DIGITAL SYSTEM DESIGN WITH LSI BIT-SLICE LOGIC

Glenford J. Myers

DIGITAL SYSTEM DESIGN WITH LSI BIT-SLICE LOGIC

Glenford J. Myers
Senior Staff Member
IBM Systems Research Institute

A Wiley-Interscience Publication
JOHN WILEY & SONS
New York • Chichester • Brisbane • Toronto

Other books by GLENFORD J. MYERS

Reliable Software Through Composite Design, 1975
Software Reliability: Principles and Practices, 1977
Composite/Structured Design, 1978
Advances in Computer Architecture, 1978
The Art of Software Testing, 1979

Library of Congress Cataloging in Publication Data:

Myers, Glenford J 1946–
Digital system design with LSI bit-slice logic.

"A Wiley-Interscience publication."
Includes bibliographical references and index.
1. Electronic digital computers—Design and construction. 2. Digital electronics. 3. Logic circuits. 4. Logic design. 5. Integrated circuits—Large scale integration. 6. Microprogramming. I. Title.

TK888.3.M93 621.3819'58'3 79-26258
ISBN 0-471-05376-7

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Preface

If one takes the nature of the fundamental unit of design or atomic building block as a measure of "generations" of digital engineering, the field can be said to be in its fourth generation. In the first generation, the building blocks were discrete components, such as vacuum tubes and, later, transistors. The second generation can be categorized as the use of integrated-circuit gates as the basic units of design. In the third generation, the engineer dealt with more powerful building blocks, such as registers, multiplexers, and arithmetic and logic units. In the fourth generation, the units of design grew into such sophisticated single components as microprocessors, CRT controllers, and analog-to-digital converters. A more recent component in the fourth generation is bit-slice logic, a powerful and flexible digital building block. This is the subject of this book.

The motivation for writing this book was that, in comparison to the large amount of literature on other fourth-generation building blocks such as microprocessors, there is relatively little literature (except for semiconductor manufacturers' specifications) on bit-slice logic. A further motivation was that, beginning in 1978, a variety of manufacturers announced that new large-scale computers were being produced using bit-slice logic.

The main thrust of the book is as a tutorial and reference for the system architect and digital design engineer. The book also has application in electrical-engineering and computer-science curricula as a supplemental text in such courses as computer organization, digital system design, digital logic design, microprocessors, and microprogramming.

Chapter 1 introduces the concept of bit-slice logic. It also introduces the 2901, the most widely used bit-slice component. Since bit-slice logic is usually used with the control concept of microprogramming, this concept is introduced in Chapter 2. Chapters 3 and 4 describe many of the bit-slice components available from semiconductor manufacturers. In addition to covering these components in detail, Chapters 3 and 4 put the slices into perspective and discuss problems in the designs of some of the slices.

Chapter 5 discusses more advanced microprogramming topics, such as pipelining, encoding, and optimization. Chapter 6 illustrates some auxiliary LSI logic components, some of which are designed as bit slices. Complementing the devices

discussed in Chapter 7 are general devices that can be customized or "programmed" as specialized logic components; these devices, such as programmable logic arrays and gate arrays, are discussed in Chapter 7.

Given that the microprogramming concept is closely allied with the use of bit-slice logic, the last two chapters discuss tools, facilities, and principles for the development of microprograms.

I gratefully acknowledge the help of two colleagues—Dan O'Donnell of the IBM Systems Research Institute and Poughkeepsie Laboratory, and Dave Hocker of the IBM Poughkeepsie Laboratory—in reviewing the manuscript of this book and providing many helpful suggestions. I also acknowledge the cooperation of the following companies for providing material and permission to use selected parts of it:

Advanced Micro Devices
Fairchild Camera and Instrument Corp.
Intel
Monolithic Memories
Motorola
Texas Instruments

The views and opinions herein are solely those of the author, who also takes responsibility for any errors.

GLENFORD J. MYERS

New York, New York
January 1980

Contents

1

An Introduction to Bit-Slice Logic

Bit-slice logic is the most recent generation of fundamental building blocks available to the digital design engineer. Not only do such devices give the engineer a set of powerful, fast, and flexible building blocks, but also they attempt to solve the major problem of LSI technology, namely, the potential proliferation of unique part types.

When one looks at the history of the engineer's basic building blocks, one can identify roughly four generations of components, although the process has really been one of evolution, rather than a clearly separated sequence of distinct steps. Not only did each generation introduce significant improvements in component speed, cost, and reliability, but they also significantly improved the productivity of the system design and manufacturing processes. In reviewing the latter, it is helpful to see the effects of the generations on the traditional types of design that must be performed, namely:

Circuit design —the interconnection of components such as transistors, resistors, and capacitors, to form logic devices such as ANDs and ORs.

Logic design —the interconnection of logic devices to form combinatorial and sequential devices such as registers, counters, and adders.

System design —the interconnection of such devices as adders, registers, and memory arrays to form digital systems such as processors and I/O device controllers.

Physical design—the physical layout of the components, for instance, on printed-circuit or wire-wrap boards.

The building blocks of the first technology generation, beginning in the 1940s and lasting into the early 1960s, were discrete components such as transistors (earlier, vacuum tubes), diodes, resistors, and capacitors. Here the engineer was faced with the full tasks of circuit, logic, system, and physical design.

The second generation, occurring in the early- and mid-1960s, saw the building blocks grow into integrated circuits, each of which contained perhaps 10 to 50 elementary components and performed such logic functions as AND and NOR. The jobs of logic and system design remained the same (actually, they grew because of the increased sophistication of systems), but the task of circuit design was greatly reduced (except, of course, for the few people designing the integrated circuits themselves), and the task of physical design was reduced somewhat, since the integrated circuit reduced the total part count of a system.

The building blocks of the third generation, occurring in the late 1960s and early 1970s, were integrated circuits representing logic devices (MSI) containing perhaps 50 to 200 elementary components and forming such devices as registers, counters, multiplexers, and arithmetic-and-logic units. This generation continued the reduction of the task of physical design (again because of a reduction in the number of physical parts needed to build a system) and lessened the task of logic design.

The fourth generation, which began in the early 1970s, brought about an immense leap in the size and scope of the building blocks. One notable change occurred in storage devices; single chips containing 4096 bits of storage, and later, 16,384 and 65,536 bits, became widely available. Another notable change was the microprocessor; on one chip, embodied in 20,000 or more elementary components, is a full-fledged central processing unit and perhaps a small amount of memory.

Given that a microprocessor and its support chips (e.g., disk controllers, keyboard controllers, communication interfaces, memory-refresh logic, bus controllers) can be used, the tasks of digital system design and physical layout are substantially reduced. However, the microprocessor is not a viable solution to all design problems (i.e., the problem of designing a high-speed, single-processor computer); microprocessors are relatively slow and have static and primitive instruction sets. This precipitated a requirement for a more flexible set of fourth-generation building blocks. The bit-slice device is an answer to this requirement.

THE EVOLUTION OF BIT-SLICE DEVICES

Fourth-generation building blocks are LSI devices. The motivations of LSI are lower costs (large amounts of circuitry are mass produced on a single silicon chip, reducing the costs of the primitive components and eliminating most of the human-assembly costs of prior generations), higher speeds (by reducing transistor sizes, path lengths between components, and circuit capacitance), higher reliability (by reducing the number of mechanical interconnections among components), and shorter design times. However, the designer of LSI devices now faces two new problems: the "pin-out" problem and the "part-proliferation" problem.

The pin-out, or pin-count, problem is the easiest to understand. A typical LSI silicon chip may have a size of 0.15 x 0.2 inches. Obviously only a limited number of external connections can be made between this chip and the outside world. Currently the feasible upper limit is in the neighborhood of 100 connections (pins on

the package holding the chip), and it is unlikely that significant increases in this limit will be made. This restricts the type of circuitry that can be placed on a chip. For instance, it is not feasible to construct a 32-bit ALU chip (in spite of the fact that the amount of circuitry is well within the state of the art), since such a device would require over 100 pins (32 outputs, 64 inputs, several control inputs, and several status outputs). Hence the pin-count limitation is one of the great barriers to the use of LSI.

The second problem is the potential proliferation of unique LSI devices. In producing an LSI device, the design costs are extremely high, but the production cost per unit is extremely low. Hence the economics of LSI are attractive only when a large number (e.g., tens of thousands or more) of units of each type can be used.

Here one is faced with a dilemma. The large number of circuits on an LSI device could easily imply that such devices (with the exception of memory arrays and microprocessors) are specialized toward a particular system and unusable in other designs. For instance, suppose that a computer company were developing a family of processors, each with distinct cost and performance objectives. If one could design several LSI chips for use in processor A, it is unlikely that they could be used in processor B, because the processors are likely to have dissimilar internal designs (e.g., different data-path widths, different ALU functions, different degrees of internal parallelism). That is, the mere nature of an LSI chip (containing thousands of gates) means that it is likely to absorb much of the nature of the system in which it is a part, rendering it unusable in other designs, and hence restricting the volume in which it can be manufactured. Thus one is faced with another dilemma that could prevent the designer from taking advantage of LSI. Note that the pin-count restriction compounds the part-proliferation problem, since it might require one to develop a larger number of LSI chip types, increasing the proliferation of specialized chips.

The road to a solution is the realization that, in LSI-based design, one should not be preoccupied with minimizing the number of elementary components or gates in a system but with minimizing the number of chip types. What is needed is a small set of universal chip types. Rather than containing static functions, the functions of these devices should be capable of being controlled externally (i.e., by logic external to the device). To serve as building blocks in a large number of systems, these devices should be capable of performing a large number of functions, many of which might not be used in a particular design. At the same time, these universal devices should place few, if any, restrictions on the designs in which they might be used (e.g., restrictions on data-path sizes). Last, such devices must conform to the pin-count limitations. The bit-slice device is an answer to these requirements.

THE NATURE OF A SLICE

The question then is determining how to partition a system into a set of LSI building blocks such that they can be used in a variety of other designs. Consider-

ing central processing units for the moment, the answer is the realization that the heart of most processors, independent of their instruction-set architecture, is similar to that illustrated in Figure 1.1. That is, the heart of a processor normally consists of an array of registers, a multifunction ALU, and shift logic. One thought might be to incorporate all this on a chip, but this is not a solution for several reasons. First, one is likely to encounter the pin-count problem. Second, the chip will not be universal, because it will contain too many dependencies (such as data-path width) on the original design. For instance, if Figure 1.1 is the heart of a 16-bit processor, such a chip would prove to be of no use in 32-bit, 36-bit, and 8-bit designs.

The solution is to view Figure 1.1 as the three-dimensional equivalent in Figure 1.2, and then make vertical slices through the design. That is, one creates devices having the appearance of Figure 1.3. Figure 1.3 illustrates a device that might be termed a 4-bit ALU/register slice. By designing such a chip and bringing out appropriate signals (e.g., ALU carry-in and carry-out, both sides of the shift register), one can devise a universal building block by allowing a set of these devices to be interconnected to yield an ALU/register section of arbitrary width

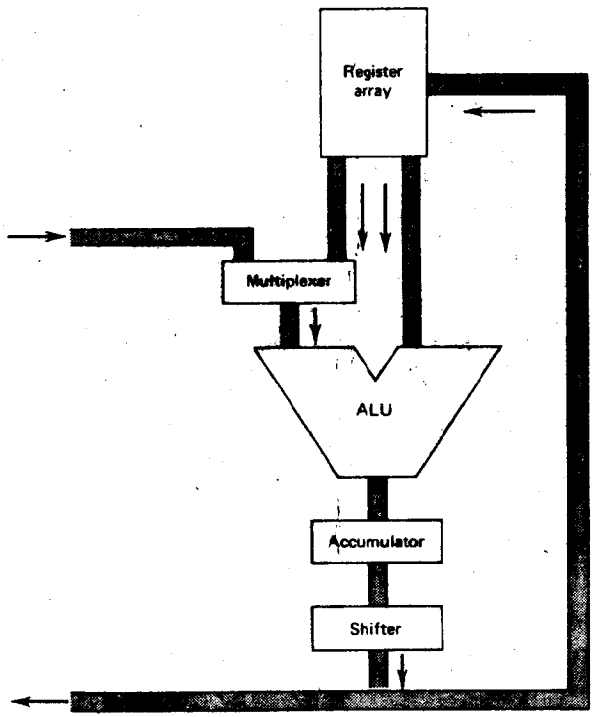

Figure 1.1. Typical processing-section organization.

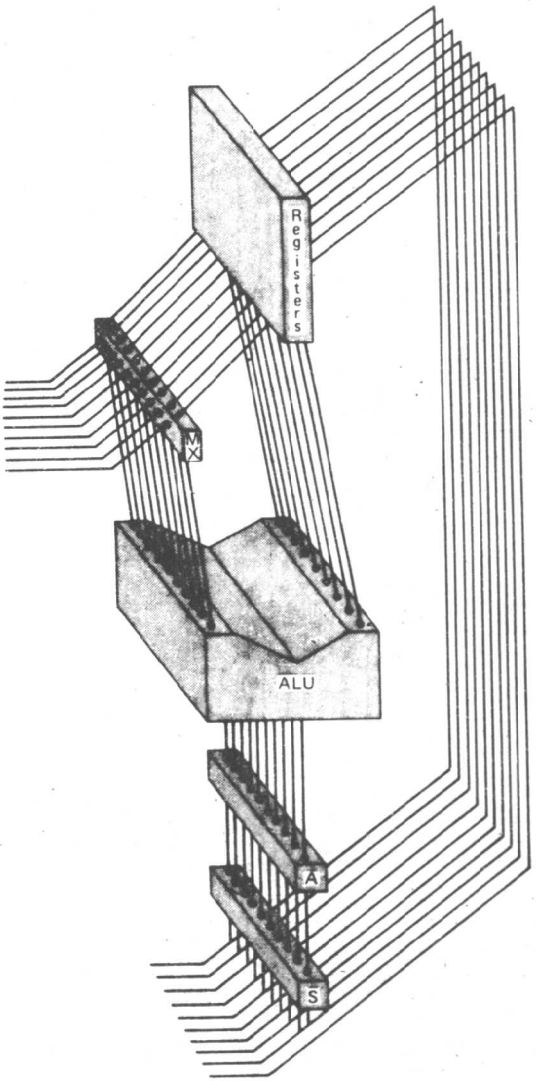

Figure 1.2. Three-dimensional view of a typical processing section.

(e.g., 8, 12, 16, . . . bits). Such a device solves the problems mentioned earlier by having the following attributes:

1. The pin-count problem is not present because of the "narrowness" of the device. A device of the form in Figure 1.3 might have 24 to 40 pins.

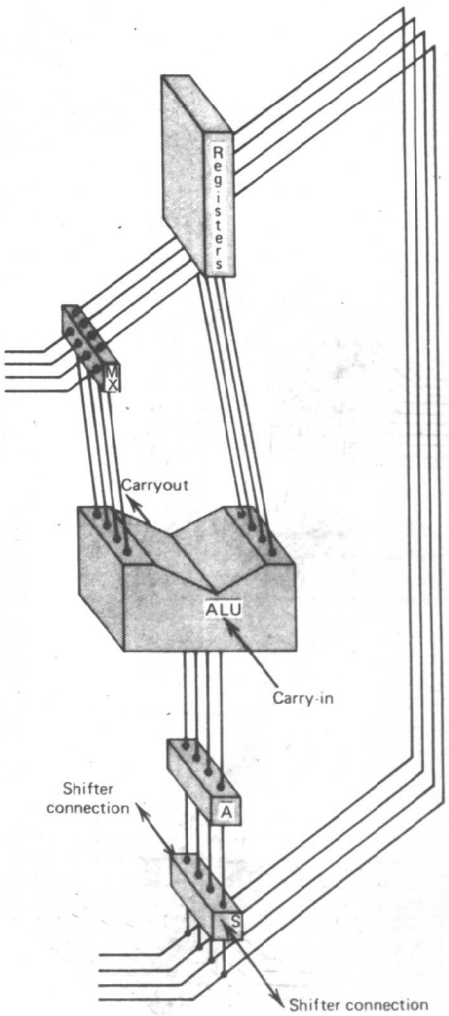

Figure 1.3. A vertical slice through the processing section.

2. It is an LSI device because, although it is a narrow slice through a set of registers, an ALU, shifters, and so on, it contains a large number of circuits.
3. It serves as a universal building block, because the device is designed to perform a large number of functions, some of which are likely not to be used in a particular application (the cost of the unused functions is virtually zero, assuming the device is manufactured in large quantities), and the devices can be cascaded together to form a processing section of any width.

Note that only one type of slice—the ALU/register slice—has been introduced at this point. Other types of slices have been devised. These will be introduced in later chapters.

THE 2901 ALU/REGISTER SLICE

The best way to gain an understanding of bit-slice logic is to examine an actual device, and the best device to start with is the 2901 ALU/register slice. The 2901, first produced by Advanced Micro Devices and now second-sourced by many other firms, is the most widely used bit-slice device. The 2901 is to bit-slice logic what the 8080 has been to microprocessors. The 2901 is used as the heart of such CPU products as DEC's DECsystem-2020, Data General's Nova 4, National Semiconductor's System/400, Functional Automation's F6400, and the Ampex Model 12.

The 2901 is a 40-pin LSI chip; most versions of the 2901 employ low-power Schottky TTL technology. The chip contains about 500 gates. Only an introduction to the 2901 is presented here; it is discussed in more detail in Chapter 3.

Figure 1.4 illustrates the organization of the 2901. Its data paths are 4-bits wide. The basic sections are a 16-word by 4-bit, 2-port RAM, a working register (Q), an ALU, and shifting, decoding, and multiplexing logic.

Any of the 16 registers can be read onto the A bus. Likewise, any of the 16 registers can be read onto the B bus. Each of these busses contains a latch, used to prevent race conditions when the output of the ALU is being written back into the register array.

Both inputs of the ALU are fed from multiplexers. The R input of the ALU can be selected to be the register value on the A bus or a value on the D bus, an external input bus. The S input of the ALU can be selected from the A bus, the B bus, or the Q register. Both multiplexers have an inhibit capability, meaning that the value zero can also be fed into the R and/or S ALU inputs.

The Q register is a working register available for general use, although the motivation for its existence is for use in implementing multiplication and division algorithms. Looking at the connections to the Q register, one can feed it from the ALU output bus or from itself. If Q is fed from itself, the shifter allows one to feed Q with its current value shifted right one bit, shifted left one bit, or not shifted (*no* change).

Figure 1.4 shows that the output of the ALU can be gated to three places—to the Y bus (an external output bus), to the Q register, or to the register array. When it is gated to the register array, the data moves into the register that was designated as the register to be gated to the B bus. Between the ALU and the register array is a shifter, allowing the ALU output to be shifted right or left one bit, or not shifted, before the value is placed in the register array. Note that the external output bus Y is controlled by a multiplexer, meaning that the data placed on the Y bus can be the ALU output or the A bus. Y is a three-state output, meaning that if nothing is to be moved onto the Y bus during the current operation, it can be held in the high-impedence state.

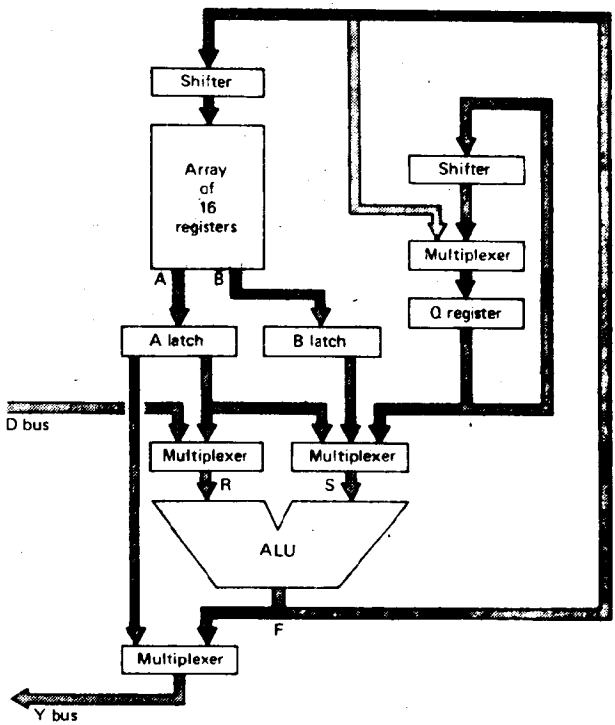

Figure 1.4. Organization of the 2901.

The 2901's external connections are shown in Figure 1.5. The inputs are the 4-bit D bus and 20 control lines. The outputs are the 4-bit Y bus and six status or condition signals. The four lines at the bottom are sometimes inputs, sometimes outputs, and sometimes neither (i.e., in a high-impedence state), depending on the functions specified by the I controls. For instance, if the I signals specify that the Q shifter should shift left by one bit, Q_0 is an input (allowing the low-order bit to be generated externally) and Q_3 is an output (allowing the displaced bit to be tested or fed into another 2901).

As a convention here, bits in registers or busses are numbered such that the least significant bit is numbered 0. Also, the overscore (e.g., $\overline{X}$) is used to indicate either (a) the one's complement of a value, (b) that an input control is active when in the low state, or (c) that an output condition is present when in the low state. The meaning that pertains in a particular situation should be obvious from the context.

Most of the control inputs should be obvious by inspecting Figure 1.5, with the exception of the I controls. The I lines are subdivided into three groups of three lines each. One group controls the multiplexers feeding the ALU. The second group controls the function of the ALU, and the third controls the two shifters, the

Q-register multiplexer, the Y-bus multiplexer, and the gating of the F bus into the register array.

Group one specifies the inputs to the ALU. The items that can be fed into the R side of the ALU are the A register (i.e., the word in the array selected by the A inputs), the D bus, and zero. The items that can be fed into the S side of the ALU are the A, B, and Q registers, and zero. However, only eight combinations are permitted (the combinations A-A, 0-0, A-0, and D-B are not provided).

Group two (three of the I lines) specifies the function to be performed by the ALU. The ALU can perform three arithmetic and five logic functions, but since the C_n (carry-in) input is used during the arithmetic functions, the ALU can perform six different arithmetic functions if C_n is used as a control input. The functions are

Add	$(F = R + S)$
Add plus one	$(F = R + S + 1)$
Subtract minus one	$(F = R - S - 1)$
Subtract	$(F = R - S)$
Subtract minus one	$(F = S - R - 1)$
Subtract	$(F = S - R)$
And	$(F = R \wedge S)$
Mask	$(F = \overline{R} \wedge S)$
Or	$(F = R \vee S)$
Exclusive or	$(F = R \veebar S)$
Exclusive nor	$(F = \overline{R \veebar S})$

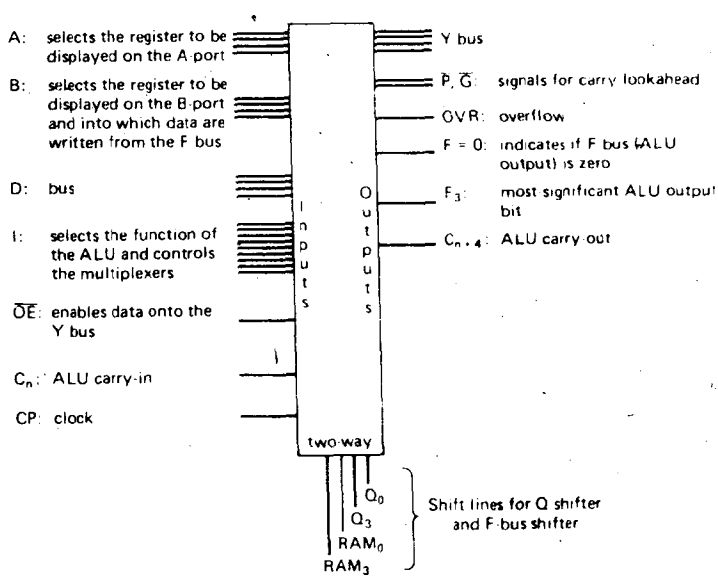

Figure 1.5. 2901 external connections.