–1081, October 1990.

[Scheuerlein00] R. Scheuerlein et al., "A 10-ns Read-and-Write Non-Volatile Memory Array using a Magnetic Tunnel Junction and FET Switch in Each Cell," *ISSCC Digest of Technical Papers*, pp. 128–129, February 2000.

[Sedra87] A. Sedra and K. Smith, *Microelectronic Circuits*, 2d ed., Holt, Rinehart and Winston, 1987.

[Snow67] E. Snow, "Fowler-Nordheim Tunneling in SiO_2 Films," *Solid State Communications*, vol. 5, pp. 813–815, 1967.

[Taguchi91] M. Taguchi et al., "A 40-ns 64-Mb DRAM with 64-b Parallel Data Bus Architecture," *IEEE Journal of Solid State Circuits*, vol. 26, no. 11, pp. 1493–1497, November 1991.

[Takada91] M. Takada and T. Enomoto, "Reviews and Prospects of SRAM Technology," *IEICE Transactions*, vol. E74, no. 4, pp. 827–838, April 1991.

[Tosaka97] Y. Tosaka et al., "Cosmic Ray Neutron-Induced Soft Errors in Sub-Half Micron CMOS Circuits," *IEEE Electron Device Letters*, vol. 18, no. 3, pp. 99–101, March 1997.

[Weste93] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, 2d ed., Addison-Wesley, 1993.

[Yoo95] H. Yoo et al., "A 159-MHz 8-Banks 256-M Synchronous DRAM with Wave Pipelining Methods," *ISSCC Digest of Technical Papers*, pp. 374–375, 1995.

Exercises and Design Problem

The web site of the book (*http://bwrc.eecs.berkeley.edu/IcBook*) contains up-to-date and challenging exercises and design problems on memory.