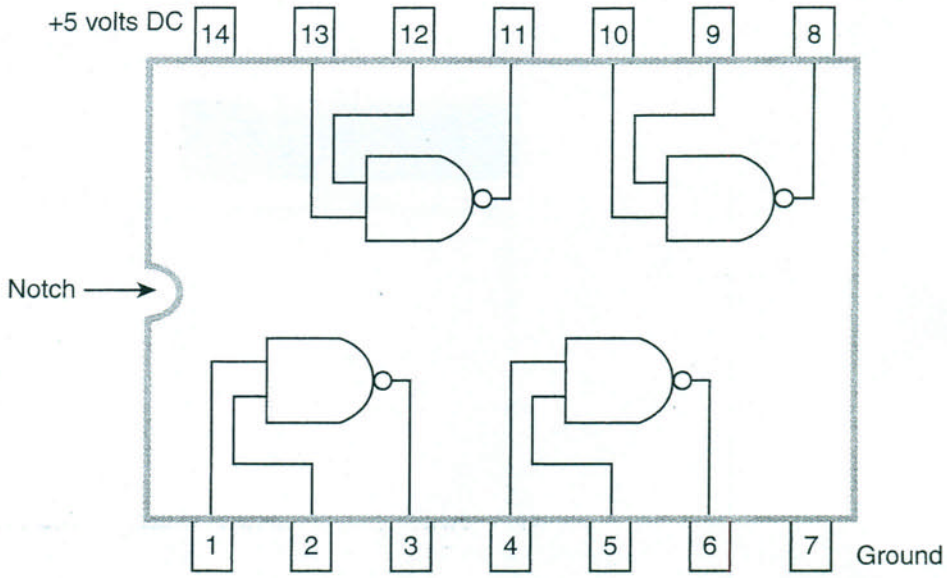


FIGURE 3.10 Simple SSI Integrated Circuit

to create larger modules, which, in turn, are used to implement various functions. The number of gates required to create these "building blocks" depends on the technology being used. Because the circuit technology is beyond the scope of this text, you are referred to the reading list at the end of this chapter for more information on this topic.

Typically, gates are not sold individually; they are sold in units called **integrated circuits** (**ICs**). A chip (a silicon semiconductor crystal) is a small electronic device consisting of the necessary electronic components (transistors, resistors, and capacitors) to implement various gates. As described in Chapter 1, components are etched directly on the chip, allowing them to be smaller and to require less power for operation than their discrete component counterparts. This chip is then mounted in a ceramic or plastic container with external pins. The necessary connections are welded from the chip to the external pins to form an IC. The first ICs contained very few transistors. As we learned in Chapter 1, the first ICs were called SSI chips and contained up to 100 electronic components per chip. We now have ultra large-scale integration (ULSI) with more than 1 million electronic components per chip. Figure 3.10 illustrates a simple SSI IC.

3.5 COMBINATIONAL CIRCUITS

Digital logic chips are combined to give us useful circuits. These logic circuits can be categorized as either **combinational logic** or **sequential logic**. This section introduces combinational logic. Sequential logic is covered in Section 3.6.

3.5.1 Basic Concepts

Combinational logic is used to build circuits that contain basic Boolean operators, inputs, and outputs. The key concept in recognizing a combinational circuit

is that an output is always based entirely on the given inputs. Thus, the output of a combinational circuit is a function of its inputs, and the output is uniquely determined by the values of the inputs at any given moment. A given combinational circuit may have several outputs. If so, each output represents a different Boolean function.

3.5.2 Examples of Typical Combinational Circuits

Let's begin with a very simple combinational circuit called a **half-adder**. Consider the problem of adding two binary digits together. There are only three things to remember: $0 + 0 = 0$, $0 + 1 = 1 + 0 = 1$, and $1 + 1 = 10$. We know the behavior this circuit exhibits, and we can formalize this behavior using a truth table. We need to specify two outputs, not just one, because we have a sum and a carry to address. The truth table for a half-adder is shown in Table 3.9.

A closer look reveals that Sum is actually an XOR. The Carry output is equivalent to that of an AND gate. We can combine an XOR gate and an AND gate, resulting in the logic diagram for a half-adder shown in Figure 3.11.

The half-adder is a very simple circuit and not really very useful because it can only add two bits together. However, we can extend this adder to a circuit that allows the addition of larger binary numbers. Consider how you add base 10

Inputs		Outputs	
x	*y*	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

TABLE 3.9 Truth Table for a Half-Adder

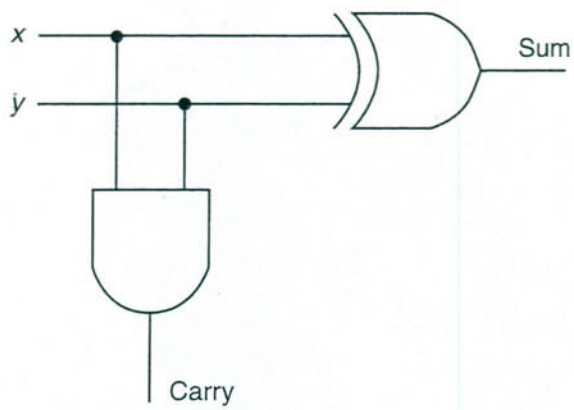


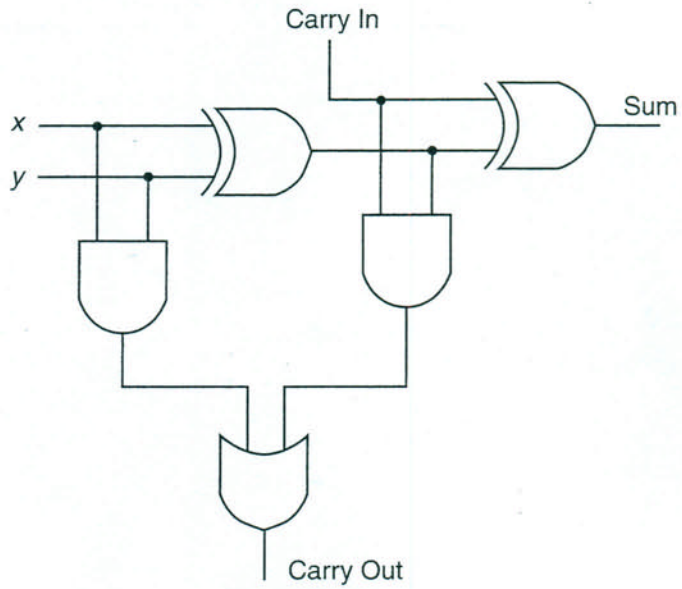
FIGURE 3.11 Logic Diagram for a Half-Adder

numbers: You add up the rightmost column, note the units digit, and carry the tens digit. Then you add that carry to the current column and continue in a similar fashion. We can add binary numbers in the same way. However, we need a circuit that allows three inputs (x, y, and Carry In) and two outputs (Sum and Carry Out). Figure 3.12 illustrates the truth table and corresponding logic diagram for a **full-adder**. Note that this full-adder is composed of two half-adders and an OR gate.

Given this full-adder, you may be wondering how this circuit can add binary numbers; it is capable of adding only three bits. The answer is, it can't. However, we can build an adder capable of adding two 16-bit words, for example, by replicating the above circuit 16 times, feeding the Carry Out of one circuit into the Carry In of the circuit immediately to its left. Figure 3.13 illustrates this idea. This type of circuit is called a **ripple-carry adder** because of the sequential generation of carries that "ripple" through the adder stages. Note that instead of drawing all the gates that constitute a full-adder, we use a **black box** approach to depict our

Inputs			Outputs	
x	y	Carry In	Sum	Carry Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a)



(b)

FIGURE 3.12 a) Truth Table for a Full-Adder
b) Logic Diagram for a Full-Adder

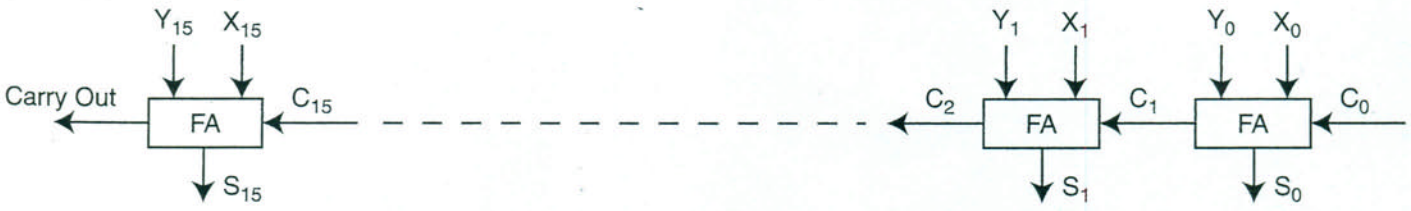


FIGURE 3.13 Logic Diagram for a Ripple-Carry Adder

adder. A black box approach allows us to ignore the details of the actual gates. We concern ourselves only with the inputs and outputs of the circuit. This is typically done with most circuits, including decoders, multiplexers, and adders, as we shall see very soon.

Because this adder is very slow, it is not normally implemented. However, it is easy to understand and should give you some idea of how addition of larger binary numbers can be achieved. Modifications made to adder designs have resulted in the carry-look-ahead adder, the carry-select adder, and the carry-save adder, as well as others. Each attempts to shorten the delay required to add two binary numbers. In fact, these newer adders achieve speeds 40% to 90% faster than the ripple-carry adder by performing additions in parallel and reducing the maximum carry path.

Adders are very important circuits—a computer would not be very useful if it could not add numbers. An equally important operation that all computers use frequently is decoding binary information from a set of n inputs to a maximum of 2^n outputs. A **decoder** uses the inputs and their respective values to select one specific output line. What do we mean by "select an output line"? It simply means that one unique output line is asserted, or set to 1, while the other output lines are set to 0. Decoders are normally defined by the number of inputs and the number of outputs. For example, a decoder that has 3 inputs and 8 outputs is called a 3-to-8 decoder.

We mentioned that this decoder is something the computer uses frequently. At this point, you can probably name many arithmetic operations the computer must be able to perform, but you might find it difficult to propose an example of decoding. If so, it is because you are not familiar with how a computer accesses memory.

All memory addresses in a computer are specified as binary numbers. When a memory address is referenced (whether for reading or for writing), the computer first has to determine the actual address. This is done using a decoder. Example 3.6 should clarify any questions you may have about how a decoder works and what it might be used for.

EXAMPLE 3.6 A 3-to-8 decoder circuit

Imagine memory consisting of 8 chips, each containing 8K bytes. Let's assume chip 0 contains memory addresses 0–8191, chip 1 contains memory addresses 8192–16,383, and so on. We have a total of 8K $\times$ 8, or 64K (65,536) addresses available. We will not write down all 64K addresses as binary numbers; however, writing a few addresses in binary form (as we illustrate in the following paragraphs) will illustrate why a decoder is necessary.

Given $64 = 2^6$ and $1K = 2^{10}$, then $64K = 2^6 \times 2^{10} = 2^{16}$, which indicates we need 16 bits to represent each address. If you have trouble understanding this, start with a smaller number of addresses. For example, if you have 4 addresses—addresses 0, 1, 2, and 3, the binary equivalent of these addresses is 00, 01, 10, and 11, requiring two bits. We know $2^2 = 4$. Now consider eight addresses. We have to be able to count from 0 to 7 in binary. How many bits does that require? The answer is 3. You can either write them all down, or you recognize that $8 = 2^3$. The exponent tells us the minimum number of bits necessary to represent the addresses.

All addresses on chip 0 have the format: 000*xxxxxxxxxxxxx*. Because chip 0 contains the addresses 0–8191, the binary representation of these addresses is in the range 0000000000000000 to 0001111111111111. Similarly, all addresses on chip 1 have the format 001*xxxxxxxxxxxxx*, and so on for the remaining chips. The leftmost 3 bits determine on which chip the address is actually located. We need 16 bits to represent the entire address, but on each chip, we only have 2^{13} addresses. Therefore, we need only 13 bits to uniquely identify an address on a given chip. The rightmost 13 bits give us this information.

When a computer is given an address, it must first determine which chip to use; then it must find the actual address on that specific chip. In our example, the computer would use the 3 leftmost bits to pick the chip and then find the address on the chip using the remaining 13 bits. These 3 high-order bits are actually used as the inputs to a decoder so the computer can determine which chip to activate for reading or writing. If the first 3 bits are 000, chip 0 should be activated. If the first 3 bits are 111, chip 7 should be activated. Which chip would be activated if the first 3 bits were 010? It would be chip 2. Turning on a specific wire activates a chip. The output of the decoder is used to activate one, and only one, chip as the addresses are decoded.

Figure 3.14 illustrates the physical components in a decoder and the symbol often used to represent a decoder. We will see how a decoder is used in memory in Section 3.6.

Another common combinational circuit is a **multiplexer**. This circuit selects binary information from one of many input lines and directs it to a single output line. Selection of a particular input line is controlled by a set of selection variables, or control lines. At any given time, only one input (the one selected) is

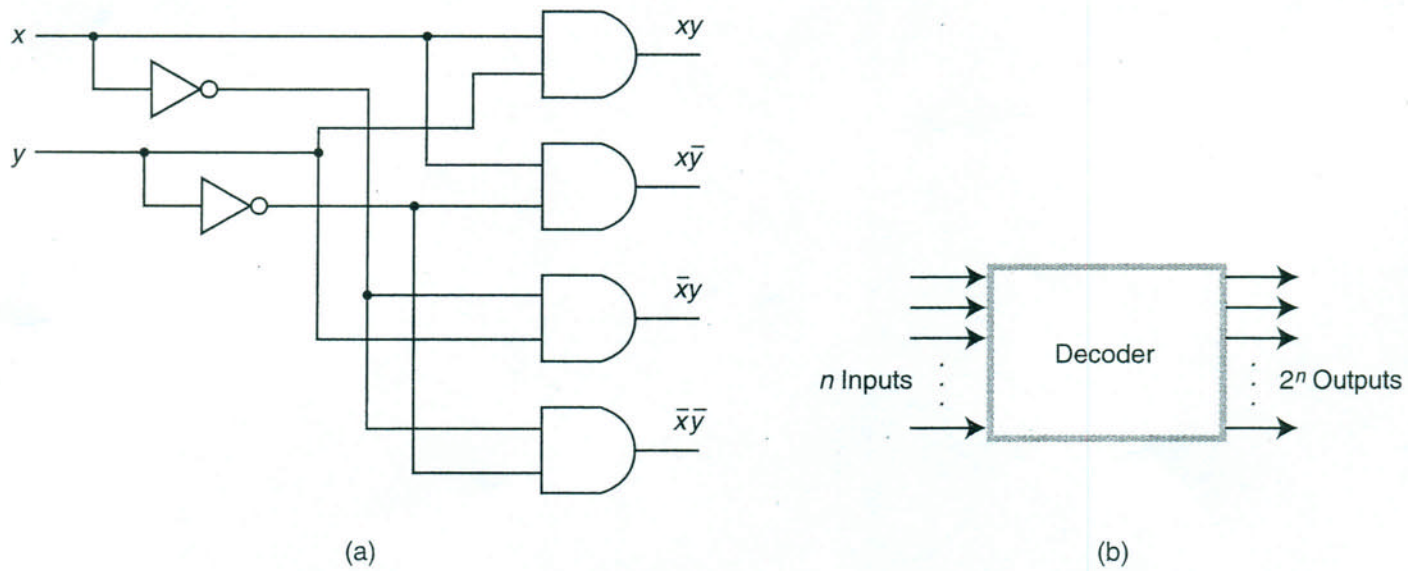


FIGURE 3.14 a) A Look Inside a Decoder
b) A Decoder Symbol

routed through the circuit to the output line. All other inputs are "cut off." If the values on the control lines change, the input actually routed through changes as well. Figure 3.15 illustrates the physical components in a multiplexer and the symbol often used to represent a multiplexer.

Can you think of some situations that require multiplexers? Time-sharing computers multiplex the input from user terminals. Modem pools multiplex the modem lines entering the computer.

Another useful set of combinational circuits to study includes a parity generator and a parity checker (recall we studied parity in Chapter 2). A **parity generator** is a circuit that creates the necessary parity bit to add to a word; a **parity checker** checks to make sure proper parity (odd or even) is present in the word, detecting an error if the parity bit is incorrect.

Typically, parity generators and parity checkers are constructed using XOR functions. Assuming we are using odd parity, the truth table for a parity generator for a 3-bit word is given in Table 3.10. The truth table for a parity checker to be used on a 4-bit word with 3 information bits and 1 parity bit is given in Table 3.11. The parity checker outputs a 1 if an error is detected and 0 otherwise. We leave it as an exercise to draw the corresponding logic diagrams for both the parity generator and the parity checker.

Bit shifting, moving the bits of a word or byte one position to the left or right is a very useful operation. Shifting a bit to the left takes it to the position of the next higher power of two. When the bits of an unsigned integer are shifted to the left by one position, it has the same effect as multiplying that integer by 2, but using significantly fewer machine cycles to do so. The leftmost or rightmost bit is lost after a left or right shift (respectively). Left shifting the nibble, 1101, changes it to 1010, and right shifting it produces 0110. Some buffers and encoders rely on shifters to produce a bit stream from a byte so that each bit can be processed in sequence. A 4-bit shifter is illustrated in Figure 3.16. When the control line, *S*, is

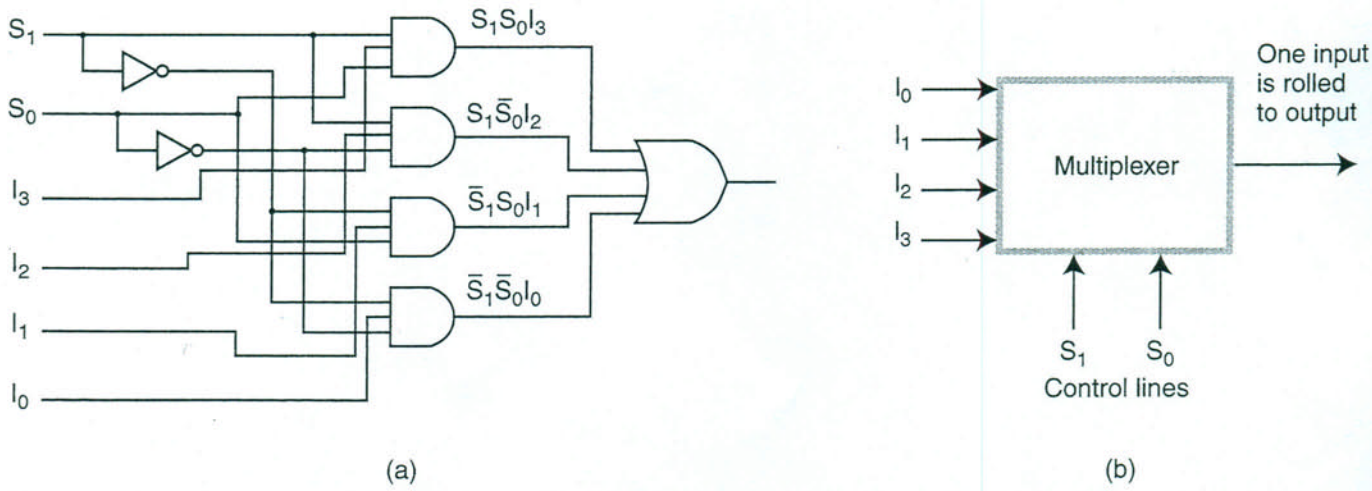


FIGURE 3.15 a) A Look Inside a Multiplexer
b) A Multiplexer Symbol

x	y	z	Parity Bit
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

TABLE 3.10 Parity Generator

x	y	z	P	Error detected?
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

TABLE 3.11 Parity Checker

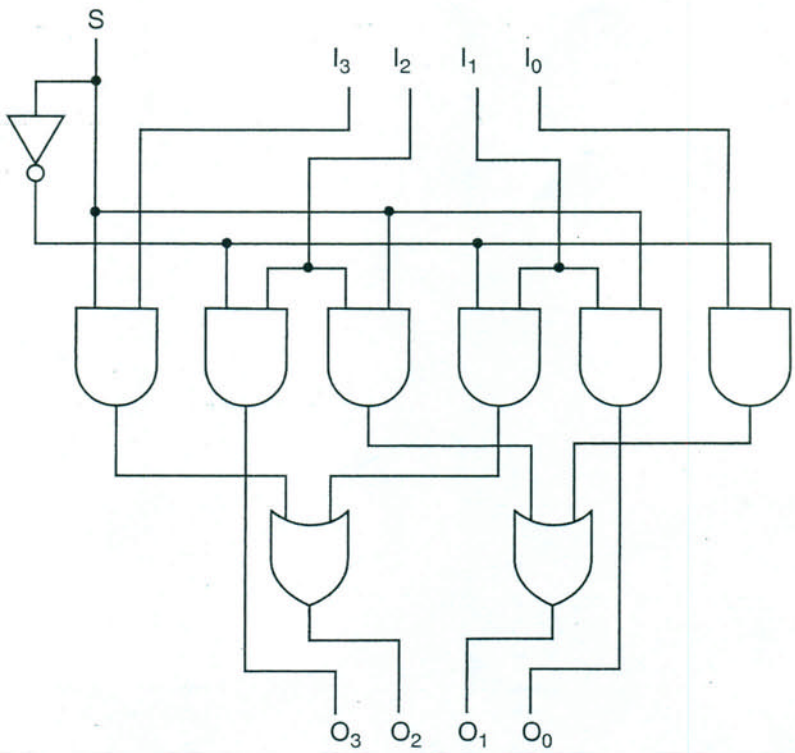


FIGURE 3.16 4-Bit Shifter