KARNAUGH MAPS FOR FAST DIGITAL DESIGN

A neat, simple method for working with logic.

BY ART DAVIS

HAVE you ever tackled a digital design project with vim and vigor—only to find yourself entangled in a morass of logic ones and zeros and a "this goes up, and that goes down" nightmare? If you have, don't despair. There is a much neater, much simpler method than the brute force approach. This article provides a coherent approach to digital design. The method is not a substitute for intuition and practical seat-of-the-pants experimentation, but a tool for getting the end results quickly.

Before getting down to actual techniques, it might be wise to do a little reviewing. The truth tables for the AND, OR, and NOT (or COMPLEMENT, or INVERTER) functions are shown in Fig. 1. The function *a* AND *b* is written *ab*; *a* OR *b* is written $a + b$; and NOT *a* is written $\overline{a}$. Note that + as defined here is different from ordinary addition, and merely symbolizes

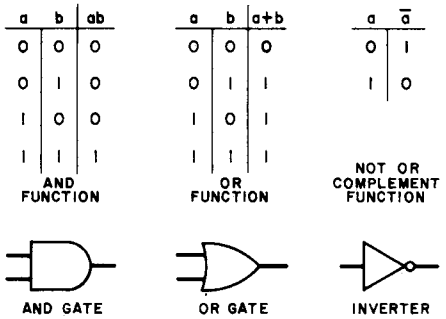


Fig. 1

the function defined by the truth table of Fig. 1. A truth table is simply an array, one side of which contains all possible combinations of the input variables and the other side of which contains the corresponding values of a logic function—or output. Figure 1 also shows the digital logic gate symbols for the three functions.

Any logic function can be constructed from these three basic types of functions or gates. It is often convenient, though, when working with a particular type of logic family (TTL, DTL, etc.) to use two other types of function, the NAND and the NOR. The NAND function of *a* and *b* is written $\overline{ab}$, and the NOR function, $\overline{a + b}$. Their truth tables and logic symbols are illustrated in Fig. 2. All of these functions except the NOT, or INVERTER, can be extended in an obvious way to include more than two inputs. With these functions at hand, it becomes possible to construct any logic function desired.

In manipulating the basic functions to form more complex ones, it is expedient to have available two important, yet simple, rules of basic logic theory known as DeMorgan's Laws. Figure 3 contains truth tables for the logic functions $\overline{ab}$, $\overline{a} + \overline{b}$, $\overline{a + b}$, and $\overline{a}\,\overline{b}$. Comparing them yields the formulas of DeMorgan's Laws:

1) $\overline{ab} = \overline{a} + \overline{b}$
2) $\overline{a + b} = \overline{a}\,\overline{b}$

These formulas are useful in implementing digital functions using only NAND or only NOR gates.

Why Map Techniques? A truth table is one way of specifying a logic function—the Karnaugh map (pronounced Kar-no) is another. To get an idea of what such a map is, and why it is a convenient tool, let's look at a practical digital design problem.

Suppose we are faced with designing the digital black box of Fig. 4, which has three inputs *a*, *b* and *c*, and a single output *f(a,b,c)*. The black box is to provide a logic one output under the following input conditions:

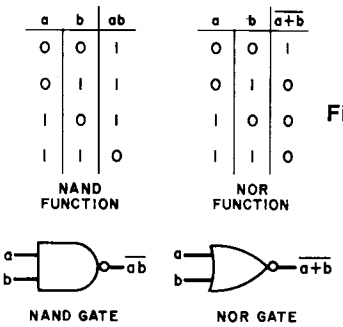


Fig. 2

$a=b=c=1$, $a=c=1$ and $b=0$, $a=0$ and $b=c=1$, or $a=b=0$ and $c=1$. How can we manufacture the digital logic inside the box from this specification?

One possible answer is to be methodical. A person unfamiliar with map techniques—but very methodical—might reason in the following way.

"The output function *f(a,b,c)* is logic one whenever $a=b=c=1$. An AND gate puts out a one whenever all inputs are logic one, so let's use an AND. But the AND output is zero for all other input combinations, and *f(a,b,c)* is a one for several other input conditions.

"Well, the AND gate did pretty well for the first input combination, so why not try it for the second? Let's take the complement of *b* by passing it through an INVERTER and run it into an AND gate with *a* and *c*. This AND will put out a one when $a=c=1$ and $b=0$, as desired. This seems to be working well, so let's do the same with each of the other two combinations."

With all the AND gates and INVERTERs arranged as above, our methodical experimenter will then observe that, since *f(a,b,c)* is to be a logic one whenever the input variables form the first combination, *or* the second, *or* the third, *or* the fourth, all he has to do is OR the outputs of the four ANDs to generate *f(a,b,c)*. The resulting logic is shown in Fig. 5.

Now this logic design works. It will do the digital job, but it is inefficient. It requires four AND gates, one OR, and two INVERTERs. This is costly, and it would cause quite a few layout problems because of the numerous interconnections. In addition, the design procedure outlined above is slow and, for more complicated circuits, error prone. What can be done to streamline the procedure?

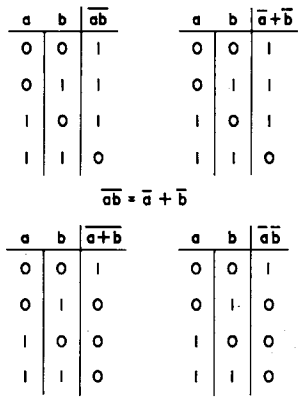


Fig. 3

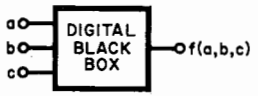
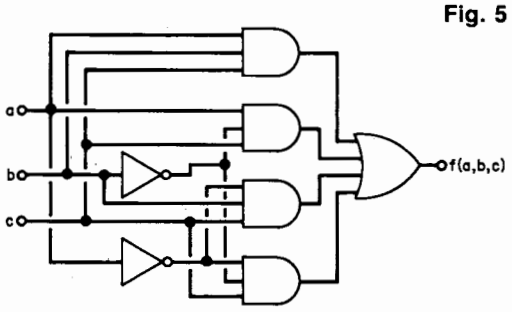


Fig. 4

Fig. 5



The answer is the Karnaugh map. This is just a rectangle divided up into a number of squares, each square corresponding to a given input combination. The Karnaugh map of our function *f(a,b,c)* is shown in Fig. 6. The right half of the map corresponds to $a=1$, the left half to $a=0$ ($\bar{a}=1$), the middle half to $b=1$, and so on. The basic idea is that there is one square for each input combination. If we write into that square the value of the output function for that particular input combination, we will have completely specified the function. The ones and zeros in Fig. 6 are the values which *f(a,b,c)* assumes for the associated input variable combinations.

Now recalling our methodical design procedure, it is easy to see that each square which has a one in it corresponds to the AND function of those input variables, and *f(a,b,c)* can be generated as the OR function of all of the ANDs.

A key factor arises here. It isn't necessary to include all of these AND functions, and the Karnaugh map tells us how to eliminate some of the terms. For example, looking at Fig. 6, we see that *f(a,b,c)* is a one for four *adjacent* boxes forming the bottom half of the map. (We will consider squares on opposite edges to be adjacent.) It is also easy to see the following: The only variable which does not change as we go from one square with a one to another with a one is c. It remains at one. What this means is that *f(a,b,c)* cannot depend on *a* and *b* because, regardless of their values, *f(a,b,c)* is a one as long as $c=1$. Therefore we can forget about *a* and *b*, and implement our black box as shown in Fig. 7. We have grouped together the four adjacent squares to eliminate *a* and *b*. Notice that we have simplified things a great deal, since we now need no gates at all.

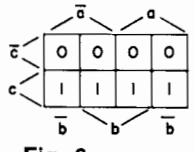


Fig. 6

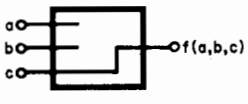


Fig. 7

Using a Karnaugh Map. Maps of one, two, three, and four variables are shown in Fig. 8. Maps of one variable are rarely used, and maps with more than four variables are seldom needed—even if such a problem were to chance along, the Karnaugh map would not be the tool to use. Its value depends on the pattern recognition capability of the user, and it becomes hard to recognize pattern groupings in maps of more than four variables.

Using the three-variable map as an example, note that there is one box for *abc*, one for $a\bar{b}c$, another for $\bar{a}b\bar{c}$, and so on, with *abc* corresponding to the input combination $a=1$, $b=1$, $c=1$; $a\bar{b}c$ to $a=1$, $b=0$, $c=1$; and $\bar{a}b\bar{c}$ to $a=0$, $b=1$, $c=0$; etc. Each box, then, corresponds to a single row in the truth table. The map is arranged in such a way that half of it corresponds to the uncomplemented form of a given variable and the other half to its complemented form; and the variables are interleaved so that every input combination corresponds to exactly one square, and conversely. Usually only the uncomplemented form of each variable is written—it being clear that the other half of the map corresponds to the complemented form.

Now, a logic function is displayed by placing ones and zeros in the boxes on the map. If a given input combination produces an output, or function value, of one, a one is placed in the corresponding square on the map. If the output is zero, a zero is placed in the square. As an example, look at the logic function in Fig. 9. On the Karnaugh map, the box given by *abc* has a 1 in it. This means that *f* is a logical one when the input variables have the value $a=1$, $b=1$, and $c=1$. The box given by $ab\bar{c}$ has a 0 in it. This means that $f=0$ when $a=1$, $b=1$, and $c=0$. These entries, as well as all the others, can be verified by looking at the truth table.

The logic function in Fig. 9 is not at all simple looking. The question is, how can the function be reduced to its simplest form? Variables can be eliminated from the function by use of the following definition and rules:

Definition: Two boxes are adjacent if the corresponding variables differ in only one place, for example if one box corresponds to $a\bar{b}\bar{c}$ and the other to $a\bar{b}c$. Notice that boxes on opposite edges of the map are adjacent under this definition.

Rule 1: If two boxes containing ones are adjacent, the single variable which differs between the two (uncomplemented for one, complemented for the other) can be eliminated and the two boxes combined. These two boxes correspond to the AND function of all the variables except the one eliminated.

Rule 2: If four boxes containing ones are adjacent in such a way that each box is adjacent to at least two others, these boxes can be combined and the two variables eliminated—those two which appear in both complemented and uncomplemented form somewhere within the group. The group of four corresponds to the AND function of all the variables except for the two which have been eliminated.

Rule 3: The same procedure holds for eight, sixteen, and so on, adjacent boxes. Each box in a grouping must be adjacent to three, four, etc., others within that group.

Rule 4: The various AND functions produced by the above groupings are "ORed" together to yield the simplest function.

It should be noted that a given box can be included in more than one grouping if that will simplify the overall function, but each grouping should contain at least one box which doesn't belong to an existing group-

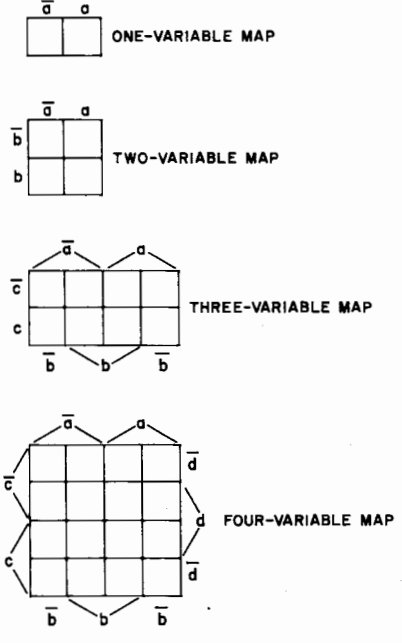


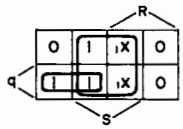
Fig. 8

R	S	Q
0	0	q
0	1	1
1	0	0
1	1	x

RS=0
INCOMPLETE TRUTH TABLE

R	S	q	Q
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	x
1	1	1	x

RS=0
COMPLETE TRUTH TABLE



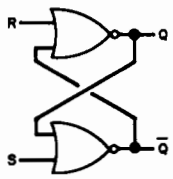
KARNAUGH MAP

$Q = S + \bar{R}q$

SO: $\bar{Q} = \overline{S + \bar{R}q}$

$= \bar{S}\,\overline{\bar{R}+\bar{q}}$

(DEMORGAN'S LAW)



NOR-GATE R-S FLIP FLOP

Fig. 13

keep from violating the condition that $R=S=1$ must never occur. Figure 13 also shows how DeMorgan's Law is used to get the function into a form requiring only NOR gates for its construction. By assuming all three possible combinations of input variables (remembering the $R=S=1$ is disallowed from ever occurring) and computing outputs, the truth table can easily be verified. It is also easy to show in this way that the output labeled Q is, indeed, the complement of the output labeled $\bar{Q}$ for all input conditions except the disallowed $R=S=1$.

The Clocked R-S Master-Slave Flip-Flop. It is often desirable to have available an R-S flip-flop which changes state only on, for example, the trailing edge (or 1-0 transition) of a clock signal. It is possible to use the Karnaugh map to derive the form of such a flip-flop, but the end result, although economical in number of gates and number of inputs per gate, would not shed much light on the internal workings.

This type of sequential machine is depicted in Fig. 14. When the clock goes high, the R and S inputs are passed through the input gates and stored in the master. When the clock goes low, the input gates are disabled, and the information is coupled through the transfer gates into the slave flip-flop. The function of the preset and clear inputs is evident. Try assuming a set of input values for R and S, and trace the information flow, letting the clock change as described above, to convince yourself that the unit performs the R-S function.

The J-K Flip-Flop. Let's return to our newly developed map technique now and develop the (clocked) J-K flip-flop as a last example. For convenience, since output changes are allowed only on clock transitions, let's denote the unstable state q by Q_n and the stable state Q by Q_{n+1}. This is reasonable, because Q_n is the stable state just prior to the n^{th} 1-0 clock transition, and is the unstable state just afterward, with the flip-flop settling down into the stable state Q_{n+1} before the next clock transition occurs.

The incomplete and complete truth tables are shown in Fig. 15, along with the Karnaugh map and the resulting simplified function. The J serves as the S and K as the R, respectively, of an R-S flip-flop. The only difference is that the $J=K=1$ output is now defined ($\bar{Q}_n$).

Let's use the clocked R-S flip-flop to build the J-K from our derived equation. For this purpose, let $S=J\bar{Q}_n$ and $R=KQ_n$ be the inputs to the clocked R-S. According to the R-S equation,

$$Q_{n+1} = S + \bar{R}Q_n = JQ_n + (\overline{KQ}_n)Q_n$$

Now, applying DeMorgan's law to $\overline{KQ}_n$, we get

$$Q_{n+1} = J\bar{Q}_n + (\bar{K} + \bar{Q}_n)Q_n = J\bar{Q}_n + \bar{K}Q_n + \bar{Q}_nQ_n$$

But $\bar{Q}_nQ_n = 0$ always, so

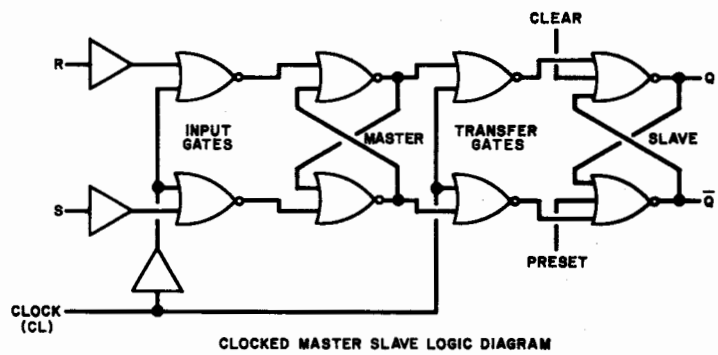
$$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$$

which is the J-K flip-flop equation. Notice that the R-S constraint is satisfied, since

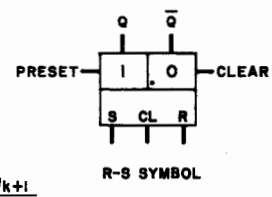
$$RS = (J\bar{Q}_n)(KQ_n) = JK(\bar{Q}_nQ_n) = 0$$

Fig. 16 shows the construction of the J-K from the R-S using two AND gates. Again, test the operation by assuming a set of conditions for J and K and tracing the logic flow. A glance back at the incomplete truth table will reveal that if J=K=1 (J and K inputs tied to a logic one) the J-K forms a toggle flip-flop.

The preceding examples have been intended to accomplish two things. In the first place, knowledge of the logical operation of the various types of flip-flops is essential in order to use them intelligently in an original design. As a second objective, they have provided an effective demonstration of the economy of thought which results when the Karnaugh may is used in a digital design effort. ◈



CLOCKED MASTER SLAVE LOGIC DIAGRAM



R-S SYMBOL

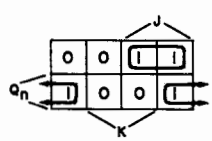
Fig. 14

J	K	Q_{k+1}
0	0	Q_k
0	1	0
1	0	1
1	1	$\bar{Q}_k$

INCOMPLETE TRUTH TABLE

J	K	Q_k	Q_{k+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

COMPLETE TRUTH TABLE



KARNAUGH MAP

$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$

Fig. 15

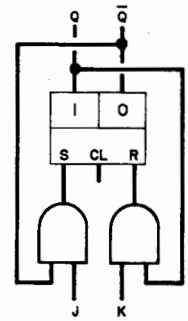


Fig. 16

Signal-driven exclusive-OR gate detects state changes

An exclusive-OR gate can detect a change in state of a logic signal more conveniently than can a one-shot, says N. Ghani of the Computer Laboratory at Newcastle University, Newcastle-upon-Tyne, England. A one-shot can directly sense state changes in only one direction, so additional circuitry is needed to handle both positive and negative changes. **But a quad 7486 exclusive-OR gate, driven by the signal and its delayed version, can sense both types of changes,** Ghani says.

Three of the gates can be used as inverters producing a delay. The fourth is in the exclusive-OR mode, taking in the signal and its delayed counterpart. Whenever the input changes state—whether it is positive- or negative-going—the 7486 will produce a narrow negative pulse with a width equal to the delay. If no inverters are used in producing the delay, then a positive pulse will be generated.

Graphic symbols clarified

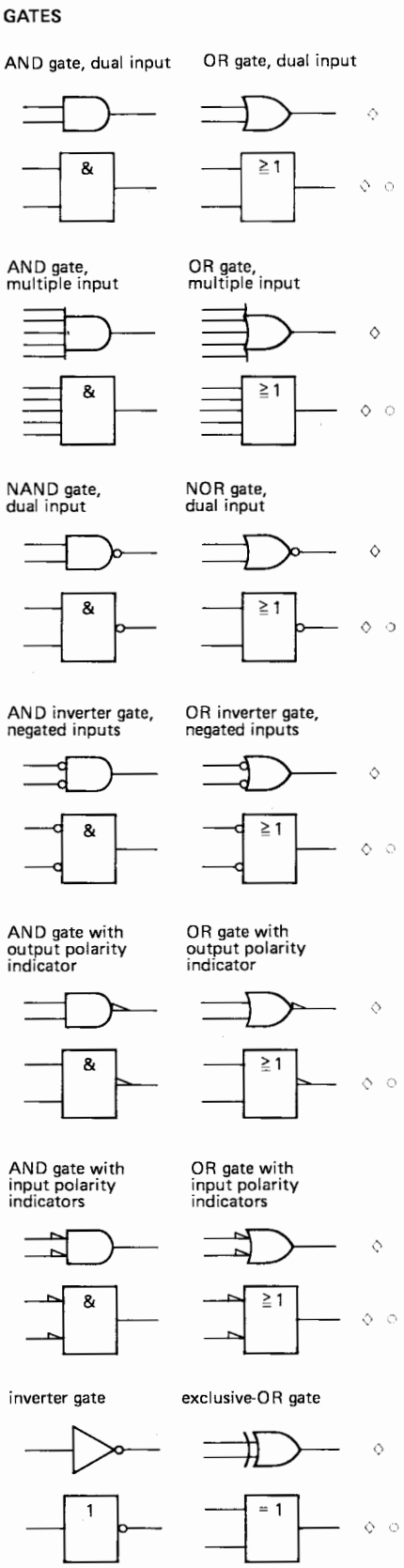
A number of readers were apparently confused by the gates section of two-state logic devices in the "Graphic Symbols for Electronics Diagrams," April 3, 1975. To clear up this confusion, the gates section has been modified and reproduced here. It can be clipped out and placed over the original section for an instant revision. To summarize, the revision includes changes in the labels for the inverting-type gates, and the symbols showing polarity indicators (small right triangles) or negation indicators (small open circles) have been separated.

One source of confusion, as a number of readers pointed out, is that either polarity indicators or negation indicators, but not both, should be used on a drawing. The polarity triangle inverts only voltage level and does not invert logic state. The negation circle, on the other hand, inverts only logic state, not voltage level. When the polarity triangle is placed on a line, that line is associated with a logic 1 when the voltage is low.

If a logic drawing makes use of the polarity indicators, there is no need for a statement as to whether the design is based on positive logic or negative logic. However, when negation circles are used, the drawing must state whether the design is employing positive or negative logic.

One other symbol should be modified. The labels for the input terminals to the upper R-S flip-flop are transposed. The letter S should designate the top input line, and the letter R the bottom input line.

Again, as indicated in the introductory comments to the original symbols guide, it is a compilation of those symbols most often needed by today's designer. It is not meant to be a complete listing of all possible symbols and their applications. For such thorough documentation, the standards published by the Institute of Electrical and Electronic Engineers should be consulted. They are: "Graphic Symbols for Electrical and Electronics Diagrams" [IEEE No. 315, 1971]/[ANSI Y32.2, 1970] and "Graphic Symbols for Logic Diagrams (Two-State Devices)," [IEEE Std. 91, 1973]/-[ANSI Y32.14, 1973].



DIPs verify strobe within time window

by Robert A. Dougherty
RAD Technical Consulting, Dunedin, Fla.

It is often necessary, in testing digital equipment, to ascertain that the equipment under test can deliver a pulse (strobe) during a particular time interval (window). The circuit shown here can verify the presence of a strobe pulse coming from equipment under test during a window pulse from the test set; if the strobe does not appear, an error signal goes high. This circuit operates with no external clock, and uses only two dual-in-line-packaged integrated circuits.

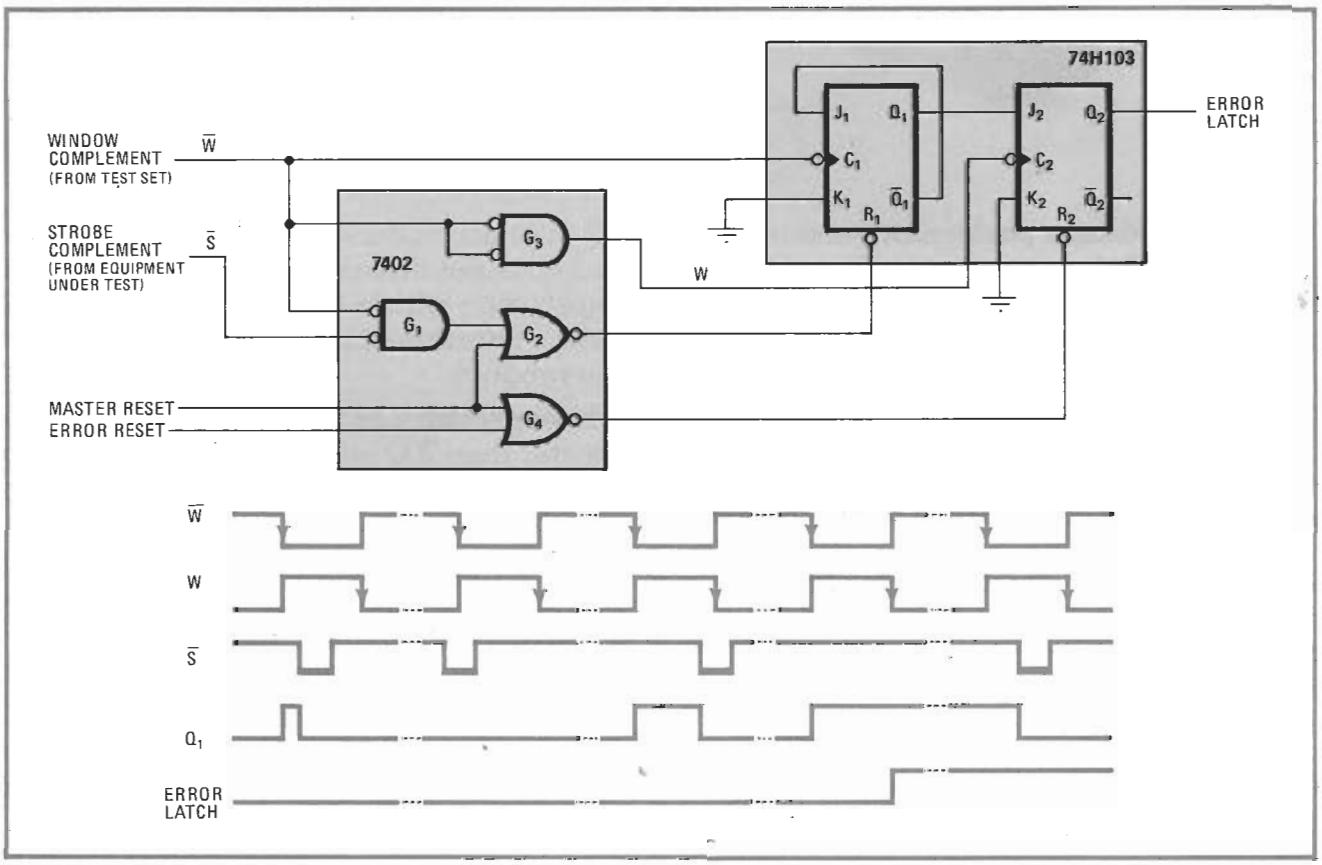
As shown in the diagram, one of the ICs is a dual edge-triggered J-K flip-flop; the other is a quad NOR gate. Assume that both J-Ks are initially in the reset condition—that is, Q_1 and Q_2 are both low. In this case the falling leading edge of a window complement pulse, $\overline{W}$, sets Q_1 high. Also, the low condition of $\overline{W}$ enables gate G_1. Therefore if a strobe complement pulse, $\overline{S}$, appears while $\overline{W}$ is low, G_1 goes high and resets J-K_1 through G_2.

The output from inverter G_3 is the window pulse, W. Its falling trailing edge clocks the condition of Q_1 (which is also J_2) to Q_2, which is the error latch. Therefore, if Q_1 is low, the error latch stays low. If Q_1 is high, the error latch goes high and remains high until cleared by the error reset or the master reset.

The master reset initializes both J-Ks. The error reset clears the error latch through G_4.

The timing diagram illustrates the operation of the circuit. Note that the error latch stays low if a strobe is totally within the window, or if it overlaps the beginning and/or end of the window. But if a strobe does not coincide with any portion of the window, the latch goes high; it can ring a bell, light a light, or otherwise indicate that the equipment under test has failed to deliver a pulse when one was required. □

Engineer's Notebook is a regular feature in Electronics. We invite readers to submit original design shortcuts, calculation aids, measurement and test techniques, and other ideas for saving engineering time or cost. We'll pay $50 for each item published.



Checking the windows. Indicating whether digital equipment can deliver a pulse at the proper time, this circuit signals an error if a strobe pulse does not coincide with a window pulse generated by the test station. Note that 74H103 J-K flip-flops are clocked by falling edge of pulse. Gates in the 7402 quad NOR are drawn to indicate their function; by DeMorgan's Theorem, a negative NAND is equivalent to a NOR. Timing diagram shows that error latch goes high on falling edge of window pulse (W) unless strobe pulse has occurred some time during W.

Circuit adds BCD numbers faster with less hardware

by Dharma P. Agrawal
Federal Polytechnic Institute of Lausanne, Switzerland

To add two binary-coded decimal numbers, at least four full adders are needed, not to mention the gates and inverters that correct the sums from each adder and generate the decimal carry-out. But this extra logic hardware can be simplified, as has already been shown ["Simplifying sum-correction logic for adding two BCD numbers," by Robert D. Guyton: *Electronics*, May 30, 1974, p. 108], and the new approach proposed here economizes on hardware and improves speed still further.

The circuit in the accompanying diagram uses a neat dodge to reduce the number of logic elements required to add the two BCD numbers $A_8A_4A_2A_1$ and $B_8B_4B_2B_1$. The dodge is to obtain the decimal carry-out, C_0, from the uncorrected sums S_2, S_4, and S_8, and the uncorrected carry C_{16} first, and only then to use C_0 to obtain the corrected sums S_1', S_2', S_4', and S_8'.

The boolean expression for the decimal carry-out can be written as

$$C_0 = C_{16} + S_8S_4 + S_8S_2$$

The circuit schematic shows how to obtain this value for C_0 by using just three NAND gates and one inverter.

The truth table for the corrected sums S_8', S_4', S_2', and S_1' as functions of C_0, S_8, S_4, S_2, and S_1 can be prepared, and their boolean expressions can be obtained as

$$S_8' = \overline{C}_0S_8$$

$$S_4' = S_4S_2 + \overline{C}_0S_4$$

or $$= S_4S_2 + \overline{S}_8S_4$$

$$S_2' = C_0\overline{S}_2 + \overline{C}_0S_2$$

or $$= C_0\overline{S}_2 + \overline{S}_8S_2$$

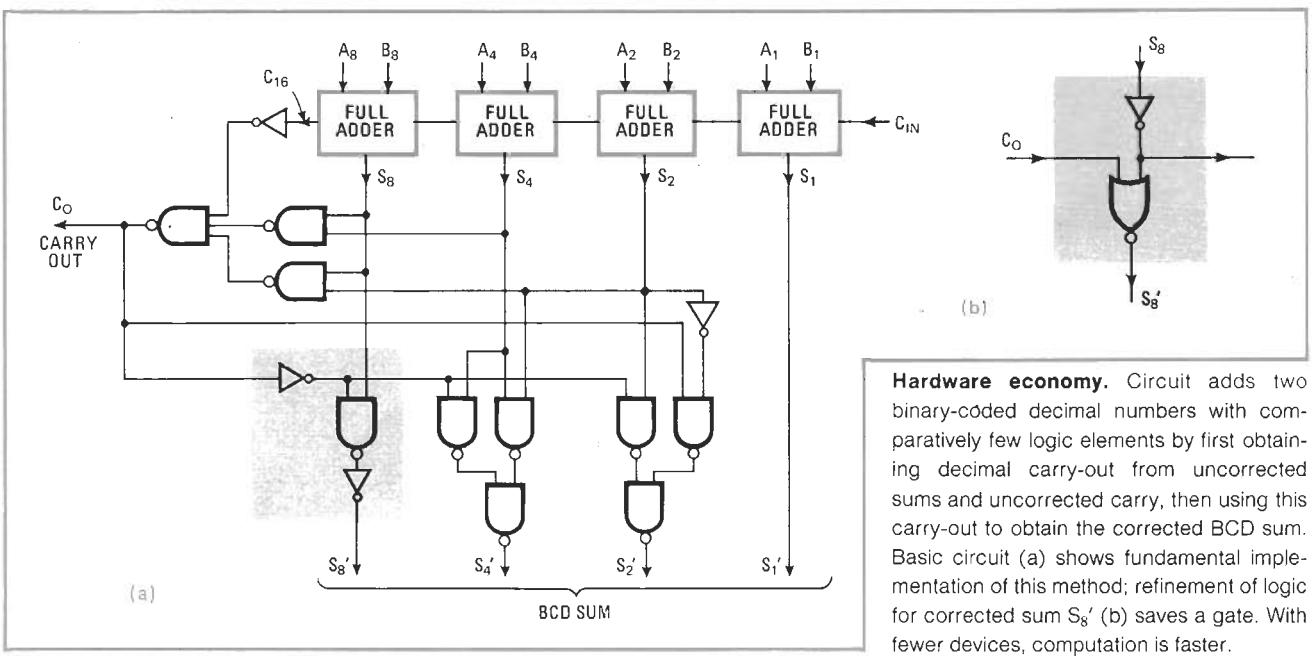
and $S_1' = S_1$

The circuit diagram clearly indicates the hardware accomplishment of each of these corrected sums. Note that the portion of (a) that is inside the shaded box can be replaced by the arrangement (b) to produce S_8' with one less gate and implement the alternative expressions for S_4' and S_2'.

The numbers in the accompanying table demonstrate how effectively this BCD adder reduces parts count and time delay, compared with some earlier circuits.□

COMPARISON OF BCD ADDERS

	Excess-3 adder	Guyton's adder	Proposed adder (a) shown here	Proposed adder (a), partially replaced by (b)
Number of full adders	5	4	4	4
Number of half adders	2	–	–	–
Number of 3-input NAND gates	20	4	1	1
Number of 2-input NAND gates		10	9	8
Number of 2-input NOR gates		–	–	1
Number of inverters	9	6	4	3
Time delay in terms of number of:				
Full adders	5	4	4	4
Half adders	2	–	–	–
Gates	?	6	5	4



Hardware economy. Circuit adds two binary-coded decimal numbers with comparatively few logic elements by first obtaining decimal carry-out from uncorrected sums and uncorrected carry, then using this carry-out to obtain the corrected BCD sum. Basic circuit (a) shows fundamental implementation of this method; refinement of logic for corrected sum S_8' (b) saves a gate. With fewer devices, computation is faster.

NAND gates and inverter synchronize control signal

by Robert L. White
Applied Research Laboratories, University of Texas, Austin, Texas

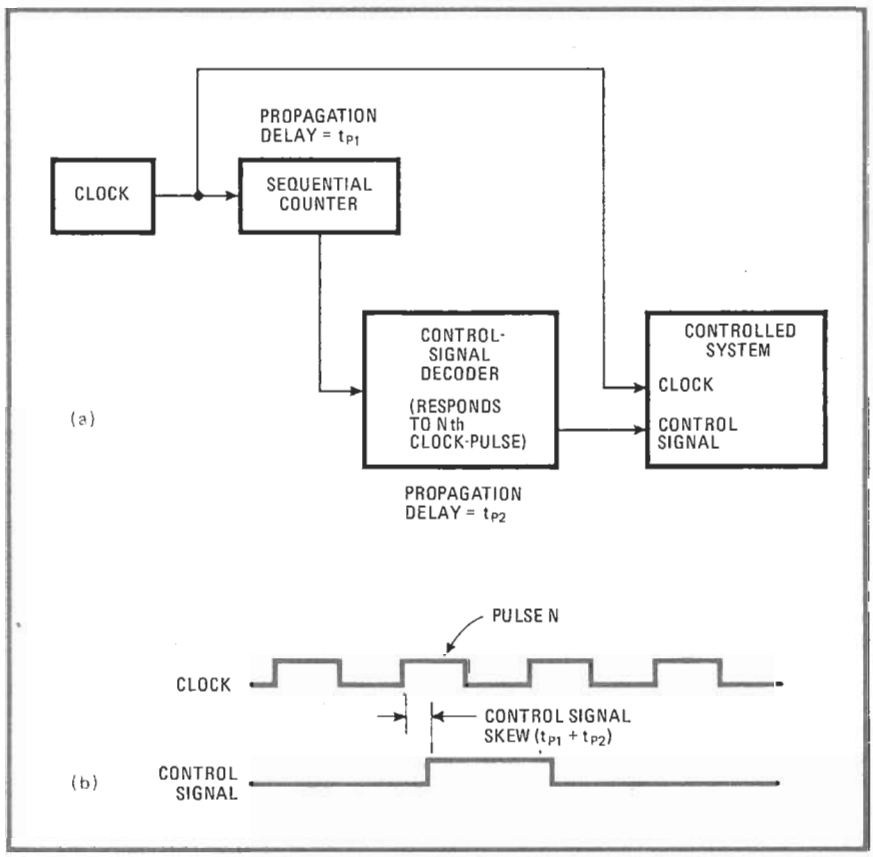
In many sequential digital systems that respond to a rising pulse edge, control signals must be synchronized to the clock pulses. These control signals are usually obtained by decoding the desired states of a sequential counter that is driven by the system clock (Fig. 1a). Since the counter and decoder have propagation delays, the decoded control signal is delayed or skewed relative to the system clock, as illustrated in Fig. 1b. But as the clock frequency of the system gets higher, the skew time may become a significant portion of the clock cycle and cause erratic circuit operation.

One example of this problem is encountered with an accumulator operating with a 15-megahertz clock. On the rising edge of every 40th clock pulse, a control signal transfers data from the accumulator to a data register. The control-signal skew causes the data to be transferred at some instant between the clock edges, rather than at the edges. Since the accumulator output is changing between clock edges, its value is uncertain at the instant of data transfer, and the system's operation is erratic.

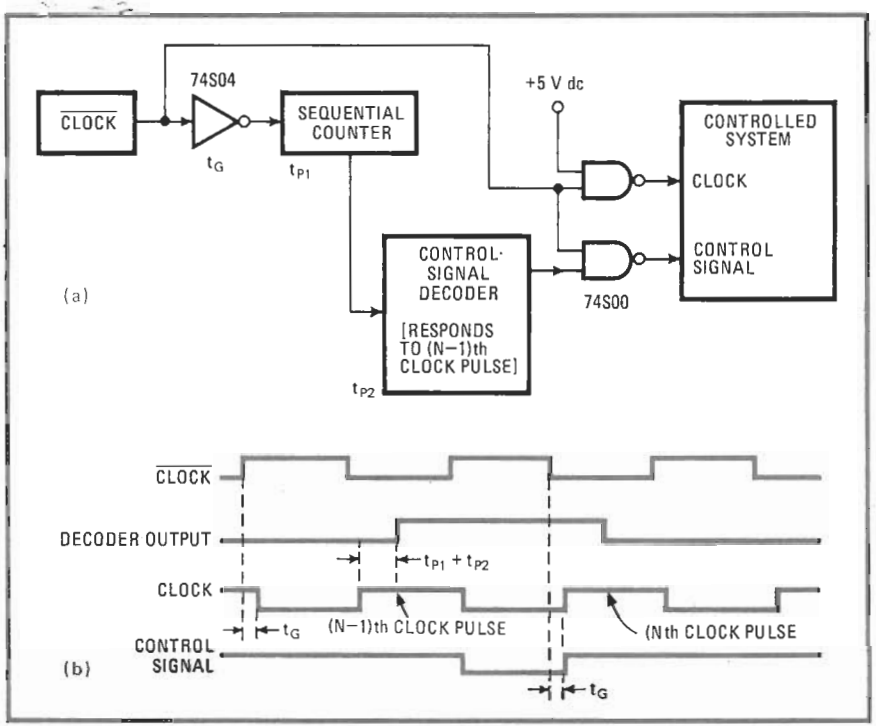
But the skew of the control signal can be reduced to a negligible value by configuring the sequential system as shown in Fig. 2a. The clock of Fig. 1 becomes $\overline{\text{clock}}$, and inverters are added to the $\overline{\text{clock}}$ line. One inverter is shown as a 74S04, but it could be one gate of a 74S00 IC. The other inverter is the top NAND gate in the 74S00. Also, $\overline{\text{clock}}$ drives an input of a NAND gate in the control-signal line. The decoder in Fig. 2 responds one clock-pulse sooner than the one in Fig. 1.

The synchronizing effect of the inverter and NAND gates can be seen in Fig. 2b. Line 3 shows that the control signal waveform is in the high state until the decoder output goes high. This event occurs at the (N – 1)th clock pulse, i.e., one clock period before the rising edge of the control signal.

After the decoder output goes high, the control signal remains high until $\overline{\text{clock}}$ also goes high. After $\overline{\text{clock}}$ goes high, the inputs of the two NAND gates are the same. Therefore, the control-signal and clock waveforms are alike; their falling edges and the subsequent rising edges virtually coincide at the leading edge of the Nth clock pulse. The only skew remaining is the difference between the propagation delays in the two NAND gates. With a 74S00 IC, the difference between the delays in



1. Skewed up. In a digital system that requires synchronization of clock and control signal, propagation delays in counter and decoder cause erratic circuit operation. Block diagram (a) shows how control signal is delayed relative to clock pulse, and timing diagram (b) shows waveforms for clock and control signal.



2. Squared away. By adding inverter and NAND gates to circuit and changing decoder to respond one pulse earlier than in Fig. 1, the control signal and clock pulse are synchronized accurately. After rising, the control signal is always high except for the half clock period before its next rising edge.

the two NAND gates was less than 1 nanosecond.

The circuit discussed above provides a control signal with a rising edge that is closely synchronized with the rising edge of the clock. Its applications involve the transfer of data in high-speed digital systems. The circuit can be used for any application requiring a precisely timed signal transition that does not occur on every edge of the clock pulse. □

How to Simplify Logic Circuits

Introducing the decision-accounts table

*by N. Darwood**

The theory of logical circuits is sometimes called switching logic. This is because the function of a combination of switches, such as is shown in Fig. 1(a), is easily described. Whether the components of which it is finally built are NAND, AND or OR gates or switches, the description of a logical circuit is the same. To explain: two switches in series make an AND gate, Fig. 2(a); two switches in parallel make an OR gate, Fig. 2(b). A switch which is short-circuit when 'on' and open-circuit when 'off' makes a logical inverter, Fig. 2(c).

Although the individual switches of a circuit may switch at electronic speeds, the function, i.e. the logic of the circuit, is described as though it is static and such that there is continuity across the circuit; the circuit is then said to be 'on'. For

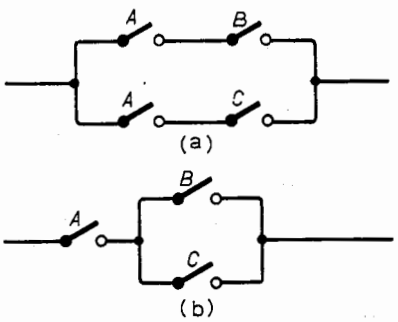


Fig. 1. All the possible switch states in these two (equivalent) combinations are shown in the truth table (first two tables in the text).

example, the logic of Fig. 1(a) is: either (A and B) or (A and C) which is written in shorthand as AB+AC. An instance of the usefulness of switching theory is to factorize AB+AC into A(B+C), which is simpler to construct—Fig. 1(b).

In the paper and pencil analysis of a logic circuit, it is sometimes useful to consider all the possible combinations of states of the switches. In the circuit of Fig. 1(b) there are three switches, A, B and C. All the possible states are shown below. Also shown is the condition of the circuit in Figs 1(a) and (b).

A	*B*	*C*	*AB+AC*
off	off	off	off
on	off	off	off
off	on	off	off
on	on	off	on
off	off	on	off
on	off	on	on
off	on	on	off
on	on	on	on

Rather than having to write many 'on's and 'off's, it is usual[1] to use 1 for 'on' and 0 for 'off'. Whence the above table becomes

Truth table for circuits of Fig. 1

State	*A*	*B*	*C*	*AB+AC*
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	1
4	0	0	1	0
5	1	0	1	1
6	0	1	1	0
7	1	1	1	1

From the above table we can describe the logic of Fig. 1 as being on when the switches are in state 3 (i.e. 110 in binary form where the least significant bit is on the left), state 5 or state 7. This is the truth-table. One way of designing a logic circuit is to write out the truth-table which shows all the possible states of the switches, then to enter the required 'on' or 'off' circuit condition. Suppose a table as shown below is required.

Truth table 2

State	*A*	*B*	*C*	
0	0	0	0	0
1	1	0	0	1
2	0	1	0	1
3	1	1	0	0
4	0	0	1	1
5	1	0	1	1
6	0	1	1	1
7	1	1	1	0

The first step would be to derive a logical expression by extracting the states of the switches *A*, *B* and *C* that will produce an 'on' condition. From the table the circuit-on conditions are

states: 1 or 2 or 4 or 5 or 6

logical expression: $A\bar{B}\bar{C}+\bar{A}B\bar{C}+\bar{A}\bar{B}C+A\bar{B}C+\bar{A}BC$

Armed with this expression we can draw the logic diagram. There are five terms in the expression, hence five AND gates which feed into a five input OR gate are needed. The logic diagram is shown in Fig. 3. The logic diagram may now be converted into some other type of logic using, say, NAND

*Decca Navigator Co. Ltd.

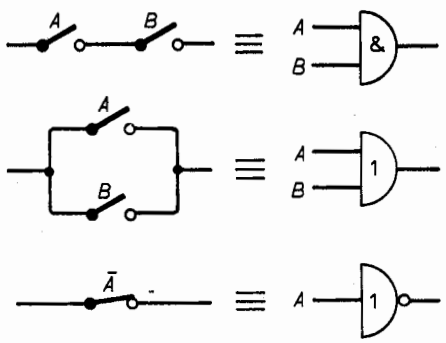


Fig. 2. In writing the logic diagram equivalent of switch diagrams, the notation shown is used.

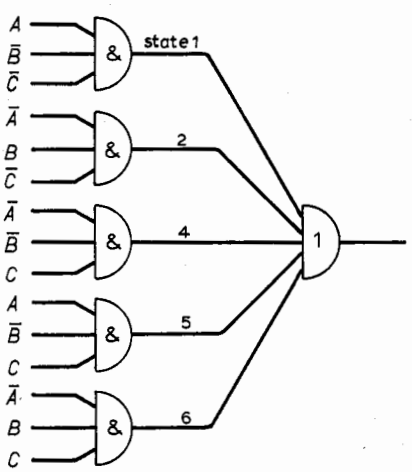


Fig. 3. To draw a logic circuit, the 'on' condition is normally written from the desired truth table. Truth table 2 gives an expression with five 'on' terms, realised by the logic gates shown.

gates. The conversion technique is a separate branch of logic circuit theory.

Returning to the logic diagram of Fig. 3, the circuit uses six logical elements. Using the truth-table or the rules of Boolean algebra (for example AB+AC = A(B+C); A+Ā = 1, A+A = A) the full expression can be simplified, for example, to $A\bar{B}+\bar{A}C+\bar{A}B\bar{C}+\bar{B}C$. The logic diagram of this expression, logically equivalent to Fig. 3, is shown in Fig. 4. Thus we can have two different, but logically equivalent, expressions one of which is simpler to construct than the other.

The question can be asked—are there other different ways a logical circuit can be built? Many methods exist for finding if a

simpler logical expression is possible, e.g. the Harvard method[2]; but they require a knowledge and a skill of Boolean algebra which usually only professional logic designers, or logicians, attain through constant practice.

Decision-accounts table

However, a new approach to the problem is being made, whereby not only are simpler expressions derived but also all the other equivalent expressions can be listed. This gives a complete analysis of the circuit. A table lists all the states to which a particular logic term, such as $\bar{A}C$, applies—in this instance states 4 and 6. (See logic tables opposite for three factors.) The table does not contain the states to which expressions apply, found by looking up the states for each term. To illustrate: the expression derived for Fig. 4 is

$$A\bar{B}+\bar{A}C+\bar{A}B\bar{C}+\bar{B}C$$

Enter the tables at the section for three factors A, B and C, with each term, and extract the corresponding state, thus

$$\begin{aligned} A\bar{B} &= 1,5 \\ \bar{A}C &= 4,6 \\ \bar{A}B\bar{C} &= 2 \\ \bar{B}C &= 4,5 \end{aligned}$$

Therefore the expression, when implemented in hardware, will be on for states 1, 2, 4, 5 or 6, see Fig. 3.

Having found the states that apply to the circuit, to find logically equivalent expressions reverse the procedure by entering the tables with the states 1, 2, 4, 5 and 6, and extract all the terms that apply, as shown below.

Decision-accounts table 1

State no. 1	2	4	5	6	term found
✓			✓		$\bar{A}B$
	✓			✓	$\bar{A}B$
		✓	✓		$\bar{B}C$
		✓		✓	$\bar{A}C$

Now as long as we take a combination of these terms that account for all the required states, then that combination will suffice. For example, the three terms $A\bar{B}$, $\bar{A}B$ and $\bar{B}C$ suffice. The circuit, Fig. 5, uses less hardware than Fig. 4.

The above table is called a decision-accounts-table. In conjunction with the logic table, they form a Boolean expression simplification method[3]. The tables are easily derived[4]. This same decision procedure is useful also to programmers and systems analysts who have to formulate and programme complex logical decisions.

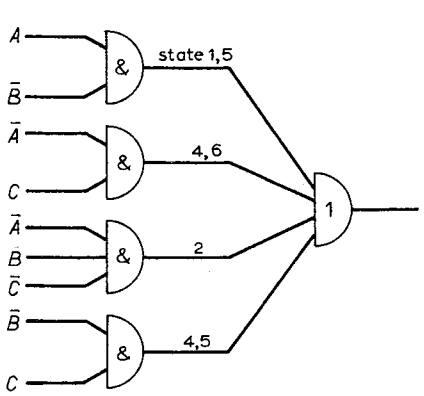


Fig. 4. The logic circuit of Fig. 3 could be simplified by using the rules of Boolean algebra on the logic expression of truth table 2, resulting in five instead of six gates.

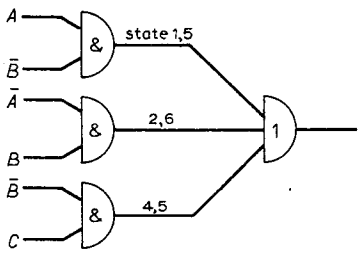


Fig. 5. To find other equivalent expressions or logic circuits of Fig. 3, a 'decision-accounts' table is compiled using logic tables which show all the terms that account for the required states. This allows the simpler circuit shown to be drawn.

By extending the method we look for other expressions, perhaps simpler or perhaps just as simple but different, that will produce the same output for the same combinations of switch states. Because this is a new procedure it may be a little difficult to grasp. One logical law need be used to perform the manipulation. The law is $X+XY=X$, which allows us to eliminate a term if it is ORed with a factor of itself. The logic diagram of this law is shown in Fig. 6, in terms of switches and gates. If, for clarity, we label the rows of the decision-accounts table P, Q, R and S, then a more complex instance of the absorption law, as it is called, could be

$$PQS+PQ=PQ$$

Having drawn up the accounts table, we can now calculate the different ways of constructing the logic of Fig. 3. (The example is purposely obvious to show the reasoning). The accounts table, re-labelled, is

Row	State 1	2	4	5	6
P	✓			✓	
Q		✓			✓
R			✓	✓	
S			✓		✓

The first column is accounted for by row P,
the second by Q
the third by R or S (= R+S)
the fourth by P+R
and the fifth by Q+S.
Also we have to account for the first column and the second and the third, and so on. Hence an expression for this particular decision-accounts table could be

$$P\cdot Q\cdot(R+S)(P+R)(Q+S)$$

which can be expanded to

$$\begin{aligned}(PQR+PQS)(PQ+PS+RQ+RS) = \\ PQR+PQRS+PQR+PQRS+PQS \\ +PQS+PQRS+PQRS\end{aligned}$$

$= PQR+PQS$ (by application of the absorption law, $X+XY=X$). The final expression $PQR+PQS$ shows that either rows P, Q and R or rows P, Q and S may be used to construct the initial logic expression.

Because $P=A\bar{B}$, $Q=\bar{A}B$ and $R=\bar{B}C$, then one expression that could be used is

$$A\bar{B}+\bar{A}B+\bar{B}C$$

or, alternatively, because $S=\bar{A}C$, the following expression could be used

$$A\bar{B}+\bar{A}B+\bar{A}C$$

That the alternatives are equivalent to the original may be proved by the truth table,

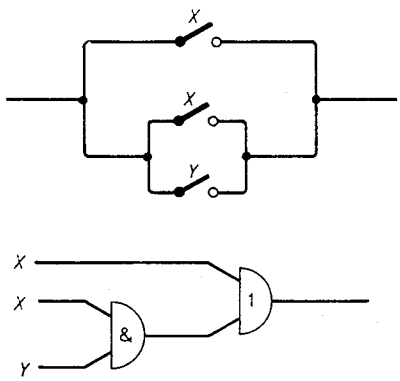


Fig. 6. In extending the decision-accounts technique the basic Boolean absorptive law $X+XY=X$ is used, which allows elimination of a term which is ORed with a factor of itself.

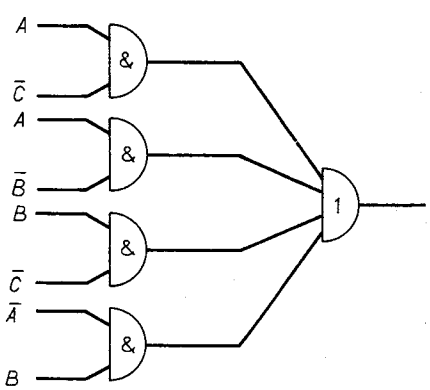


Fig. 7. In finding equivalent expressions for this circuit, the decision-accounts table 2 shows it can be achieved in either of two ways with one gate less.

or by showing via the logic tables that the three expressions produce the same states, which is the same thing.

Further example

Suppose the example is the circuit shown in Fig. 7; the Boolean expression is derived from the circuit as

$$A\bar{C}+A\bar{B}+B\bar{C}+\bar{A}B$$

Enter the logic tables at the section for three factors A, B and C to extract the states to which each term applies

$$\begin{aligned} A\bar{C} &= 1,3 \\ A\bar{B} &= 1,5 \\ B\bar{C} &= 2,3 \\ \bar{A}B &= 2,6 \end{aligned}$$

Therefore the expression is ON for states 1, 2, 3, 5 or 6. Next draw up an accounts table. To ease the working the rows are labelled P, Q, R and S.

Logic tables

two factors	
$0 = \bar{A}\bar{B}$	$0, 1 = \bar{B}$
$1 = A\bar{B}$	$0, 2 = \bar{A}$
$2 = \bar{A}B$	$1, 3 = A$
$3 = AB$	$2, 3 = B$

three factors	
$0, 1 = \bar{B}\bar{C}$	$2, 6 = \bar{A}B$
$0, 2 = \bar{A}\bar{C}$	$3, 7 = AB$
$0, 4 = \bar{A}\bar{B}$	$4, 5 = \bar{B}C$
$1, 3 = A\bar{C}$	$4, 6 = \bar{A}C$
$1, 5 = A\bar{B}$	$5, 7 = AC$
$2, 3 = B\bar{C}$	$6, 7 = BC$
$0 = \bar{A}\bar{B}\bar{C}$	$0, 1, 2, 3 = \bar{C}$
$1 = A\bar{B}\bar{C}$	$0, 1, 4, 5 = \bar{B}$
$2 = \bar{A}B\bar{C}$	$0, 2, 4, 6 = \bar{A}$
$3 = AB\bar{C}$	$1, 3, 5, 7 = A$
$4 = \bar{A}\bar{B}C$	$2, 3, 6, 7 = B$
$5 = A\bar{B}C$	$4, 5, 6, 7 = C$
$6 = \bar{A}BC$	
$7 = ABC$	

Decision-accounts table 2

	State					
Row	1	2	3	5	6	Term
P	✓		✓			$A\bar{C}$
Q	✓			✓		$A\bar{B}$
R		✓	✓			$B\bar{C}$
S		✓			✓	$\bar{A}B$

The accounts table expression is

$$\begin{aligned} &(P+Q)(R+S)(P+R)\,QS \\ &= (R+S)\,QS\,(P+QR) \\ &= QS\,(P+QR) = QSP+QRS \end{aligned}$$

The final expressions show we can construct the circuit by either rows Q, R or S, i.e.

$$A\bar{B}+B\bar{C}+\bar{A}B$$

or, alternatively, from rows Q, S and P, i.e.

$$A\bar{B}+A\bar{C}+\bar{A}B$$

On looking back at the accounts table, it is intuitive that to account for each column ('on' state of the switches) at least once, then rows Q and S are essential (called prime-implicants in the literature). Rows Q and S account for states (columns) 1, 5, 2 and 6, leaving state 3 outstanding. With the prime implicants we may choose either row P or row R. Hence Q & S & (P or S) is the choice of decisions.

Readers may like to simplify and/or find alternative expressions for the following

$$AB+\bar{A}C+BC$$
$$A\bar{B}+B\bar{C}+\bar{B}C+\bar{A}B$$
$$A+\bar{B}C+\bar{A}B$$
$$\bar{A}BC+A\bar{B}C+ABC$$

References

1. Boole, G. Mathematical Analysis of Logic (1847). Blackwell, 1965.
   Bowran, A. P., Boolean Algebra, Macmillan 1965.
2. Hohn, F. E. Applied Boolean Algebra, Macmillan, 1966.
3. Darwood, N., Representation of Logic Tables, *Electronic Engineering*, Jan. 1971.
4. Darwood, N., Simplification of Decision Tables, *Computer Weekly*, 29 July 1971.

Counter and switches select pulse-train length and dead time

by Héctor Gellón and Enrique Marcoleta
San Luis, Argentina

Only the more expensive pulse generators can repeatedly generate a pulse train of selectable length followed by an off time also of selectable length. But this common requirement is easily met by one cascaded counter, two switches, and some logic. The number of pulses in the train is selectable from 1 to 99, and irrespective of the number of pulses delivered to the output, the off, or dead, time can also be varied between 1 and 99 clock periods. The lengths of both the pulse train and the off time may be extended by adding to the number of stages in the counter.

As shown in the figure, a system clock drives two cascaded 7490 binary-coded-decimal decade counters, and their outputs are converted to a decimal equivalent by the 7442 BCD-to-decimal decoders. The decoded outputs are active low, and when the count reaches the pulse-train length desired (preset in this case to 8 by switches S_{1a} and S_{1b}), gate G_1 moves high.

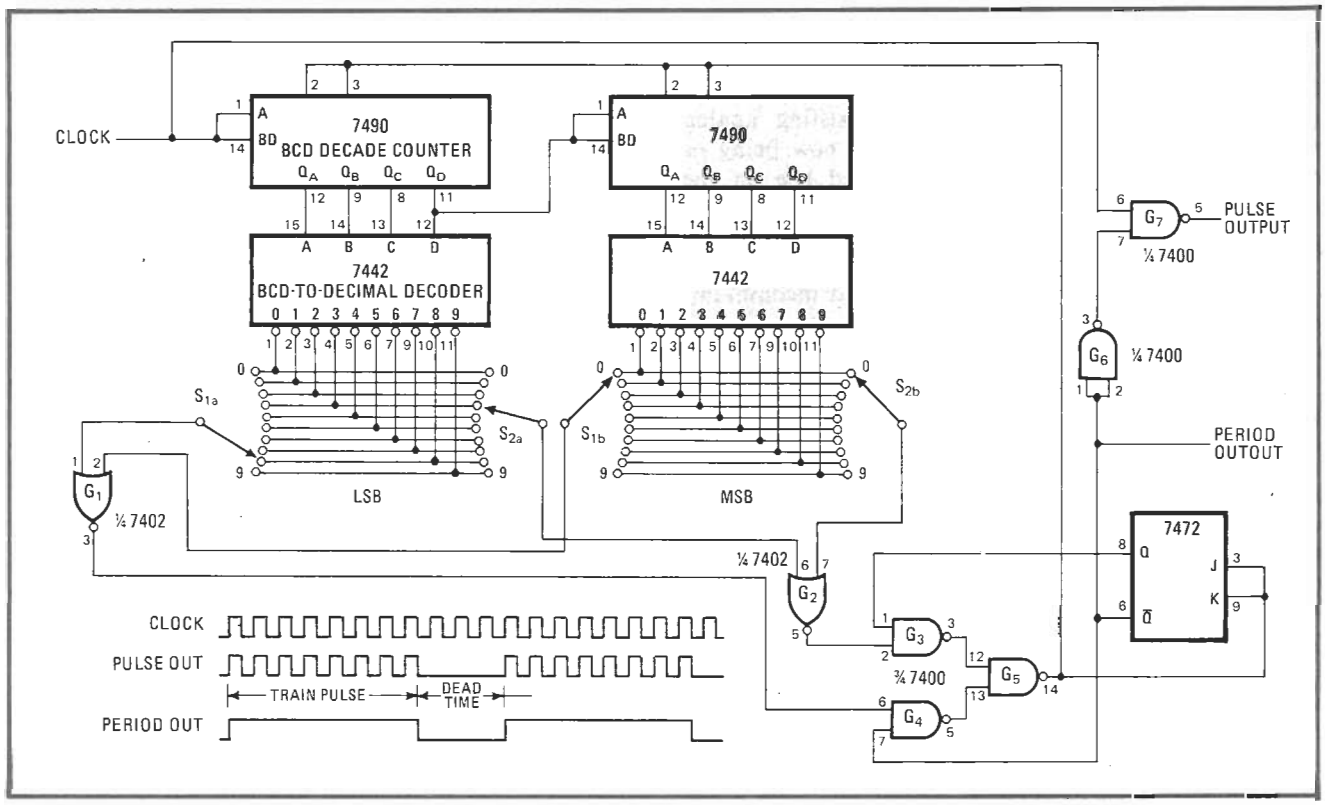
If the $\overline{Q}$ output of the 7472 flip-flop is high, as it will be once the circuit settles after initialization, G_4 moves low. This causes G_5 to assume a high state, resetting the counter and toggling the flip-flop, which in turn disables G_4 and output gate G_7.

There is no output until the counter reaches the number set by switches S_{2a} and S_{2b}. G_2 moves high, causing G_3 to move low and G_5 to go high. The counter is reset, and the flip-flop is toggled, thus once more enabling G_4 and G_7. The pulse train now appears at the output until the settings of switches S_{1a} and S_{1b} are reached, and the cycle repeats.

The $\overline{Q}$ output of the flip-flop is a signal having a duty cycle that may be anything from 1 to 99 times the period of the system clock. It is essentially an ungated version of the signal at the pulse-output port.

Cascading additional BCD counters and decoders to the circuit will extend its pulse-counting and dead-time limits. Of course, more switches and gates must also be added, to accommodate a greater number of inputs. □

Designer's casebook is a regular feature in *Electronics*. We invite readers to submit original and unpublished circuit ideas and solutions to design problems. Explain briefly but thoroughly the circuit's operating principle and purpose. We'll pay $50 for each item published.



Selectable. Generator produces train of N pulse lengths followed by dead time of M clock pulses, set by 2-pole, 10-position switches S_{1a} through S_{2b}. N and M may assume any value of from 1 to 99. Switches are shown set for pulse train of 8, dead time of 3.

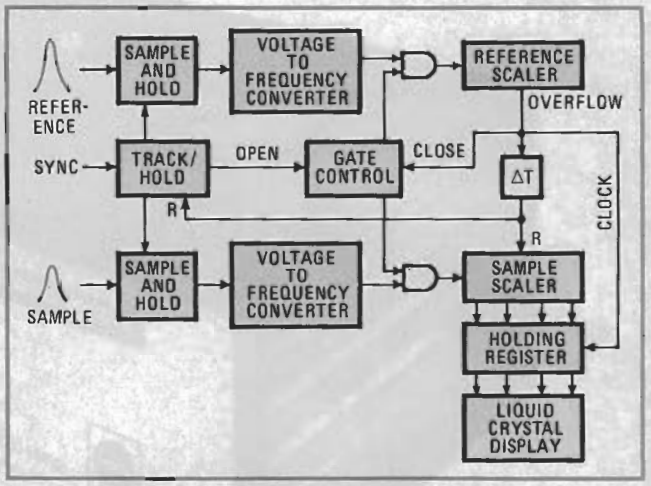
Digital normalizer derives ratio of two analog signals

by James H. McQuaid
University of California, Lawrence Livermore Laboratory, Livermore, Calif.

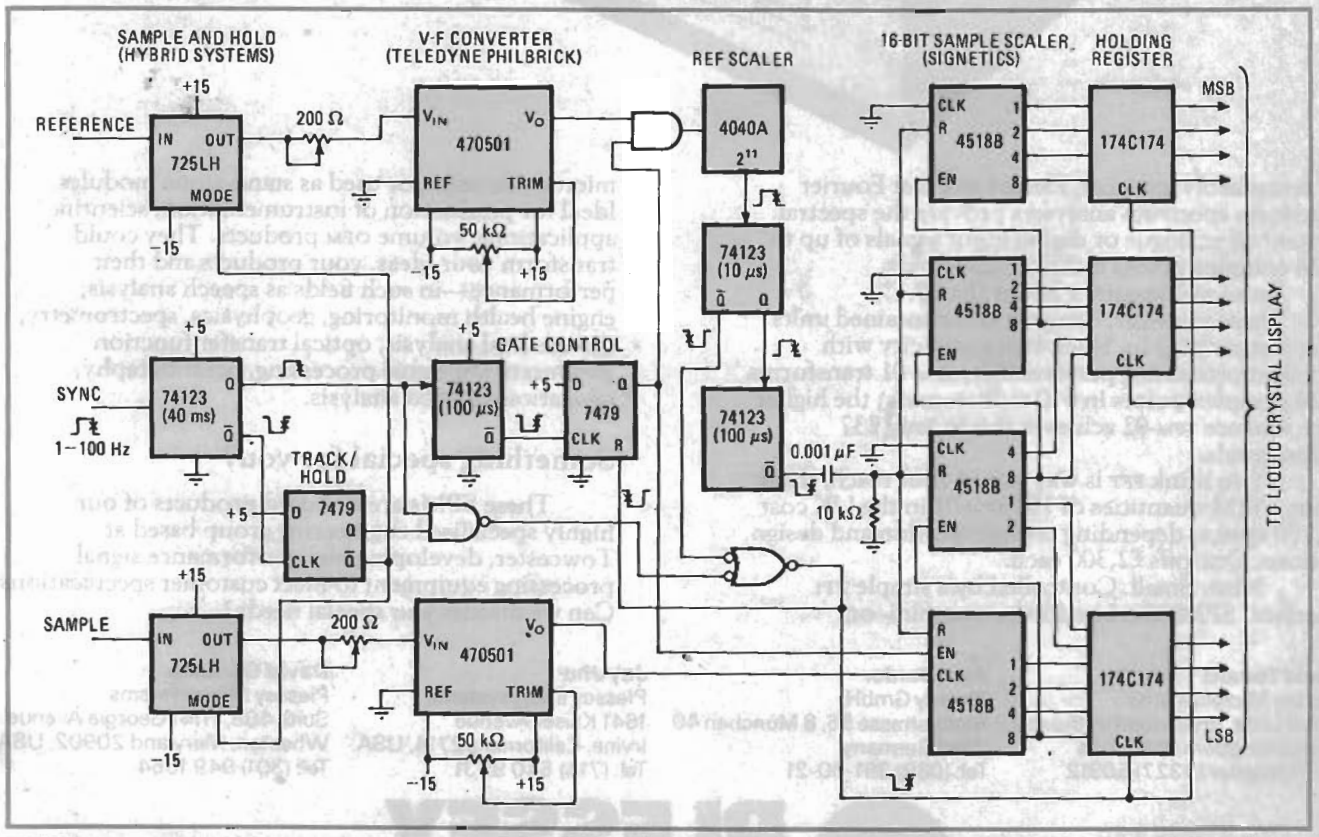
The absolute value of a voltage or current at a circuit point is frequently less important than the ratio of that quantity to a reference. This circuit compares two analog signals by using a high-accuracy digital technique for normalizing the reference voltage, thus simplifying circuitry and avoiding the use of analog dividers or microprocessors. It is invaluable in many light-chopping applications, such as lasers, where the measurement of light intensity at a specific frequency must take into account total source-intensity variations. It is also useful in atmospheric physics when measuring the concentration of a specific gas in a mixture by infrared techniques in which the intensity of the beam is subject to drift.

As shown in the block diagram (Fig. 1), the reference and sample signals are each introduced to a sample-and-hold circuit. Peak-detection circuitry in this device, in conjunction with track-and-hold logic, which controls the sample rate, produces a voltage output that is presented to their respective voltage-to-frequency converters. Each converter's output is a pulse train with a frequency directly proportional to the input voltage.

The track-and-hold logic and associated gating circuits simultaneously initialize both scalar circuits and allow the output pulse train of both converters to be counted. When the count in the reference scaler exceeds its capacity, an overflow pulse is generated that closes the gating circuit; at this time, the contents of the sample scaler are clocked into the holding register and then sent to the display. The contents of the reference scaler are regarded as a unit voltage, and the reference scaler



1. Ratio meter. Voltage comparison of analog sample to time-dependent reference uses analog-to-digital converters. Voltages are converted to frequency and counted. Reference count normalized to unity serves as gating signal to count number in sample counter. Output of circuit yields ratio of sample to reference voltage.



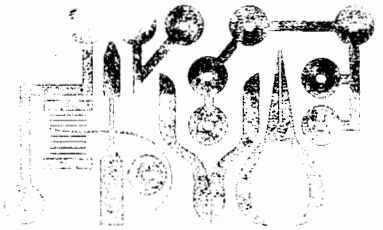
2. Digital normalization. Circuit detects relationship of analog sample amplitude to reference voltage by digital means. Several one-shots are used to obtain proper timing and gating. Circuit uses C-MOS devices where possible to reduce power consumption.

controls the gating time to the sample scaler. Thus the contents of the holding register will normally be some fraction of the unity-set voltage. The sample scaler and sampling circuits are then reset, ready to process the next sample.

As shown in Fig. 2, the sample-and-hold devices are Hybrid Systems 725LH devices, which are accurate to within 0.01% and have a droop rate of 15 millivolts per second using their internal holding capacitor. The sync input used to control the sampling period is a 5-volt, 1-to-100-hertz pulsed voltage. The converters are Teledyne Philbrick 470501 devices with an upper limiting frequency of about 1 megahertz. This frequency is produced at an input of 10 v, and the device's voltage-frequency characteristic is linear to 0.005%. The converter may be easily calibrated with its 50-kilohm trimmer potentiometer and the 200-ohm rheostat at the output of the 725LH device.

The 4040 reference scaler is a 12-bit binary counter. After reaching its counting capacity (1,024) during a given sampling period, it clocks the contents of the Signetics 4518B into the 174C174 C-MOS holding register while resetting the gating circuits. The sample scaler capacity (16 bits) and the large reference scaler capacity (12 bits) ensure a high counting accuracy, typically to within 0.1%. In addition, the larger capacity of the sample scaler allows the signal amplitude to exceed the reference amplitude while the correct ratio is still displayed.

The ΔT function in the block diagram is a small but important part of the circuit. It is implemented as shown in Fig. 2 with a number of one shots to achieve correct timing and edge triggering for data transfer. The digitizing time for the circuit shown is 4 milliseconds. Greater speed (with less accuracy) can be easily achieved by reducing the number of bits in the reference scaler. For instance, a digitizing time of 250 microseconds may be achieved with an 8-bit reference scaler. □



Experimenter's Corner

By Forrest M. Mims

THREE-STATE LOGIC

IF YOU want to stay abreast of the latest developments in digital logic and microprocessor technology, you need to know something about three-state logic. This month, we're going to experiment with circuits that will teach you the basics of three-state logic in an hour.

Suppose you need to connect the outputs from two or more gates to a common terminal, perhaps the input to another gate. This is OK in the unlikely event *all* the outputs are consistently low or high; but what happens if the outputs are at different logic states? Obviously, it's not possible to place logic 0's and 1's on a common terminal without creating mass confusion—and possibly damaging one or more gates.

Three-state logic provides an efficient solution to this design problem. The output of a conventional logic gate is *always* low or high as long as power is applied. A three-state gate, however, employs a clever circuit that effectively isolates the gate from the output terminal. This requires that a special control terminal called the *enable input* be added to the gate.

Figure 1 shows two buffers with three-state outputs. When their enable inputs are activated, these buffers pass the logic state of their inputs to their outputs. When the buffers are not enabled, the outputs enter a high-impedance state. This high-impedance output state means the outputs of a dozen or more buffers (or any other three-state logic gate) can be connected to a common terminal if only one is enabled at any one time.

Many digital circuits, particularly microprocessors and memories, use common terminals called *buses* to transmit binary bits or words (a group of bits). Thanks to three-state logic it's possible to connect many different circuits to a common bus so long as one simple rule is followed: The output of only *one* circuit connected to a bus can be enabled at any one time. If more than one output is enabled, logic 0's and 1's will be placed on the bus at the same time, and we're back to the problem that first caused us to employ three-state logic.

We'll look at three-state buses again later. First, let's get some hands-on experience with a three-state buffer.

Three-State Buffer Demonstrator. Figure 2 shows a simple circuit you can quickly build on a solderless breadboard to demonstrate how three-state logic works. It uses one of the gates in a 74125 quad three-state buffer. The two LED's indicate the logic state applied to the input of the buffer when the enable input is at logic 0. When *LED1* is on, the input is low. When *LED2* is on, the input is high.

When the enable input is high, the output of the buffer enters and remains in the high-impedance state irrespective of the logic state at the buffer's input. Both LED's will then glow at about half their normal brightness, conducting a limited amount of current along the path between 5 volts and ground through the series resistors and the LED's.

Here's a truth table that sums up the operation of the demonstrator circuit:

Enable	Input	Output	
		LED1	LED2
0	0	ON	OFF
0	1	OFF	ON
1	0	*	*
1	1	*	*

*Both LED's at half brightness.

Three-State Multiplexer. A multiplexer is a data selector. Apply an appropriate input select signal and one of several inputs will be applied to a single output. Figure 3 shows how you can make a 4-to-1-line multiplexer from a quad, three-state buffer like the 74125. The enable inputs of the buffers are used as the data select inputs. Remember, only one buffer can be enabled at any one time. With that in mind, here's the truth table for the multiplexer:

Data Inputs				Data Select				Output
A	B	C	D	A	B	C	D	
0	X	X	X	0	1	1	1	0
1	X	X	X	0	1	1	1	1
X	0	X	X	1	0	1	1	0
X	1	X	X	1	0	1	1	1
X	X	0	X	1	1	0	1	0
X	X	1	X	1	1	0	1	1
X	X	X	0	1	1	1	0	0
X	X	X	1	1	1	1	0	1

Note: The X means "don't care"; the input can be either a 0 or 1.

If you build the circuit in Fig. 3, you can apply the data select inputs with a 4-position rotary switch (rotating contact connected to ground) or a 1-of-4 decoder like half of a 74139. The decoder will condense the data select inputs to four 2-bit addresses.

Three-State Bus Demonstrator. Figure 4 shows a simple circuit that will teach you how a three-state bus works. The circuit uses a 74173 4-bit data register with a built-in, three-state output buffer. This means you can connect both the inputs and outputs of the register to the same bus (!) *and* control the transfer of data into and out of the register by applying appropriate signals to the register's read and write inputs.

For best results, build this circuit on a solderless breadboard. Use four rows of adjacent terminal receptacles for the bus and an 8-position DIP switch for the data input and control switch. To write a data word into the register, place the word on the bus by loading it into the first

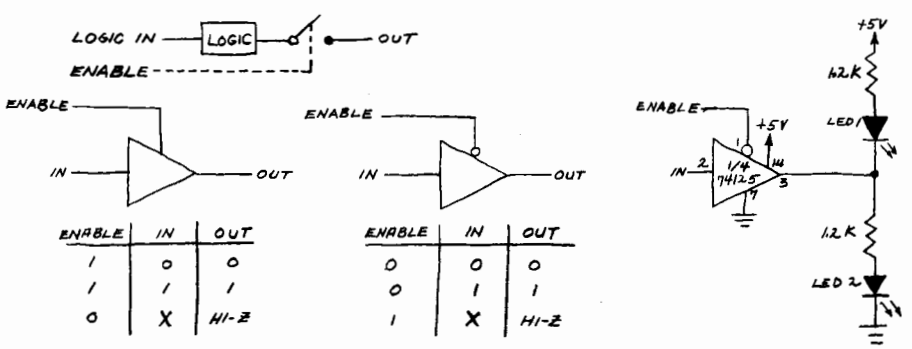


Fig. 1. Two three-state buffer configurations (left).
Fig. 2. Three-state buffer demonstrater (right).

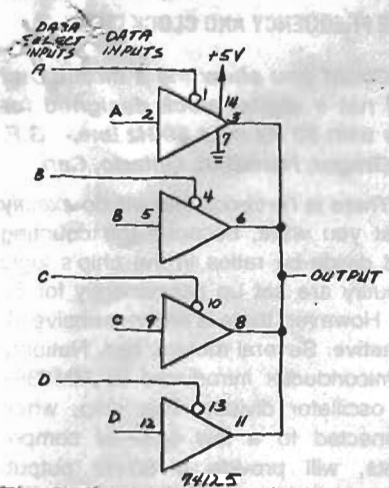


Fig. 3. Three-state multiplexer.

four poles of the DIP switch (let on = 1 and off = 0) and turning switch 8 on. The LED's will display the word you've switched into the input (LED one = logic 1 and LED off = logic 0).

The register will accept a data word from the bus when the WRITE input is low and the positive edge of a clock pulse arrives at pin 7. Prepare to load the data word into the register by turning switches 6 and 7 on. Then apply a clock pulse by turning switch 7 off. This disconnects the CLOCK input of the 74173 from ground, which is the equivalent of applying a positive pulse (unconnected TTL inputs go high). Don't worry about extra clock pulses from the bouncing that occurs when you throw the switch. The data word is copied on the first rising bounce, and any subsequent bounces simply recopy the same word.

After the data word is written into the 74173, turn switch 8 off to remove the input data from the bus. Then turn switch 6 off. To see the word stored in the register, just turn switch 5 on. This will activate the READ input of the 74173 and connect the register's output to the bus. This will display the stored word.

Going Further. You can expand the three-state demonstrator by adding a second 74173 to the data bus. You can connect the CLOCK input of the new register to the CLOCK input of the original 74173, but you'll need a couple of switches on a second DIP switch for the additional READ and WRITE inputs.

Can you think of a practical use for the three-state bus demonstrator? A bus system like this can send data between registers in *either* direction. Therefore, it's often called a *bidirectional data bus*. If that rings a bell, it's because the bidirectional data bus is used in most microprocessors. In fact the simple three-state bus demonstrator we've been experimenting with is functionally equivalent to part of a microprocessor.

In a real microprocessor, of course, the signals that activate the control inputs of the various registers and circuits are automatically supplied by a circuit called a *controller*. The signals from the controller are binary bit patterns called *microinstructions*. ◇

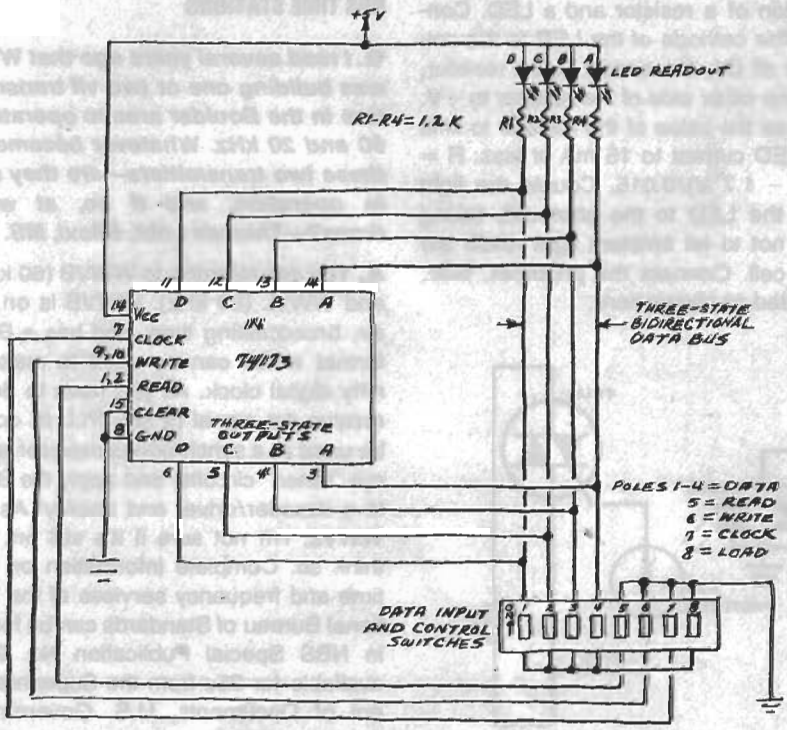


Fig. 4. Three-state bus demonstrater.

Capacitance-coupled logic fills unusual jobs

by Stephen R. Pareles
Cook College of Environmental Science, New Brunswick, N.J.

Capacitively coupling logic signals may prove to be a simple way to do several not-so-simple jobs. For instance, capacitive coupling can make short work of bidirectional pulse-edge detection, as well as comparison of an analog signal and a digital signal.

With the circuit of Fig. 1 and a single-trace oscilloscope, an analog signal and a digital signal can be displayed at the same time, allowing the two signals to be compared or synchronized. The circuit's output is the analog signal with superimposed digital cursors.

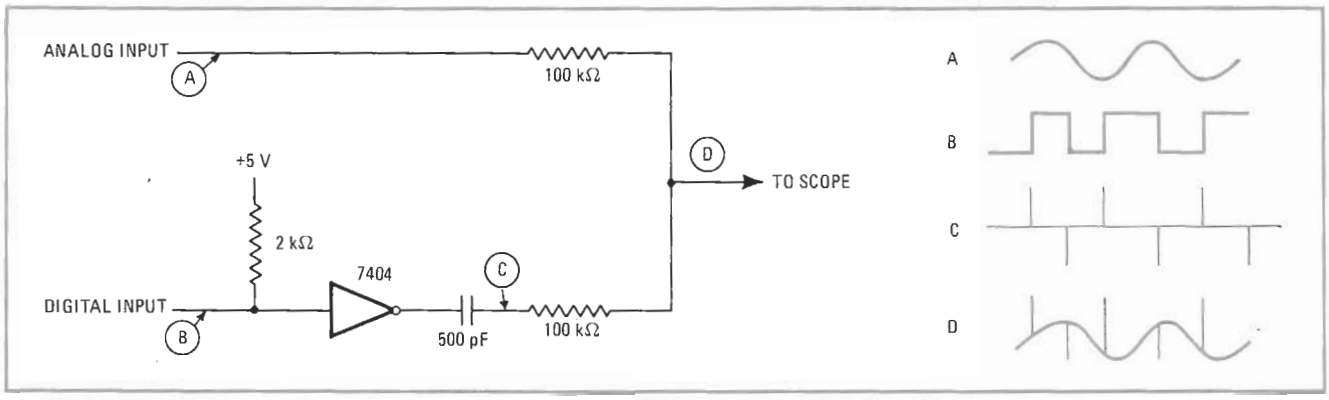
The capacitor serves as a bidirectional edge-detector for the buffered arbitrary logic train. Analog-level transients are produced by the capacitor from this input logic train. They are positive for leading pulse edges and negative for trailing pulse edges.

These transients are then cross-coupled with the analog signal through resistors that provide cross-current isolation (100-kilohm resistors are sufficient for most applications). A capacitance of 500 picofarads is ideal for slow horizontal sweep rates of up to about 100 hertz. Smaller capacitance values should be used for faster sweep rates to prevent the trailing edges of the transients from becoming observable.

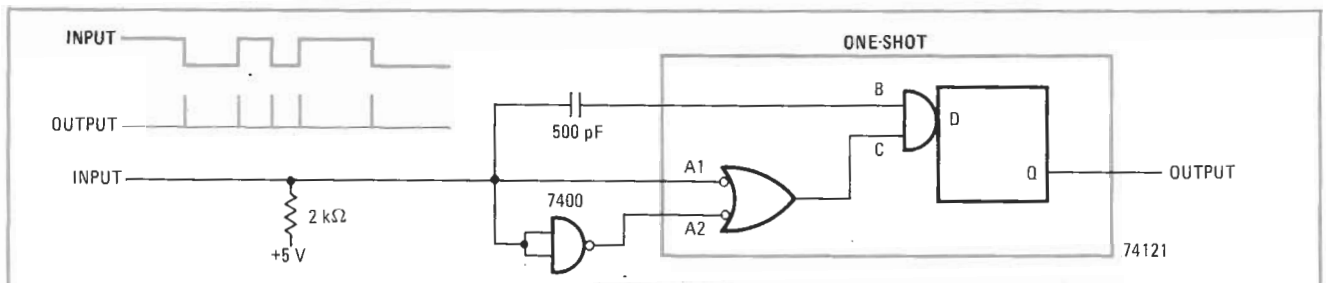
Capacitive coupling can also be used to perform bidirectional edge-detection when a logic-level output is desired. The detector circuit, which is drawn in Fig. 2, can even handle variable pulse widths.

Normally, a 74121-type one-shot is only triggered by a positive transition at point D, following a low condition at points D and Q. When the input first goes high, point A1 goes high. Since point A2 is still high, point C momentarily remains low. When A2 goes low and C high, the one-shot is triggered by the positive edge at D. Point B is kept high throughout.

When the input goes low, A1 goes low before A2 goes high, so that C remains high. Point B, however, is momentarily low. When B goes high again, the one-shot is triggered by the positive edge at C, as before. The tables in Fig. 2 detail the circuit's operation at key points. □



1. Two-signal display. A capacitor simplifies the task of observing two signals on a single-trace oscilloscope. The circuit's output becomes the analog input with superimposed digital timing cursors. The two 100-kilohm resistors provide the necessary cross-current isolation.



CIRCUIT OPERATION

TIME	POSITIVE INPUT TRANSITION							NEGATIVE INPUT TRANSITION						
	INPUT	A1	A2	C	B	D	Q	INPUT	A1	A2	C	B	D	Q
t_n	0	0	1	1	1	1	0	1	1	0	1	1	1	0
TRANSITION	1	1	1	0	1	0	0	0	0	0	1	0	0	0
t_{n+1}	1	1	0	1	1	1	ONE-SHOT	0	0	1	1	1	1	ONE-SHOT

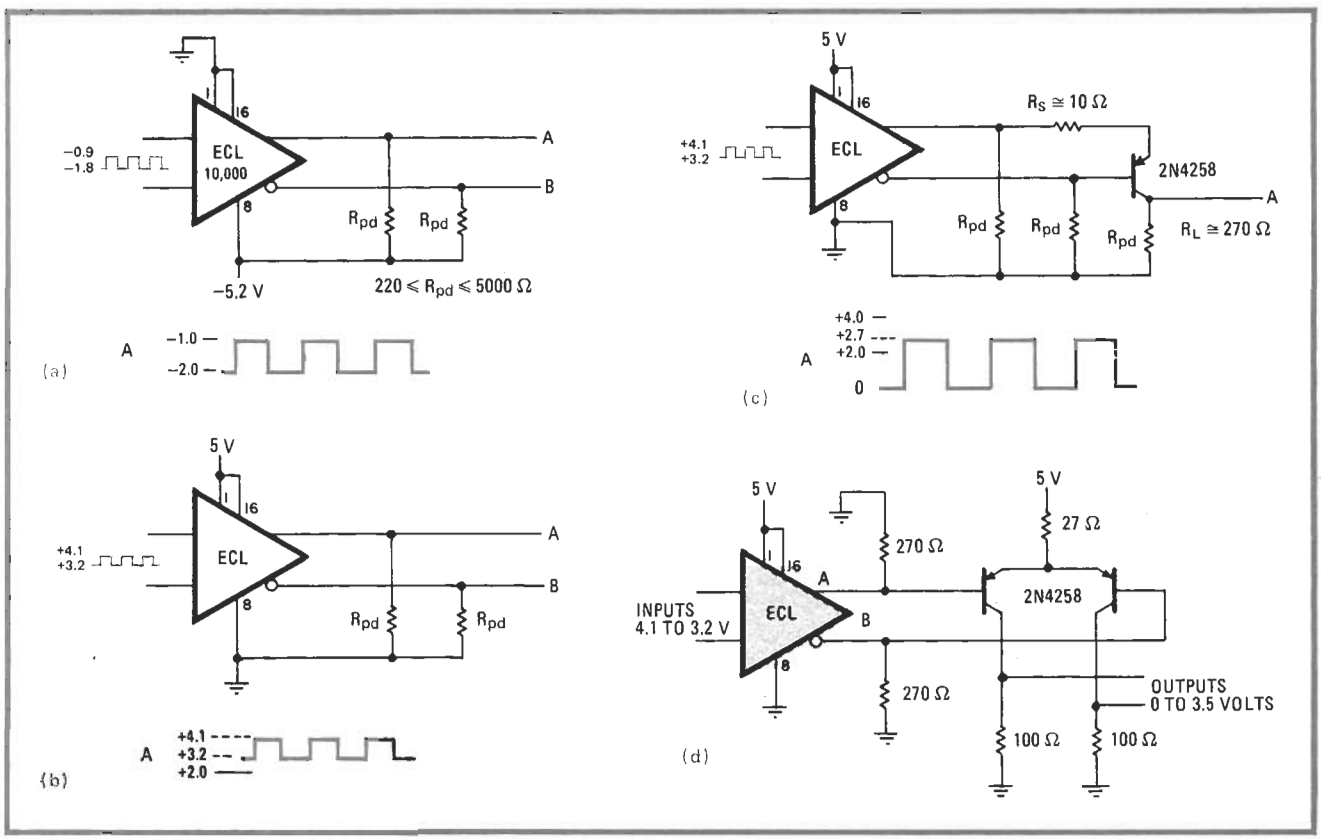
2. Dual edge-detection. Both the leading and trailing edges of the input-pulse train are detected by this capacitively coupled circuit.

Appropriate biasing mates ECL and TTL families

by William A. Palm
Magnetic Peripherals Inc., Minneapolis, Minn.

The speed and flexibility of emitter-coupled logic can be combined with the convenience of transistor-transistor logic in circuits that work over a wide range of frequencies. The easiest and most economical way to mate the two logic families is to adapt ECL to the 5-volt operation of TTL, since the rebiasing of ECL elements is easily accomplished. Then the circuits can be powered by a single supply voltage.

Emitter-coupled logic is costly and draws considerable power at higher frequencies. So there is little sense in using it throughout in such a circuit as a 100-megahertz counter/divider, for example, when the part of the circuit that operates at lower frequencies could be implemented with TTL. As shown in (a), the typical dual-output logic gate in the Motorola 10,000 series requires a supply voltage of −5.2 V, conventionally wired as shown in (a). ECL circuitry is so configured that such a



Evolution of an interface. Standard emitter-coupled logic is powered by at least one negative voltage, cannot drive or be driven by TTL (a). Re-biased ECL device generates positive output voltage, but not within proper TTL switching threshold range (b). Adding a transistor enables 0–2.7-V swing, suitable for TTL (c). Five-volt ECL gate and two transistors drive 100-ohm TTL loads at 0–3.5 V (d).

gate will operate if there is a supply voltage differential of 5 v between pin 8 and pins 1 and 16, independent of the actual values as long as they are within device limits. Thus, it is permissible to place a 5-v supply voltage at pins 1 and 16 and to ground pin 8 as shown in (b).

This arrangement is not suitable for driving TTL because an input of 3.2 to 4.1 v results in an output voltage swing at point A of only 3.2 v (logic 0) to 4.1 v (logic 1). But note that the voltage at B is the inverted output of A. By using both outputs and adding a transistor to shift the output swing levels (c), approximately 10 milliamperes is made to flow through the 270-ohm collector resistor at point A when the transistor is saturated. Thus there will be a 2.7 v drop across R_L when the transistor is on, and the voltage will go to zero when the transistor is off, enabling TTL elements (or any other elements, for that matter) to be driven.

The fourth circuit (d) is useful in applications where considerable speed and output current are required to drive a balanced load. There will be a 1-v drop across the 27-Ω resistor during the time each transistor conducts, and so 35 mA will flow through the 100-Ω collector resistors. Thus 3.5 v will be developed at each output. This circuit is suitable for driving a 100-Ω twisted-pair cable. □

Engineer's notebook is a regular feature in *Electronics*. We invite readers to submit original design shortcuts, calculation aids, measurement and test techniques, and other ideas for saving engineering time or cost. We'll pay $50 for each item published.

Calculator notes

Standard symbols let designers grasp logic operation quickly and easily

ANSI Y32.14 specifies set of symbols for clearly depicting logic, from gates to systems

by Bill King, *Hewlett-Packard Co., Santa Clara (Calif.) Division*

☐ The more complex integrated circuits become, the greater the detail in which designers and technicians need to understand their workings. To give them that information at a glance, manufacturers must depict the logic operation of their chips clearly and concisely.

In 1973, therefore, the American National Standards Institute approved and published ANSI Y32.14, which set the logic-symbol specifications for most devices—notably gates, flip-flops, and counters—and for systems containing them. But few users are familiar with the standard, because so far it has been adopted by only two manufacturers—by Hewlett-Packard Co. and, to some extent, by Texas Instruments Inc. Nonetheless, since ANSI does set standards for IC makers, its symbolism is likely to become widely accepted.

ANSI Y32.14 specifies:

- Definitions for the basic logic elements.
- Logic symbols, which show the defining shapes corresponding to the logic function performed.
- Qualifiers, which consist of letters, numbers, or arrows placed inside the logic-device symbol to indicate its logic function or special properties.
- Indicators, which show primarily if the input and output are active high or low.
- Dependency notation, which defines the logic-state relationship between the inputs required to activate the device.
- Control and contiguous blocks, which integrate with gates, flip-flops, and other elements to form shift registers, counters, and other commonly used devices.

TABLE 1: BASIC LOGIC ELEMENTS

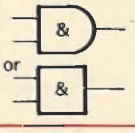
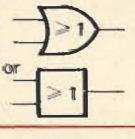
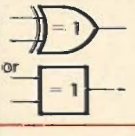
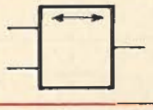
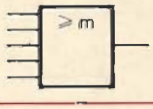
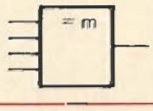
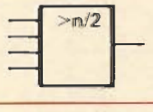
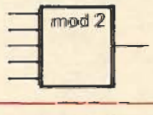
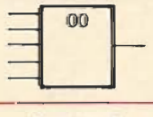
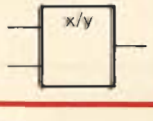
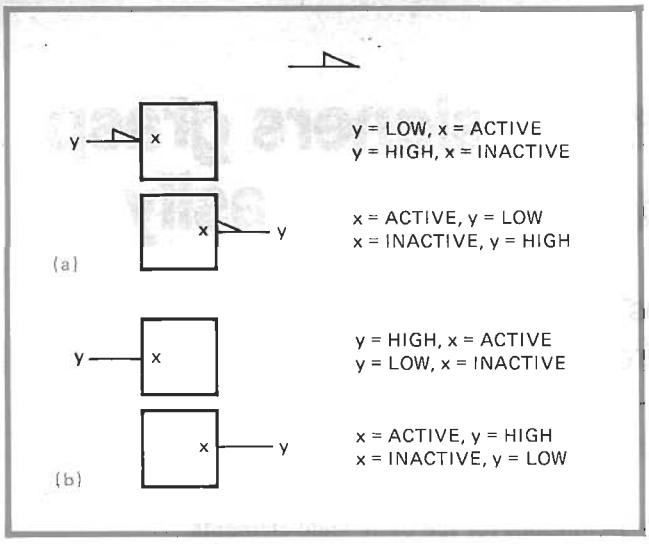
Symbol	Function	Description
	amplifier	The output will be active only when the input is active (can be used with polarity or logic indicator at input or output to signify inversion).
	AND	The output will assume its indicated active state only when all its inputs assume their indicated active levels.
	OR	The output will assume its indicated active state only when any of its inputs assume their indicated active levels.
	exclusive-OR	The output will assume its indicated active level if, and only if, only one of the inputs assumes its indicated active level.
	wired-AND	This is a connection of outputs of two or more elements that are joined together to achieve the effect of an AND function.
	wired-OR	This is a connection of outputs of two or more elements that are joined together to achieve the effect of an OR function.

TABLE 2: SELECTED QUALIFIER DESIGNATIONS

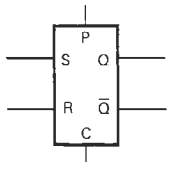
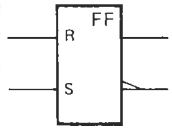
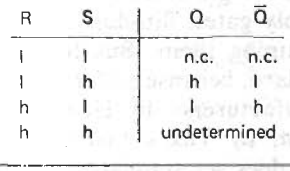
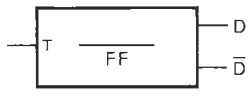
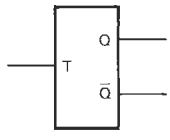
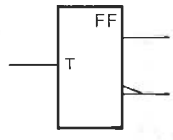
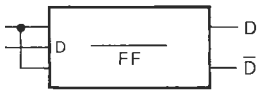
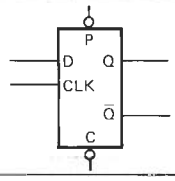
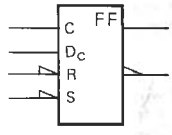
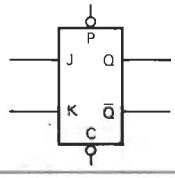
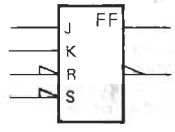
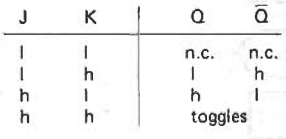
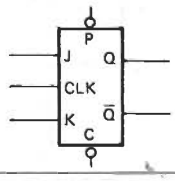
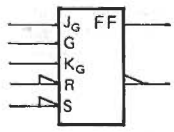
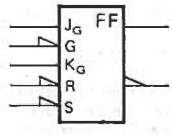
Symbol	Description
	Bilateral switch: a binary-controlled circuit that acts as an on-off switch to analog or binary signals flowing in both directions.
	Logic threshold: output will assume its active state if m or more inputs are active.
	m and only m: output will be active when m and only m inputs are active (for example, exclusive-OR).
	Majority function: output will be active only if more than half the inputs are active.
	Odd function: output is active only if an odd number of inputs are active.
	Even function: output is active only if an even number of inputs are active.
	Signal-level converter: input levels are diffenent from output levels.



1. Polarity convention. Indicator symbol (top) signifies that corresponding inputs or outputs are active low (a), thereby characterizing circuit operation without use of labeled outputs. The absence of the symbol (b) indicates inputs and outputs are active high.

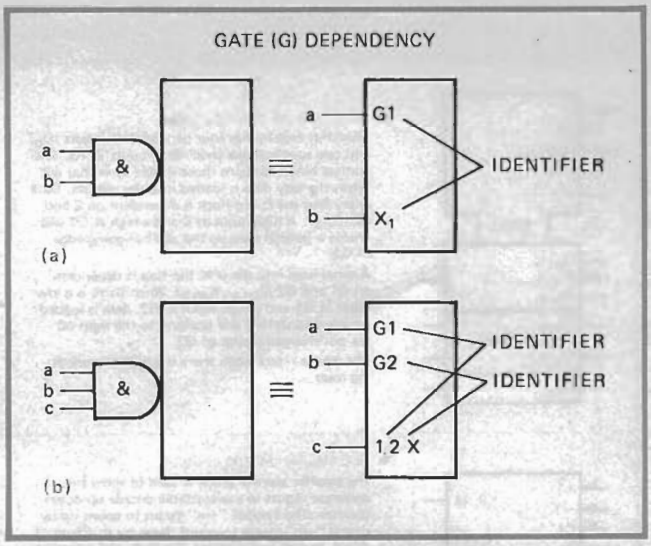
Table 1 gives the definitions of the basic elements—the amplifier, AND, OR, exclusive-OR, wired-AND, and wired-OR circuits—and their logic symbols. Note that the AND, OR, and exclusive-OR can be shown by their assigned shape or by a rectangle, since the presence of an identifying symbol within those elements specifies the device function. The inverting function for these elements (that is, inverter, NAND, or NOR) is indicated by placing the negation symbol (a small circle) at the corresponding output ports of the devices—the same symbol used currently. In addition to specifying the function of a logic element, qualifier symbols are used for classifying logic blocks. Table 2 shows the symbols

TABLE 3: FLIP-FLOP SYMBOLISM

Flip-flop	Original symbols	Previous standard MIL-STD-806B	ANSI Y32.14 Control designations description for flip-flop	
R-S				
T				Toggling occurs with every clock pulse.
D				Data output follows data input; input is gated by C.
J-K				
J-K (gated)				J and K inputs are gated by C.
J-K (master-slave)	—	—		Outputs are dependent on the negative-going edge of the clock.

n.c. = no change

2. Dependency notation. Block-diagram equivalent of two-input AND gate (a), which drives one-shot, provides quick overview of circuit operation. Identifier indicates dependency between inputs a and b, showing data on b is gated in by a. Approach to coding up three-input AND gate follows logical extension of method (b).

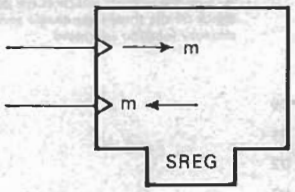


for some of the most widely used ones.

The polarity indicator symbol, shown in Fig. 1 (top), establishes the active states of the input leads required to switch on the logic element or indicates whether the output leads are active high or low. Any input or output so labeled is active low (a). Otherwise, the inputs or outputs are active high (b).

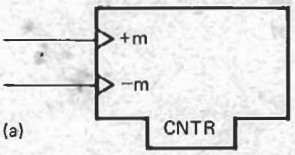
Although this symbol provides the same information as the negation symbol, it offers the advantage of visually representing the signal polarity required to activate the device. Furthermore, it eliminates the inconsistent labeling of logic devices. For example, the inverted

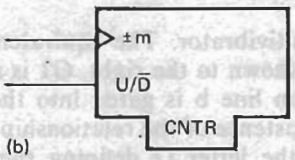
TABLE 4: COMMON CONTROL-BLOCK DEFINITIONS



Shift register block

This symbol is used with an array of flip-flop symbols to form a shift register. The data will shift to the right (→) or to the left (←) on the positive-going edge of the input signal.



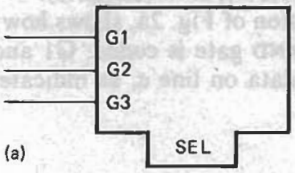


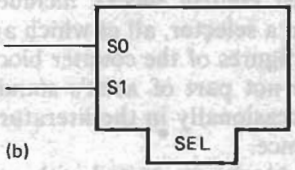
(not ANSI-defined)

Counter block

This symbol is used with an array of flip-flops or other circuits serving as a binary or decade counter. In (a), the positive-going edge of an input to either the +m or −m input causes the counter to count upward or downward m times.

In (b), a positive-going edge of an input to the ± m port will cause the counter to increment or decrement m times depending on the input to the up-down control (U/D).



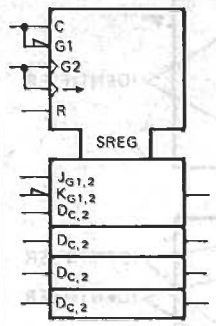


(not ANSI-defined)

Selector block

This control block is used with an array of OR symbols to provide for the gating lines (a) or selection lines (b). The gating lines have an AND relation with the respective input of each OR function: G1 with the inputs numbered 1, G2 with the inputs numbered 2, etc. The selection lines enable the input designated 0, 1, . . . n of each OR function.

TABLE 5: COMMON DEVICE BLOCKS

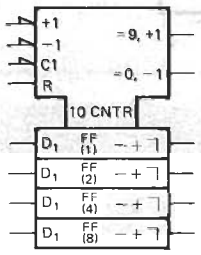


- 4-bit universal shift register

This shift register has four parallel data inputs (D_C) and two synchronous serial data inputs (J, K). The control block contains those enable lines that will select the way data is loaded into the register. Data entry into the D flip-flops is dependent on C and G2 ($D_{C,2}$). A high input at C and a high at G2 will enable a parallel load on the positive-going edge of G2.

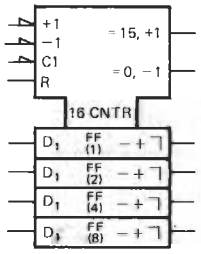
A serial load into the J-K flip-flop is dependent on G1 and G2 ($J_{G1,2}$, $K_{G1,2}$). When there is a low input at G1 and a high input at G2, data is loaded serially and shifted one position to the right on the positive-going edge of G2.

The device resets when there is a high present at the reset.



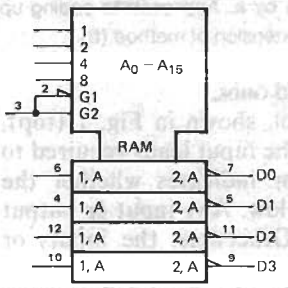
- Presettable decade counter

The counter control block is used to show the common inputs to a presettable decade up-down counter. The symbol "+m" means to count up by m and "−m" means to count down by m. (Note: if m = 1, it may be omitted.) The "=9, +1" symbol is the carry output or the terminal count up when the count equals 9. The "=0, −1" symbol is the borrow output or the terminal count down when the count equals 0. C1 is the control input for the D flip-flops and R is the master reset. When C1 is enabled, it loads all four flip-flops (D_1) in parallel. The presence of the output delay indicator is used to indicate that the D flip-flops are master-slave. Flip-flop weights are indicated in the parentheses. The symbol "10 CNTR" indicates that the counter is modulus 10.



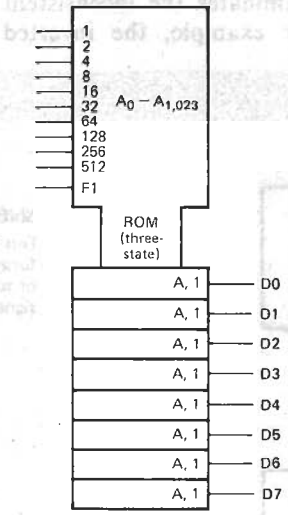
- Presettable binary counter

Same as above except that the carry output is indicated with "=15, +1," indicating that the terminal count up occurs when the count equals 15. The symbol "16 CNTR" indicates that the counter is modulus 16.



- Random-access memory (without identical input/output pins)

Address selection is determined by the 4-bit address input codes in the upper left corner of the control block. These address codes are weighted to correspond to the possible address ($A_0 - A_{15}$). G1 and G2 are the read/write enable. A low on pin 2 will enable data to be read into the chip in the memory location addressed; a high on pin 3 will enable the chip to output data from the particular memory location addressed. The inputs on the lower left corner of the symbol are labeled 1, A. This indicates that the information will be stored in the memory location addressed. 2, A on the outputs indicates that the data will be written from the memory location when G2 is enabled.



- Read-only memory

This is a read-only memory with 1,024 addresses. Address selection is determined by the 10-bit address input in the upper left corner of the control block.

F1 is the three-state enable line. A high signal on this line will enable the outputs. A, 1 on the outputs indicates the dependency on the three-state enable and the memory location addressed.

output of a flip-flop is normally designated $\overline{Q}$, and the inverted R and S inputs have negation symbols at their ports, rather than being labeled $\overline{R}$ and $\overline{S}$. These ports will now be labeled Q, R, and S with appropriate polarity indicators. This change can be seen in Table 3, which shows the development of flip-flop symbolism.

Block form

By providing for dependency symbols and one-block devices, ANSI Y32.14 makes it easier for system designers to understand the operation of large circuits. So-called control blocks, which group the common control inputs, can be joined to contiguous blocks, which depict the remainder of the circuit (an array of gates, flip-flops, etc.). A combination of control and contiguous blocks forms a device block.

Figure 2 shows the application of dependency symbols. Dependency is indicated by subscripts, prefixes, or suffixes. For example, in the case of D_1, the 1 indicates a logic connection between the input, D, and a control line assigned the numeral 1. In prefix form, the notation becomes 1D; in suffix form, D1.

In the simple example of Fig. 2a, a two-input AND gate drives a one-shot multivibrator. The equivalent dependency for the gate is shown to the right. G1 is an input, through which data on line b is gated into the device. The 1 identifies the existence of the relationship between lines a and b, with the letter G defining the type of relationship (AND-gate dependency). The appropriate letter identifies other relationships: A (address), C (control), F (free, or three-state), or V (OR-gate).

Figure 2b, an extension of Fig. 2a, shows how a circuit having a three-input AND gate is coded. G1 and G2 are the gating inputs for data on line c, as indicated by the 1,2 of the input 1,2 X.

Symbol buildup

ANSI's recommended control blocks include a shift register, a counter, and a selector, all of which are shown in Table 4. The lower figures of the counter block and of the selector block are not part of ANSI's standard, but they have appeared occasionally in the literature and so are included for reference.

When these control blocks are united with contiguous blocks, such as flip-flops, then entire devices can be built. Several are illustrated in Table 5. □

Solid-State Developments

By Forrest M. Mims

Do-It-Yourself Logic Chips

IN THIS day of ultra-sophisticated semiconductor technology, large-scale and very-large-scale integrated circuits (LSI and VLSI respectively) containing hundreds or even thousands of logic gates have become commonplace. Nevertheless, examine any board containing one or more LSI or VLSI chips and you'll probably find an assortment of small- and medium-scale integrated circuits (SSI and MSI) with relatively few gates or flip-flops package.

Circuit designers have long wanted to combine in a few packages the relatively small number of gates and flip-flops required to support most LSI and VLSI chips. Custom ICs are usually out of the question because of their high price and long development time. And what happens if a design change is necessary?

Semi-custom integrated circuits are a better choice. These chips contain arrays of gates which have not been *metalized*. In other words, the gates are independent of one another since they've not yet been connected together electrically by a metalization pattern on the top surface of the chip. The customer tells the custom IC house how he wants the gates interconnected, and the gate chips are then metalized according to the customer's specifications and installed in DIPs.

This procedure is faster and cheaper than the custom IC route, but it's still relatively expensive since the customer usually must agree to buy a thousand or more chips. And as in the case of the custom IC, what happens if a design change is necessary?

A third alternative is the do-it-yourself logic chip. Included in this category are *field programmable logic array* (FPLA) and *programmable array logic* (PAL, a trademark of Monolithic Memories, Inc.) chips. These chips contain arrays of logic gates interconnected via the same kind of fusible links used to make programmable read-only memories (PROMs). By selectively applying high-current pulses to the programming pins of an FPLA or PAL, fusible links can be opened in various patterns to produce a customized integrated circuit.

The PROM is itself a versatile do-it-yourself logic chip since it can be used to implement any truth table for which it has sufficient inputs and outputs.

You can better understand the operation and compare the differences of PROMs, FPLAs and PALs be referring to Figs. 1, 2 and 3. They show the internal circuitry of ultra-simple, hypotheti-

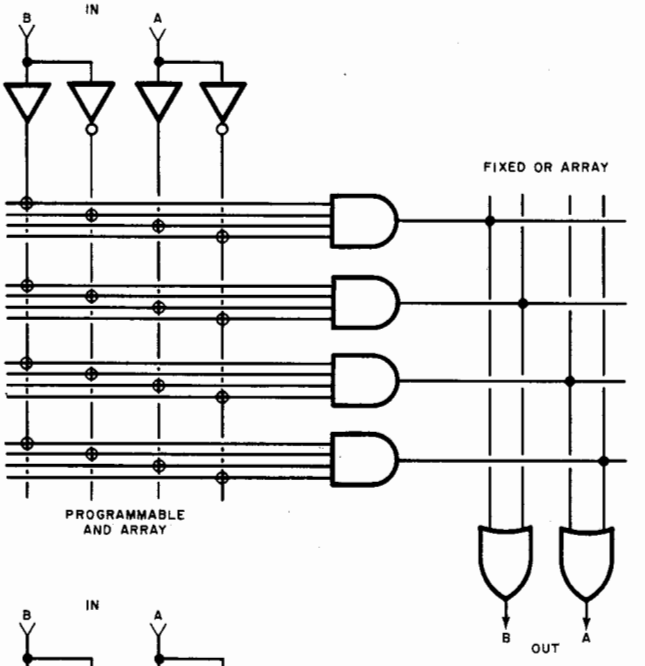


Fig. 2. A hypothetical PAL of two four-bit words. It is a backward PROM since the AND array is programmable while the OR is fixed.

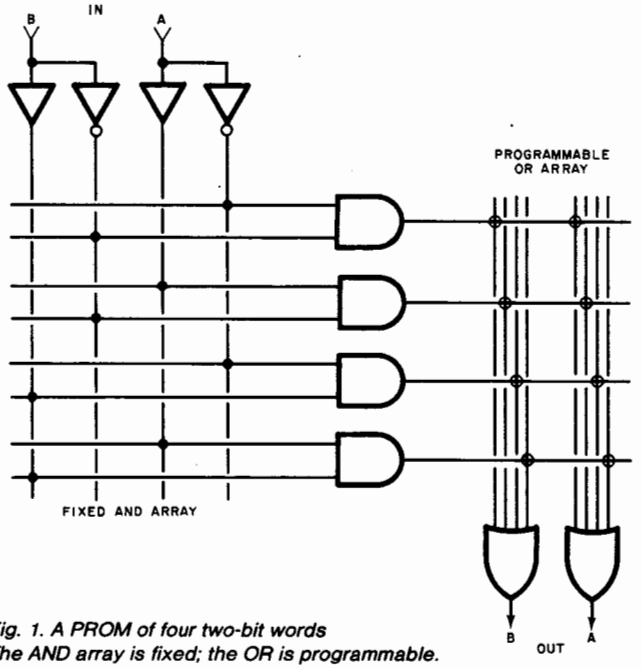


Fig. 1. A PROM of four two-bit words The AND array is fixed; the OR is programmable.

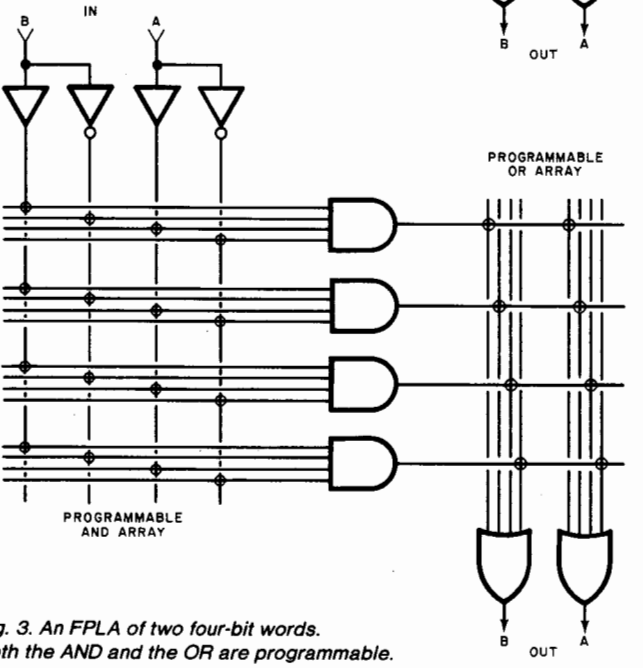


Fig. 3. An FPLA of two four-bit words. Both the AND and the OR are programmable.

cal versions of each of these three kinds of programmable logic arrays.

As is readily apparent from these figures, all three circuits contain an AND array followed by an OR array. The input word applied to the AND array can be considered an address, data word or bit pattern. In any case, the effect is the same since a particular input switches the output of one of the AND gates from low to high. The outputs then reflect whether or not connections are present at the junction of the output line from a selected AND gate and the input lines to the OR gates.

A solid dot at the intersection of two array lines means the connection was unalterably programmed when the chip was made. User programmable fusible links are indicated by small circles at intersection array lines.

In the PROM (Fig. 1), the AND array is permanently programmed or *fixed* while the OR array is programmable. The AND array in Fig. 1 is programmed to address in turn each of the AND gates from top to bottom according to a standard 00, 01, 10, 11 input sequence.

The PAL (Fig. 2) is a backward PROM since the AND array is programmable while the OR array is fixed. In real PALs the OR array is factory programmed to give some of the most commonly used logic functions.

The FPLA (Fig. 3) is the ultimate do-it-yourself logic chip since *both* the AND and OR arrays are programmable. While this provides the highest

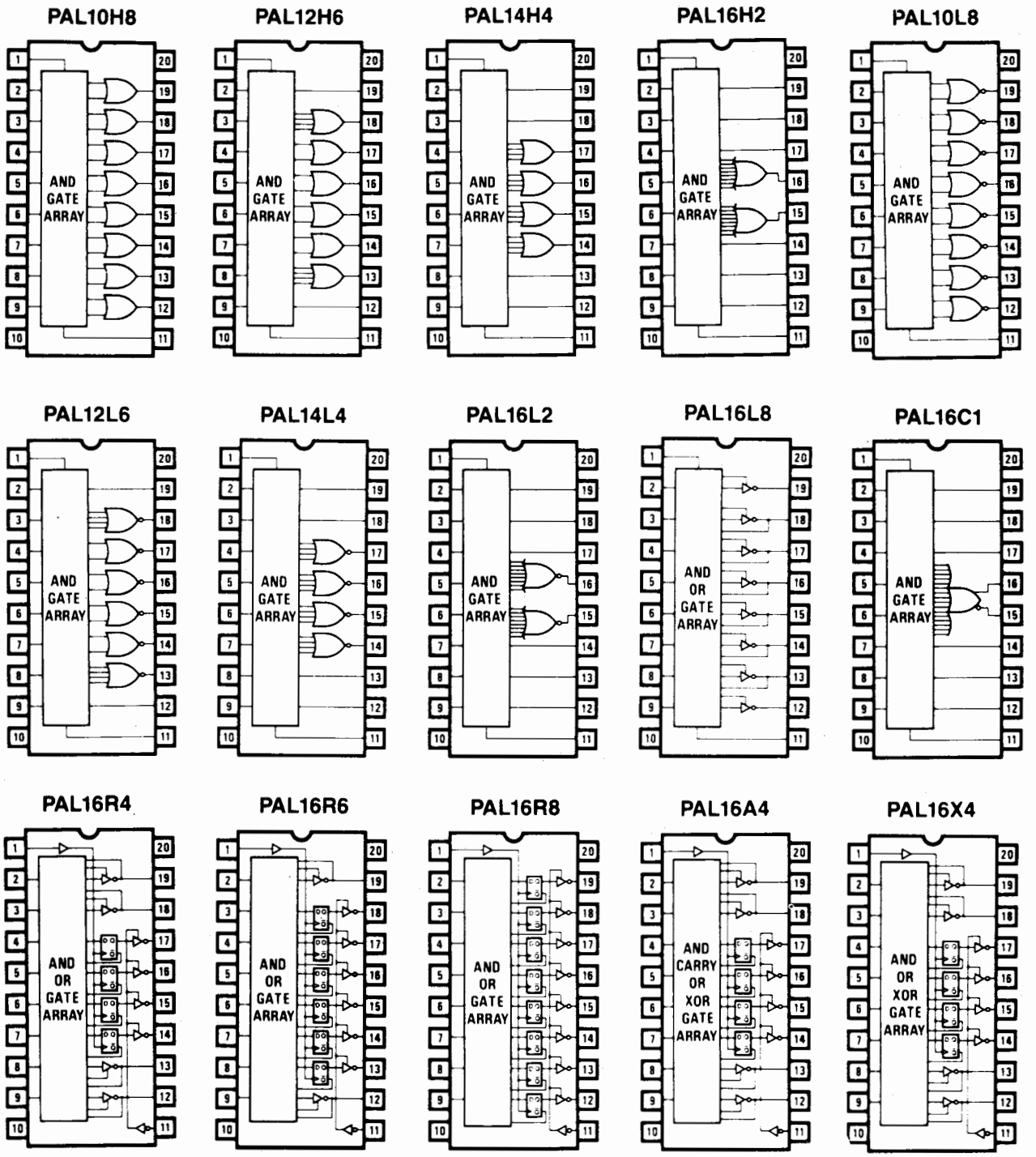


Fig. 4. Pin outlines and internal block diagrams of the PAL family of chips.

degree of flexibility, in practice the FPLA is much more dificult to use and more expensive than either the PROM or the PAL. All three kinds of chips can be programmed using standard PROM programmers, but the programming procedure for the FPLA is at least twice as cumbersome since both the AND and OR arrays must be programmed.

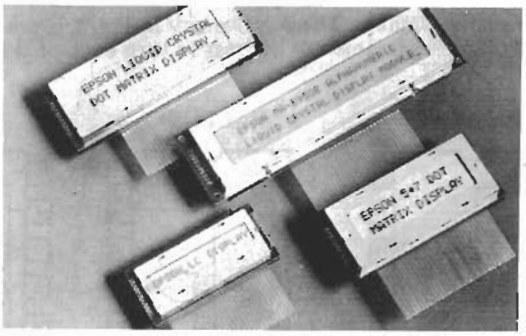


Fig. 5. Alphanumeric liquid-crystal displays from Epson America, Inc.

Some PALs and FPLAs include flip-flops to store output states and feed results back to the inputs. This makes possible such functions as counting, shifting and sequencing.

PALs without flip-flops can perform virtually any task now accomplished with SSI and MSI logic chips up to and including a 4-bit arithmetic logic unit! In many applications a single PAL can replace up to ten SSI/MSI packages.

A clever feature of PAL chips is a data security fuse. After the PAL has been programmed, the security fuse is blown to disable the circuit's internal verification logic. This prevents the internal program from being read out by a potential copier, thereby making the chip proprietary.

The PAL concept was pioneered by John Birkner of Monolithic Memories, Inc., and that firm now makes a family of fifteen PAL chips with National Semiconductor as a second source. Figure 4 shows the pin outlines and internal block diagrams for all fifteen chips. As you can see, considerable flexibility is provided by this lineup.

Information about PALs and FPLAs is not too abundant. The best way to learn more about PALs is to contact a Monolithic Memories or National distributor or representative. Try to obtain a copy of the excellent "PAL Programmable Array Logic Handbook" published by Monolithic Memories (1165 E. Arques, Sunnyvale, CA 94086).

Signetics (P.O. Box 9052, Sunnyvale, CA 94086) is a major maker of FPLAs. Their "Bipolar and MOS Memory Data Manual" contains FPLA data sheets and related information. Two Signetics engineers, Napoleone Cavlan and Stephen J. Durham, have written an excellent two-part article of the subject for *Electronics* (July 5, 1979, pp. 109-114 and July 19, 1979, pp. 132-139). In an article for *Computer Design* (April 1980, pp. 141-147), Mr. Durham described a complete 60-character keyboard encoder complete with key debouncing and made from a *single* Signetics 82S105 FPLA!

You can find the aforementioned articles in any good public or university library. For manufacturer's literature, check the yellow pages and call local electronics distributors or reps. If they can't help you, ask for the phone number of an authorized rep in any nearby city or state. If necessary, call the company direct. The cost of a few long distance calls may be well worth the results you'll harvest.

Do-it-yourself logic chips require careful design procedure and a PROM programmer so they're not necessarily suited for the typical hobbyist or experimenter. But if you want to greatly simplify a favorite logic circuit while learning about one of the latest trends in digital circuit design, get your hands on some manufacturer's literature and warm up your PROM zapper.

Component News. In a packet of recently received specification sheets for new National ICs was one which immediately attracted my attention. The new chip is the LH0082 Optical Communication Receiver. It's housed in a 14-pin metal DIP and includes a fast FET-input amplifier, output comparator with hysteresis (to prevent output oscillations near the reception threshold) and the feedback and coupling resistors and capacitors necessary for a complete receiver. With a suitable photodiode connected to its input, the LH0082 can receive 20-MHz analog signals sent via lightwaves through free space or by way of an optical fiber.

The new chip is housed in a metal DIP to reduce stray noise pickup and to

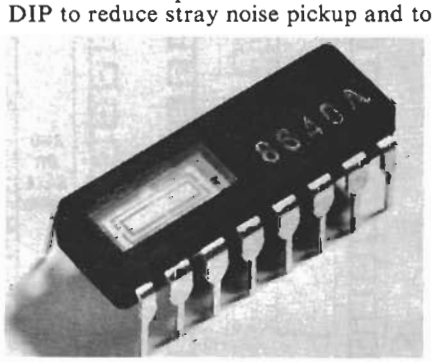


Fig. 6. Epson's programmable clock-pulse generator contains a quartz crystal oscillator.

CUT ALONG DOTTED LINE

Digital Reference Charts

NUMBER SYSTEM

Our present number system (decimal) uses base 10 to perform all necessary arithmetic operations. Example: $312 = 3 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$. Binary number system uses combinations of 0's and 1's to represent and perform all arithmetic operations that are possible in the decimal system. Binary number system uses base 2:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Table progresses to the left indefinitely as 2^n, where n = 0, 1, 2, 3, 4, . . . To change any binary number to base 10, write binary number underneath table and add decimal equivalent where 1's appear. Example, change 11000100 to decimal:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	0	0	0	1	0	0

$2^7 + 2^6 + 2^2 = 128 + 64 + 4 = 196$. The number then is 196.

To change decimal to binary: Divide number by 2 and keep remainder that will always give 1 or 0. Example: 196.

	Remainder
196 ÷ 2 = 98	0
98 ÷ 2 = 49	0
49 ÷ 2 = 24	1
24 ÷ 2 = 12	0
12 ÷ 2 = 6	0
6 ÷ 2 = 3	0
3 ÷ 2 = 1	1
1 ÷ 2 = 0	1

To write binary number from decimal begin at bottom row of remainder table and write the digits from left to right. Example: 196 = 11000100

BINARY WORDS

A bit is a digit in binary language. A digit can be either a 1 or 0. Binary numbers are also known as binary *words*. Consequently a *5 bit* binary number is referred as a *5 bit* word. The longer the word the greater the decimal number that can be represented. Example: A 5 bit word can represent 32 different combinations. A 2 bit word can represent 4 different combinations. By the equation: combinations = 2^n where n is the number of bits in the digital word. Example: Represent the maximum number of combinations with a 2 bit word. Combinations = $2^n = 4$. See table.

Decimal	Binary
0	00
1	01
2	10
3	11

The highest decimal number that can be expressed for a given word is the total number of combinations minus one. Equation: Decimal Number = $2^n - 1$. Example: What is the highest decimal number that can be expressed by a 2 bit word. The total decimal number is $2^n - 1 = 2^2 - 1 = 3$.

ADDITION

Binary addition is manipulated the same as decimal addition but keep in mind that 1 + 1 = 0 and 1 carries over.

Carry 1
Carry 1
Carry 1

```
  1 1 1
  1 1 1
1 0 1 0
```

Addition table	
	Carry
0 + 0 = 0	0
0 + 1 = 1	0
1 + 0 = 1	0
1 + 1 = 0	1

Examples:

```
Add  101011        Add  10000
   + 111001           + 00100
   -------            -------
   1100100            10100
```

SUBTRACTION

Reversed procedure from addition remembering to borrow one from the left column.

```
  1001
- 100
------
  0101
```

Here borrow one from the left which makes the next number 1 − 1 = 0

Subtraction table	
	Borrow
0 − 0 = 0	0
1 − 0 = 1	0
1 − 1 = 0	0
0 − 1 = 1	1

MULTIPLICATION

Proceed as arithmetic multiplication.

Example:

```
  10011
X   101
-------
  10011
 00000
10011
-------
1011111
```

TABLE

0 X 0 = 0
1 X 0 = 0
0 X 1 = 0
1 X 1 = 1

DIVISION

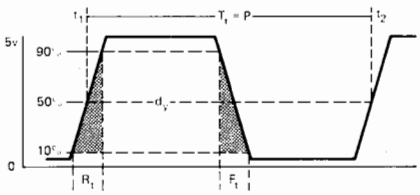
Binary division: proceed as arithmetic division.

```
    1100
10 | 11000
     10
     ---
     010
      10
      --
      0000
```

Division Table
0 ÷ 1 = 0
1 ÷ 1 = 0

DIGITAL PULSES

In digital logic, pulses are defined as voltage transitions that occur during a determined lapse of time. Timing pulses are usually generated by a circuit called a clock. A set of digital pulses is known as a train of pulses. The duration and length of the pulses can be changed to fit the needed application. A pulse is an abrupt change in a voltage level.



For the pulse shown in the figure: the maximum amplitude is 5 volts; the reference level is 0 or ground. The total time duration is from t_2 to $t_1 = t_{total}$. If the pulse occurs 1 time in one second, the frequency is one pulse per second. The period = 1/f. The duty cycle = d_y/P.

P = 1/frequency

The left-hand side of the pulse is called the leading edge. The right-hand side of the pulse is called the trailing edge. The width of the dotted area under the leading edge is known as the pulse risetime. The width of the dotted area under the trailing or decay edge is known as the falltime. Digital pulses are fast pulses with very sharp leading edges and very sharp trailing edges. *Positive logic* equates the maximum amplitude of the pulse as the digital number 1 and the ground or zero level as digital logic zero. *Negative logic* equates the maximum amplitude of the pulse as the digital number 0 and the ground or zero level as digital 1.

OCTAL NUMBER SYSTEM

Number system widely used in computer language. The octal system is a simpler way to store and recall values stored in computer memory banks. The binary numbers to be stored are converted to octal saving considerable time and possible errors. The octal number system has a base of 8. The symbol to represent the numbers are: 0, 1, 2, 3, 4, 5, 6 & 7. Notice the absence of the number 8. The conversion from decimal to octal follows the same procedure as the decimal to binary, but the base is 8.

8^4	8^3	8^2	8^1	8^0
4096	512	64	8	1

The table progresses indefinitely to the left as 8^n; where n = 0, 1, 2, 3, 4 . . . .

Convert the octal number 421_8 to decimal.

8^4	8^3	8^2	8^1	8^0
4096	512	64	8	1
		$4X8^2$	$2X8^1$	$1X8^0$
		256	16	1

Add the powers of 8 to get the decimal number. Example: 421 octal = $4 \times 8^2 + 2 \times 8^1 + 1 \times 8^1 = 273_{10}$ (decimal). To convert decimal number to octal, divide decimal number by 8 and keep remainder that will give octal number.

	Remainder
273 ÷ 8 = 34	1
34 ÷ 8 = 2	2
4 ÷ 8 = 0	4

To write octal number begin at the bottom row or last digit of the remainder and write octal number from left to right. Example: 421_8 (octal) = 273_{10} (decimal). The conversion of a binary digit to octal is accomplished by grouping the binary number into groups of 3 digits. Then proceed to evaluate the octal number in each group of 3 digits. Example: Change binary number 111100111110 to octal. Separate from right to left in groups of 3. Example: 111-100-111-110 = 7476_8. Change each group of 3 digits to octal converting binary to decimal. $111 = 7_8$; $100 = 4_8$; $110 = 6_8$.

CUT ALONG DOTTED LINE

BOOLEAN ALGEBRA RULES

▲ symbol indicates a logic variable. This can be either a logic 0 or a logic 1 level.

1	$0 + ▲ = ▲$
2	$1 + ▲ = 1$
3	$▲ + ▲ = ▲$
4	$▲ + \overline{▲} = 1$
5	$0 \cdot ▲ = 0$
6	$1 \cdot ▲ = ▲$
7	$▲ \cdot ▲ = ▲$
8	$▲ \cdot \overline{▲} = 0$
9	$\overline{(\overline{▲})} = ▲$
10	$▲ + Y = Y + ▲$
11	$▲ \cdot Y = Y \cdot ▲$
12	$▲ + (Y + Z) = (▲ + Y) + Z$
13	$▲ (YZ) = (▲Y) Z$
14	$▲ (Y + Z) = ▲Y + ▲Z$
15	$▲ + ▲Z = ▲$
16	$▲ (▲ + Y) = ▲$
17	$(▲ + Y) (▲ + Z) = X + YZ$
18	$▲ + \overline{▲}Y = ▲ + Y$

BOOLEAN SIMPLIFICATION

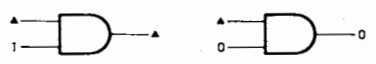
Simplify $P + X + \overline{X}$. Using Rule 4 from above table, $P + X + \overline{X}$ reduced to $P + 1$. Using Rule 2 reduced $P + 1$ to 1. Therefore, $P + \overline{X} + X$ is always equal to a logic 1.

Simplify $PQ\overline{Q}L$. By Rule 8 of above table, $Q \cdot \overline{Q} = 0$. Therefore, $PQ\overline{Q}L = P \cdot Q \cdot L$. By Rule 5, $P \cdot Q \cdot L = 0$. Therefore, $PQ\overline{Q}L = 0$.

Simplify: $\overline{M} + C\overline{S}C + F$. Use Rule 8: $C\overline{S}C$; $C\overline{C} = 0$. Use Rule 5: $0 \cdot S = 0$. Expression becomes $\overline{M} + 0 + F$, which reduces to $\overline{M} + F$. Note we should not apply Rule 4 of boolean table $\overline{M} + F$ because variables are not identical. The answer is $\overline{M} + F$ or an OR gate with inputs $\overline{M}$ and F.

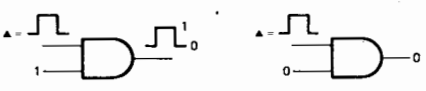
LAW OF PRODUCTS

The law of *Products* is also called the law of *Intersection.* This law explains the behavior of an AND gate. It follows Rules 5 and 6 from the above table.

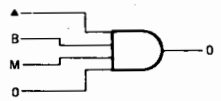


According to Rule 6 ($1 \cdot ▲ = ▲$), if we apply a logic 1 and the variable ▲ to the input of an AND gate, the output will be equal to the variable ▲. According to Rule 5 ($0 \cdot ▲ = 0$), if we set the variable ▲ equal to a binary 1, the output of an AND gate is still 0.

For a pulsed input:

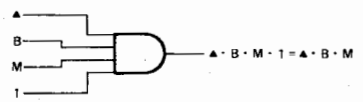


Consequently for a four input AND gate, applying Rule 5 of the boolean angebra table:



$▲ \cdot B \cdot M \cdot 0 = 0$. It is obvious that in the preceding AND gate expression if any of the variables is a logic 1 level, but either of the inputs is zero, the output will be 0.

If the input 0 becomes 1, then:



THE LAW OF UNIONS

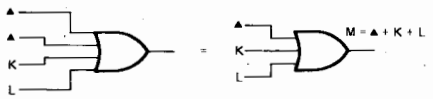
This law pertains to the OR gate and is related to Rule 1 and 2 of the boolean table.

According to Rule 1 ($0 + ▲ = ▲$), if one of the inputs of an OR gate is 0 and we apply the variable ▲ to the other input, the output will be the variable ▲. According to Rule 2 ($1 + ▲ = 1$),

if we apply a 1 and the variable ▲ to the inputs of an OR gate, the output will be 1.

LAW OF TAUTOLOGY

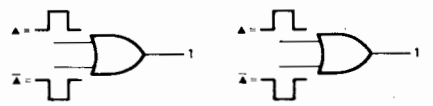
The known *Law of Tautology* applies to Rules 3 and 7 of the boolean table. Rules 3 and 7 apply to AND gates and OR gates. Using this law, simplification of long algebraic expressions becomes simple. The rules merely state that equal variables in an equation should be omitted. Example: Simplify the equation ($M = ▲ + ▲ + K + L$). It is obvious that the variable ▲ repeats twice. By Rule 3, the equation simplifies to $M = ▲ + K + L$.



THE LAW OF COMPLEMENTS

If a logic signal and the complement of this logic signal is applied to a logic gate the resulting output is 1 or 0 depending on the logic gate being used. The law of the complement is stated in Rules 4 and 8 of the boolean table. Let's apply this rule to an OR gate.

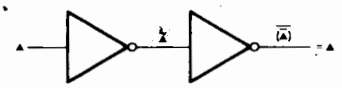
According to Rule 4 ($▲ + \overline{▲} = 1$), if one of the inputs of the OR gate is logic ($1 = ▲$) and the other input is ($0 = \overline{▲}$) the output will be 1. Example: Pulsed



According to Rule 8 ($▲ + ▲ = 0$), if one input to an AND gate is variable ▲ and the other input is 0, the output of the gate will be 0.

THE LAW OF DOUBLE NEGATION

The Law of Double Negation is expressed by Rule 9 of the boolean algebra table. This law states that feeding the negation of a variable through an inverter produces the original variable.



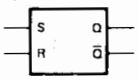
Complementing a signal an even number of times produces the original signal.

FLIP-FLOPS

A digital logic circuit able to memorize by storing logic levels. A flip-flop has two stable states: It will remain in either set or reset state until its state is changed by external signals. The data stored in a flip-flop can be quickly checked by using an oscilloscope or meter to detect the state of its output. There are three basic types of flip-flops.

1 The RS
2 The D type
3 The JK

The logic symbol for an RS flip-flop is

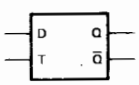


The inputs are S and R. The outputs are Q and $\overline{Q}$. Application of a logic 1 level on the S input will make the Q output go to a logic 1 level and the $\overline{Q}$ output go to a logic 0 level. If the logic 1 level is applied to the R input, the output levels are reversed. The unused input must be held at a logic 0 level.

RS FLIP-FLOP LOGIC TABLE			
INPUTS		OUTPUTS	
R	S	Q	$\overline{Q}$
High	Low	Low	High
Low	High	High	Low
Low	Low	Unchanged	
High	High	Not Permitted	

When the S input is 1 and the R is 0 the flip-flop is reset. When the S is 0 and R is 1, it is set. All the other input combinations produce ambiguous or race states.

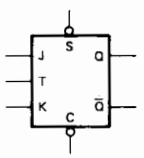
The D-type flip-flop logic symbol.



The D-type flip-flop generally behaves like the RS flip-flop but the main difference is that a low-to-high transition must be applied to the T input for the D flip-flop to toggle and store information.

D-TYPE FLIP-FLOP LOGIC TABLE			
INPUT		OUTPUT	
D	T	Q	$\overline{Q}$
Low	Low	Previous state	
Low	High	Low	High
High	Low	Previous state	
High	High	High	Low

The JK flip-flop symbol



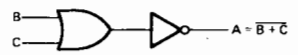
The S and C inputs presets the JK flip-flop to a desired state before another operation is begun. The S and C inputs are referred to as asynchronous inputs because they don't require a transition on the T input. The J and K inputs only affect the Q and $\overline{Q}$ outputs when a transition occurs on the T or clock input. If the J input is 1 and the K input is 1, the flip-flop will reset from the previous state in the presence of a low-to-high transition on the T input. To set the JK flip-flop, apply a 1 to the J input and a 0 to the K input, then apply a low-to-high transition (clock pulse) to the T input. This operation is referred as synchronous with the clock operation.

CUT ALONG DOTTED LINE

THE NOR GATE

A logic circuit with two or more input capable of resolving the equation $A = \overline{C + B}$. The NOR gate is a combination of an OR logic gate followed by an inverter.

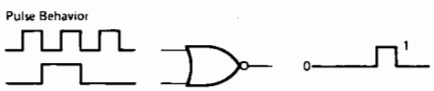
The NOR gate could be constructed using an OR gate followed by an inverter.



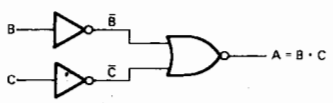
The operation of a *NOR* gate is represented in the following truth table.

Input		Output
B	C	$A = \overline{B + C}$
0	0	1
0	1	0
1	0	0
1	1	0

Notice that the NOR table is the exact opposite or complement of the OR truth table. Summary: The 2 input NOR gate produces an output when both of the inputs are logic 0 level. If either of the inputs is 1, the output is always a logic 0 level.



A NOR gate using inverters in the input will act as an AND gate.



Truth Table for inverted input NOR gate shown above.

B	C	$\overline{B}$	$\overline{C}$	A
1	0	0	1	0
0	1	1	0	0
0	0	1	1	0
1	1	0	0	1

DUALITY OF LOGIC GATES

Gates can provide different functions depending on the assumed reference logic level applied to the input. There are two widely used types of combinational logic levels used in present logic circuits. These are known as *positive logic* and *negative logic*.

POSITIVE LOGIC LEVELS

Input	Output
Logic 1 = +5 volts	+5 volts
Logic 0 = 0 volts to +0.2 volts	0 volts to +0.2

The logic 0 is relatively close to the 0 or ground reference level but in practical gate design the 0 reference is usually a few tenths of a volt above ground level.

NEGATIVE LOGIC LEVELS

Input	Output
Logic 1 = 0 volts to +0.2 volts	0 volts to +0.2 volts
Logic 0 = +5 volts	+5 volts

Truth Table for Positive Logic 2-input AND gate

Voltage Table

Input		Output
B	C	A
0 V	0 V	0 V
0 V	+5 V	0 V
+5 V	0 V	0 V
+5 V	+5 V	+5 V

Truth Table

Input		Output
B	C	A
0	0	0
0	1	0
1	0	0
1	1	1

Compare above Truth Table with Truth Table below for a negative logic 2-input AND gate.

Negative Logic AND gate.

Voltage Table

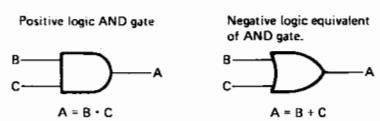
Input		Output
B	C	A
0V	0V	0V
+5V	0V	0V
0V	+5V	0V
+5V	+5V	+5V

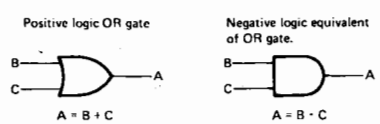
Truth Table

Input		Output
B	C	A
1	1	1
0	1	1
1	0	1
0	0	0

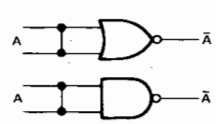
Notice that the Truth Table for the negative logic AND gate is exactly opposite to the Truth Table for the positive logic AND gate. The negative logic AND gate acts as a positive logic OR gate.

Consequently an AND gate can provide the OR function and an OR gate can provide the AND function by selecting positive or negative logic level assignments.

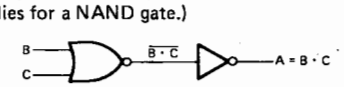




Flexibility in implementation of the basic gate functions. NOR gates and NAND gates can be used to implement any of three basic logic functions. Example: By connecting all the inputs of a NOR or NAND gate together we can implement an inverter.

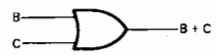


By connecting an inverter in the output of a NOR gate, we can implement an OR gate. (Same applies for a NAND gate.)

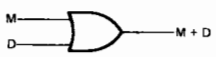


BOOLEAN ALGEBRA

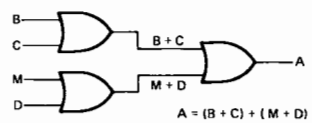
Is the mathematical method of analyzing logic circuits. Boolean equations describe the operation and provides the mathematical tool for the manipulation of logic circuits. For example, draw the logic circuit that solves the boolean equation A = (B+C) + (M+D): The expression indicates that there are two OR gates being OR'ed by another single OR gate: Analysis: Draw the sumbol for the first member of the equation. (B+C):



Then draw the symbol for the second member of the equation (M+D).



Use a single OR gate to combine the two OR gate outputs as required by the indicated + symbol.

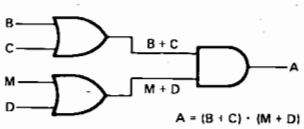


Draw symbol for the second term of the equation (M+D).

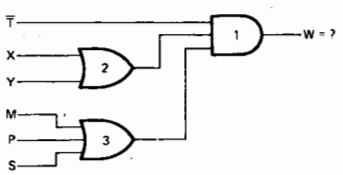
Draw the logic circuit to solve the boolean equation A = (B+C) · (M+D).

The first term (B+C) of the equation indicates an OR gate with inputs B and C.

Combine the two OR gate outputs using a single AND gate as required by the multiplication.



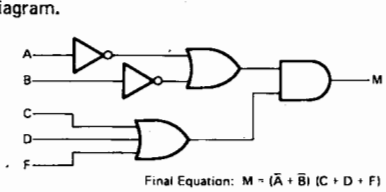
Write the boolean algebraic expression from the given logic circuit.



First write the expression describing the output of gate 2. This is an OR gate, therefore, the expression is X+Y. Secondly write the output equation for gate 3. This is another OR gate, so the expression is M + P + S. Notice that the algebraic expressions are being AND'ed by gate 1. Consequently the output expression so far is (X+Y) · (M+P+S). Input T could have been included anywhere in the equation because it is being AND'ed by gate 1 with the other two equations. The complete output equation is:

$$W = (X + Y)(M + P + S)\,\overline{T}.$$

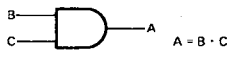
Write the boolean equation for the logic diagram.



CUT ALONG DOTTED LINE

AND GATE

Logic circuits with two or more inputs and a single output capable of resolving an output with combinations of input variables. The two-input AND gate resolves the equation A = BC. The output (A) is expressed in terms of the two variables (B) and (C). The expression A = BC does not imply multiplication but rather (A) is the result of quantity (B) *AND* quantity (C) presented at the input. AND gate symbol



The operation of the AND gate is better represented by the use of a Truth Table that indicates the output for the various input combinations.

Truth Table for 2-input AND gate:

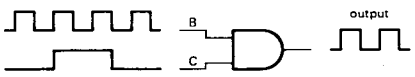
IN		OUT
B	C	A
0	0	0
1	0	0
0	1	0
1	1	1

The AND gate performs binary multiplication.

MULTIPLICATION TABLE
0 X 0 = 0
1 X 0 = 0
0 X 1 = 0
1 X 1 = 1

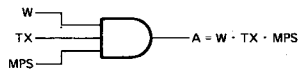
The total number of possible input combinations for a gate with an even number of inputs is given as $(\text{inputs})^2$ = outputs. For a two input AND gate; $2^2 = 4$. The truth table then contains 4 possible input combinations. Summary: A two-input AND gate gives an output only when both inputs are logic 1.

For a pulse input.



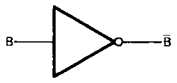
Notice that the AND gate produces 2 output pulses out of 4 input pulses arriving at the B input because the duration of pulse C is exactly twice the total duration of input pulses B.

Algebraic equations. Example: For A = W · TX · MPS. Using AND gate



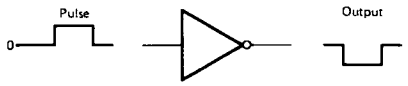
THE INVERTER

A digital gate that inverts the input signal. It is also known as a complementary gate because the output is inverted in relation to the input. The inverted output is written with a bar over the inverted variable.



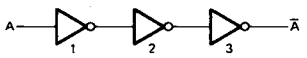
The inverter has only one input connection and one output connection.

For a pulse input:

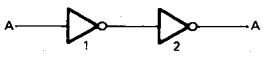


CASCADED INVERTERS

If an odd number of inverters are connected in series, the output will always be the negation of complement or the input variable.

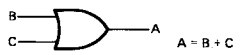


If an even number of inverters are connected in series, the output will be the same as the input with no inversion.



THE OR GATE

Logic circuit with two or more inputs that provides an output when any input is a logic (1). The two input OR gate resolves the equation A = B + C. The output is expressed in terms of either variable B or C acting on the input. The expression A = B + C does not imply addition but rather A is the result of either B or C acting on the input.



The operation of the OR gate is better represented by the use of a Truth Table that indicates the output when the inputs are modified by (1) or (0):

2 INPUT OR GATE

INPUT		OUTPUT
B	C	A
0	0	0
0	1	1
1	0	1
1	1	1

Below is a 3-input OR gate Truth Table.

INPUT			OUTPUT
A	B	C	D
0	0	0	0
1	0	0	1
1	1	0	1
1	1	1	1
0	0	1	1
0	1	1	1
0	1	0	1
1	0	1	1

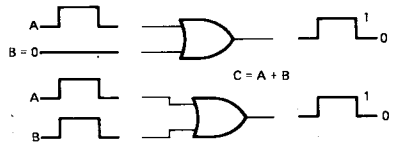
The number of inputs determines the number of combinations in the same manner as for the AND gate: input = combinations − 1. Example: $3^2 - 1 = 8$.

Note: The equation; $\text{inputs}^2 = \text{combinations} - 1$; is for an ODD number of inputs. Use $\text{inputs}^2 = \text{combinations}$ for an EVEN number of inputs. Example: An OR gate with 2 inputs has $2^2 = 4$ possible input combinations. An OR gate with 5 inputs has $5^2 = 25 - 1 = 24$ possible input combinations.

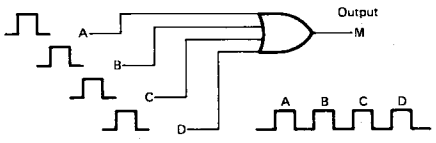
The OR gate equation satisfies the rule of binary addition.

LOGICAL ADDITION TABLE
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 1

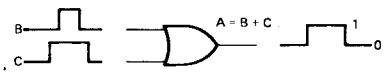
Note: Binary addition and logical addition are not EQUAL. Summary: The OR gate gives an output when any input is a logic one. Output is zero when all inputs are zero.



The OR GATE preserves the individual characteristics of pulses arriving at the input.

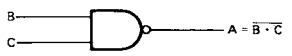


Note: Each output pulse has the same time interval at the output as it had in the input. If there is time coincidence at the input, the output pulse will be equivalent to the longest pulse at the input. Example:

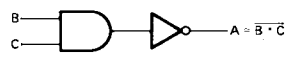


THE NAND GATE

The NAND (NOT-AND) gate is the combination of an AND gate and inverter. The operation of a NAND gate is represented by the equation $A = \overline{B \cdot C}$ and is read A is the result of B and C operating at the input of the NAND gate but inverted at the output. The solid bar over B · C means inversion.



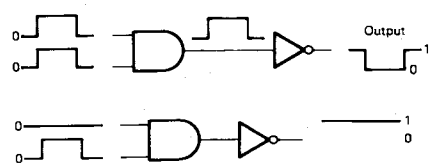
The NAND gate could be constructed by using an AND gate followed by an inverter.



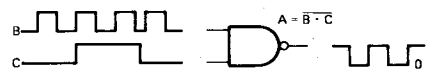
The operation of a 2-input NAND gate is easily represented by a truth table form.

INPUT		OUTPUT
C	B	$A = \overline{B \cdot C}$
0	0	1
1	0	1
0	1	1
1	1	0

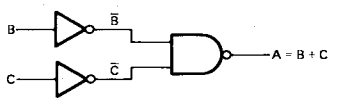
Note: The NAND gate truth table is the complement of the AND gate truth table. Both inputs must be at a logic 1 level to produce a logic 0 output. Pulse behavior:



Non-coincident input pulses have no effect on the output of a NAND gate. The output always stays at logic 1 level. Coincidence at input of logic 1 level pulses produces a negative going pulse at the output.



A NAND gate with inverters connected to the inputs will act as an OR gate.

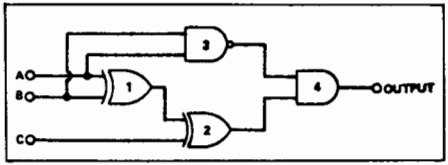


Expanding Ex-Or Gates

By L. Robertson

Exclusive–OR gates are only obtainable in 2–input packages, and simply cascading two gates does not give the correct truth table. Any application, therefore, which requires an Ex–OR gate with three or more inputs is going to require some tricky logic combinations.

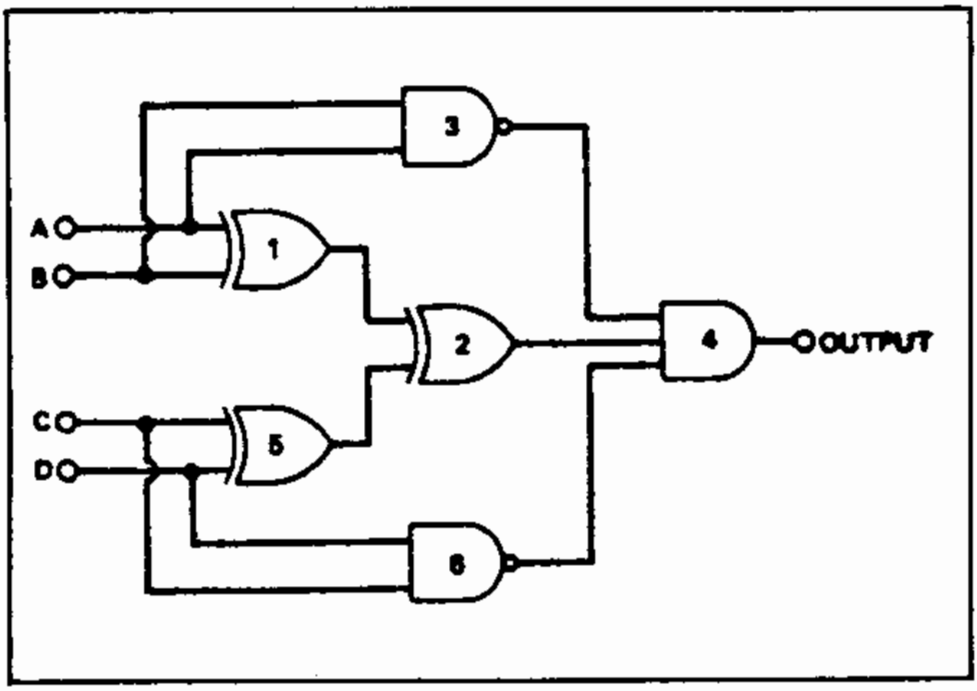
In the first circuit, inputs A and B are fed into gate 1 and the output of the gate is combined with input C at gate 2. This arrangement satisfies every part of the truth table except A = B = C = 1, where the output from gate 2 will be 1 instead of 0. To overcome this problem, inputs A and B are also fed to gate 3 so that when both are high the consequent high output from that gate will disable gate 4 and so produce a final output of 0.

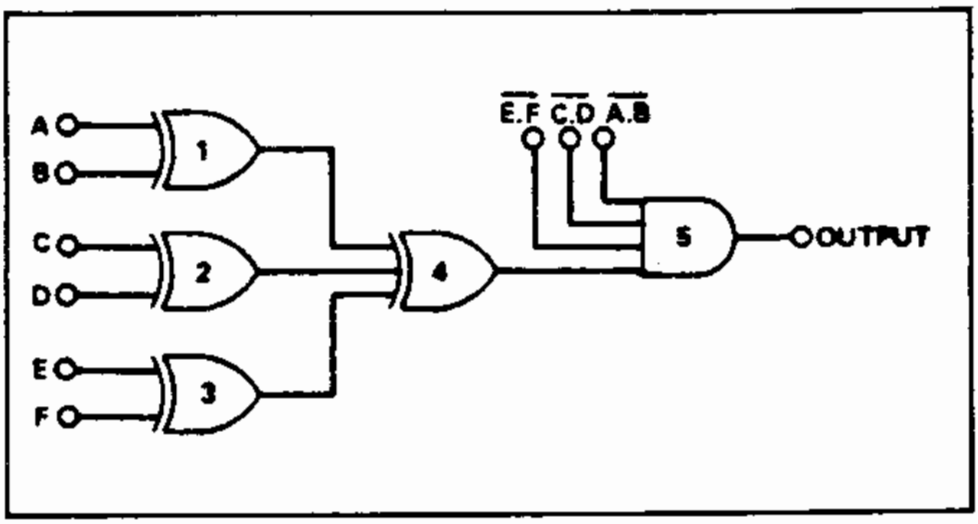


If a four–input arrangement is required, the expansion can be achieved by treating inputs C and D in the same way as inputs A and B in the first circuit. Thus, in the second circuitt gate 5 performs a similar function to gate 1 and gate 6 behaves in the same way as gate 3.

The final permutation is a six–input gate, shown in simplified form in the third circuit diagram. The three–input Ex– OR gate shown as gate 4 is made up as shown

continued on next page





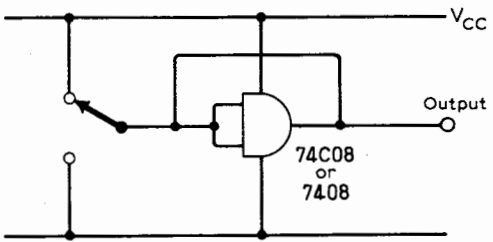
in the first diagram above and the pairs of inputs AB, CD, EF are combined in three NAND gates and fed to three of the inputs of the final AND gate.

Simple bounce-free switch

A single non-inverting gate or buffer wired as shown forms a bistable circuit because the positive loop gain is greater than unity. Whilst the switch is in the up position, the output will be high. When the switch leaves this position and is in transit, the output remains high because the input is still high. When the switch first makes contact with the lower position, the output of the gate is momentarily shorted. This situation is however remedied within a few nanoseconds because the input is also taken to ground which drives the output of the gate low. Thereafter, if the switch contact bounces, the output will stay low because the input is low.

This single non-inverting gate arrangement is simpler than the usual SR flip-flop, and the annoying pull-up resistors are eliminated.

P. Seligman
Victoria
Australia



One's complement adder eliminates unwanted zero

by John F. Wakerly
Digital Systems Laboratory, Stanford University, Stanford, Calif.

To enable an adder to subtract, a binary system can use the one's complement representation for negative numbers. However, since the negative of a number is created by replacing 1s by 0s and 0s by 1s, two forms of zero result—00 . . . 0 and 11 . . . 1—to the complication of later zero-checking operations. Fortunately, it's possible to eliminate the 11 . . . 1 version if a NAND gate is included in the adder circuitry.

In a one's complement binary system, the most significant digit in a positive number must be 0 and in a negative number must be 1. Thus the eight possible values that can be represented by 3-bit number are no longer 0, 1, . . . 7. Instead, they are −3, −2, −1, −0, +0, +1, +2, +3. The following table shows why both +0 and −0 occur:

One's complement form	Value represented
111	−0
110	−1
101	−2
100	−3
011	+3
010	+2
001	+1
000	+0

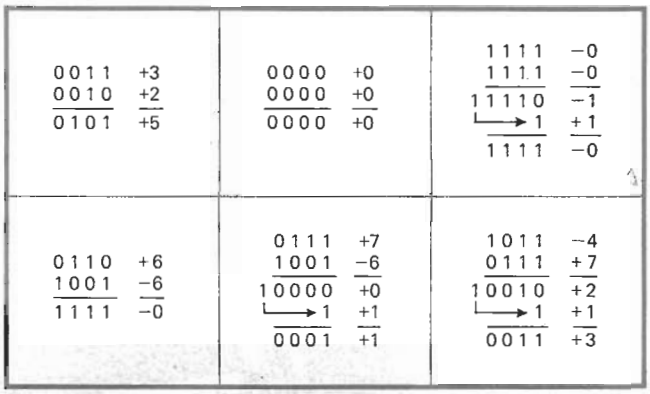
More generally, positive zero is represented by 00 . . . 0, and negative zero by 11 . . . 1.

When two numbers are added in this representation, any carry from the most significant position is added into the least significant position—a process termed "end-around carry." As it happens (see Fig. 1), positive zero is produced only when positive zero is added to itself.

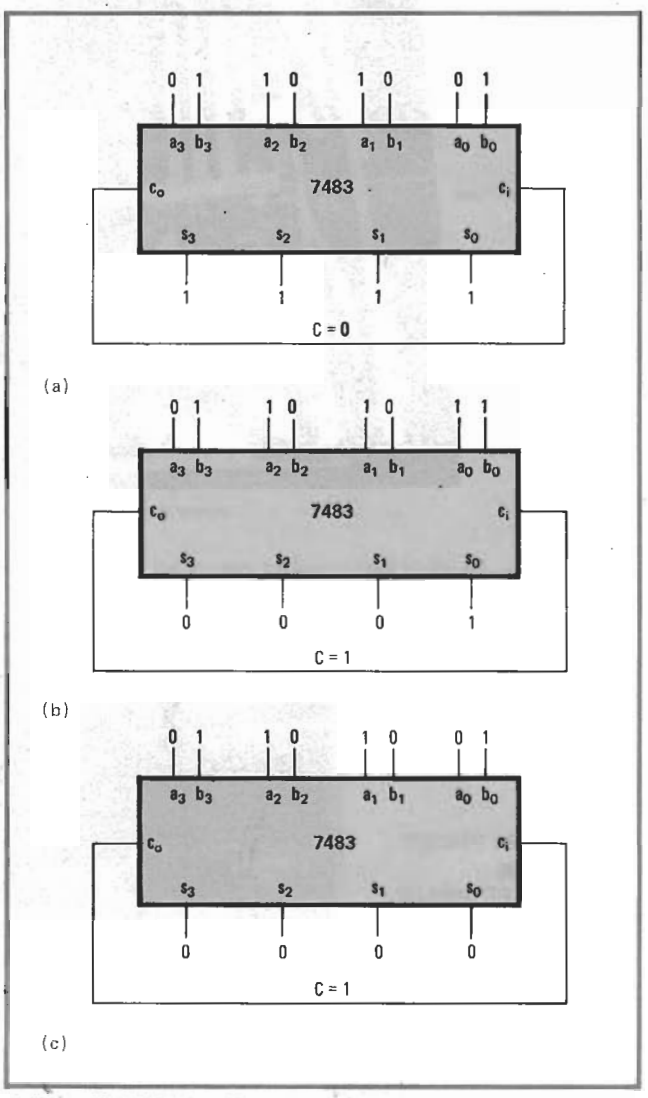
The standard implementation of a one's complement adder uses a conventional binary adder with the carry output connected to the carry input to achieve the end-around carry. This direct connection of the carry output to input in effect turns the adder into an asynchronous sequential circuit, whose state depends on its previous state. (A D-type flip-flop is a familiar example of a sequential circuit.)

To see this, consider Fig. 2(a), which shows a 4-bit one's complement adder that can be implemented with a single MSI circuit such as a 7483. The two input numbers are A = $a_3a_2a_1a_0$ and B = $b_3b_2b_1b_0$, the carry input is c_i, and the carry output is c_o. The state of the sequential circuit is the state of the carry line C.

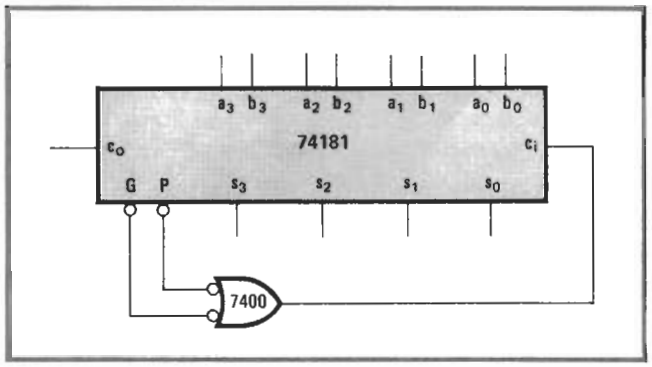
In Fig. 2(a), A equals 0110, B equals 1001, and C equals 0, making the output 1111 as expected. In Fig. 2(b), A is changed by one bit to 0111, C changes to 1, and the output changes to 0001. When A is changed back to 0110, as shown in Fig. 2(c), the carry line C, which was 1 before the change, produces the sum of



1. One's complement addition. Examples show the one's complement representation in which a negative number is just the bit-by-bit complement of the positive number; the "end-around carry" rule for addition, and the two forms of zero.

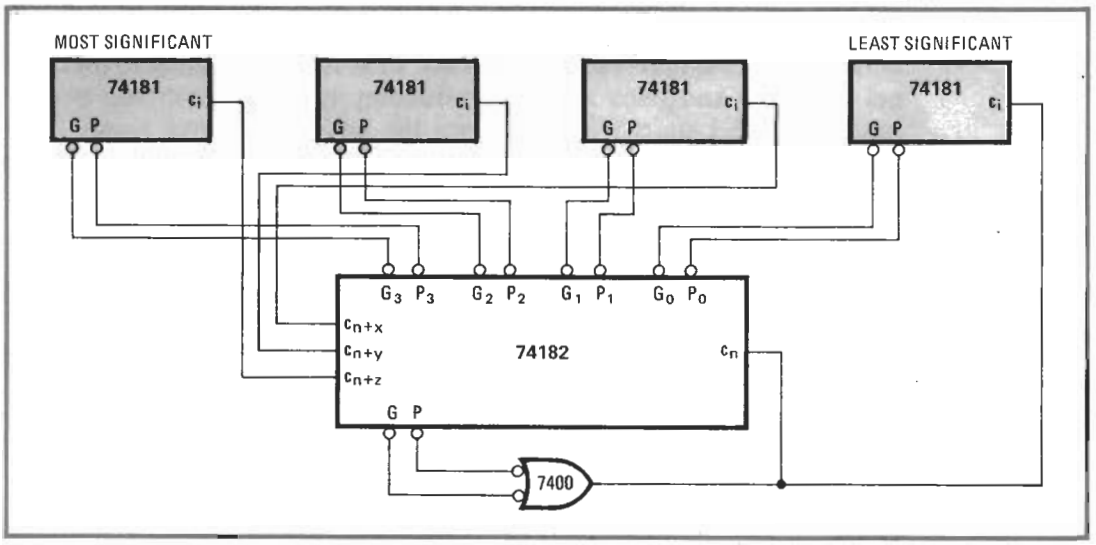


2. Unpredictable zero. Standard binary adder has carry output connected to carry input for "end-around carry" in adding one's complement numbers. A result that is zero may come out as 1111 (negative zero) or 0000 (positive zero), depending upon prior condition of circuit. In both (a) and (c), computation of (6 − 6) is represented by (0110 + 1001); results are 1111 and 0000, respectively, because intervening computation in (b) changed state of circuit.



3. Eliminates negative zero. This 4-bit one's complement adder has only one form of zero—0000. The 1111 representation of zero is eliminated by use of NAND gate and arithmetic logic unit that has carry-generate (G) and carry-propagate (P) outputs.

4. More bits. Here a 16-bit one's complement adder uses NAND gate to assure that zero appears as 00 . . . 0. NAND-gate arrangement introduces no more delay than the conventional end-around carry.



0110 + 1001 + 1 = 10000 after the change, so that C remains 1 after the change.

Thus the circuit has two stable states, either C = 0 or C = 1, for any input combination having exactly one 0 and one 1 at each bit position (that is, a number and its one's complement are being added). Which state the circuit attains for a particular input combination depends on the state of the carry line C for the previous operation. Unless the inputs of the adder are set to a known value (say 0) before each operation, it is quite impossible to predict which form of zero will be produced by adding a number and its complement.

Nevertheless, despite its unpredictability, the unwanted form of zero (11 . . . 1) can be eliminated very easily with most MSI and LSI arithmetic logic units (ALUs) and data path slices. MSI circuits such as any 74181, 74S281, and 74S381 ALUs, and LSI circuits such as Monolithic Memories' 6701, Intel's 3002, and Advanced Micro Devices' 2901 data path slices, all have carry-generate (G) and carry-propagate (P) outputs for fast carry lookahead, in addition to the normal ripple carry output (c_o). Examination of the carry-propagate equations shows that P equals 1 if a number and its one's complement are being added, i.e., the sum equals 11 . . . 1. Therefore the 11 . . . 1 representation of zero can be eliminated by producing a carry input of 1 whenever a carry is generated (G = 1) or P equals 1. For typical devices this process requires a single two-input NAND gate, as shown in Fig. 3 for a 4-bit one's complement adder using one 74181.

For larger adders, the G and P outputs of the carry lookahead generator may be used, as shown in Fig. 4 for a 16-bit one's complement adder using four 74181's and one 74182 lookahead carry generator. In all cases the resulting circuit is no longer a sequential circuit, because generate and propagate outputs do not depend on the carry input.

The total propagation delay of a one's complement adder using a conventional end-around carry is $t_{ADD} = t_{ICO} + t_{CIS}$, where t_{ICO} is the propagation delay from any data input to the carry output and t_{CIS} is the propagation delay from the carry input to the sum output.

For the scheme illustrated in Fig. 3, the delay is $t_{ADD} = t_{IPG} + t_N + t_{CIS}$, where t_{IPG} is the delay from any data input to the P and G outputs and t_N is the NAND-gate delay. Typical values for standard 7400-series parts are t_{ICO} = 28 ns, t_{CIS} = 13 ns, t_{IPG} = 17 ns, and t_N = 11 ns. Hence the total delay for both schemes is the same: $t_{ADD} = 28 + 13 = 17 + 11 + 13 = 41$ ns.

For larger adders, as in Fig. 4, the delay for both schemes is still approximately the same, since the delay of the external NAND gate is comparable to the delay of the internal gate used to compute the ripple carry output from P, G, and c_i in the ALU or lookahead generator.

The scheme of Fig. 3 or Fig. 4 automatically converts all arithmetic results of 11 . . . 1 to 00 . . . 0. However, it should be noted that in logic operations a result of 11 . . . 1 is not converted. In most applications this is the desired behavior. □