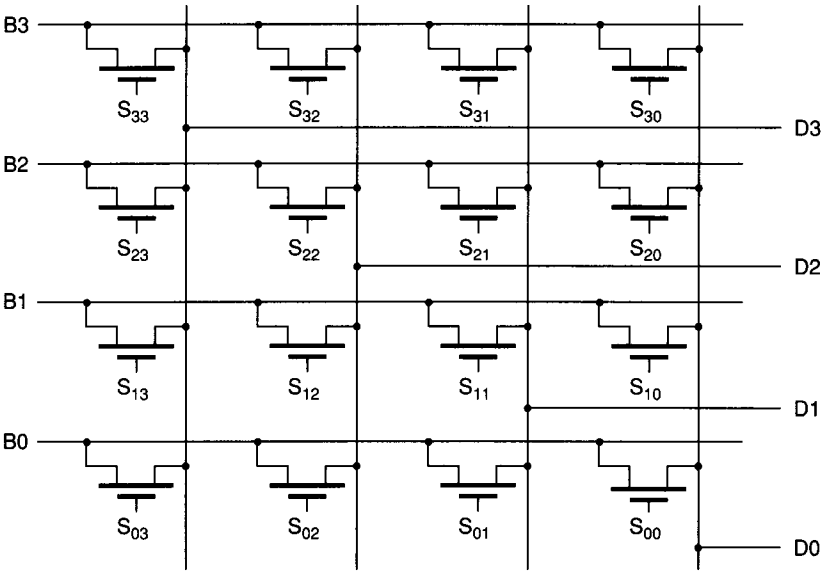


**FIGURE 9.15-5**
General bus multiplexer (16 control lines, $S_{00}$ . . . $S_{33}$).

Studying the barrel shifter structure of Fig. 9.15-6 allows some interesting observations. First, note that if the data path runs horizontally, it is necessary to provide a vertical path for control to implement the shift function. This vertical connection is sometimes used to insert or extract external data to or from the data path along D4–D6 or D0–D3. Remembering that control signals S0–S3 for the data path are provided vertically, the vertical path of the barrel shifter can be used to insert data that is part of the data path control instruction. Data that is part of the data path instruction is usually called *literal* or *immediate* data. Second, recognition of the simplicity of the barrel shifter structure suggests that the vertical pitch of the data path layout will probably be limited by the vertical pitch of the register array or the ALU rather than by the barrel shifter. This observation allows minimization of the horizontal dimension of a barrel shifter while the vertical dimension is stretched to match the rest of the data path to obtain area efficiency of the layout.

## 9.15.3 Arithmetic Logic Unit

The arithmetic logic unit (ALU) is the last important part of the data path to be discussed. As its name suggests, the ALU must provide arithmetic and logic operations on data furnished from the data path. The ALU accepts two operands, performs a specified operation, and outputs the result. A block diagram of a 32-bit wide ALU showing the inputs, control, and outputs is given in Fig. 9.15-7. The *A* and *B* inputs of the ALU along with the dual-port register array discussed earlier suggest that two parallel 32-bit buses should be provided in the data path

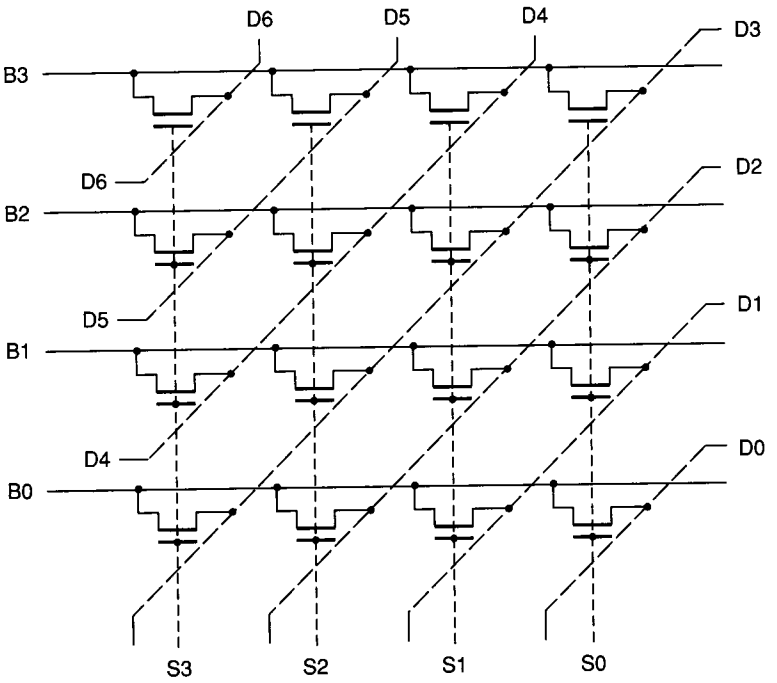


**FIGURE 9.15-6**
Barrel shifter (4 control lines, S0-S3).

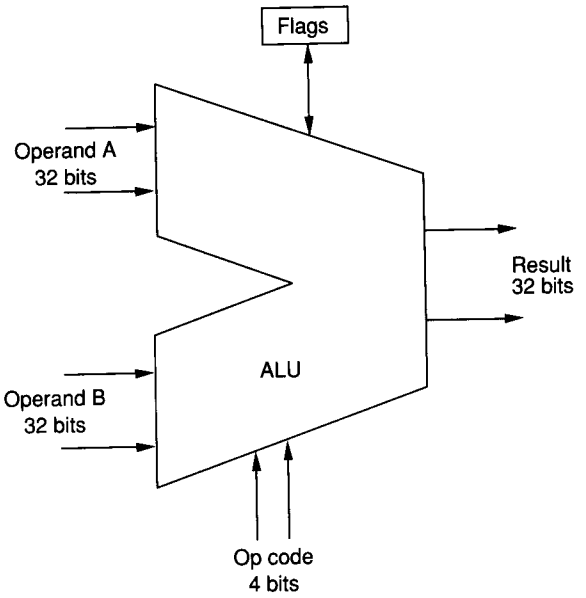


**FIGURE 9.15-7**
Arithmetic logic unit.

TABLE 9.15-1
**Common ALU operations**

| Mnemonic | Operation |
|---|---|
| ADD | Add |
| ADC | Add with carry |
| SUB | Subtract |
| SUBC | Subtract with borrow |
| NEG | Negate (2's complement) |
| AND | Logical AND |
| OR | Logical OR |
| EOR | Logical exclusive-OR |
| COM | Complement (1's complement) |
| CMP | Compare |
| ASL | Arithmetic shift left |

between the register array and the ALU. This allows both inputs to receive data simultaneously. One of the two parallel buses is used to return the result to the register array after the ALU operation is complete.

The ALU for a microprocessor is normally expected to accomplish the operations of Table 9.15-1 as a minimum. These 11 functions require a 4-bit encoded operation code (often abbreviated as *op code*) for their selection, although a 3-bit op code could be used if certain of the operations were combined. For example, the operations ADD and ADC could be implemented by specifying the ALU ADC operation with a separate control line to choose 0 or the previous carry as the carry input to implement ADD or ADC, respectively. The function COM could be implemented by using the EOR operation with the second ALU operand set to all 1s. And the ASL function could be implemented by providing the same operand to both ALU inputs and executing the ADD operation. Although a 3-bit ALU operation code would suffice, other control lines are required to select the carry input or set the second ALU operand to all 1s.

The ALU execution time may limit the maximum clock frequency of the microprocessor unless special care is taken for arithmetic operations. These operations are slowed by carry or borrow propagation delays across the width of the ALU. This problem has worsened as data bus widths have moved from 8 through 16 to 32 bits. Most microprocessors use a precharged carry line with each bit position of the ALU required to generate a carry propagate or a carry generate signal. In addition, newer microprocessors include one or more levels of carry skip circuits to speed carry propagation across groups of adjacent stages.

## 9.15.4 Microcoded Controller

Most present microprocessors use a form of microcoded controller (described earlier in this chapter) to generate required control signals for operation. Both the Motorola 68030 shown in Fig. 9.1-3 and the Intel 80386 shown in Fig. 9.15-2 are examples of microcoded microprocessors. Many times, bit fields of the microprocessor instruction word can be used directly to simplify the control field of

the microROM output. For example, encoded register specification fields can be gated from the instruction register to the data path to save microcode bits. Also, the operation code field of the instruction can directly specify the address in the microROM where execution of an instruction should start. Specific details of controller implementation vary considerably with different microprocessors and different manufacturers.

## 9.16 SYSTOLIC ARRAYS

The ability to place hundreds of thousands of transistors on a single integrated circuit and then replicate that circuit inexpensively has led many researchers to propose the use of connected sets of identical integrated circuits to solve problems in parallel. In particular, one class of parallel processors has been given the name *systolic arrays* because the data flow through the array is analogous to the rhythmic flow of blood through human arteries after each heartbeat.[23] In essence, the concept of systolic processing combines a highly parallel array of identical processors with local interconnections and rhythmic data flow. The array of processors may span several integrated circuit chips. The connections are formed so that data is accepted and processed at each stage, with the result ready for output to the next stage as new data arrives at the current stage. The objective is to keep most processors busy doing useful work to reduce the time to achieve a result. Three examples of parallel computational systems using multiple, identical circuits are described in this section.

### 9.16.1 Systolic Matrix Multiplication

Figure 9.16-1 shows a systolic interconnection of processors arranged to compute the matrix multiplication product

$$C = A \times B$$

Consider any single hexagonal processing stage of this array. An element of the $A$ matrix arrives from the upper left; an element of the $B$ matrix arrives from the upper right; and a partially computed element of the $C$ matrix arrives from the bottom. The calculation $c_{ij} = c_{ij} + a_{ik}b_{kj}$ is performed, and the new value for $c_{ij}$ is passed up to the next processing stage. Two successive steps of this calculation are shown in Fig. 9.16-1*a* and *b*. With this organization, all data are moved over local interconnections, and one-third of the processors are busy at each step. Eventually, the newly computed $C$ matrix will surface at the top of the array. The total computation can be performed faster with a systolic array such as this than with a sequential computer because many of the required calculations are performed in parallel.

### 9.16.2 General Linear System Solver

A second look at the systolic array of Fig. 9.16-1 reveals that local feedback paths are not present. Although this is not a concern for matrix multiplication,

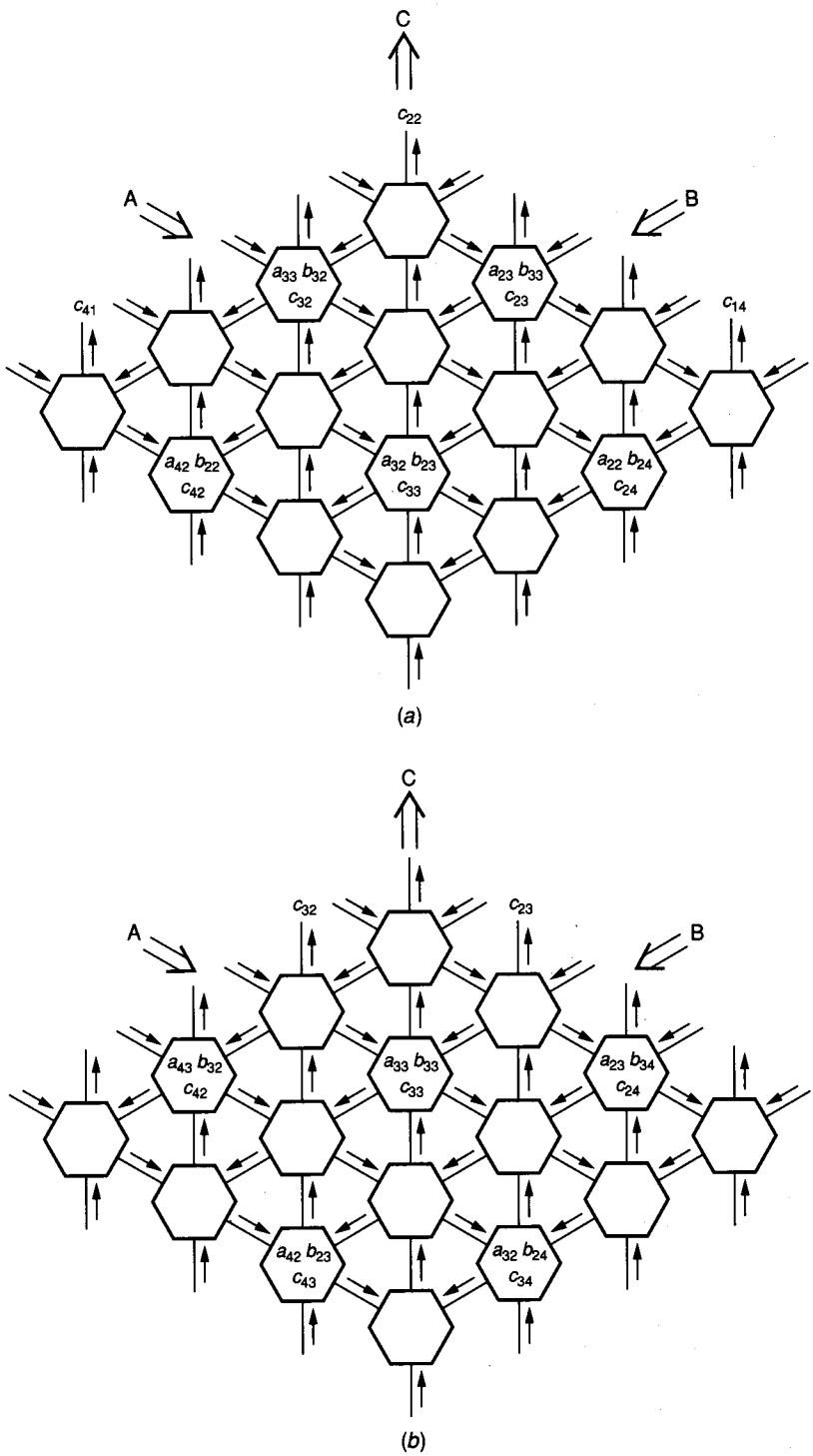


**FIGURE 9.16-1**
Systolic array multiplication: (*a*) Step $n$, (*b*) Step $n + 1$.

many engineering problems require a system of equations with feedback for their solution. Consider the general linear system representation as

$$X(k + 1) = AX(k) + Bu(k) \tag{9.16-1}$$

and

$$Y(k + 1) = CX(k) + Du(k) \tag{9.16-2}$$

The $(k + 1)$th value of the state variable vector $X$ depends on the $k$th value of the state vector. This implies a connection between the output of the state vector calculation and the inputs to the next calculation. This would be difficult with the array structure shown in Fig. 9.16-1. A systolic solution to this problem based on simple processing stages is possible. The solution, shown in Fig. 9.16-2, requires that all processors access a single common bus.

The operation of the state variable solver of Fig. 9.16-2 can be explained as follows. Rows of coefficient values from the state variable matrix are stored in circular shift registers within each processing stage. Only the output value from this circular shift register is shown for each processing element of Fig. 9.16-2. At the start of a new state vector calculation, the accumulator in each processing stage is cleared as shown in step 1. As the calculation starts, $x_1(k)$ is placed on the common bus. Each processing stage simultaneously forms the product $a_{i1}x_1(k)$, where $i$ is the process stage identifier. The resulting product is added to the contents of the accumulator. Next, at the second step, $x_2(k)$ is placed on the common bus; the product $a_{i2}x_2(k)$ is formed; and the result is added to the accumulator. Finally, for an $n$th order system, at the $(n + 1)$th step $u(k)$ is placed on the bus, the product $b_iu(k)$ is formed, and the result is accumulated. With the completion of this step, the $(k + 1)$th state vector is computed.

If $n + 1$ processing stages are provided, then the output $y(k + 1)$ can be computed simultaneously with the next-state vector. Note that a total of $n + 1$ processors are used to solve for the next-state vector and output value in $n + 1$ iterations. Direct approaches to this problem require $n^2$ iterations. The single common bus required here is less desirable than completely local interconnections, but a solution for feedback problems with only local communication is not apparent. For the structure of Fig. 9.16-2, a single, large bus driver can be placed at the left end to minimize delays associated with driving the long bus.

### 9.16.3 Bit-serial Processing Elements

To conclude this section, a look at a simple integrated circuit processing element is appropriate. Even as integrated circuit technology scales to smaller dimensions, the prospect for placing $n$ parallel processing stages on a single silicon die is dim for large $n$. The size of the typical processing stages and the interconnection buses require too much silicon area. A partial solution to this problem uses bit-serial processing stages. These stages are smaller than their $m$-bit parallel counterparts by at least a factor of $m$, allowing $m$ times as many stages to be placed on a silicon die. If the processing stages are designed properly, individual stages will interconnect directly as they are placed, thereby eliminating interconnection buses. This technique of "interconnection by default" is generally useful within integrated circuit design, saving both layout time and silicon area.

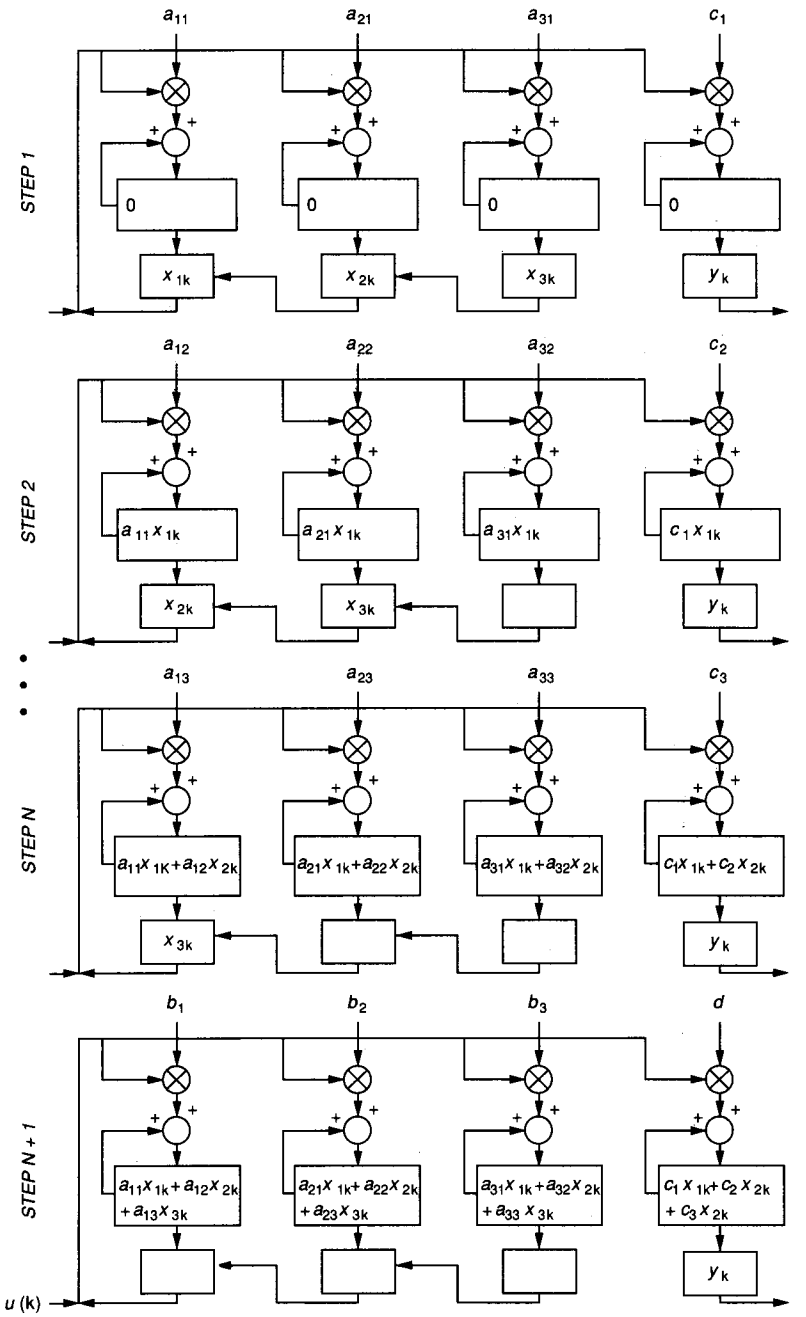


**FIGURE 9.16-2**
Block diagram for linear equation solver.

Initially, it might appear that bit-serial processing stages are a factor of $1/m$ as fast as parallel processing stages. This could negate the gain achieved by placing $m$ times as many processors on a silicon die. However, bit-serial multipliers and adders can be designed to eliminate carry propagation delay. This allows a net processing speed advantage when $m$ bit-serial processing stages replace a single $m$-bit parallel processing stage.

An example of a simple bit-serial multiplier is given in Fig. 9.16-3. The algorithm for this particular bit-serial multiplier requires the multiplicand $b$ in parallel and the multiplier $a$ in bit-sequential form. During the first iteration, the first bit of the multiplier $a_0$ is input and ANDed with the parallel multiplicand $b$, producing a set of variables called *summands*. The summands are added by the full adders to compute a set of partial product bits and the first product bit $P_0$. Carries and partial product bits are saved and shifted through unit delay registers so they are available for the next step. As the second multiplier bit $a_1$ is shifted in, it is ANDed with the multiplicand and added to previous carries and partial product bits. This produces a second bit of the product $P_1$. At each iteration the weight of each stage doubles, allowing the carry to be fed back within the same stage. This operation continues until the multiplier $a$ is exhausted and the entire product has been shifted out of the multiplier.

The simple multiplier just presented requires one operand in parallel. It is often desirable to accept both inputs in bit-sequential form. This is easily accomplished by using pipeline techniques or through other, slightly more complex bit-serial multipliers.[24]

The capability to map algorithms into hardware with large-scale integrated circuits is revolutionizing the way signal processing is accomplished. The ability to create special-purpose processors to provide parallel solution of time-consuming problems may be the next major step in increasing computational speeds. The concept of systolic processing, with many processors working in lock step fashion, is an important means to achieve this goal.

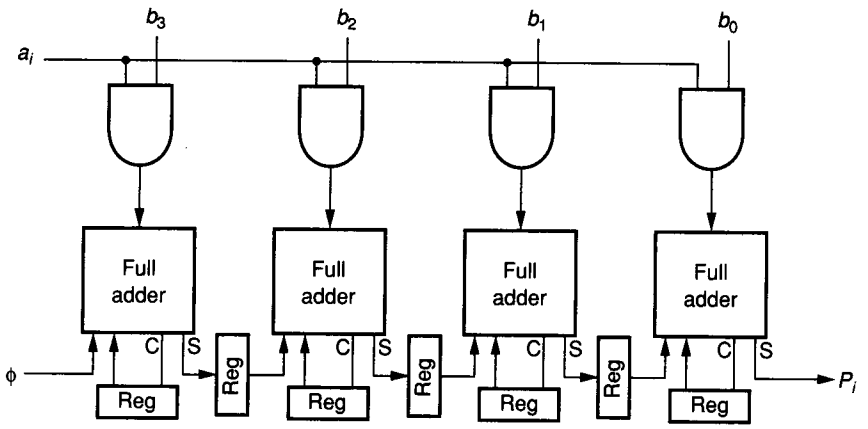


**FIGURE 9.16-3**
Serial-parallel multiplier.

## 9.17 SUMMARY

The scope of a chapter describing structured digital circuits and systems is, of necessity, quite broad. The topics presented include structured logic forms such as PLAs, Weinberger arrays, gate matrix layout, and gate arrays (also sea-of-gates). Clocking schemes for digital circuits were introduced as a prerequisite to the treatment of dynamic storage, clocked logic, and sequential machines. Three styles of clocked logic, including $C^2$MOS, precharge evaluate logic, and domino CMOS logic, were explained. A prototypical semiconductor memory architecture was presented, followed by an investigation of the prominent types of memory storage cells that are used. Salient limitations of both the architecture and the individual cells were provided, usually by example. The memory sections were followed by brief introductions to two widely used forms of sequential machine: PLA FSMs and microcoded FSMs. With these concepts available, an overview of microprocessor architecture was presented to show how the digital subsystems just described can be formed into a complex digital system. And finally, systolic arrays were introduced along with three examples: an array multiplier, a linear system solver, and bit-serial multiplication.

The goal of this chapter was to provide an awareness and basic understanding of structured digital circuits and examples of how these structures are used to form complex digital systems. The information presented on digital integrated circuit structures, along with the prerequisite background in digital system design, allows the design of large digital systems within the integrated circuit medium.

## REFERENCES

1. B. Lattin: "VLSI Design Methodology: The Problem of the 80's for Microprocessor Design," *Proc. Caltech Conf. on VLSI*, pp. 247–252, January 1979.
2. D. A. Patterson and C. H. Sequin: "A VLSI RISC," *Computer*, pp. 8–21, September 1982.
3. R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli: *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.
4. G. De Micheli and A. Sangiovanni-Vincentelli: "Pleasure: A Computer Program for Simple/-Multiple Constrained Unconstrained Folding of Programmable Logic Arrays," UCB/Electronics Research Laboratory Memorandum M82/57, University of California, Berkeley, August 9, 1982.
5. G. D. Hachtel, A. R. Newton, and A. L. Sangiovanni-Vincentelli: "Techniques for Programmable Logic Array Folding," *Proc. 19th Design Automation Conf.*, pp. 147–155, June 1982.
6. A. Weinberger: "Large Scale Integration of MOS Complex Logic: A Layout Method," *IEEE J. Solid-State Electronics*, vol. SC-2, no. 4, pp. 182–190, December 1967.
7. J. M. Siskind, J. R. Southard, and K. W. Crouch: "Generating Custom High Performance VLSI Designs from Succinct Algorithmic Descriptions," *Proc. MIT Conference on Advanced Research in VLSI*, pp. 28–39, January 1982.
8. S. M. Kang, R. H. Krambeck, H. F. S. Law, and A. D. Lopez: "Gate Matrix Layout of Random Control Logic in a 32-Bit CMOS CPU Chip Adaptable to Evolving Logic Design," *Proc. 19th Design Automation Conf.*, pp. 170-174, 1982.
9. A. D. Lopez and H-F. S. Law: "A Dense Gate Matrix Layout Method for MOS VLSI," *IEEE J. Solid-State Circuits*, vol. SC-15, no. 4, pp. 736–740, August 1980.
10. G. Dupenloup: "A Wire Routing Scheme for Double-Layer Cell Arrays," *Proc. 21st Design Automation Conf.*, pp. 32–35, June 1984.
11. C. Sechen and A. L. Sangiovanni-Vincentelli: "The TimberWolf Placement and Routing Package," *IEEE J. Solid-State Circuits*," vol. SC-20, no. 2, pp. 510–522, April 1985.

12. J. Y. Chen: "CMOS—The Emerging VLSI Technology," *Proc. IEEE Int. Conf. on Comp. Design*, pp. 130–141, October 1985.
13. W. N. Carr and J. P. Mize: *MOS/LSI Design and Application*, McGraw-Hill, New York, 1972.
14. N. H. E. Weste and K. Eshraghian: *Principles of CMOS VLSI Design*, Addison-Wesley, Reading, Mass., 1985.
15. R. H. Krambeck, C. M. Lee, and H-F. S. Law: "High-Speed Compact Circuits with CMOS," *IEEE J. Solid-State Circuits*," vol. SC-17, no. 3, pp. 614–619, November 6, 1981.
16. J. A. Marques and A. Cunha: "Clocking of VLSI Circuits," *VLSI Architecture*, B. Randell and P. C. Treleaven, eds., Prentice-Hall, Englewood Cliffs, N. J., pp. 165–178, 1983.
17. T. H. Bennett: "Split Low Order Internal Address Bus for Microprocessor," U.S. Patent No. 3,962,682, Motorola, Austin, Texas, June 8, 1976.
18. W. I. Fletcher: *An Engineering Approach to Digital Design*, Prentice-Hall, Englewood Cliffs, N. J., 1980.
19. B. T. Murphy, R. Edwards, L. C. Thomas, and J. J. Molinelli: "A CMOS 32 Bit Single Chip Microprocessor," *ISSCC Digest of Technical Papers*, p. 230, 1981.
20. J. W. Beyers, L. J. Dohse, J. P. Fucetola, R. L. Kochis, C. G. Lob, G. L. Taylor, and E. R. Zeller: "A 32-Bit VLSI CPU Chip," *IEEE J. Solid-State Circuits*, vol. SC-16, pp. 537–542, October 1981.
21. G. Moore: "VLSI: Some Fundamental Challenges" *IEEE Spectrum*, pp. 30–37, April 1979.
22. R. W. Sherburne, Jr., M. G. H. Katevenis, D. A. Patterson, and C. H. Sequin: "Datapath Design for RISC" *Proc. MIT Conf. on Advanced Research in VLSI*, pp. 52–62, January 1982.
23. H. T. Kung: "Let's Design Algorithms for VLSI Systems," *Proc. Caltech Conf. on VLSI*, pp. 65–90, January 1979.
24. N. R. Strader and V. T. Rhyne: "A Canonical Bit-sequential Multiplier," *IEEE Trans. Comput.*, vol. C-31, no. 8, pp. 791–795, August 1982.

## PROBLEMS

**Section 9.1**

**9.1.** If a manufacturer's layout rate is eight transistors/day and a chip regularization factor of 20 is achieved, how many man-years are required to lay out a 300,000 transistor chip?

**9.2.** Estimate the layout time for the Intel 8086 and the Motorola 68000 if a layout rate of 10 transistors/day is assumed for each (use Table 9.1-1).

**Section 9.2**

**9.3.** Determine the sizing triplet $(i, p, o)$ for a PLA that implements the following logic equations.

$$X = PB + D\overline{B} + \overline{P}\,\overline{D}A$$

$$Y = P\overline{A} + DA + \overline{P}\,\overline{D}A$$

$$Z = PB + DA$$

**9.4.** In terms of $\lambda$, estimate the minimum pitch for repeating the AND plane section of Fig. 9.2-5*c* using the design rules of Table 2B.2 of Appendix 2B. Consider the spacings required to connect the rotated AND section into the OR plane. *Pitch* is defined as the repetition spacing for repeated placements of layout segments.

**9.5.** Find a folded PLA realization for the following equations so that the PLA sizing triplet is reduced from a straightforward implementation. Can you determine if this is the minimum realization? Why or why not?

$$X = ABC + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}$$

$$Y = ABC + \overline{B}\,\overline{C}$$

$$Z = ABC$$

$$R = \overline{A}BC + AB\overline{C}$$

$$S = \overline{A}\,\overline{B} + AB\overline{C} + \overline{B}C$$

**Section 9.3**

**9.6.** Show an NMOS Weinberger NOR array implementation for the logic equations in Prob. 9.3.

**9.7.** If no rows are shared between input terms and intermediate terms in a Weinberger NOR array, compute the area required to implement the logic equations in Prob. 9.3 as the product of the rows and columns. Compare with a PLA implementation for the same logic equations using the same metric.

**9.8.** What logic functions are defined by the symbolic gate matrix layout of Fig. 9.3-5?

**9.9.** Provide a symbolic gate matrix layout for the logic equations of Prob. 9.3.

**Section 9.4**

**9.10.** The logic equations of Prob. 9.3 are to be implemented in a CMOS gate array using logic function blocks from the library of Table 9.4-1. Provide a list of the blocks required and the specific logic equations implemented by each block for this design. Use five or fewer types of standard logic function blocks.

**9.11.** Compare the number of transistors required for a straightforward CMOS implementation of the logic equations of Prob. 9.3 and the number of transistors required with a gate array implementation using blocks from Table 9.4-1. Note that each standard logic function block in Table 9.4-1 is composed of a multiple of 12 transistors (6 p-channel and 6 n-channel).

**Section 9.5**

**9.12.** For the circuit of Fig. 9.5-4*a*, let NOR2 be the gate connected directly to $\phi_2$, NOR1 be the gate connected to $\phi_1$, and INV represent the inverter. Identify the gate or gates that cause each of the four delays shown in Fig. 9.5-4*b*. If the delays are 3, 4, and 5 ns for the INV, NOR2, and NOR1 gates, respectively, find the nonoverlap time following (*a*) $\phi_2$ and (*b*) $\phi_1$.

**9.13.** If a pair of inverters is added between each NOR gate output and its feedback connection in Fig. 9.5-4*a*, show the resulting clock waveforms as in Fig. 9.5-4*b*. Assume unit delays for all gates and inverters. (Note that such inverter pairs could be sized geometrically to increase clock drive capability.)

**Section 9.6**

**9.14.** For an NMOS dynamic storage circuit with a pass transistor drain diffusion area of 20 $\mu^2$, a metal interconnect area of 20 $\mu^2$, a non-gate polysilicon area of 10 $\mu^2$ and an inverter gate area of 6 $\mu^2$, calculate an equivalent dynamic storage time constant if $C_{\text{diff}} = 0.04$ fF/$\mu^2$, $C_{\text{met}} = 0.03$ fF/$\mu^2$, $C_{\text{poly}} = 0.03$ fF/$\mu^2$, $C_{\text{ox}} = 0.4$ fF/$\mu^2$, and the leakage current $I_r = 0.5$ fA/$\mu^2$.

**9.15.** In what ways does the use of a CMOS transmission gate rather than an NMOS pass transistor affect the storage time of a dynamic storage circuit? Explain.

**9.16.** The level restorer transistor of Fig. 9.6-1*c* must be sized above some minimum value and below some maximum value. What limits the minimum and maximum allowable resistances for this transistor?

**9.17.** Consider the linear shift register of Fig. 9.6-2. If the input to stage A is brought high during $\phi_1$ of a first clock cycle and then left low, prepare a timing diagram showing the clock signals $\phi_1$ and $\phi_2$ and the outputs of each of the four shift stages for five consecutive clock cycles.

**9.18.** Prepare a symbolic layout diagram (see Fig. 9.6-4) that shows a clocked shift register that provides a left shift, no shift, or a right shift for four parallel bits.

**Section 9.7**

**9.19.** Generate geometrical layouts for the shift register circuits of Figs. 9.7-1 and 9.7-2. Compare the complexity of the two layouts, particularly noting the number of contacts required.

**9.20.** Provide circuits to implement the following logic equations using complex P-E gate structures as in Fig. 9.7-4.

$$R = AB + BC + AC$$

$$S = ABC + \overline{A}\,\overline{B}\,\overline{C}$$

**9.21.** Assume the drain and source of each transistor in Fig. 9.7-4 contribute an equal incremental capacitance at each node. If the node between the transistors gated by signals E and F is low, the signals D, B, and A are each low, the gate output is precharged high, the clock is high, and the input signal F is then driven high, calculate the effect of charge sharing on the output voltage if load capacitance caused by subsequent connections is ignored.

**9.22** P-E logic stages cannot be directly cascaded. Consider alternating P-E stages of n-channel logic clocked by clk1 with P-E stages of p-channel logic clocked by clk2. Can clk1 and clk2 be complementary signals? What conditions must clk1 and clk2 satisfy for this cascade configuration to work correctly?

**9.23.** Is it possible to realize the exclusive-OR function using only domino logic circuits? Why or why not?

**9.24.** Create a table to compare static NMOS, static CMOS, P-E CMOS, and domino CMOS logic gates in terms of total transistor count, static power dissipation (yes or no), and output availability (duty cycle).

**Section 9.8**

**9.25.** *IEEE Spectrum* publishes a technology update issue in January of each year. Prepare a 1–2-page summary of the state of the art in semiconductor memories based on the most recent update issue.

**9.26.** Quantify the row and column decode circuitry required for a 1M × 1-bit memory (*a*) organized as a square array and (*b*) organized with four more address lines feeding the row decoder than the number of address lines feeding the column decoder.

**Section 9.9**

**9.27.** Assume the charge on the gate of an EPROM cell is $10 \times 10^{-15}$ C. Assume the gate loses charge exponentially with a time constant of 10 years. Calculate the equivalent leakage resistance from the gate to ground if the gate capacitance is 4 fF.

**9.28.** Provide the layout for a macro defining the repeatable layout for a ROM memory cell in the technology of Appendix 2B. Now estimate the area required for a 256k-bit memory based on your cell layout. Estimate the size of a repeatable memory cell in a commercial 256k-bit memory area if the array size is 10 $mm^2$.

**Section 9.10**

**9.29.** Consider a 256k SRAM with row lines fabricated as 2 $\mu$ polysilicon runs. Assume the row select transistors are 2 $\mu$ × 4 $\mu$ and the select lines are 5 mm long. Assuming a square memory array organization, (*a*) provide a reasonable estimate of the delay across the select line, and (*b*) provide a reasonable estimate of the delay across the select line if the row decoder divides the memory array into two equal parts to halve the select line length. Use parameters from Appendix 2B.

**9.30.** For a 256k SRAM that dissipates 0.4 W in the memory array, estimate the resistance of the polysilicon load resistors for the memory cells.

**Section 9.11**

**9.31.** If a 1M DRAM is built from cells with 40 fF storage capacitance and the minimum differential voltage that will reliably trigger the sense amplifiers is 80 mV, determine the maximum allowable capacitance for the word lines.

**9.32.** Assume that a DRAM storage cell is built using a 50 fF capacitor and that a voltage of at least 3.8 V can be reliably recognized as a logic high. If a memory refresh period of 2 s is sufficient under average conditions, estimate the total leakage resistance from the storage cell to ground. Assume a memory refresh charges the capacitor to 5 V.

**9.33.** In the design of a DRAM memory array, it is found that the delay along the polysilicon row select line is too great, and the suggestion is made to widen the row select line to reduce its equivalent resistance. Will this solution help reduce the select line delay? Why or why not? Will this have other, detrimental effects on the memory array? Explain.

**Section 9.12**

**9.34.** Show how to modify Fig. 9.12-5 for dual-port read capability.

**9.35.** The data lines of a static register array are precharged to the supply voltage and have much larger capacitances than the register cell output. If the select line couples a logic low output of the register cell to the precharged data line during a read operation, what may happen to the register cell contents? Explain.

**9.36.** For a static register cell, determine a constraint on the relative size of the inverter pulldown transistors and the select transistors to prevent disturbing the register cell state through charge sharing during a read operation with precharged data buses.

**9.37.** Using the cross-coupled NOR structure for the static register cell array, is it reasonable to provide a dual-port write capability? Explain.

**Section 9.13**

**9.38.** A Gray Code counter has the distinguishing feature that successive counts never differ in more than one bit. Thus, a 2-bit Gray Code counter counts as 00, 01, 11, 10, 00, 01, etc. An application requires a Gray Code counter that can count up when an UP control line is asserted or down when a DOWN control line is asserted. If both controls are low, the counter remains in its present state. External

circuitry prevents the condition of both control inputs being high simultaneously. Provide a state diagram, a state transition table, logic equations, and a logic/circuit diagram of a PLA FSM to implement this counter.

**Section 9.14**

**9.39.** For a typical microROM such as that of Fig. 9.14-2, consider the number of unique 72-bit control words available. With an execution rate of 10 MHz, how long would it take to execute each control word exactly once?

**9.40.** Considering the typical ROM organization described in previous sections, compare the maximum number of rows and columns that must be traversed during a bit access using the single-level microROM of Fig. 9.14-2 and the two-level microprogram memory of Fig. 9.14-3. Explain how this would affect access time for the two memories.

**9.41.** Using the description of Example 9.14-1, list the control words necessary to perform the same add operation if the register array is single-port read rather than dual-port read.

**Section 9.15**

**9.42.** Estimate and compare the percentages of total area consumed by regular structures on the die photographs of Fig. 9.15-1 and Fig. 9.15-2.

**9.43.** Assume the data path of a microprocessor has a 32-bit by 16 register array, a 0 to 32-bit shift capability for the barrel shifter, and the ALU operations of Table 9.15-1. Ignoring timing and temporary results storage, specify the control lines required to operate this data path.

**9.44.** Using the barrel shifter of Fig. 9.15-6, indicate the states of the control lines and the source and destination buses necessary to perform (*a*) a shift left by 2 bits (quadruple the magnitude of the input), and (*b*) a shift right by 1 bit (halve the magnitude of the input).

**9.45.** Four flag bits are common for most microprocessor ALUs. These include the C, N, Z, and V bits (carry, negative, zero, and overflow, respectively).
(*a*) The Z bit is normally generated using a distributed NOR gate structure at the ALU output. Show how this might be accomplished.
(*b*) Explain how the N bit could be generated.
(*c*) Explain how the C bit could be generated.
(*d*) The V bit can be generated as the exclusive-OR of the final carry bit and the penultimate (next to the highest) carry bit. Demonstrate that this is logically correct for two's complement arithmetic.

**Section 9.16**

**9.46.** Using the systolic array of Fig. 9.16-1 as a guide, show the systolic array structure required to multiply two $4 \times 4$ square arrays to produce a $4 \times 4$ result. How many processors are required?

**9.47.** For the serial-parallel multiplier of Fig. 9.16-3, how many steps are required to multiply two 4-bit numbers? Show the value of the sum bits and the carry bits for each step when multiplying 6 by 5.

**9.48.** Symbolically show the hand multiplication of two 4-bit numbers $a$ and $b$. Demonstrate that the $i$th product bit is a function of only the $i$th and lower-order multiplicand and multiplier bits.