

Thus far, our discussion has focused on the theoretical design issues of computer systems. We have not yet addressed any of the actual hardware you might find inside a computer. This chapter changes that.

The following sections present different applications used either as stand-alone circuits or integrated into the circuitry of a processor. Each section will begin with a definition of a problem to be addressed. From this, a truth table will be developed which will then be converted into the corresponding boolean expression and finally a logic diagram.

8.1 Adders

Most mathematical operations can be handled with addition. For example, a simple subtraction can be performed by taking the two's complement of a binary value, and then adding it to the binary value from which it was to be subtracted. Two numbers can be multiplied using multiple additions. Counting either up or down (incrementing or decrementing) can be performed with additions of 1 or -1.

Chapter 3 showed that binary addition is performed just like decimal addition, the only difference being that decimal has 10 numerals while binary has 2. When adding two digits in decimal, a result greater than nine creates an "overflow", i.e., a one is added to the next position. For binary, a result greater than 1, e.g., 1+1=2, creates an overflow. This produces a sum of 0 with a carry of 1 to the next position.

$$\begin{array}{r}
 0 \\
 + 0 \\
 \hline
 0
 \end{array}
 \quad
 \begin{array}{r}
 0 \\
 + 1 \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 1 \\
 + 0 \\
 \hline
 1
 \end{array}
 \quad
 \begin{array}{r}
 1 \\
 1 \\
 + 1 \\
 \hline
 10
 \end{array}$$

Figure 8-1 Four Possible Results of Adding Two Bits

A well-defined process such as this is easily realized with digital logic. Figure 8-2 shows the block diagram of a system that takes two binary inputs, A and B, and adds them together producing a bit for the

sum and a bit indicating whether or not a carry occurred. This well-known circuit is commonly referred to as a *half-adder*.

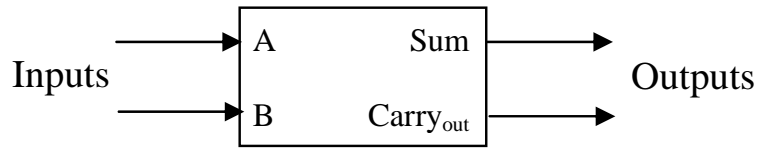


Figure 8-2 Block Diagram of a Half Adder

With two inputs, there are four possible combinations of ones and zeros that can be input.

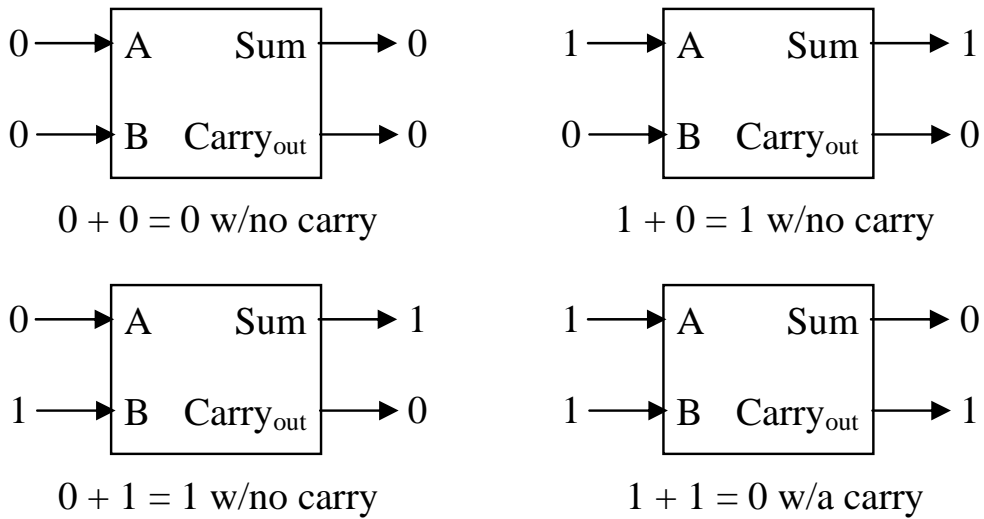


Figure 8-3 Four Possible States of a Half Adder

A truth table can be derived from Figure 8-3 from which the boolean expressions can be developed to realize this system.

A	B	Sum	Carry _{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

The simplicity of a two-input truth table makes the use of a Karnaugh map unnecessary. Examining the Sum column shows that we should have an output of one when A=0 and B=1 and when A=1 and B=0. This gives us the following SOP expression:

$$\text{Sum} = \overline{A} \cdot B + A \cdot \overline{B}$$

Note that the output Sum is also equivalent to the 2-input XOR gate.

For $\text{Carry}_{\text{out}}$, the output equals 1 only when both A and B are equal to one. This matches the operation of the AND gate.

$$\text{Carry}_{\text{out}} = A \cdot B$$

Figure 8-4 presents the logic circuit for the half adder.

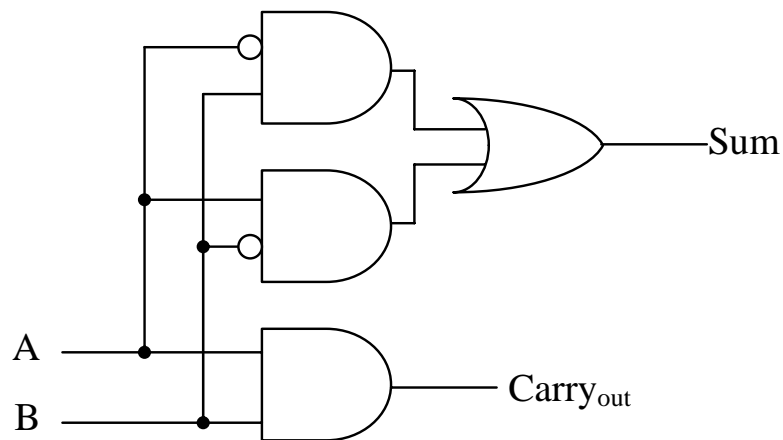


Figure 8-4 Logic Circuit for a Half Adder

The half-adder works fine if we're trying to add two bits together, a situation that typically occurs only in the rightmost column of a multi-bit addition. The remaining columns are responsible for adding two bits along with a possible carry from a previous column.

For example, assume we want to add two four bit numbers, $A = 0110_2$ and $B = 1011_2$. The addition would go something like this.

$$\begin{array}{r} 1\ 1\ 1 \\ 0\ 1\ 1\ 0 \\ +\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 0\ 1 \end{array}$$

Adding the least significant bits together is exactly like the half-adder described above. The inputs to the half-adder are the least significant bits of each number, and the outputs are the least significant digit of the sum and a possible carry to the next column.

What is needed for the remaining columns is an adder similar to the half-adder that can add two bits and the carry from the previous column to produce a Sum and the Carry_{out} to the next column. Figure 8-5 represents this operation where A_n is the bit in the n^{th} position of A, B_n is the bit in the n^{th} position of B, and S_n is the bit in the n^{th} position in the resulting sum, S.

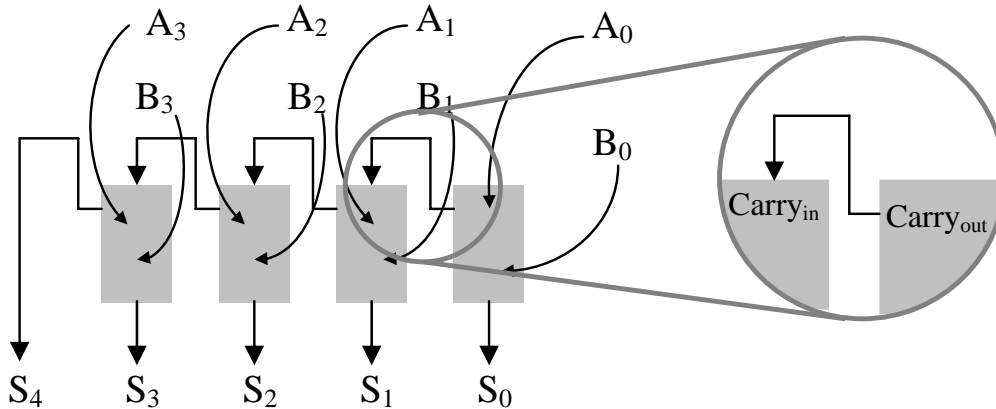


Figure 8-5 Block Diagram of a Multi-bit Adder

Notice that a Carry_{out} from the addition of a pair of bits goes into the carry input of the addition for the next bit. We will call this input Carry_{in}. This implies that we need to create a circuit that can add three bits, A_n , B_n , and the carry result from the $n-1$ position. This system has two outputs, the resulting sum and the carry to the $n+1$ position. The resulting circuit is called a *full adder*. A block diagram of the full adder is shown in Figure 8-6.

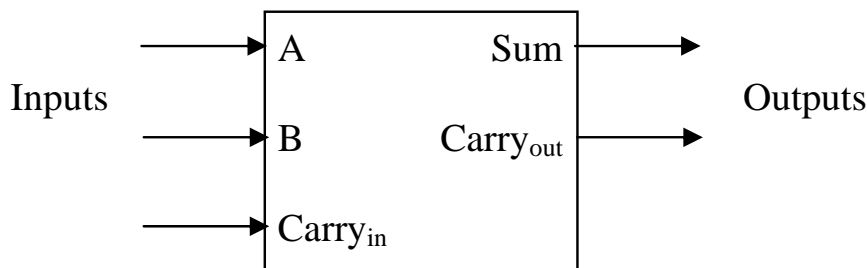


Figure 8-6 Block Diagram of a Full Adder

With three inputs, there are $2^3 = 8$ possible combinations of ones and zeros that could be input to our full adder. Table 8-1 below lists these combinations along with the results of their addition which range from 0 to 3_{10} .

Table 8-1 Addition Results Based on Inputs of a Full Adder

Inputs			Result	
A	B	Carry _{in}	Decimal	Binary
0	0	0	0 ₁₀	00 ₂
0	0	1	1 ₁₀	01 ₂
0	1	0	1 ₁₀	01 ₂
0	1	1	2 ₁₀	10 ₂
1	0	0	1 ₁₀	01 ₂
1	0	1	2 ₁₀	10 ₂
1	1	0	2 ₁₀	10 ₂
1	1	1	3 ₁₀	11 ₂

The two-digit binary result in the last column of this table can be broken into its components, the sum and a carry to the next bit position. This gives us two truth tables, one for the Sum and one for the Carry_{out}.

Table 8-2 Sum and Carry_{out} Truth Tables for a Full Adder

A	B	Carry _{in}	Sum	A	B	Carry _{in}	Carry _{out}
0	0	0	0	0	0	0	0
0	0	1	1	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	1	1	0	1
1	1	1	1	1	1	1	1

With three inputs, a Karnaugh map can be used to create the logic expressions. One Karnaugh map will be needed for each output of the circuit. Figure 8-7 presents the Karnaugh maps for the Sum and the Carry_{out} outputs of our full adder where C_{in} represents the Carry_{in} input.

The Carry_{out} Karnaugh map has three rectangles, each containing two cells and all three overlapping on the cell defined by A=1, B=1, and C_{in}=1. By using the process presented in Chapter 7, we can derive the three products for the SOP expression defining Carry_{out}.

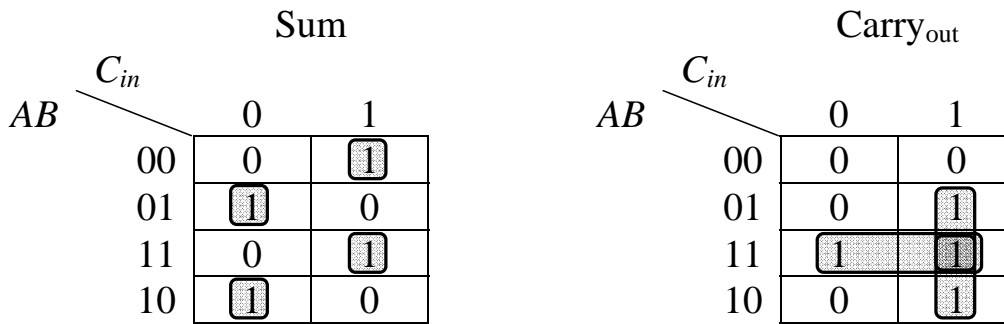


Figure 8-7 Sum and Carry_{out} Karnaugh Maps for a Full Adder

Rectangle 1:

A	B	C_{in}	
0	1	1	$B \cdot C_{in}$
1	1	1	

Rectangle 2:

A	B	C_{in}	
1	1	0	$A \cdot B$
1	1	1	

Rectangle 3:

A	B	C_{in}	
1	1	1	$A \cdot C_{in}$
1	0	1	

$$\text{Carry}_{out} = B \cdot C_{in} + A \cdot B + A \cdot C_{in}$$

The Karnaugh map for the Sum output is less promising. In fact, there is no way to make a more complex 3-input Karnaugh map than the one that exists for the Sum of the full adder. The addition or removal of a '1' in any cell of the map will result in a simpler expression. The four single-cell rectangles result in the four products of the SOP expression for the Sum output shown below.

$$\text{Sum} = \overline{A} \cdot \overline{B} \cdot C_{in} + \overline{A} \cdot B \cdot \overline{C_{in}} + A \cdot B \cdot C_{in} + A \cdot \overline{B} \cdot \overline{C_{in}}$$

Figure 8-8 presents the circuit for the full adder.

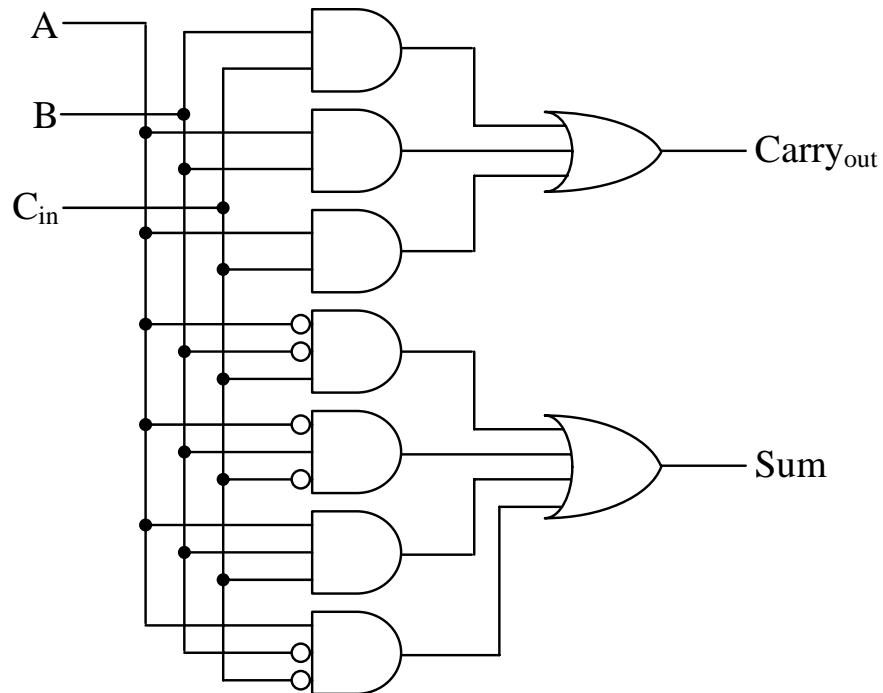


Figure 8-8 Logic Circuit for a Full Adder

Now we have the building blocks to create an adder of any size. For example, a 16-bit adder is made by using a half adder for the least significant bit followed by fifteen full adders daisy-chained through their carries for the remaining fifteen bits.

This method of creating adders has a slight drawback, however. Just as with the addition of binary numbers on paper, the sum of the higher-order bits cannot be determined until the carry from the lower-order bits has been calculated and propagated through the higher stages. Modern adders use additional logic to predict whether the higher-order bits should expect a carry or not well before the sum of the lower-order bits is calculated. These adders are called *carry look ahead adders*.

8.2 Seven-Segment Displays

Most everyone has seen a seven-segment display. It is the most common way to display time on a clock radio, and it is one of the easiest ways to implement a numeric output for a digital circuit. The use of seven-segment displays is so extensive that special integrated circuits (ICs) have been developed to take a four-bit binary numeric input and create the output signals necessary to drive the display.

A seven-segment display consists of seven long, thin LEDs arranged in the pattern of an eight. Each segment is controlled individually so that any decimal digit can be displayed. Using a combination of both upper- and lower-case letters, A, B, C, D, E, and F can be displayed too allowing a seven-segment display to output all of the hexadecimal digits too.

Figure 8-9 shows a diagram of the typical seven-segment display with each segment lettered for identification.

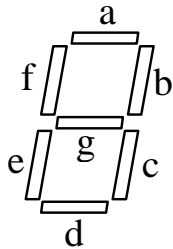


Figure 8-9 Diagram of a Seven-Segment Display

To make a digit appear, the user must know which segments to turn on and which to leave off. For example, to display a '1', we need to turn on segments b and c and leave the other segments off.

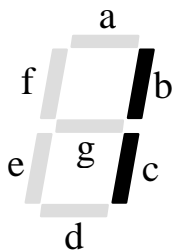


Figure 8-10 A Seven-Segment Display Displaying a Decimal '1'

This means that the binary circuit driving segment 'a' would output 0, the binary circuit driving segment 'b' would output 1, the binary circuit driving segment 'c' would output 1, the binary circuit driving segment 'd' would output 0, and so on. Displaying the number 2 would result in the following:

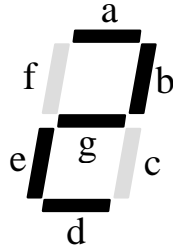


Figure 8-11 A Seven-Segment Display Displaying a Decimal '2'

The binary inputs for this are $a=1$, $b=1$, $c=0$, $d=1$, $e=1$, $f=0$, and $g=1$.

The digital circuitry used to drive a seven-segment display consists of seven separate digital circuits, one for each LED. Each circuit takes as its input the binary nibble that is to be displayed. For example, if the binary nibble $0010_2 = 2_{10}$ is input to the digital circuitry driving the display, then the digital circuit for segment 'a' would output 1, the digital circuit for segment 'b' would output 1, the digital circuit for segment 'c' would output 0, and so on. Figure 8-12 shows a block diagram of the seven-segment display driver.

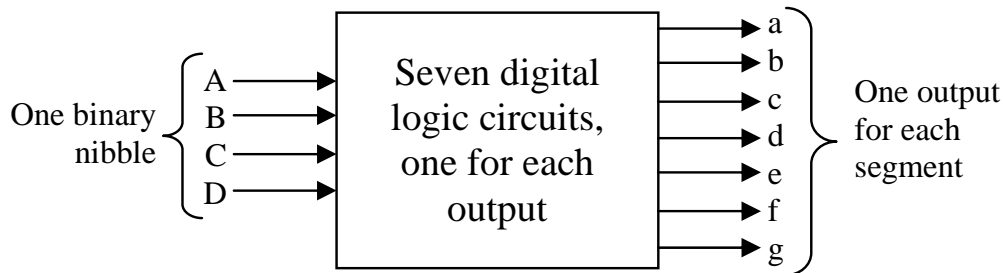


Figure 8-12 Block Diagram of a Seven-Segment Display Driver

To begin with, we need seven truth tables, one for the output of each circuit. The individual bits of the number to be displayed will be used for the inputs. Next, we need to know which segments are to be on and which are to be off for each digit. Figure 8-13 shows the bit patterns for each hexadecimal digit.

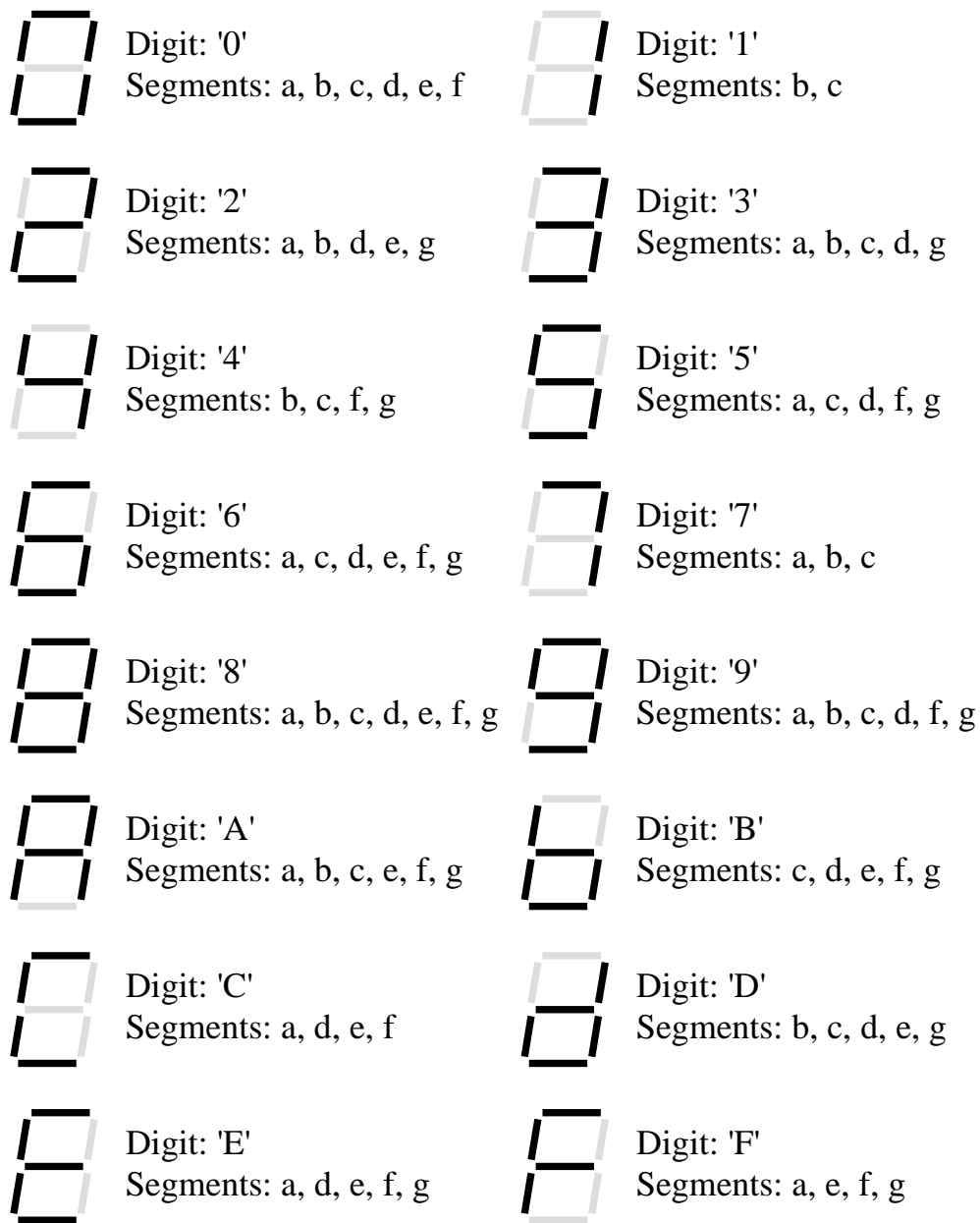


Figure 8-13 Segment Patterns for all Hexadecimal Digits

Using the information from Figure 8-13, we can build the seven truth tables. The truth table in Figure 8-14 combines all seven truth tables along with a column indicating which digit is displayed for the corresponding set of inputs. Note that the capital letters denote the input signals while the lower case letters identify the segments of the seven-segment display.

Inputs				Hex Value	Segments						
A	B	C	D		a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	1	0	1	1	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1
0	0	1	1	3	1	1	1	1	0	0	1
0	1	0	0	4	0	1	1	0	0	1	1
0	1	0	1	5	1	0	1	1	0	1	1
0	1	1	0	6	1	0	1	1	1	1	1
0	1	1	1	7	1	1	1	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1
1	0	0	1	9	1	1	1	1	0	1	1
1	0	1	0	A	1	1	1	0	1	1	1
1	0	1	1	B	0	0	1	1	1	1	1
1	1	0	0	C	1	0	0	1	1	1	0
1	1	0	1	D	0	1	1	1	1	0	1
1	1	1	0	E	1	0	0	1	1	1	1
1	1	1	1	F	1	0	0	0	1	1	1

Figure 8-14 Seven Segment Display Truth Table

The next step is to create a Karnaugh map for each of the seven segments in order to determine the minimum SOP expression and digital circuit to be used to drive each segment. Here we will only do one of the circuits, segment e. Figure 8-15 takes the column for segment e and maps it into a four-by-four Karnaugh map.

		<i>CD</i>			
		<i>AB</i>	00	01	11
00	00	1	0	0	1
01	01	0	0	0	1
11	11	1	1	1	1
10	10	1	0	1	1

Figure 8-15 Karnaugh Map for Segment 'e'

Next, we need to identify the optimum set of rectangles for the Karnaugh map. These rectangles are shown in Figure 8-16.

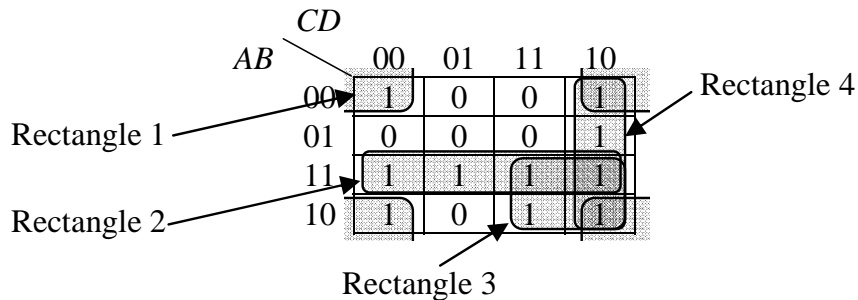


Figure 8-16 Karnaugh Map for Segment 'e'

Now that we have the rectangles, we can derive each product for the SOP expression.

Rectangle 1:	A	B	C	D	
	0	0	0	0	Product: $\bar{B} \cdot \bar{D}$
	1	0	0	0	
	0	0	1	0	
	1	0	1	0	

Rectangle 2:	A	B	C	D	
	1	1	0	0	
	1	1	0	1	Product: $A \cdot B$
	1	1	1	1	
	1	1	1	0	

Rectangle 3:	A	B	C	D	
	1	1	1	1	
	1	1	1	0	Product: $A \cdot C$
	1	0	1	1	
	1	0	1	0	

Rectangle 4:	A	B	C	D	
	0	0	1	0	
	0	1	1	0	Product: $C \cdot \bar{D}$
	1	1	1	0	
	1	0	1	0	

Our final SOP expression is then the OR of these four products.

$$\text{Segment } e = (\bar{B} \cdot \bar{D}) + (A \cdot B) + (A \cdot C) + (C \cdot \bar{D})$$

Figure 8-17 presents the digital logic that would control segment e of the seven-segment display. The design of the display driver is not complete, however, as there are six more logic circuits to design.

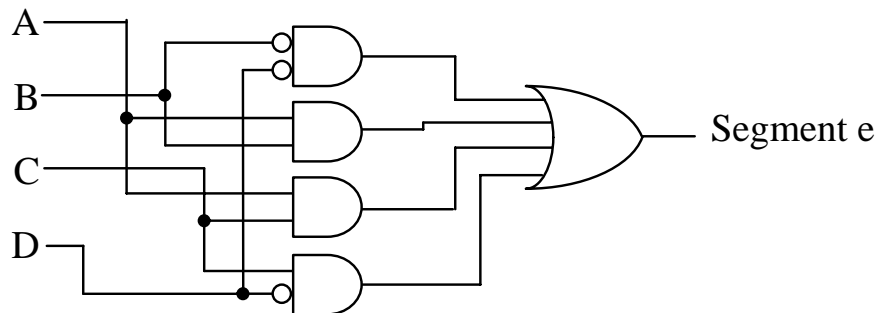


Figure 8-17 Logic Circuit for Segment e of 7-Segment Display

8.3 Active-Low Signals

Computer systems are composed of numerous subsystems, some of which may be idle, some of which may be operating independent of the processor, and some of which may be in direct contact with the processor. For systems that are in direct contact with the processor, only one may be enabled at any one time. For example, although a computer system may have multiple memory devices, when a piece of data is stored, it is sent to only one of the modules while the other modules must remain idle.

A scheme is needed to select or enable a specific device or to route data to an appropriate subsystem. This scheme is implemented with a separate binary line that is connected to each subsystem where one of the binary values enables the subsystem and the other binary value disables it, i.e., an on/off binary control.

Our discussion previous to this suggests that the "on" signal is equivalent to a logic 1, but for a number of reasons, the standard method of enabling a device is *not* to send a logic 1. Instead, due to the nature of electronics, it is standard practice to enable devices with a logic 0 and disable them with a logic 1. This is called *active-low* operation, i.e., the device is active when its enable signal is low or logic 0. The device is inactive when the enable is high or logic 1.

There is a special notation that goes along with active-low signals. If you see an input to or an output from a system labeled with the NOT or inverter bar over it, then that signal is an active-low signal. Sometimes,

the line that is drawn into or out of the system diagram will also pass through an inverter circle to identify the signal as active-low. For example, in the system shown in Figure 8-18, the input C and the output EN are both active-low.

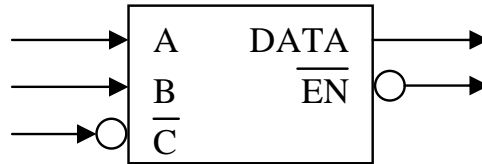


Figure 8-18 Labeling Conventions for Active-Low Signals

Many of the circuits discussed in the following sections of this chapter use active-low signals. Any time this happens, the notation shown above will be used.

8.4 Decoders

One application where digital signals are used to enable a device might be to tell that device that all of the conditions are correct for its operation. For example, the magnetron in a microwave is enabled when a number of conditions are met, i.e., when the timer is running *and* the start button is pushed *and* the oven door is closed.

This method of enabling a device based on the condition of a number of inputs is common in digital circuits. One common application is in the processor's interface to memory. It is used to determine which memory device will contain a piece of data.

In the microwave example, the sentence used to describe the enabling of the magnetron joined each of the inputs with the word "and". Therefore, the enabling circuit for the magnetron should be realized with an AND gate as shown in Figure 8-19.

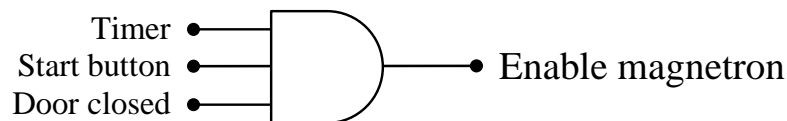


Figure 8-19 Sample Circuit for Enabling a Microwave

There are many other types of digital systems that enable a process based on a specific combination of ones and zeros from multiple inputs. For example, an automobile with a manual transmission enables the

starter when the clutch is pressed *and* the key is turned in the ignition. A soda drops from a vending machine when a proper amount of money is inserted *and* a button is pushed *and* the machine is not out of the selected soda.

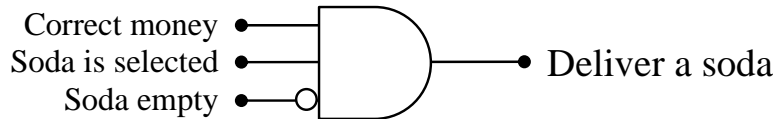


Figure 8-20 Sample Circuit for Delivering a Soda

An AND gate outputs a one only when all of its inputs equal one. If one or more inputs are inverted, the output of the AND gate is one if and only if all of the inputs without inverters equal one and all of the inputs with inverters equal zero.

The truth table for this type of circuit will have exactly one row with an output of one while all of the other rows output a zero. The row with the one can change depending on which inputs are inverted. For example, Figure 8-21 presents the truth table for the circuit that enables a device when A and B are true but C is false.

A	B	C	EN
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Figure 8-21 Truth Table to Enable a Device for A=1, B=1, & C=0

When SOP expressions were introduced in Chapter 6, we found that each row of a truth table with a '1' output corresponded to a unique product. Therefore, the circuit that is used to enable a device can be realized with a single AND gate. The conditions that activate that AND gate are governed by the pattern of inverters at its inputs.

When we apply the tools of Chapter 6 to the truth table in Figure 8-21, we get the following boolean expression.

$$EN = A \cdot B \cdot \overline{C}$$

Decoder circuits are a group of enable circuits that have an individual output that satisfies each row of the truth table. In other words, a decoder has a unique output for each combination of ones and zeros possible at its inputs.

For example, a 2-input decoder circuit with inputs A and B can have an output that is 1 only when A=0 and B=0, an output that is 1 only when A=0 and B=1, an output that is 1 only when A=1 and B=0, and an output that is 1 only when A=1 and B=1. The boolean expressions that satisfy this decoder circuit are:

$$EN_0 = \overline{A} \cdot \overline{B} \quad EN_1 = \overline{A} \cdot B \quad EN_2 = A \cdot \overline{B} \quad EN_3 = A \cdot B$$

This two-input circuit is called a *1-of-4 decoder* due to the fact that exactly one of its four outputs will be enabled at any one time. A change at any of the inputs will change which output is enabled, but never change the fact that only one is enabled. As for the logic circuit, it has four AND gates, one satisfying each of the above boolean expressions. Figure 8-22 presents this digital circuit.

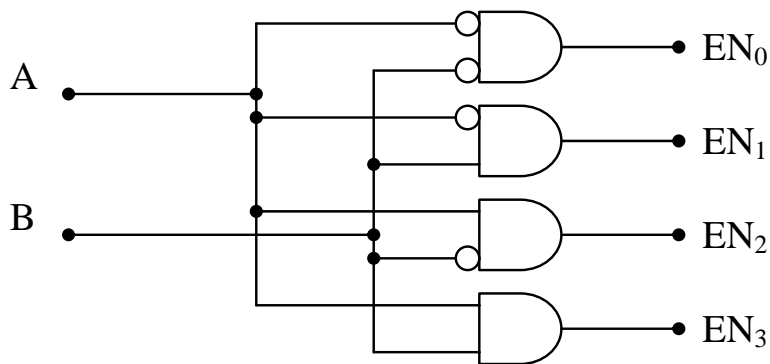


Figure 8-22 Digital Circuit for a 1-of-4 Decoder

As suggested in the previous section, it is common to implement enable signals as active-low due to the nature of electronics. In order to do this, the output of each AND gate needs to be inverted giving us a single condition where an output of zero occurs. This means that the active-low decoder circuit is implemented with NAND gates as shown in Figure 8-23. Notice the bar over the output names. This indicates the active-low nature of these signals.

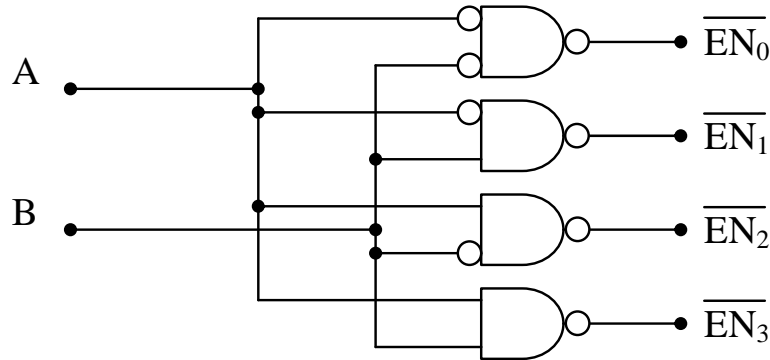


Figure 8-23 Digital Circuit for an Active-Low 1-of-4 Decoder

Decoder circuits can have any number of inputs. The number of outputs, however, is directly related to the number of inputs. If, for example, a decoder has four input signals, A, B, C, and D, then there are $2^4 = 16$ unique combinations of ones and zeros, each of which requires a NAND gate for its output. A decoder with four inputs is called a 1-of-16 decoder. Figure 8-24 presents the truth table for an active-low 1-of-8 decoder with three inputs.

A	B	C	\overline{EN}_0	\overline{EN}_1	\overline{EN}_2	\overline{EN}_3	\overline{EN}_4	\overline{EN}_5	\overline{EN}_6	\overline{EN}_7
0	0	0	0	1	1	1	1	1	1	1
0	0	1	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

Figure 8-24 Truth Table for an Active-Low 1-of-8 Decoder

8.5 Multiplexers

A multiplexer, sometimes referred to as a MUX, is a device that uses a set of control inputs to select which of several data inputs is to be connected to a single data output. With n binary "select lines," one of 2^n data inputs can be connected to the output. Figure 8-25 presents a block diagram of a multiplexer with three select lines, S_2 , S_1 , and S_0 , and eight data lines, D_0 through D_7 .

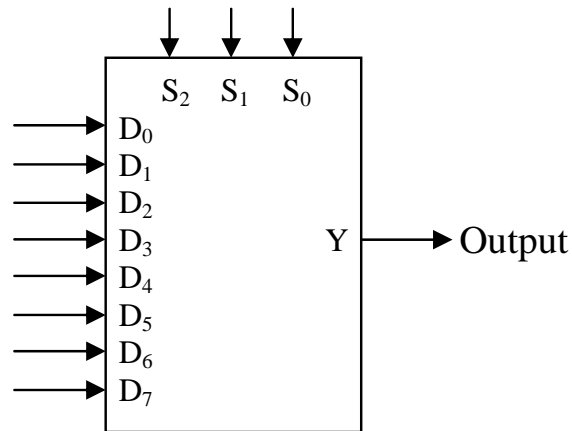


Figure 8-25 Block Diagram of an Eight Channel Multiplexer

A multiplexer acts like a television channel selector. All of the stations are broadcasted constantly to the television's input, but only the channel that has been selected is displayed. As for the eight-channel multiplexer in Figure 8-25, its operation can be described with the truth table shown in Figure 8-26.

S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

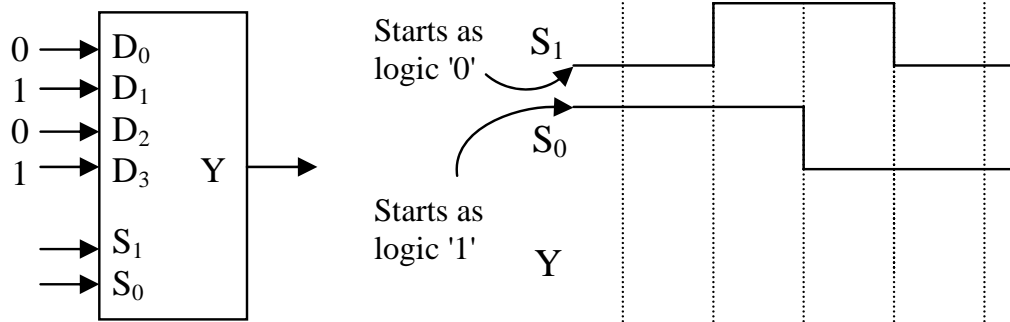
Figure 8-26 Truth Table for an Eight Channel Multiplexer

For example, if the selector inputs are set to $S_2 = 0$, $S_1 = 1$, and $S_0 = 1$, then the data present at D_3 will be output to Y. If $D_3 = 0$, then Y will output a 0.

The number of data inputs depends on the number of selector inputs. For example, if there is only one selector line, S_0 , then there can only be two data inputs D_0 and D_1 . When S_0 equals zero, D_0 is routed to the output. When S_0 equals one, D_1 is routed to the output. Two selector lines, S_1 and S_0 , allow for four data inputs, D_0 , D_1 , D_2 , and D_3 .

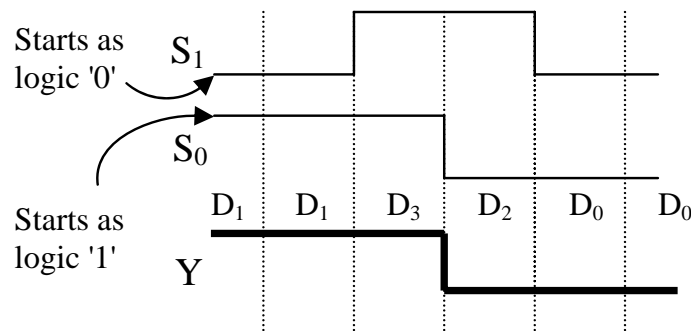
Example

For the multiplexer shown below, sketch the output waveform Y for the inputs S_1 and S_0 shown in the graph next to it. Assume S_1 is the most significant bit.

*Solution*

The decimal equivalent to the binary value input by the selector inputs indicates the subscript of the channel being connected to the output. For example, when S_1 equals one and S_0 equals zero, then their decimal equivalent is $10_2 = 2_{10}$. Therefore, D_2 is connected to the output. Since D_2 equals zero, then Y is outputting a zero.

The graph below shows the values of Y for each of the states of S_1 and S_0 . The labels inserted above the waveform for Y indicate which channel is connected to Y at that time.

**8.6 Demultiplexers**

The previous section described how multiplexers select one channel from a group of input channels to be sent to a single output. Demultiplexers take a single input and select one channel out of a group of output channels to which it will route the input. It's like having multiple printers connected to a computer. A document can only be printed to one of the printers, so the computer selects one out of the group of printers to which it will send its output.

The design of a demultiplexer is much like the design of a decoder. The decoder selected one of many outputs to which it would send a zero. The difference is that the demultiplexer sends data to that output rather than a zero.

The circuit of a demultiplexer is based on the non-active-low decoder where each output is connected to an AND gate. An input is added to each of the AND gates that will contain the demultiplexer's data input. If the data input equals one, then the output of the AND gate that is selected by the selector inputs will be a one. If the data input equals zero, then the output of the selected AND gate will be zero. Meanwhile, all of the other AND gates output a zero, i.e., no data is passed to them. Figure 8-27 presents a demultiplexer circuit with two selector inputs.

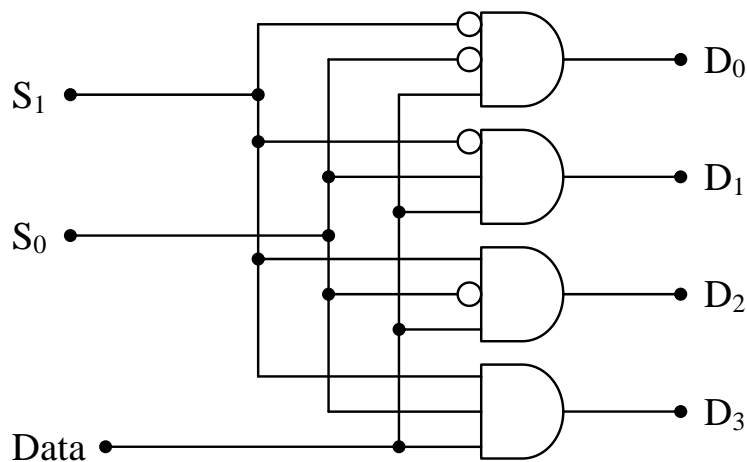


Figure 8-27 Logic Circuit for a 1-Line-to-4-Line Demultiplexer

In effect, the select lines, S_0, S_1, \dots, S_n , "turn on" a specific AND gate that passes the data through to the selected output. In Figure 8-27, if $S_1=0$ and $S_0=1$, then the D_1 output will match the input from the Data line and outputs D_0, D_2 , and D_3 will be forced to have an output of zero. If $S_1=0, S_0=1$, and $Data=0$, then $D_1=0$. If $S_1=0, S_0=1$, and $Data=1$, then $D_1=1$. Figure 8-28 presents the truth table for the 1-line-to-4-line demultiplexer shown in Figure 8-27.

S_1	S_0	Data	D_0	D_1	D_2	D_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

Figure 8-28 Truth Table for a 1-Line-to-4-Line Demultiplexer

8.7 Integrated Circuits

It may appear that much of our discussion up to this point has been theoretical, but in reality, each of the circuits we've presented can easily be implemented given the right tools. Prototypes used to test or verify circuit designs can be made by wiring together small plastic chips that offer access to the internal components through thin metal pins. These chips, called integrated circuits (ICs), come in a wide variety of shapes, sizes, and pin configurations. Figure 8-29 presents a sample of some ICs.

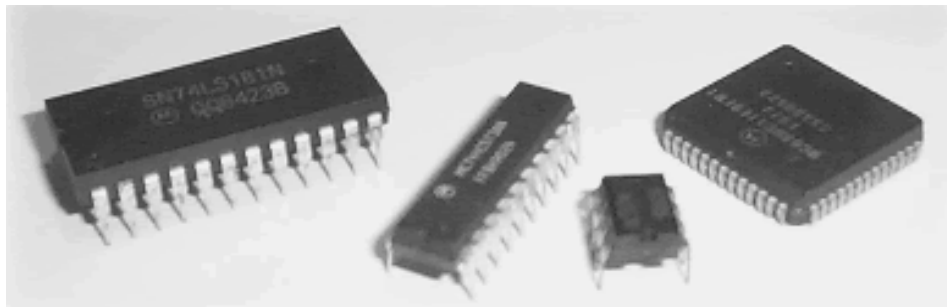


Figure 8-29 Examples of Integrated Circuits

Connecting the metal pins of these chips with other metal pins from the same chip or additional chips is what allows us to create digital circuits.

As for what we are connecting to them, the metal pins of the ICs allow us access to the internal circuitry such as the inputs and outputs of logic gates. Detailed information is available for all ICs from the manufacturer allowing designers to understand the internal circuitry.

The documentation defining the purpose of each pin of the IC is usually referred to as the IC's "pin-out description." It provides information not only on the digital circuitry, but also any power requirements needed to operate the IC.

Figure 8-30 presents an example of the pin-out of a quad dual-input NAND gate chip, commonly referred to as a 7400.

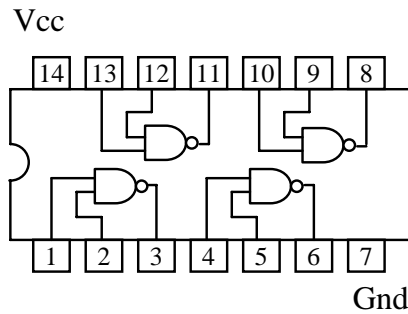


Figure 8-30 Pin-out of a Quad Dual-Input NAND Gate IC (7400)

Note that the pins are numbered. In order to properly use one of these ICs, you must be able to identify the pin numbers. To help you do this, the manufacturers provide an identification for the first pin, referred to as "pin 1", on every IC. The Figure 8-31 presents some of the ways this pin is identified.

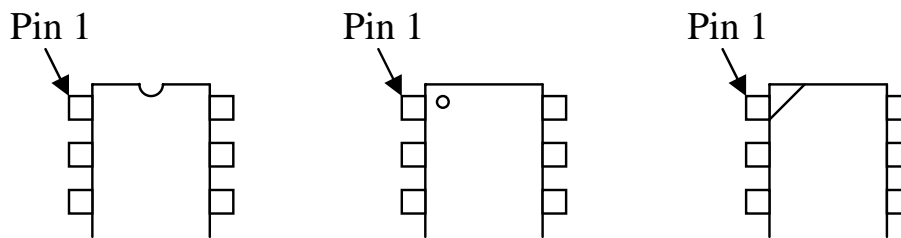


Figure 8-31 Sample Pin 1 Identifications

The pins are then numbered counter-clockwise around the chip. You can see this in the numbering of the pins in Figure 8-30.

Many circuits are then built and tested using prototype boards or protoboards. A protoboard is a long, thin plastic board with small holes in it that allow ICs and short wire leads to be plugged in. A generic protoboard is shown in Figure 8-32.

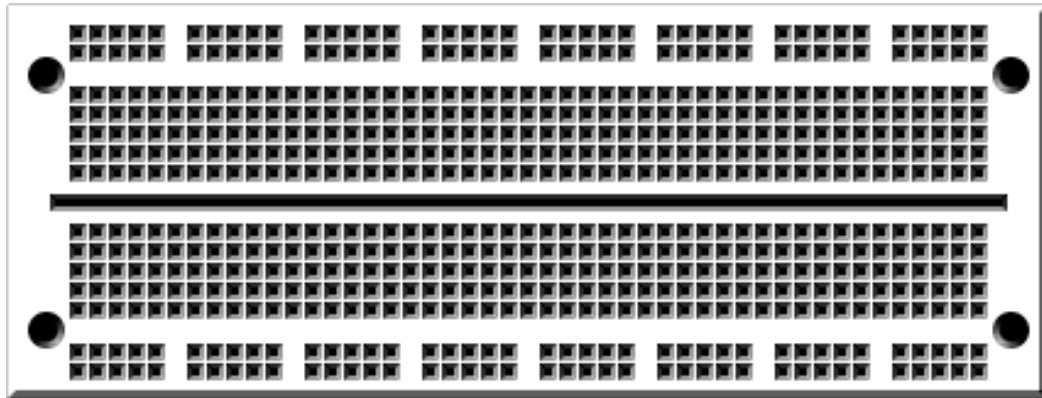


Figure 8-32 Generic Protoboard

Behind the sockets there is a pattern of metal connections that provides an electrical path between certain sockets on the protoboard. This allows us to interconnect and power ICs. Figure 8-33 below shows how the sockets are connected electrically.

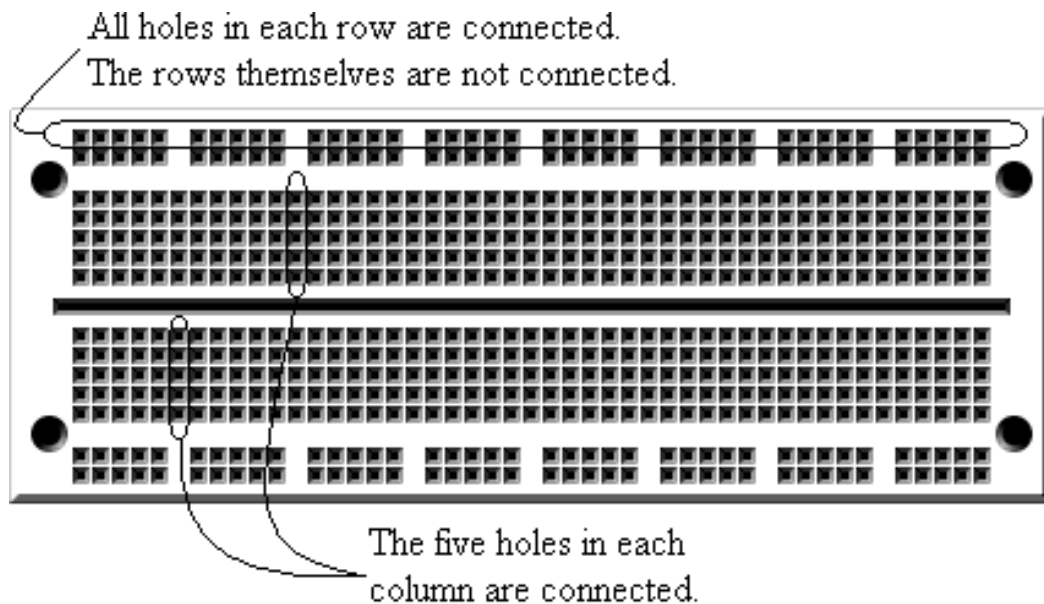


Figure 8-33 Generic Protoboard Internal Connections

The protoboard allows the user to insert an IC so that it straddles the gap running along the center of the board. Wires can then be used to connect the pins to other sockets on the protoboard. The rows on the top and bottom edges of the board in Figure 8-32 are used to connect

power (Vcc) and ground (GND) to the IC. Figure 8-34 shows a sample circuit with two chips wired together.

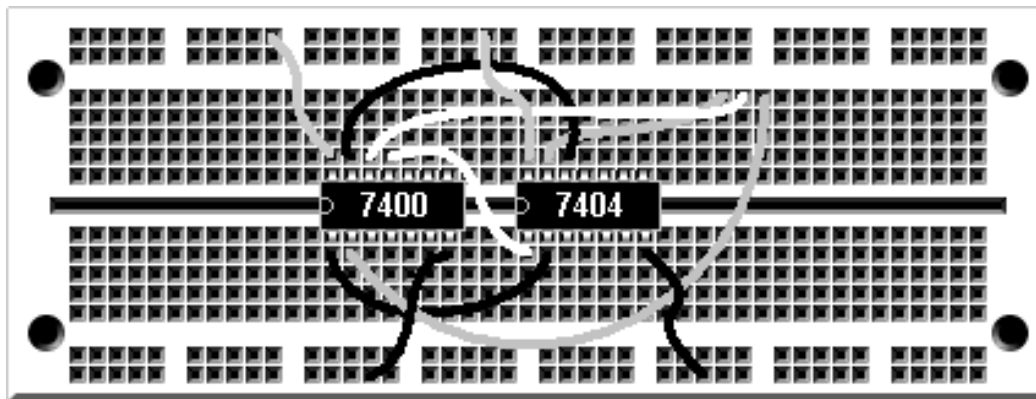


Figure 8-34 Sample Circuit Wired on a Protoboard

The next step is to add input and output that will allow us to communicate with our circuit. The simplest output from a digital circuit is an LED. Figure 8-35 presents the schematic symbol of an LED.

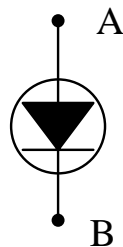


Figure 8-35 Schematic Symbol of a Light-Emitting Diode (LED)

An LED will turn on only when a small current passes through it from node A to node B. No light will appear if there is no current or if the current tries to flow in the opposite direction. By the way, if your LED doesn't work like you think it should, try to turn it around.

There are two things to note here. First, the current must be very small. In order to keep the current small enough to protect the LED, we need an electronic device called a resistor. This resistor is placed in series with the LED to limit the current. If you forget the resistor, you will hear a small pop and smell an awful burning odor when you power up your circuit. Figure 8-36 shows a typical LED circuit.

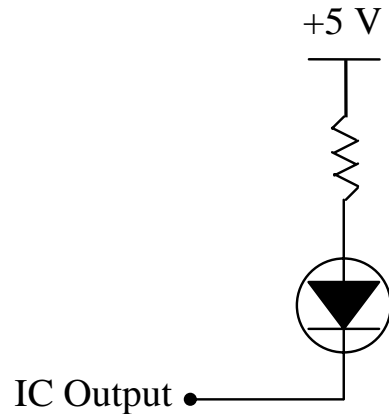


Figure 8-36 Typical LED Circuit

It is important to note that the LED will turn on only when the output from the IC equals zero. This is the best way to drive an LED. It keeps the ICs from having to supply too much current.

The simplest input to a digital circuit is a switch. It seems that the logical way to connect a switch to a digital circuit would be to connect it so that it toggles between a direct connection to a logic 1 and a direct connection to a logic 0. Switching back and forth between these connections should produce binary 1's and 0's, right?

Due to the electronics behind IC inputs, this is not the case. Instead, connections to positive voltages are made through resistors called *pull-up resistors*. This protects the IC by limiting the current flowing into it while still providing a positive voltage that can be read as a logic one. Figure 8-37 presents a generic switch design for a single input to a digital circuit. It uses a pull-up resistor connected to 5 volts which represents the circuit's power source.

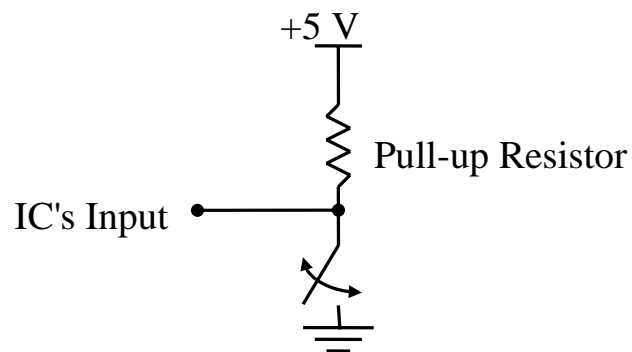


Figure 8-37 Generic Switch Circuit to an IC's Input

Any local electronics store should carry the protoboards, ICs, input switches, and output LEDs to create your prototype circuits. By using some simple circuits for switches and LEDs and the design principles outlined in this book, you can begin creating digital circuits of your own.

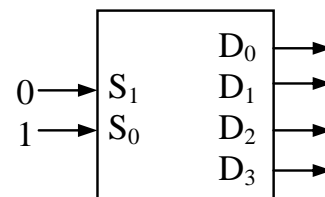
8.8 What's Next?

In this chapter, we have examined some of the lower-level hardware applications of digital logic. In the next chapter, we will present some applications that pertain more to the software and system levels of computer system design.

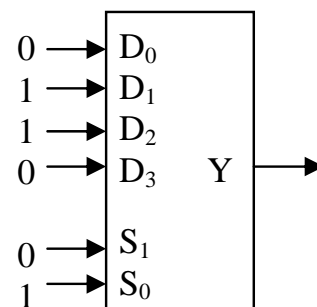
Problems

1. Design the digital logic for segments c, f, and g of the seven-segment display driver truth table in Figure 8-14.
2. Draw the decoding logic circuit with an active-low output for the inputs $A = 1$, $B = 1$, $C = 0$, and $D = 1$.

3. For the *active-low* output decoder shown to the right, fill in the values for the outputs D_0 through D_3 . Assume S_1 is the most significant bit.



4. What is the binary value being output from Y in the multiplexer circuit shown to the right?



5. What is the purpose of the resistor in the digital circuit for the LED shown in Figure 8-36?
6. What is the purpose of the resistor in the digital circuit for the switch shown in Figure 8-37?