Chapter 4
Building
More-Complex Static Figures

While it is possible to achieve a wide range of static patterns using the principles outlined in the previous chapter, they generally lack the interest that typifies some of the better TV games on the market today. The line and bar patterns are all made from straight lines and right angles; and building something as geometrically simple as a triangle calls for an exceedingly complex Line/Bar Tinkerbox cir cuit.

This chapter presents one basic approach to building static figures of all kinds, geometric figures as well as an unlimited variety of far-more-interesting figures such as rockets, airplanes, tanks, people, guns, and so on. And to some extent, it is possible to use this technique to build up some of the more complicated line, bar, and rectangle figures described in the previous chapter.

Although this chapter presents only one basic approach, I think you will find it is adequate for just about any figure-building problem you might ever encounter. The range of possible figures that can be built is really limited only by your own understanding of how the scheme works and, of course, your own imagination.

Follow the discussions carefully and in the sequence as pre-sented here. You might get the impression you are studying a textbook from time to time, but if you work out the specific examples and then try a few of your own, you'll have a lot of fun learning how to build complex figures on the TV screen.

This technique, incidentally, can be extended later on to include some animation and figure-motion effects. So the figures you create

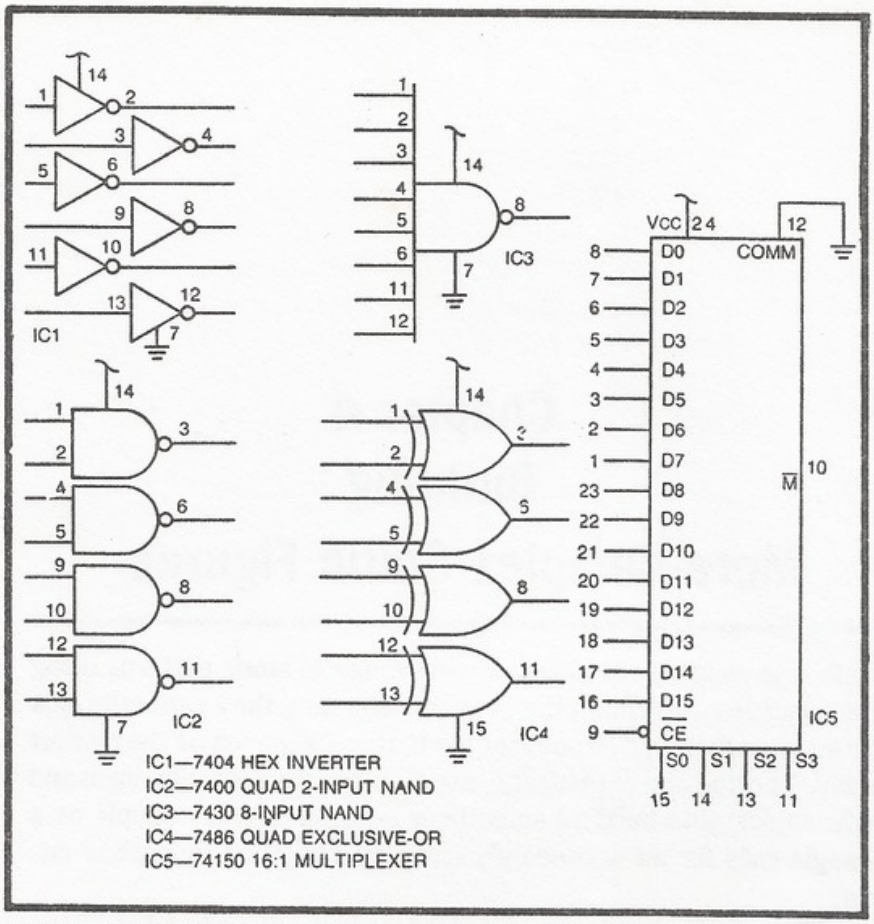


Fig. 4-1. Most-used IC components for the Complex-Figure Tinkerbox.

as part of the discussions in this chapter can be modified at a later time so that they can be moved freely around the screen. Keep a good set of notes concerning the specifications for interesting and potentially useful figures. Such notes will prove invaluable when you decide to work out a TV game of your own.

COMPLEX-FIGURE TINKERBOX

Assemble a Tinkerbox assembly similar to the one described in Chapter 3 for building lines, bars, and rectangles. As indicated in the parts list in Fig. 4-1, you need at least one 7450 16:1 multiplexer and a 7486 quad EXCLUSIVE-OR in addition to the inverters and NAND gates prescribed for the Line/Bar Tinkerbox. There will be a need for as many as four 7450 multiplexers for performing the more advanced experiments in this chapter. But since these large 24-pin ICs take up so much space on the breadboard, it is a good idea to attach them to the board only as they are needed.

For the sake of readers who have no special knowledge of digital electronics, the operation of the 16:1 multiplexer circuit calls for some special consideration. The truth table in Fig. 4-2 represents the operation of a 74150 16:1 multiplexer IC. Note that the device has 16 separate data inputs, labeled D0 through D15. There is a single output, however, $\overline{M}$. Then notice there are four select inputs, S0 through S3, and a chip-enable input, $\overline{CE}$. All of those terminals, plus two more for +5V and COMM, add up to 24 pins.

According to the truth table in Fig. 4-2, the M output of this IC is always at logic 1 whenever the $\overline{CE}$ input is at logic 1. The Xs in the select columns mean those inputs are not relevant as long as $\overline{CE}$=1. The $\overline{CE}$ input, in effect, is capable of disabling the chip altogether—as long as $\overline{CE}$=1, to be specific. Setting the $\overline{CE}$ input to logic 0 thus enables the IC for its normal multiplexing operations.

Suppose the $\overline{CE}$ input is set to logic 0. Whenever that is the case, output M is equal to an inverted version of one of the 16 D inputs. Furthermore, the D input that appears inverted at the M output depends on the status of the select inputs. If the select inputs are all set to logic 0, for instance, an inverted version of input D0

INPUTS					OUTPUT
$\overline{CE}$	S3	S2	S1	S0	$\overline{M}$
1	X	X	X	X	1
0	0	0	0	0	$\overline{D0}$
0	0	0	0	1	$\overline{D1}$
0	0	0	1	0	$\overline{D2}$
0	0	0	1	1	$\overline{D3}$
0	0	1	0	0	$\overline{D4}$
0	0	1	0	1	$\overline{D5}$
0	0	1	1	0	$\overline{D6}$
0	0	1	1	1	$\overline{D7}$
0	1	0	0	0	$\overline{D8}$
0	1	0	0	1	$\overline{D9}$
0	1	0	1	0	$\overline{D10}$
0	1	0	1	1	$\overline{D11}$
0	1	1	0	0	$\overline{D12}$
0	1	1	0	1	$\overline{D13}$
0	1	1	1	0	$\overline{D14}$
0	1	1	1	1	$\overline{D15}$

Fig. 4-2. Operating truth table for the 74150, 16:1 digital multiplexer.

appears at $\overline{M}$. If, on the other hand, the select inputs are set to binary 0001 (S3=0, S2=0, S1=0, S0=1), output M is equal to an inverted version of the D1 input.

The S columns in Fig. 4-2 actually represent a 4-bit binary counting sequence from binary 0000 (decimal 0) through binary 1111 (decimal 15). The D inputs are labeled with numbers running from 0 through 15, and it is no coindicence that setting a particular binary number at the S inputs causes the corresponding D input to appear inverted at $\overline{M}$.

A 16:1 multiplexer thus directs one of 16 different inputs to a single output, depending on the 4-bit number applied to the S inputs. If the select inputs happen to be connected to a 4-bit binary counter, the D outputs would appear at M in sequence—in a scanning-like fashion. And that, dear reader, is a clue to how a multiplexer can be used for generating complex figures on the TV screen.

Just to get the Complex-Figure Tinkerbox going, connect the circuit shown in Fig. 4-3. Note that select inputs S0, S1, S2, and S3 are connected to horizontal-count signals 32H, 64H, 128H, and 256H respectively. These counting signals cause the multiplexer to deliver inverted versions of the D inputs, in sequence from D0 through D15, to the $\overline{M}$output. Also note that the CE input is connected to COMM in order to permanently enable the multiplexing action. Connect the M output directly to GAME VID IN as shown in Fig. 4-3.

At this point in the experiment, the screen is blank, or should be blank if everything is going well. Then connect the D8 input, pin 23, to COMM. A white bar should appear down the middle of the screen. Its width should correspond to the width of the bars for a 32H signal.

Now remove the COMM jumper from D8 and connect it to other D-input terminals on the multiplexer, inputs D7 (pin 1), D9 (pin 22), and D10 (pin 21). What happens to the bar? It appears at a different place on the screen each time the COMM jumper is moved to a different D-input location. You will notice that the bar does not appear when connecting inputs D8, D7, and D6 to COMM. The bar is being generated by the multiplexer in those three instance, but it so happens the bar is in the horizontal blanking interval. All other inputs fix the location of this vertical white bar in some visible area of the screen.

Using the multiplexer as shown in Fig. 4-3 divides the screen horizontally into 16 equal segments. Each of these segments represents one of the inputs to the multiplexer IC. The D0 input is active during the segment where 256H, 128H, 64H, and 32H are all equal

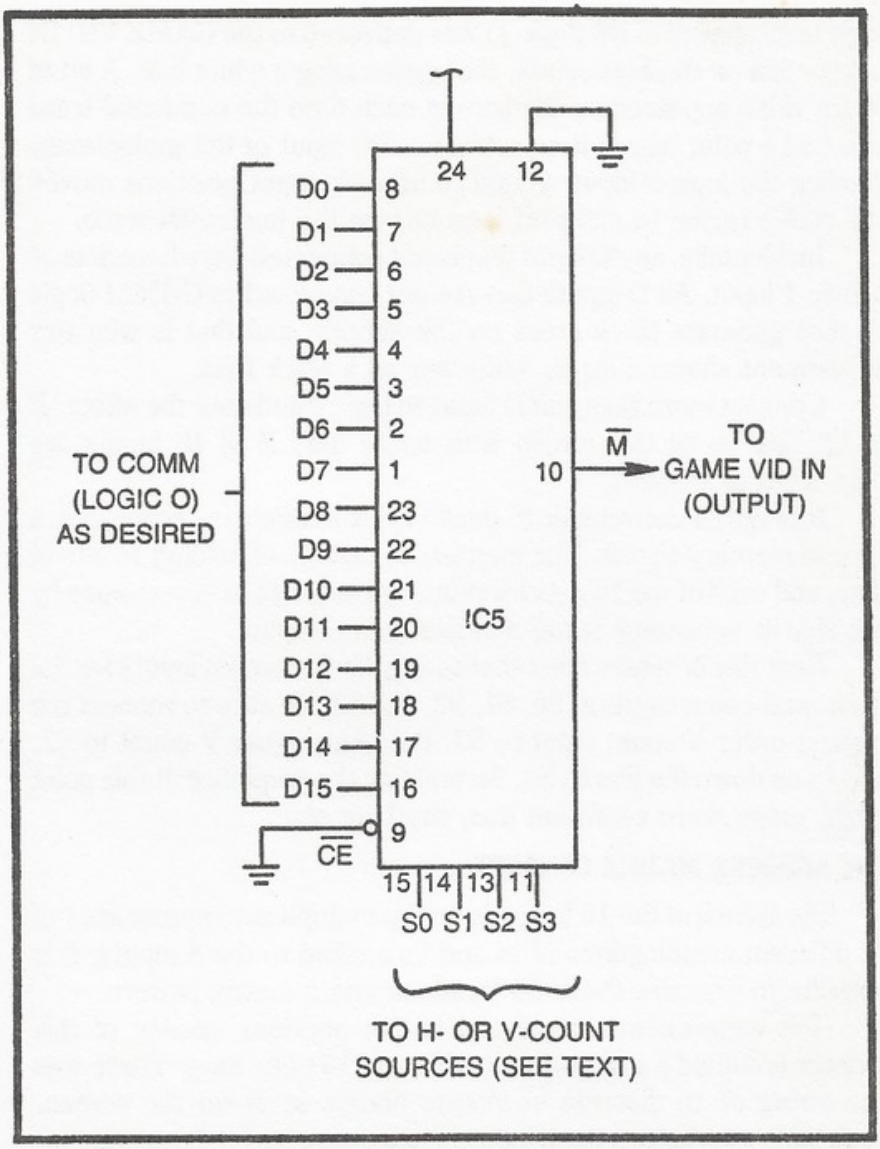


Fig. 4-3. Pinout and nomenclature for the 74150 multiplexer.

to logic 0, the point in the horizontal-count format that initiates the horizontal blanking interval. The blanking interval continues through the time the multiplexer is scanning D0, D1, and D2. But once the count reaches a point where the multiplexer is scanning the D3 input, the horizontal blanking interval is over. Segments representing inputs D3 through D15 thus appear in the useful working area of the screen.

When you connected the D8 input to COMM (logic 0), nothing really happens until the H-count inputs to the multiplexer reached the D8 scanning position. At that moment, an inverted version of the

logic level applied to D8 (logic 1) was delivered to the GAME VID IN connection on the Sourcebox, thus generating a white line. A bit of white video appeared on the screen each time the horizontal trace reached a point where it selected the D8 input of the multiplexer. Moving the logic-0 input around to other D-input positions moved the active region to different locations on the horizontal trace.

Incidentally, any D input that is not connected anywhere acts as a logic-1 input. All D inputs that are not connected to COMM (logic 0) thus generate black areas on the screen, and that is why this experiment shows a single white bar on a black field.

Connect more than one D input to logic 0 and note the effect. It is possible to fill the screen with white bars if all 16 inputs are connected to COMM.

It is rather convenient to think of this multiplexer scheme as a type of memory circuit. The memory is capable of holding 16 bits of data, and each of the 16 data locations are addressed in sequence by the signals appearing at the multiplexer's S inputs.

Bear this in mind while substituting vertical-count inputs for the horizontal-count inputs at S0, S1, S2, and S3. Be sure to connect the highest-order V-count input to S3, the next-higher V-count to S2, and so on down the line to S0. Scrambling the sequence at this point might cause more confusion than anything else.

THE ADDRESS MATRIX CONCEPT

Since each of the 16 D inputs on the multiplexer represents 1 of 16 different combinations of 1s and 0s applied to the S inputs, it is possible to organize those 16 locations into a matrix pattern.

The experiments suggested in the previous section of this chapter assumed a matrix 1 unit wide and 16 units long. There was one string of 16 discrete segments across or down the screen, depending on whether the S inputs were connected to horizontal- or vertical-count sources.

What do you suppose happens to the D-input addressing if two of the S inputs, say S0 and S1, are connected to H-count inputs, while the two remaining S inputs, S2 and S3, are connected to V-count inputs? to be more specific, try this: connect S0 to 16H, S1 to 32H, S2 to 16V, and S3 to 32V. Instead of generating a string of 16 segments, you end up with a 4×4 matrix or graph. See Fig. 4-4a.

The H-count pulses determine the horizontal positions on this little matrix, while the V-count pulses determine the vertical coordinates. Position D10, for instance, is enabled whenever the two H-count pulses show the decimal equivalent of 2 and the two V-count signals are at decimal 2.

To get a better feeling for what is happening here, make the connections to the multiplexer as prescribed in Fig. 4-4b. Every one of the D-input positions that are connected to logic 0 (COMM) generates a white area on the screen. (Remember that the multiplexer inverts the data.) So the image on the screen should correspond to the drawing in Fig. 4-4b.

Certainly there are more than one of these "Block-C" figures on the screen. That is a clear indication that the system is generating the little matrix pattern a number of times for each horizontal and vertical scan cycle.

If you are beginning to get a grasp of what is going on here, you should be able to predict what will happen if you connect the D6 input (pin 2) to logic 0. Try it. That operation should fill in the black area designated D6 in Fig. 4-4b, converting the white block-C figure into

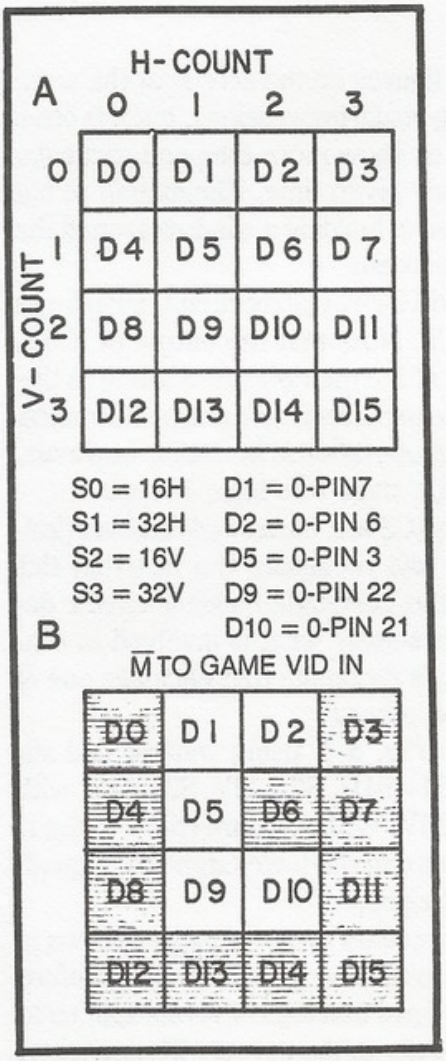


Fig. 4-4. The simplest 4 × 4 figure matrix. (a) The basic matrix format. (b) Specifications for a simple figure generated by the 4 × 4 matrix circuit.

a white rectangle. Remove the grounding wire to D5 and note the effect. You'll get backward block-Cs on the screen if you have left D6 connected to logic 0.

Play around with the logic levels at locations D1, D2, D5, D6, D9, and D10, and note the results. Keep a record of the connections that generate a pattern you think might be useful in the future. Avoid tampering with the matrix blocks surrounding the sides and bottom of the figure—unless you want the white areas of one matrix to butt up against white areas of another one.

Try running the $\overline{M}$ output of the multiplexer through an inverter before applying the signal to the GAME VID IN. You will find that the blacks and whites are then reversed on the screen. Maybe sometime you will have a need for a black block-C on a white field.

Windowing the Matrix Display

Having a number of identical figures on the screen at the same time might create some interesting visual impressions, but it is often confusing and usually undesirable to show more than one particular complex figure on the screen at any given time. Eliminating all but one of the matrix figures is a matter of building a window around the figure in the desired area of the screen.

Figure 4-5 shows the standard 16-cell matrix circuit, including a windowing feature built around IC5. Note that the output of IC5 is connected directly to the CE input of the multiplexer. As long as this terminal is at logic 0, the multiplexer circuit generates its matrix video at $\overline{M}$. The entire multiplexing operation is inhibited, however, when the output of the NAND gate rises to a logic-1 level.

Now notice that the inputs to IC3 are horizontal- and vertical-count sources. An astute reader will recognize the fact that this NAND operation is identical to the rectangle-building circuit described in Chapter 3. And that's precisely what is involved in windowing the matrix display: building a rectangle that encloses one of the matrices and eliminates all the others.

Connect the circuit shown in Fig. 4-5, using window and address inputs as follows: S0=32H, S1=64H, S2=32V, S3=64V; with window inputs $\overline{128H}$, 256H, and 128V. With no connections to the D inputs, you should see a white square in the lower right-hand quadrant of the screen. Now the fun begins.

The square on the screen represents the 4×4 matrix shown in Fig. 4-4a. Since the output of the multiplexer is now inverted before it is applied to GAME VID IN, it follows that logic-1 levels applied to the D inputs creates white matrix cells on the screen. The matrix is

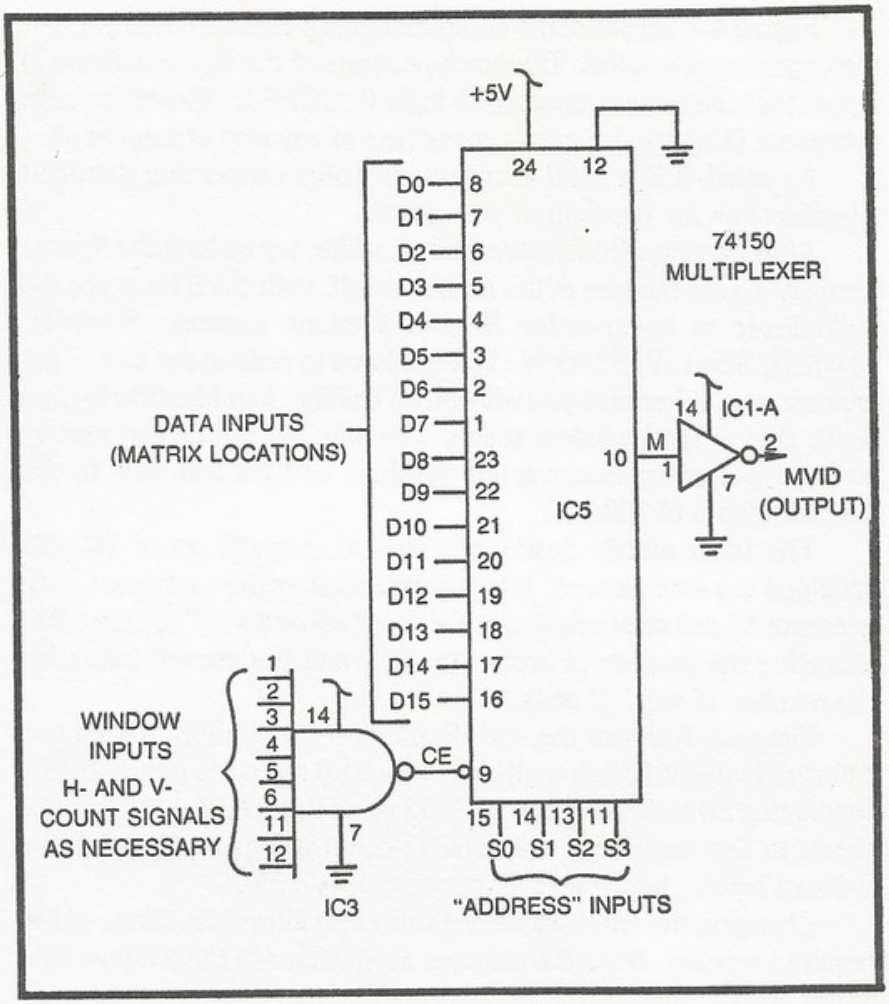


Fig. 4-5. The basic 16-cell matrix generator circuit.

now filled with white cells because any unconnected input of a TTL IC automatically assumes a logic-1 condition.

Connect a single jumper wire to COMM and touch the free end to various D inputs on the multiplexer. You will find that applying a logic-0 level to the D inputs in this fashion causes the corresponding matrix cell on the screen to go black. Connect the jumper to D0 through D15 in succession, and you will see a black square moving through the matrix as you go.

Building Complex Images in the 4x4 Windowed Matrix

Building complex figures using the circuit in Fig. 4-5 is a matter of eliminating white cells by connecting their corresponding D inputs to logic 0. And if you are really putting your heart into the project, you can have a good hour or so of fun playing with this scheme.

Figure 4-6 shows some complex figures that will help you get the experiment started. The black portions of the figure indicate D inputs that are to be connected to logic 0 (COMM). The white cells represent D inputs that aren't connected to any sort of input at all.

As usual, it is a good idea to keep notes concerning the input specifications for the figures you create.

After playing with this circuit for a while, try making the figures smaller. To cut the size of the matrix in half, shift the S inputs to the multiplexer to lower-order H- and V-count sources: S0=16H, S1=32H, S2=16V, S3=32V. You will have to reduce the size of the window, too, otherwise you will end up building four identical figures inside that original window space. The window can be reduced by adding inverted or noninverted versions of 64H and 64V to the window inputs of IC3.

The basic matrix described thus far is made up of 16 cells arranged in a 4×4 pattern. It is possible to alter the configuration to generate 16-cell matrices that are either 2×8 or 8×2. (The first digit indicating the number of horizontal cells and the second indicating the number of vertical cells.)

Figure 4-7 shows the specifications for generating these two different kinds of 16-cell matrices. The 2×8 matrix is generated by connecting S0 to 32H, S1 to 32V, S2 to 64V, and S3 to 128V. The circuit in this case uses only one H-count select input and three V-count inputs, hence it is longer than it is wide.

Changing the width-to-height ratio also alters the shape of the required window. Note the changes as specified in the window data in Fig. 4-7a.

If you ever want to create a complex pattern within an 8×2 matrix as in Fig. 4-7b, simply reorganize the select and window inputs as prescribed in that figure.

In either case you can still eliminate white cells by connecting the designated D input to logic 0, and retain a white cell by making no connection at all to that particular D input.

Carefully compare the shapes, select, and window specifications for the matrix-generating circuits in Figs. 4-4a and 4-7. All 3 have a total of 16 cells. That's quite apparent. But note a more subtle feature. The matrices must have a width-by-height product equal to 16. For example, it is not possible to build a 3×5 matrix using this 16-cell scheme. You do not have to use all the cells (the figure in Fig. 4-4b happens to occupy a 2×3 space), but you must be able to account for all of them in the basic matrix pattern.

Work with this basic 16-cell format, changing select and window specifications, until you are certain you understand how it works. As

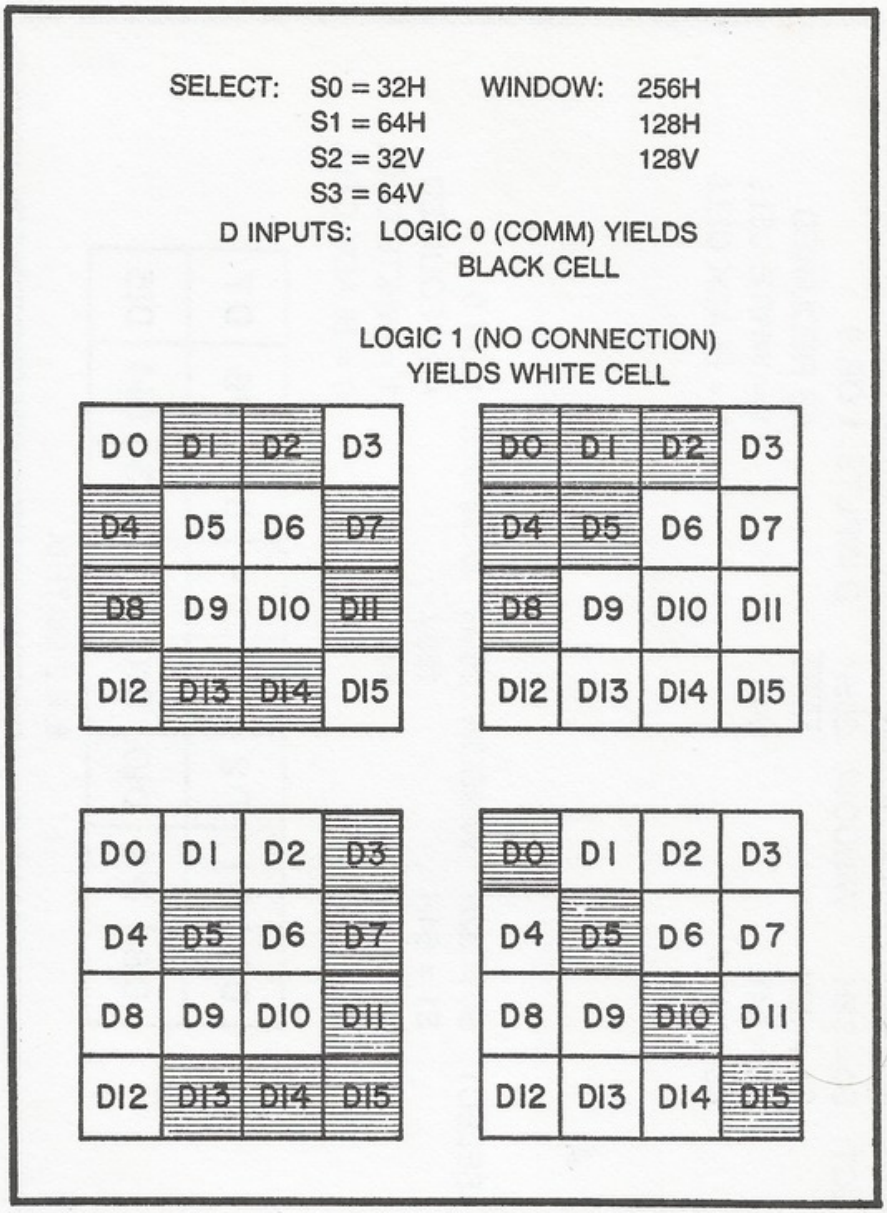


Fig. 4-6. Some experimental figures for the 16-cell matrix generator. All of these figures use a 4 × 4 format.

soon as the experiments seem to be getting a bit dull, move on to the next stage of the project, generating extended matrices.

Extending the Matrix

After experimenting with the basic 4×4 matrix for a while, you will come to the conclusion that it is incapable of generating a very wide selection of interesting figures. The versatility, resolution, and overall quality of TV-game images is directly proportional to the

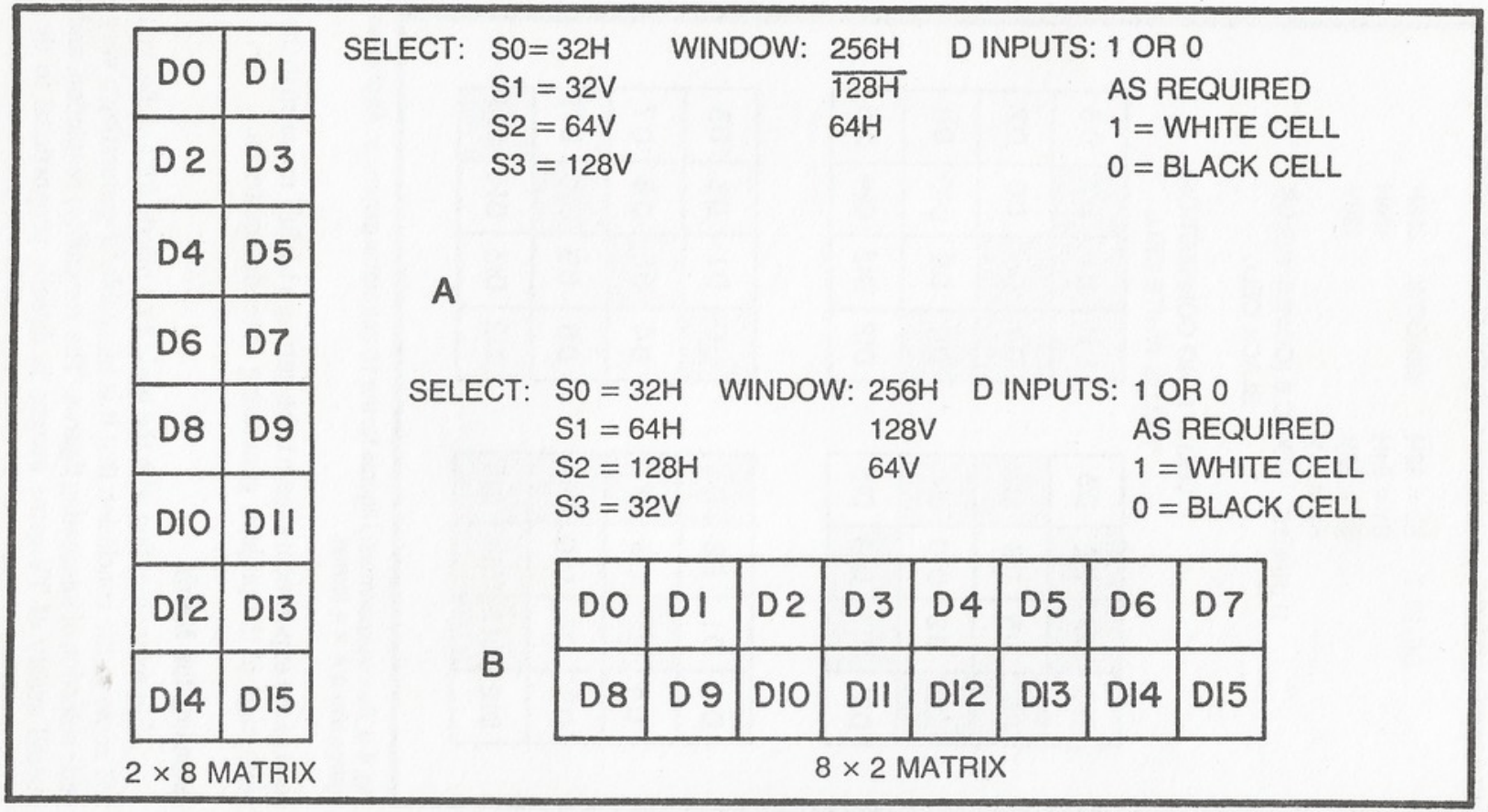


Fig. 4-7. 16-cell matrices organized in patterns other than 4 × 4. (a) 2 × 8. (b) 8 × 2.

number of cells available in the matrix, and unfortunately, the basic 16-cell matrix is a bit too elementary to be of much use.

It turns out, however, that it is possible to apply a simple matrix-extension procedure to double the number of cells in any basic matrix. Using this procedure, for instance, immediately transforms the 4×4 matrix shown in Fig. 4-4a into a 32-cell, 4×8 matrix (See Fig. 4-8).

One of the special features of the matrix-extension technique used in this book is that it does not call for adding any hardware to the basic matrix-generating circuit. The circuit for generating the 32-cell matrix in Fig. 4-8 is exactly the same as that described in connection with the 16-cell circuit in Fig. 4-5.

Thus it is possible to double the essential qualities of a matrix-generating circuit without having to add more hardware. And that is one fine example of applied engineering efficiency.

The only real difference between the specifications for a basic 16-cell matrix and its 32-cell extended version is the D-input format. Whereas a basic matrix calls for inputs of either logic 0 or 1, the extended version calls for logic 1, 0, and inverted/noninverted vertical-count signals. Compare the D-input specification for the 4×8 extended matrix in Fig. 4-8 with those for the basic 4×4 matrix in Fig. 4-6.

Rather than launching a highly technical theory of operation at this point, it is better (in this case anyway) to try the scheme first.

Using the circuit shown in Fig. 4-5, modify its select and window specifications as prescribed in Fig. 4-8. Do not make any connections to the D inputs at this time.

After carrying out this initial setup procedure, you should find a 4×16 white rectangle resting in the lower right-hand quadrant of the screen. This is the extended 4×8 matrix having all D inputs equal to logic 1.

Connect one end of a jumper wire to logic 0 (COMM) and touch the other end to the multiplexer's D0 input. You will find two black squares appearing on the screen. One is in the D0 position of the matrix and the other is in the D0E position. Here is the first important point. A logic-0 level applied to any one of the 16 D inputs on the multiplexer creates 2 black squares. One of these squares appears in the designated D-input position, while the other appears in the corresponding extended-matrix position.

Tap the logic-0 jumper wire to all 16 D inputs in succession and note how it eliminates 2 cells at one time. Of course you can connect all 16 D inputs to logic 0 and end up blacking out all 32 matrix cells—2 at a time.

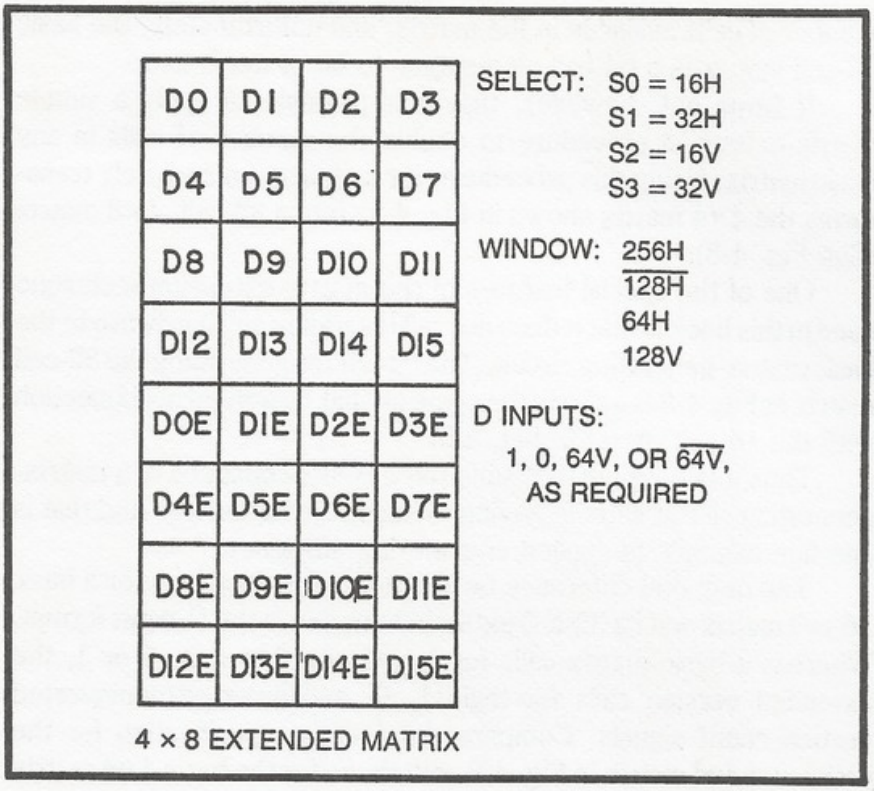


Fig. 4-8. A 4 × 8 extended matrix from a basic 16-cell circuit. The Select, Window and D-input specifications are merely examples.

Note that the extended-matrix pattern in Fig. 4-8 shows the usual 16 cells arranged in a 4×4 pattern at the top of the rectangle. The bottom half of the rectangle is composed of another 4×4 matrix with cells designated D0E, D1E, D2E, etc. The "E" suffix indicates an extended matrix cell, and whenever one of the cells in the top half of the matrix is blackened by setting that D input to logic 0, its extended counterpart in the bottom half of the matrix is also blackened.

While this procedure for generating two black squares at the same time is a cute trick, it isn't really very useful (unless the figure you want to create happens to be a symmetrical one). So here is the clincher. Remove any logic-0 connections to the D inputs of the multiplexer and connect one end of the jumper wire to the 64V source. Connect the other end of this jumper to the D0 input of the multiplexer, and presto! There is a single black square in matrix position D0. You are no longer getting the two-square effect you found when connecting the D inputs to logic 0.

Try touching this 64V source to the multiplexer's D inputs, one at a time and in succession. You will be able to blacken any one of the

16 cells in the upper half of the matrix. But notice that it is not possible to darken any of the extended-matrix cells in the lower half of the rectangle.

Next, run the 64V source through an inverter to obtain a $\overline{64V}$ signal. Connect that $\overline{64V}$ signal to the D inputs of the multiplexer and note the response on the screen. It turns out that $\overline{64V}$ inputs darken cells in the lower half of the rectangle, leaving those in the upper half unaffected.

Connect various combinations of logic 0, 64V, and $\overline{64V}$ to the D inputs of the multiplexer. Observe the response in each case, and continue experimenting in this fashion until you are convinced you understand the behavior of this valuable extended-matrix technique.

The general rules for designing figures using the extended-matrix technique go something like this:

- If a cell in the top half of the matrix and its extended counterpart in the bottom half are to be white, make no connections to that position on the multiplexer's D inputs.

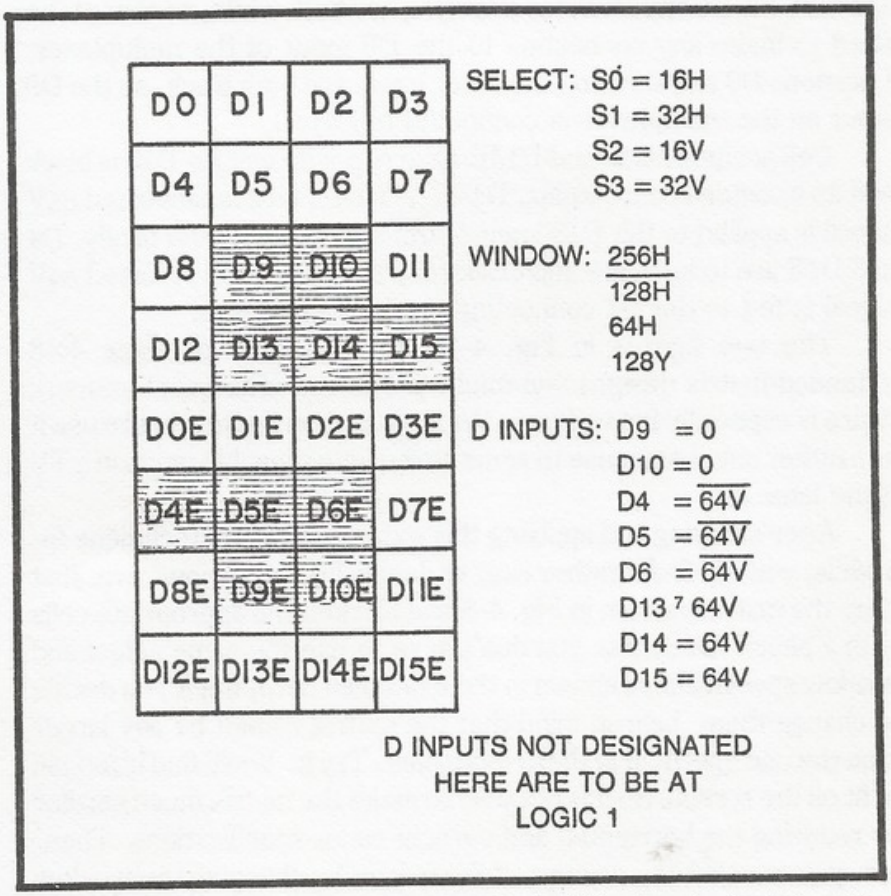


Fig. 4-9. Specifications for a stylized S figure built within a 4 × 8 extended matrix.

- If a cell in the top half of the matrix and its extended counterpart in the bottom half are to be black, connect that position on the multiplexer's D inputs to logic 0.
- If a cell in the top half of the matrix is to be black and its extended counterpart in the bottom half of the matrix is to be white, connect a noninverted 64V source to the appropriate D input on the multiplexer.
- If a cell in the top half of the matrix is to be white and its extended counterpart in the bottom half of the matrix is to be black, connect an inverted 64V source to the appropriate D input on the multiplexer.

This set of rules for designing static figures with the extended-matrix scheme might seem very complicated at first, but it can become rather obvious after playing with the circuits for a while.

Figure 4-9 shows the matrix pattern, select, window, and D-input specifications for generating a stylized S figure. Note that the specifications for the D inputs follow the general rules outlined above. Cells D0 and D0E, for example, are both white, so there is no need to make any connection to the D0 input of the multiplexer. Locations D9 and D9E, on the other hand, are both black, so the D9 input on the multiplexer is connected to logic 1.

Cell positions D13 and D13E have opposite colors: D13 is black and its extended counterpart, D13E, is white; so a noninverted 64V signal is applied to the D13 input of the multiplexer. And finally, D4 and D4E are to be white and black respectively, so an inverted 64V signal is fed to the D4 connection on the multiplexer.

The two figures in Fig. 4-10 are examples of some 4×8 extended-matrix designs you might like to try. The question-mark figure is especially fun to see on the screen. You might want to use it as a rather novel response to some sort of questionable move in a TV game later on.

After studying and applying this extended-matrix technique for a while, you will find it rather easy to design figures of your own. Just copy the matrix pattern in Fig. 4-8 and blacken the appropriate cells with a pencil. Of course you don't have to use the same select and window specifications shown in the examples here, but if you decide to change them, bear in mind that the matrix cannot be any larger than the one specified in these examples. Try it. You'll find it too tall to fit on the screen. So it is possible to make the matrix much smaller by reducing the horizontal- and vertical-count specifications. Then, too, you can shift the position of the matrix by changing the window specifications.

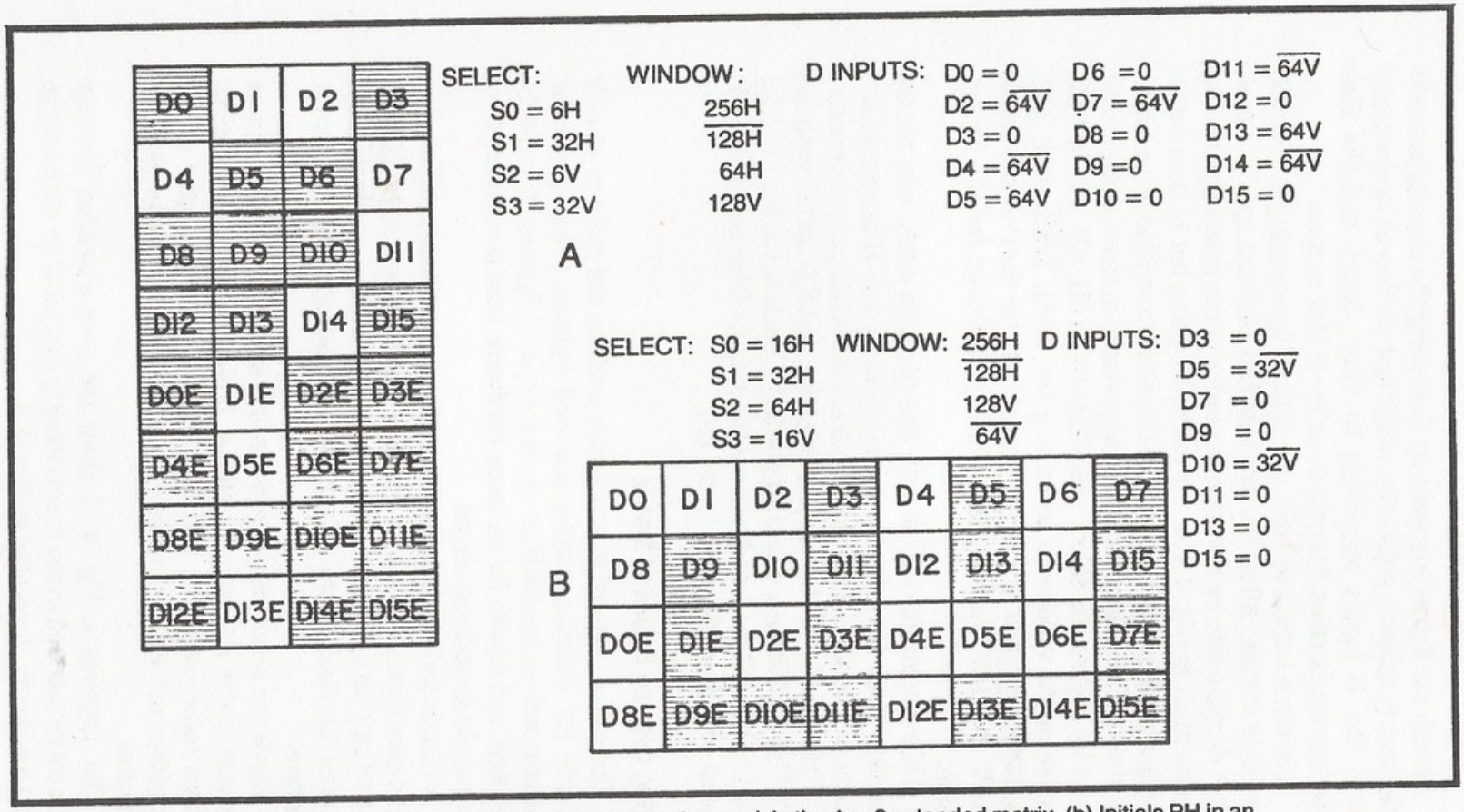


Fig. 4-10. Further examples of extended-matrix figures. (a) A question mark in the 4 × 8 extended matrix. (b) Initials PH in an extended 8 × 8 matrix.

Sketch the figure you want by darkening the appropriate cells on the matrix pattern, adjust the select and window as desired, and specify the D inputs according to your sketch and the four extended-matrix rules described earlier in this section.

It is often helpful to build the original figure using a relatively large-scale matrix. After you are satisfied with the specifications, scale it down to size by lowering the order of the select inputs. The question mark in Fig. 4-10a, for example, turns out to be a rather large figure when built according to the select and window specifications shown in that diagram. It can be reduced in size a considerable amount by setting the select specifications to 4H, 8H, 4V, and 8V, then setting the window to something like 256H, $\overline{128H}$, 64H, 32H, 16H, 126V, 64V, and 32V. The V-count signal to the D inputs should be 16V and $\overline{16V}$, rather than 64V and $\overline{64V}$ used for the larger version.

While you are having fun with this question mark, why not try something else? Use the small-figure select and D inputs as just described. But instead of using the 8-input NAND gate to generate the little window, try replacing the NAND gate with an EXCLUSIVE-OR gate. Run 32H and 32V signals to the inputs of the EXCLUSIVE-OR gate, and connect the output of that gate to the $\overline{CE}$ terminal of the multiplexer. Cute, eh?

Folding Over an Extended Matrix

The matrix-extension procedure doubles the number of cells available for building matrix-oriented figures. That particular technique has no restrictions on the type of figures that can be generated, and it calls for no more hardware than is required for a basic matrix-generating circuit.

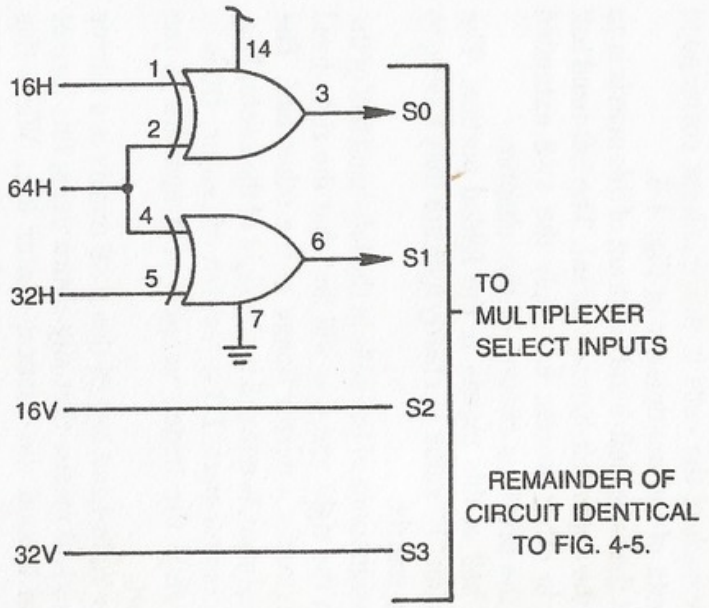
A matrix foldover scheme described here doubles the number of cells once again. A basic 4×4 matrix, for instance, can be doubled to a 4×8 pattern by means of the matrix-extension procedure, and it can then be doubled to an 8×8 matrix by applying a foldover technique.

There are some restrictions on the kind of figure that can be generated under foldover conditions, however, and the technique requires some additional EXCLUSIVE-OR gates. In spite of the restrictions and extra hardware, the foldover technique pays off quite often.

The diagrams in Fig. 4-11 show the 8×8 extended foldover matrix pattern as well as the extra circuitry that must be added to the basic 4×4 matrix generator in Fig. 4-5.

8 × 8 EXTENDED FOLDOVER MATRIX

D0	D1	D2	D3	D3F	D2F	D1F	D0F
D4	D5	D6	D7	D7F	D6F	D5F	D4F
D8	D9	D10	D11	D11F	D10F	D9F	D8F
D12	D13	D14	D15	D15F	D14F	D13F	D12F
D0E	D1E	D2E	D3E	D3EF	D2EF	D1EF	D0EF
D4E	D5E	D6E	D7E	D7EF	D6EF	D5EF	D4EF
D8E	D9E	D10E	D11E	D11EF	D10EF	D9EF	D8EF
D12E	D13E	D14E	D15E	D15EF	D14EF	D13EF	D12EF



SELECT: So = 16H F 64H
S1 = 32H F 64H
S2 = 16V
S3 = 32V

WINDOW: 256H
$\overline{128H}$
128V

D INPUTS: 1, 0, 64V, $\overline{64V}$, AS REQUIRED

Fig. 4-11. Matrix pattern and additional circuitry for creating an 8 × 8 extended foldover matrix from the basic 16-cell generator circuit. Select, Window, and D-input specifications are just examples.

A cursory study of the matrix pattern in Fig. 4-11 will show that the original 4×4 matrix is situated in the upper left-hand quadrant. The designations for each of the cells in this quadrant correspond exactly to the D inputs of the multiplexer in Fig. 4-5.

Then notice that the lower left-hand quadrant of this matrix is an extended version of the upper left-hand quadrant. The left-hand half of this 8×8 matrix, in other words, is simply the 4×8 extended matrix described in the previous section of this chapter.

The right-hand half of this matrix is the folded portion. The foldover cells all have an "F" suffix to clearly indicate they belong to the folded half of the matrix.

Compare the designations of the cells in the left-hand half of the matrix with those on the right and you will find that the right-hand half of the diagram is really a mirror image of the other half. The folded cell D3F, for instance, is one cell to the right of the center line, while its originating counter-part, D3 is just left of center. D0 is in one upper corner, while the folded version of it appears in the opposite upper corner.

The fact that the right-hand half of this 8×8 matrix is a mirror image of the left-hand half means that any figure using this format must be symmetrical around the vertical center line. While this might seem to be a rather severe restriction at first, it turns out that a great many interesting and useful figures do, indeed, have a symmetrical (mirror-image) quality. Look ahead to Fig. 4-12 for one good example.

All that is required for achieving the foldover effect in this case is to run the 16H and 32H signal through EXCLUSIVE-OR gates before applying them to the S0 and S1 select terminals of the multiplexer. One input of each of the EXCLUSIVE-OR gates is connected to 64H. So when 64H is at logic 0, noninverted versions of 16H and 32H appear at select inputs S0 and S1. When 64H switches to logic 1, however, S0 and S1 see inverted versions of 16H and 32H.

The circuit thus generates the normal matrix and extended-matrix patterns as long as 64H is at logic 0, but as soon as 64H rises to logic 1, it effectively reverses the horizontal scanning operation. Along the first row of cells, for instance, S0 and S1 see binary levels representing a normal count of 0, 1, 2, and 3. At the end of D3, though, the EXCLUSIVE-OR gates begin reversing the count: 3, 2 1, 0. A reversed image thus appears along the right-hand half of the matrix.

Connect the basic 4×4 matrix-generating circuit shown in Fig. 4-5, then add the two EXCLUSIVE-OR gates as shown in Fig. 4-10.

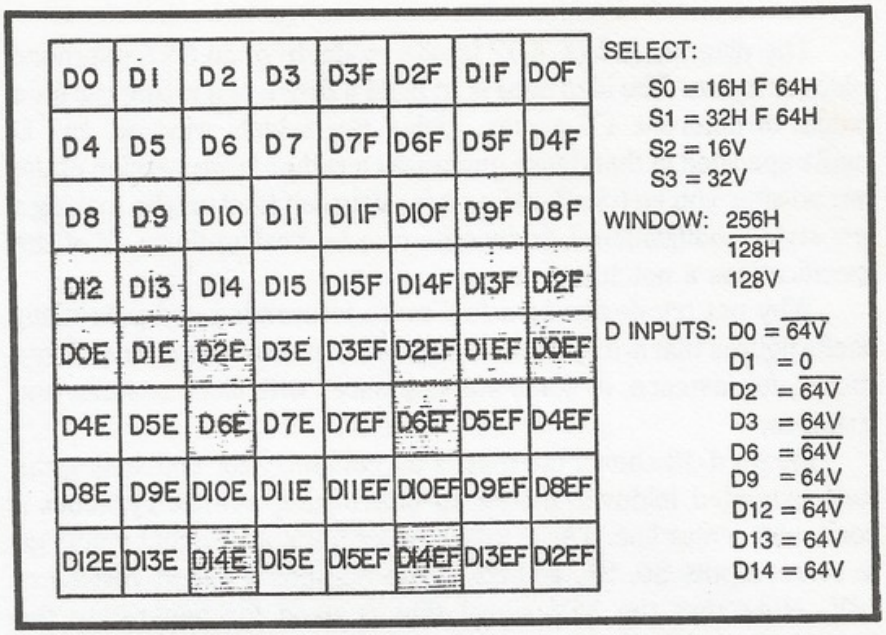


Fig. 4-12. Figure and specifications for a racing car, using the 8 × 8 extended foldover matrix circuit.

Finally, set the select and window specifications as shown. (The select statement S0=16H F 64H can be interpreted as meaning, "S0 equals 16H folded by 64H").

Using these specifications, you will find a large white square near the lower right-hand corner of the screen.

To begin getting a feeling for how this 8×8 extended foldover matrix works, connect the D inputs of the multiplexer to logic zero, beginning with D0 and working up toward D15. When you connect D0 to logic 0, you should see four black squares appearing at D0, D0F, D0E, and D0EF. In fact four black squares should appear anytime one of the multiplexer's D inputs is connected to a logic-0 source. The four squares represent the original D position, its extended counterpart, and foldover versions of both the original and extended squares.

Whenever any of the D inputs are connected to 64V, you should see a pair of black squares in the upper half of the matrix. The one on the right side is a mirror-image of the one on the left. And in the same fashion, connecting any of the multiplexer's D inputs to an inverted version of 64V creates a pair of black squares in the bottom half of the matrix.

The four rules for generating a 4×8 extended matrix pattern apply here; so if you have mastered the earlier procedure, you are fully prepared to begin building extended foldover patterns.

The diagram in Fig. 4-12 is one example of an 8×8 extended foldover figure. The idea here is to build a figure of a racing car for a couple of different TV games. Using the select, window, and D inputs specified in that figure generates a rather large version of the car; so after you've tried building it for yourself (and maybe making a few style modifications), reduce its size by scaling down all of the specifications a notch or two.

Why not try designing a few more interesting and potentially useful figures that are symmetrical about a vertical center line? Try a rocket, for instance, or some stars, squares with black centers, and triangles.

Figure 4-13 shows the matrix and circuitry for converting the 8×8 extended foldover matrix to one having symmetry about a horizontal center line. Three lower-order horizontal-count inputs go to select inputs S0, S1, and S2, while S3 sees a folded version of 16V. Note that the 32V signal that is used for generating the extended-matrix effect must also be folded before it is applied to the appropriate D inputs of the multiplexer circuit.

An example of an 8×8 extended, horizontally folded image is shown in Fig. 4-14. As with every matrix-generated figure in this chapter, it can be reduced in size by scaling down all the vertical- and horizontal-count inputs. Using the specifications given in Fig. 4-14, the airplane occupies most of the lower right-hand quadrant of the screen.

Figure 4-15 shows yet another version of the 64-cell extended foldover matrix built around the circuit in Fig. 4-5. In this instance the matrix is configured as a 4×16, with symmetry about the vertical axis. Using the select, window, and data specifications shown here, this particular matrix is quite useful for building missile images. See the suggested missile in Fig. 4-16.

See if you can apply your understanding of how the foldover principle works to generate a missile image that fits into a 16×4 matrix that is folded along the horizontal center line.

MATRIX OPERATIONS FROM A 32-CELL GENERATOR

The circuit in Fig. 4-5 is a basic 16-cell matrix generator. By applying extended-matrix and matrix foldover procedures, it is possible to build 32-cell and 64-cell matrices. The only restriction on the kinds of figures it can generate is that the 64-cell version is good only for making symmetrical figures.

And while it is a lot of fun to play with the system just described, personal experience with it shows that other people sometimes have

8 × 8 EXTENDED FOLDOVER MATRIX
(HORIZONTAL SYMMETRY)

D0	D1	D2	D3	D4	D5	D6	D7
D8	D9	D10	D11	D12	D13	D14	D15
D0E	D1E	D2E	D3E	D4E	D5E	D6E	D7E
D8E	D9E	D10E	D11E	D12E	D13E	D14E	D15E
D8EF	D9EF	D10EF	D11EF	D12EF	D13EF	D14EF	D15EF
D0EF	D1EF	D2EF	D3EF	D4EF	D5EF	D6EF	D7EF
D8F	D9F	D10F	D11F	D12F	D13F	D14F	D15F
D0F	D1F	D2F	D3F	D4F	D5F	D6F	D7F

SELECT: S0 = 16H
S1 = 32H
S2 = 64H
S3 = 16V F 64V

WINDOW: 256H
128H
128V

D INPUTS: 1, 0
32V F 64V,
32V F 64V,
AS REQUIRED

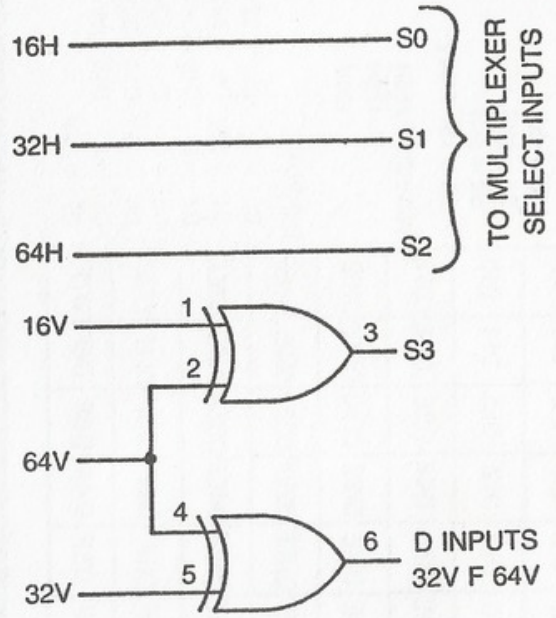


Fig. 4-13. Matrix configuration, circuit, and sample specifications for an 8 × 8 cxtended matrix that is folded horizontally.

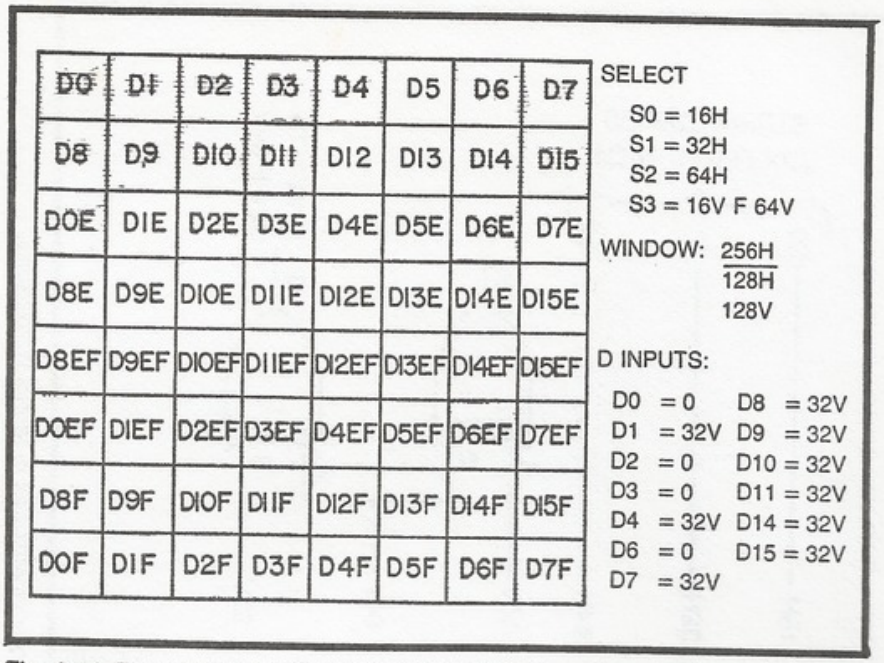


Fig. 4-14. Figure and specifications for an airplane, using a horizontally folded, 8 × 8 extended matrix.

trouble identifying the object presented on the screen. Even with 64 cells available for building an image, the resolution is often so coarse that others might, indeed, have trouble seeing what the image is supposed to represent. When this is the case, it is time to move up to a matrix generator that yields a higher resolution.

The circuit in Fig. 4-17 uses two 16:1 multiplexer circuits. The basic system generates 32 cells. (Note that there are 32 D inputs, labeled D0 through D31.) If the matrix-extension technique is applied to this circuit, the number of available cells rises to 64. And if the foldover technique is also used, the system can generate a 128-cell symmetrical figure.

This system represents a 2:1 increase in image resolution over the basic 16-cell circuit in Fig. 4-5.

IC6 and IC6 are identical ICs. In fact they are the same multiplexer IC used for all of the complex-figure circuits already described in this chapter. Note, however, that this system has five select inputs, S0 through S4, while each multiplexer IC has input provisions for only four.

We can take care of the four basic select inputs on each multiplexer IC by connecting them in parallel: pin 15 of IC5 to pin 15 of IC6, pin 14 of IC5 to pin 14 of IC6, and so on. The parallel-connected select inputs on the two multiplexer ICs then go to system-select inputs S0 through S3.

D0	D1	D1F	D0F
D2	D3	D3F	D2F
D4	D5	D5F	D4F
D6	D7	D7F	D6F
D8	D9	D9F	D8F
D10	D11	D11F	D10F
D12	D13	D13F	D12F
D14	D15	D15F	D14F
D0E	D1E	D1EF	D0EF
D2E	D3E	D3EF	D2EF
D4E	D5E	D5EF	D4E
D6E	D7E	D7EF	D6EF
D8E	D9E	D9EF	D8EF
D10E	D11E	D11EF	D10EF
D12E	D13E	D13EF	D12EF
D14E	D15E	D15EF	D14EF

4 × 16 EXTENDED
FOLDOVER MATRIX

SELECT: S0 = 8H F 16H
S1 = 8V
S2 = 16V
S3 = 32V

WINDOW: 256H
$\overline{128H}$
$\overline{64H}$
$\overline{32H}$
128V

D INPUTS: 1, 0, 64V, $\overline{64V}$,
AS REQUIRED

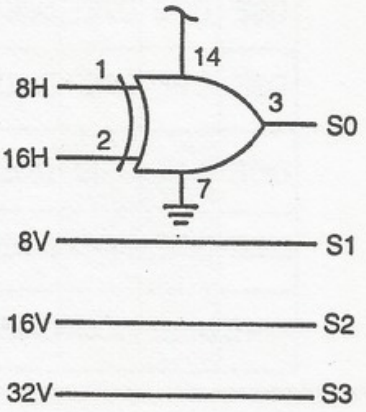


Fig. 4-15. Matrix configuration, circuit, and sample specifications for a 4 × 16 extended foldover matrix.

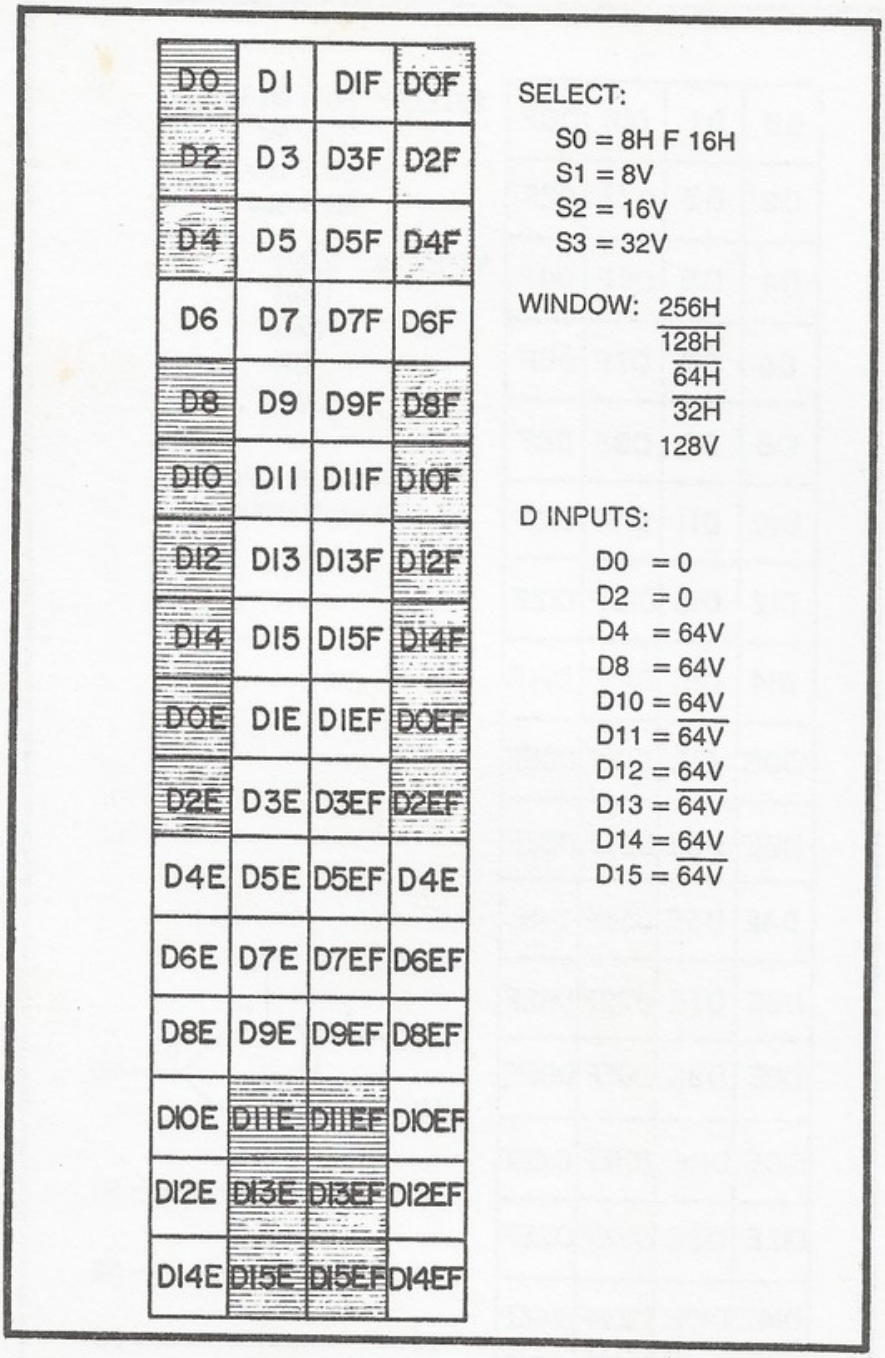


Fig. 4-16. A vertical missle figure built within a 4 × 16 extended foldover matrix.

Using this kind of parallel selection, the two multiplexers are always selected exactly the same way. If the system-select inputs happen to select D2 on IC5 (pin 6), it must also be selecting D18 (pin 6) on IC6. As far as system-select inputs S0 through S3 are con-

cerned, they select the same data input on both ICs at the same time. The scheme works very much like a two-deck rotary switch.

What, then, is the role of the S4 input? Ultimately, the purpose of the S4 input is to make certain the two multiplexers are never enabled at the same time. If this 5th-bit input happens to have a logic-0 level, for instance, it can enable multiplexer IC5 and disables IC6. Setting select input S4 to logic 1, on the other hand, makes it possible to disable IC5 and enable IC6.

ICs 5 and 6 are thus both disabled at the same time; or, according to the logic state of S4, one is enabled while the other is disabled. It is not possible to enable both multiplexers at the same time. (Both multiplexers are disabled by the window inputs, a circuit described a bit later in this section.)

Now consider all five system-select inputs at the same time. As long as the S4 select input is at logic 0, the status of the four lower-order select inputs (S0 through S3) select one of the 16 data inputs to IC5. An inverted version of that particular input to IC5 appears inverted at pin 10 of that IC. IC6, however, is disabled at that time. And though select inputs S0 through S3 might be selecting one of the 16 inputs to IC6, pin 10 of that IC remains fixed at logic 1, the disabled output condition.

IC5 and IC6 change roles when S4 goes to logic 1. IC6 then outputs its selected data to its pin-10 connection, and IC5 is disabled.

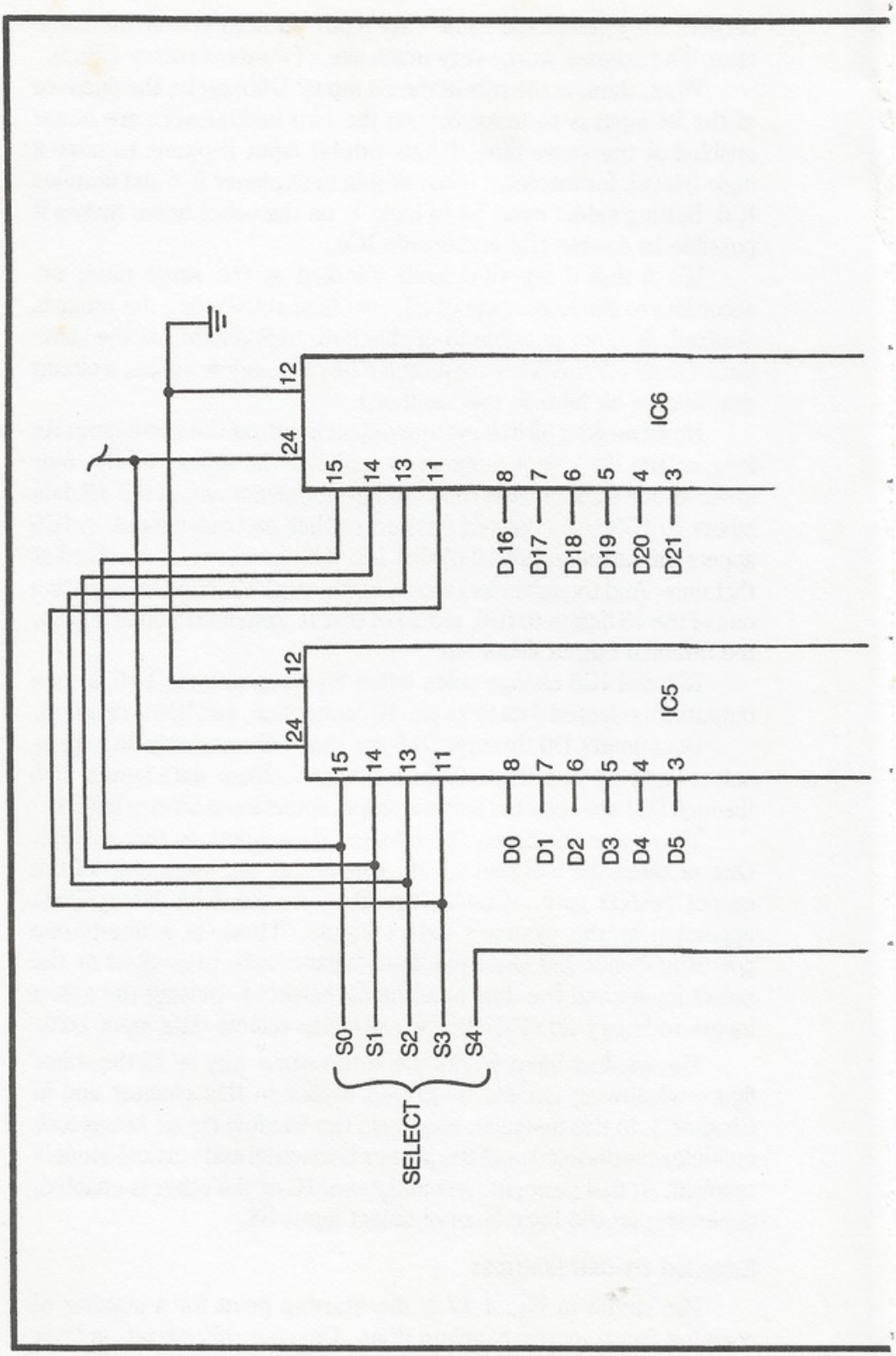
Data inputs D0 through D15 are thus relevant only as long as select input S4 is at logic 0. In a similar fashion, data inputs D16 through D31 are selected only as long as select input S4 is at logic 1.

There are effectively 32 different data inputs to this system. One of these 32 different inputs appears at the input side of the output NAND gate, IC2-C, depending on the 5-bit binary code appearing at the system's select inputs. There is a one-to-one correspondence between the 5-bit binary code presented at the select inputs and the data input being selected. Setting the select inputs to binary 20 (10100), for example, selects data input D20.

The window input works much the same way as all the other figure-windowing circuits described earlier in this chapter and in Chapter 3. In this instance, however, the window circuit keeps *both* multiplexers disabled until the proper horizontal and vertical count is reached. At that moment, on multiplexer IC or the other is enabled, depending on the logic level of select input S4.

Extended 64-Cell Matrices

The circuit in Fig. 4-17 is the starting point for a number of complex-figure matrix configurations. Connect this circuit on your


12
24
15
14
13
11
8
7
6
5
4
3
IC6
D16
D17
D18
D19
D20
D21
12
24
15
14
13
11
8
7
6
5
4
3
IC5
D0
D1
D2
D3
D4
D5
S0
S1
S2
S3
S4
SELECT

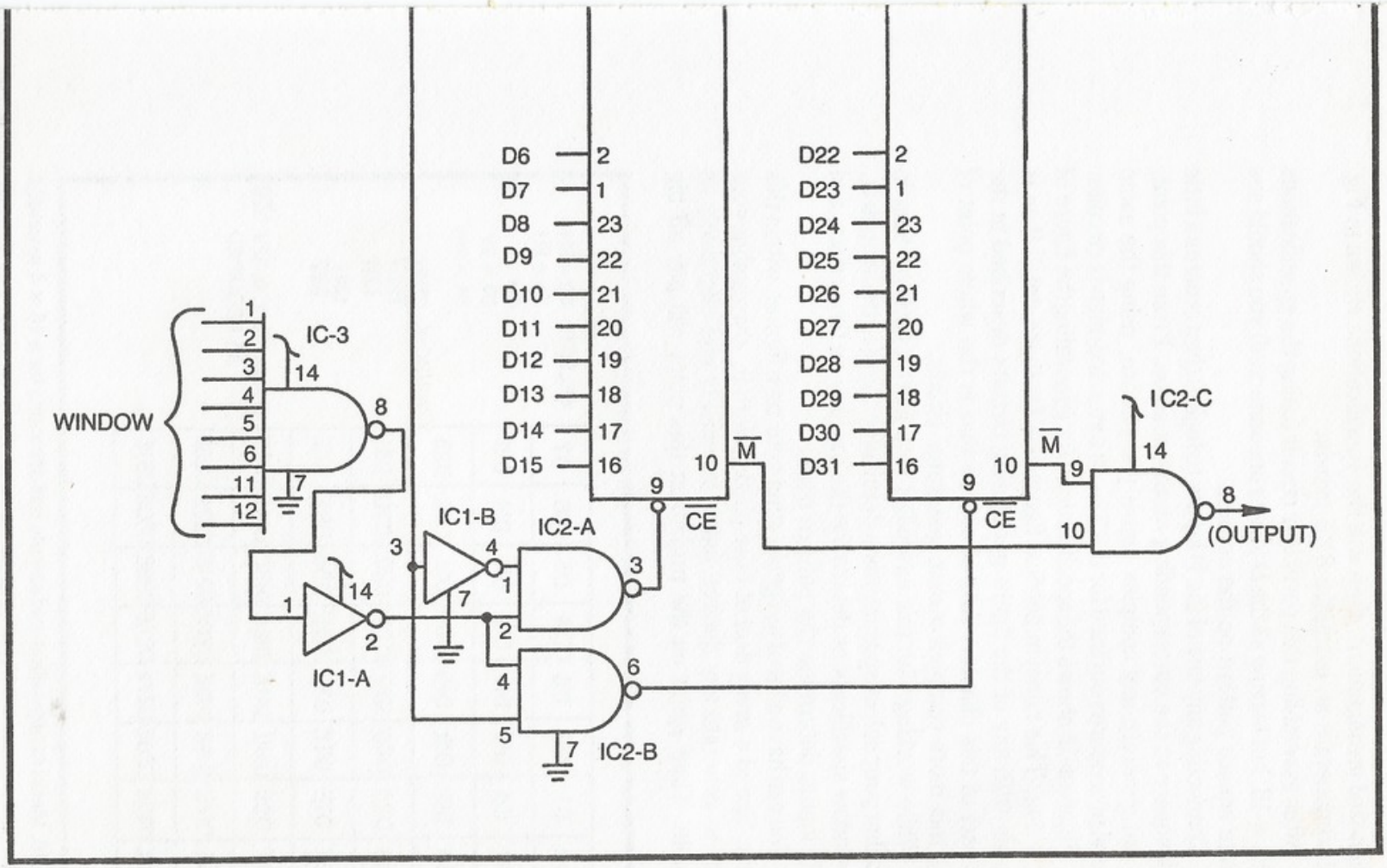


Fig. 4-17. Circuit for a basic 32-cell matrix generator.

breadboard arrangement, then use the specifications shown in Fig. 4-18 to generate an extended 8×8 matrix.

After assembling this particular circuit (using the specifications in Fig. 4-18, but leaving all the D inputs unconnected) you should see a white square pattern on the screen.

Connecting any one of the D inputs to logic 0 then creates a little black square at the corresponding matrix position. From this point, you can generate any complex figure you choose, using the same general procedures outlined for the 32-cell extended-matrix circuits.

Figure 4-19 shows the specifications for generating the figure of a little dog. This nonsymmetrical figure has far more detail than is possible with any of the figure-generating circuits described in the first part of this chapter. And of course that is the whole point of using this more complex matrix-generator circuit.

While working with this extended 8×8 matrix pattern, bear in mind that your select specifications determine the size of the matrix, the window specifications determine the position on the screen, and the D inputs determine the pattern itself.

You can have a lot of fun generating your own figures within this matrix. Spend a great deal of time playing with it, sharpening your ability to generate any desired figure as quickly and effectively as possible. Time spent on the project at this point will pay off big

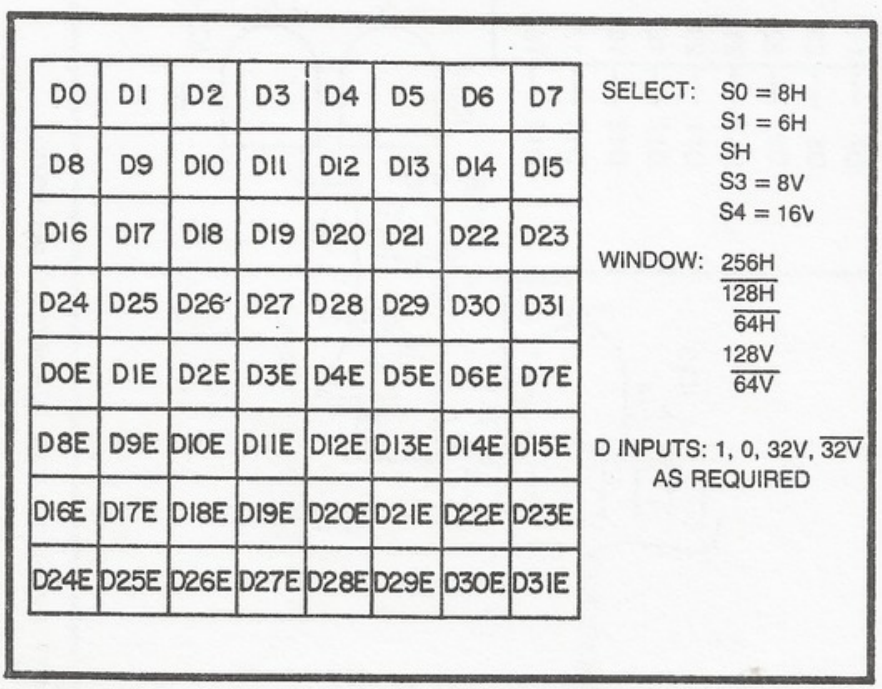


Fig. 4-18. Matrix configuration and sample specifications for a 16 × 4 extended matrix.

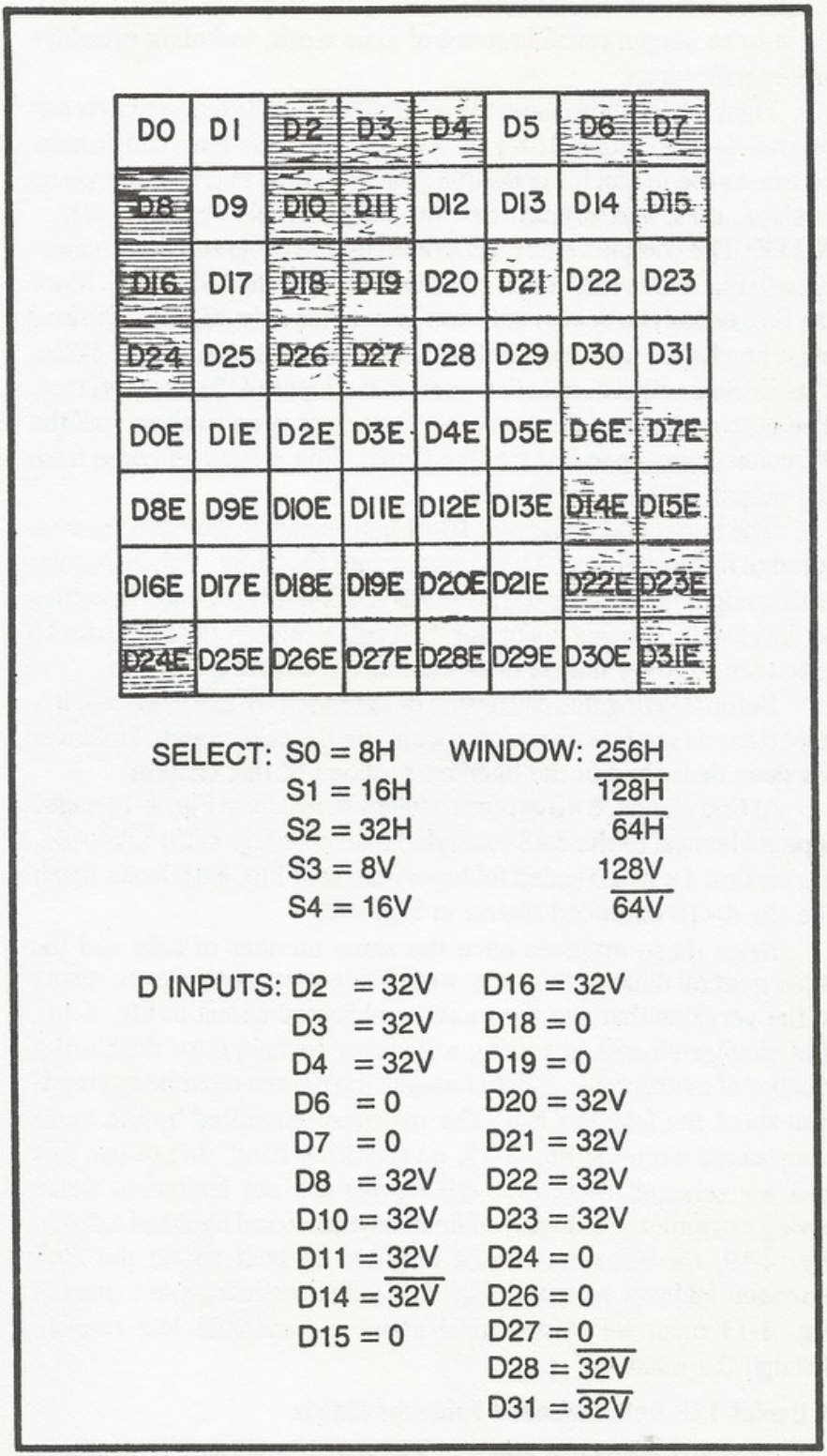


Fig. 4-19. Figure of a dog built within a 8 × 8 extended matrix.

dividends later on when you are attempting to design video games. Be sure to keep a careful record of your work, including drawings and specifications.

Figure 4-20 shows how the extended 64-cell square matrix can be transformed into a 16×4 matrix format. This long and narrow matrix can be useful for generating game figures such as side views of ships, cars, and aircraft. See the two examples in Fig. 4-21. **NOTE:** The complex figure generated by the programming connections in Fig. 4-21a calls for 21 connections from the 8V source. Since the Sourcebox can deliver sufficient power for only 20 loads, the load must be divided by means of the buffer circuit shown in Fig. 4-21a. The 8V Sourcebox connection goes to the input of the first inverter. The output of one of the two other inverters goes to about half the 8V connections specified for this figure. The other half come from the output of the third inverter.

The horizontally oriented 16×4 matrix can be likewise restructured to form a vertical, 4×16 matrix. See Fig. 4-22. This particular matrix might not be very useful as it is specified here, but it becomes an invaluable starting point for building a highly desirable 8×16 extended foldover matrix described in the following section.

Before leaving this discussion of extended 64-cell matrices, it is important to see how they differ from the 64-cell extended foldover versions described in the opening sections of this chapter.

At first glance, 8×8 extended foldover matrix in Fig. 4-13 might appear identical to the 8×8 extended matrix in Fig. 4-18. Likewise, the vertical 4×16 extended foldover matrix in Fig. 4-15 looks much like the 4×16 extended matrix in Fig. 4-22.

Since these matrices have the same number of cells and the same general dimensions, why would an experimenter ever resort to the versions that use the more complicated circuit in Fig. 4-17? The simpler circuits are using a foldover technique to double the number of matrix cells, and that means the figures must be symmetrical about the foldover line. The matrices generated by the more complicated circuit in Fig. 4-17, on the other hand, do not use this foldover scheme; therefore, the figures are not limited to those having a symmetrical display. The nonsymmetrical figure of a dog in Fig. 4-19, for instance, cannot possibly be built within the 8×8 extended foldover matrix in Fig. 4-13. Figures using the matrix in Fig. 4-13 must be symmetrical about a horizontal line running through the middle of it.

A Useful 128-Cell, Extended Foldover Matrix

Adding a 7486 quad EXCLUSIVE-OR IC package to the circuit in Fig. 4-17 makes it possible to generate a large foldover matrix that

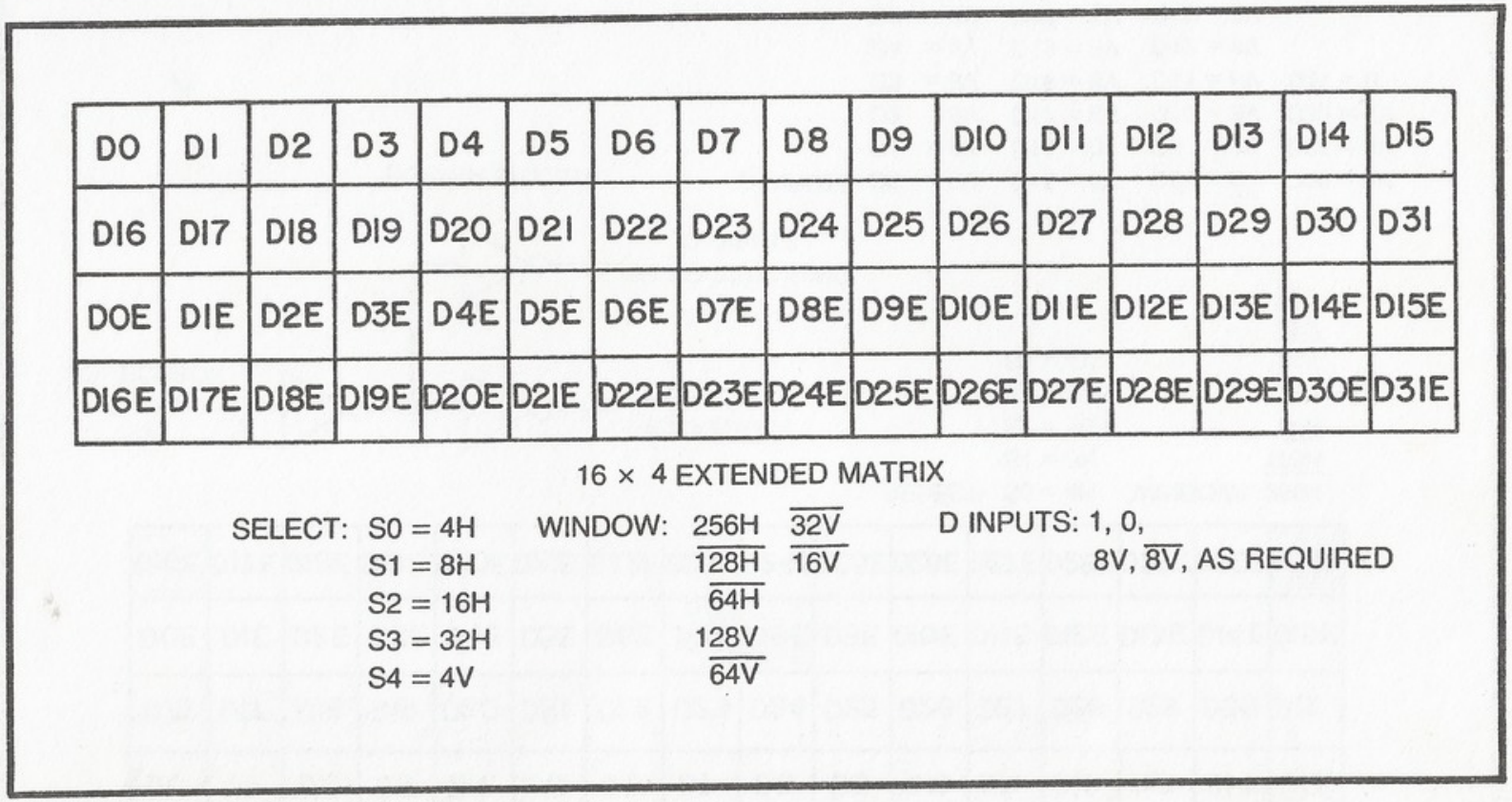
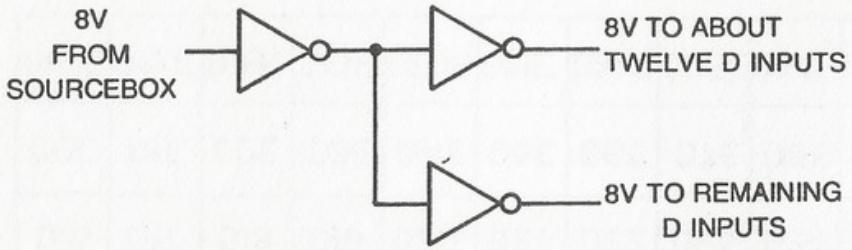


Fig. 4-20. Configuration and sample specifications for a 16 × 4 extended matrix.

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
D0E	D1E	D2E	D3E	D4E	D5E	D6E	D7E	D8E	D9E	D10E	D11E	D12E	D13E	D14E	D15E
D16E	D17E	D18E	D19E	D20E	D21E	D22E	D23E	D24E	D25E	D26E	D27E	D28E	D29E	D30E	D31E

SELECT: S0 = 4H
S1 = 8H
S2 = 16H
S3 = 32H
S4 = 4V

WINDOW: 256H
128H
64H
128V
64V
32V
16V



BUFFER CIRCUIT

D INPUTS:

D0 = 8V	D15 = 8V	D8 = 0	D26 = 8V
D1 = 8V	D16 = 0	D9 = 8V	D27 = 8V
D2 = 8V	D17 = 8V	D10 = 8V	D30 = 8V
D3 = 8V	D18 = 8V	D11 = 8V	D31 = 0
D4 = 8V	D19 = 8V	D12 = 8V	
D5 = 8V	D20 = 8V	D13 = 8V	
D7 = 0	D21 = 8V	D14 = 8V	

A

D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15
D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30	D31
D0E	D1E	D2E	D3E	D4E	D5E	D6E	D7E	D8E	D9E	D10E	D11E	D12E	D13E	D14E	D15E
D16E	D17E	D18E	D19E	D20E	D21E	D22E	D23E	D24E	D25E	D26E	D27E	D28E	D29E	D30E	D31E

B

SELECT: S0 = 4H
S1 = 8H
S2 = 16H
S3 = 32H
S4 = 4V

WINDOW: 256H
$\overline{128H}$
$\overline{64H}$
128V
64V
$\overline{32V}$
$\overline{16V}$

REMEMBER: D INPUTS THAT ARE NOT LISTED ARE TO BE AT LOGIC 1, OR NO CONNECTION

D INPUTS: D0 = 0
D1 = 8V
D2 = 8V
D5 = 8V
D7 = 8V
D8 = 8V
D9 = 8V
D10 = 8V
D11 = 8V
D12 = 8V
D13 = 8V
D14 = 0
D15 = 0
D16 = 0
D17 = 0
D20 = 8V
D21 = 0
D22 = 0
D23 = 0
D24 = 8V
D25 = 8V
D28 = 8V
D29 = 0
D30 = 0
D31 = 0

Fig. 4-21. Sample figures built within the 16 × 4 extended matrix. (a) Figure of an aircraft carrier or submarine. Note the need for inverters acting as driver circuits. (b) Profile of a racing car and driver.

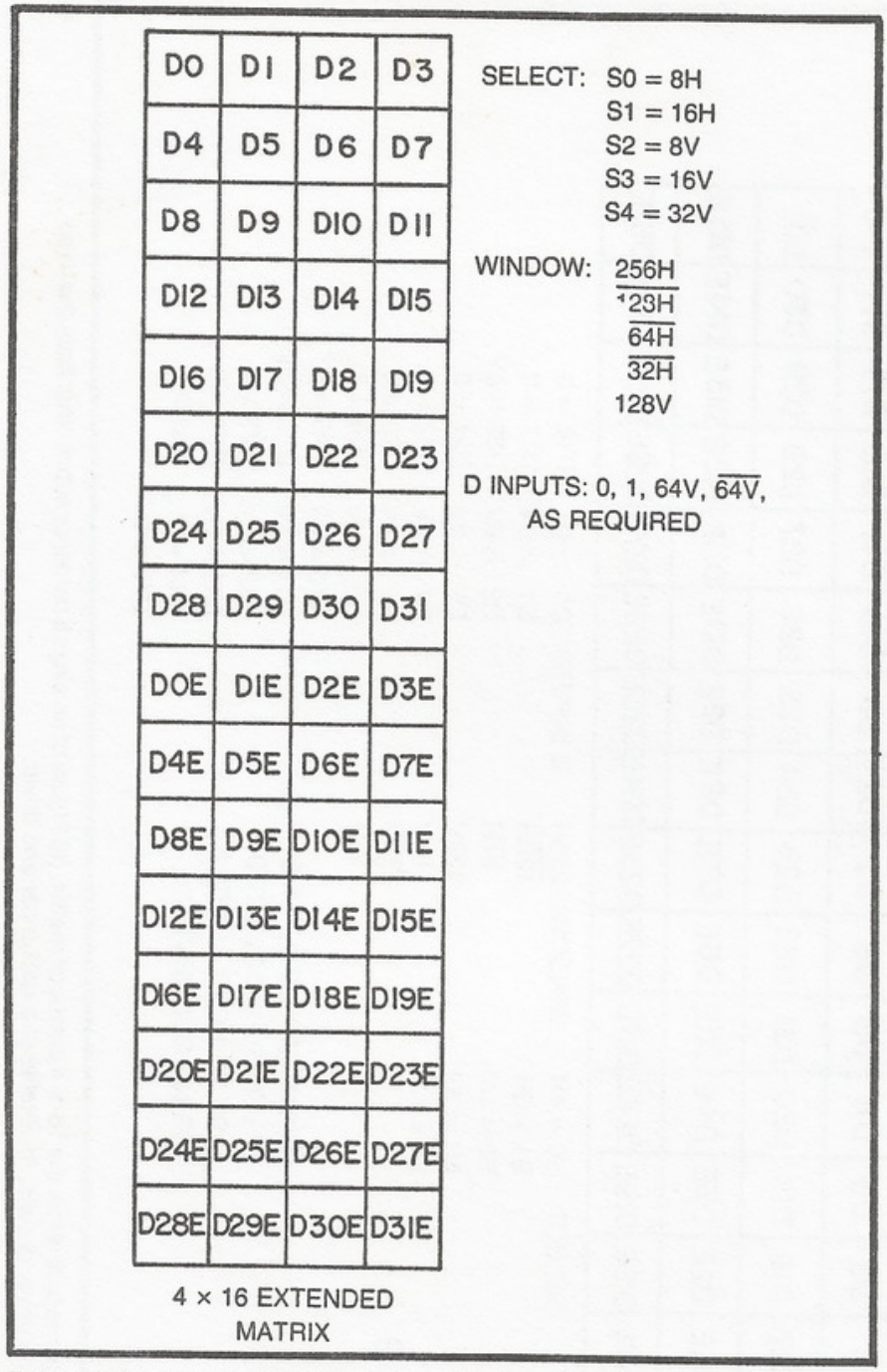


Fig. 4-22. Configuration and sample specifications for a 4 × 16 extended matrix.

is perhaps one of the most useful of all. Connected to the basic 64-cell circuit as indicated in Fig. 4-23, the EXCLUSIVE-OR circuit expands the matrix to a 8×16 format.

The matrix in Fig. 4-23 is a folded matrix that requires symmetry about the vertical line running through its center, but in actual

practice it turns out that an experimenter can generate a vast variety of interesting and useful video game figures with it. How about that cowboy figure specified in Fig. 4-24?

Turn your imagination loose on this matrix and see how many fascinating symmetrical figures you can generate. The principles for designing figures around this 8×16 extended foldover matrix with vertical symmetry are much the same as those for working out

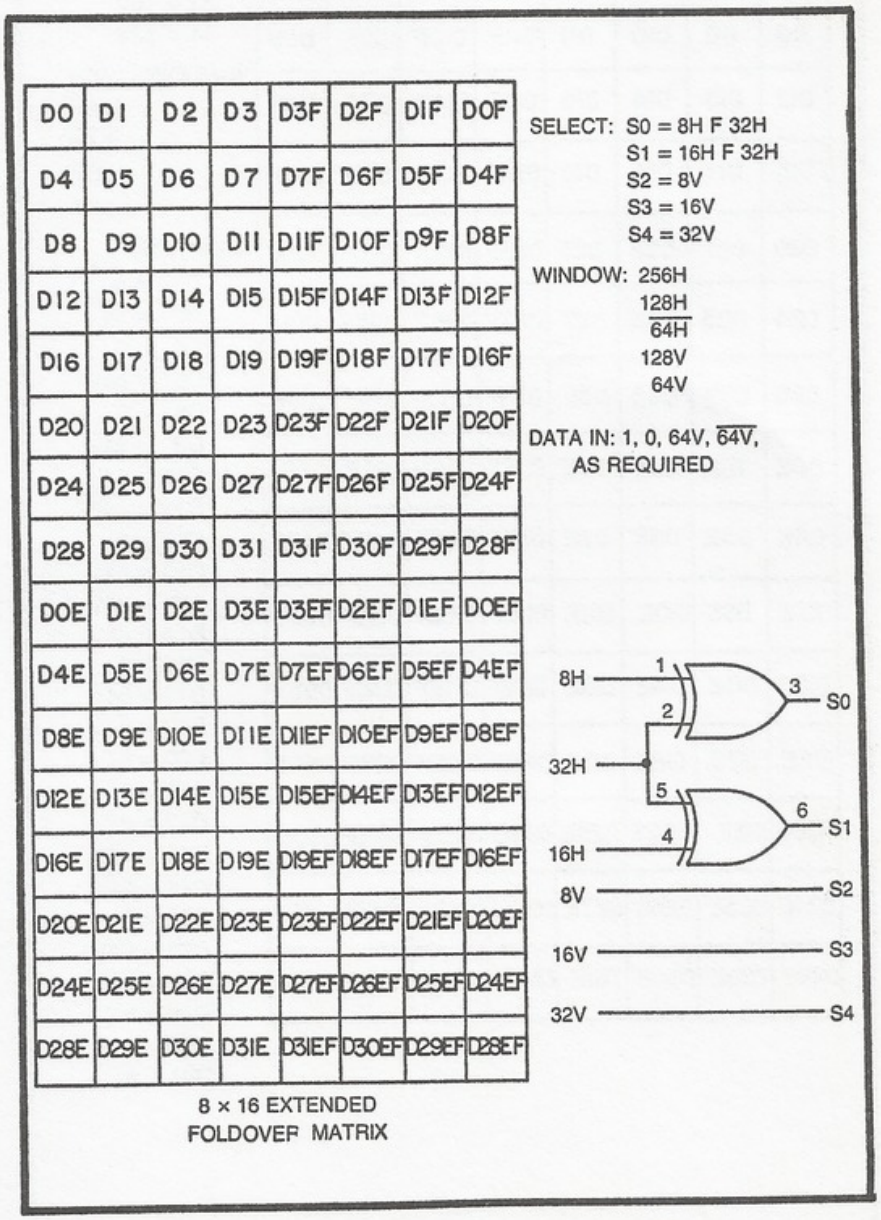


Fig. 4-23. Configuration, circuit, and sample specifications for an 8 × 16 extended foldover matrix.

D0	D1	D2	D3	D3F	D2F	D1F	D0F
D4	D5	D6	D7	D7F	D6F	D5F	D4F
D8	D9	D10	D11	D11F	D10F	D9F	D8F
D12	D13	D14	D15	D15F	D14F	D13F	D12F
D16	D17	D18	D19	D19F	D18F	D17F	D16F
D20	D21	D22	D23	D23F	D22F	D21F	D20F
D24	D25	D26	D27	D27F	D26F	D25F	D24F
D28	D29	D30	D31	D31F	D30F	D29F	D28F
D0E	D1E	D2E	D3E	D3EF	D2EF	D1EF	D0EF
D4E	D5E	D6E	D7E	D7EE	D6EF	D5EE	D4EF
D8E	D9E	D10E	D11E	D11EF	D10EF	D9EF	D8EF
D12E	D13E	D14E	D15E	D15EF	D14EF	D13EF	D12EF
D16E	D17E	D18E	D19E	D19EF	D18EF	D17EF	D16EF
D20E	D21E	D22E	D23E	D23EF	D22EF	D21EF	D20EF
D24E	D25E	D26E	D27E	D27EF	D26EF	D25EF	D24EF
D28E	D29E	D30E	D31E	D31EF	D30EF	D29EF	D28EF

SELECT.
S0 = 8H F 32H
S1 = 16H F 32H
S2 = 8V
S3 = 16V
S4 = 32V

WINDOW: 256H
128H
$\overline{64H}$
128V
64V

D INPUTS:
D1 = 0
D2 = $\overline{64V}$
D4 = 0
D5 = $\overline{64V}$
D8 = 0
D9 = 64V
D10 = 64V
D11 = $\overline{64V}$
D12 = 0
D13 = 64V
D14 = 64V
D15 = 64V
D16 = 0
D17 = $\overline{64V}$
D19 = $\overline{64V}$
D20 = $\overline{64V}$
D21 = 0
D22 = 64V
D23 = $\overline{64V}$
D24 = $\overline{64V}$
D25 = 0
D26 = 64V
D27 = $\overline{64V}$
D29 = 64V
D30 = 64V
D31 = $\overline{64V}$

Fig. 4-24. A cowboy figure built within an 8 × 16 extended foldover matrix.

symmetrical figures in the earlier section entitled Folding Over an Extended Matrix. The only difference here is that you are working with twice as many cells.

Figure 4-25 shows the 128-cell extended foldover matrix oriented horizontally. The axis of symmetry in this case is a vertical line through the center of the image. Set up this matrix, using the EXCLUSIVE-OR circuit and specifications shown in Fig. 4-25. Then work out some figures of your own, bearing in mind that the right-hand half of the images must be mirror images of the left-hand half.

Further Experiments With the 32-, 64-, 128-Cell System

If you have been performing the experiments suggested in this chapter thus far, you most likely have the know-how and confidence necessary for generating other matrix formats. What's even more important is that you ought to be coming up with additional ideas you want to try, perhaps more-novel ideas than you have time to work on.

Suppose, for instance, you are thinking about putting more than one kind of complex figure on the screen. How do you go about it? Well, you certainly have to build two different figure-generating circuits, one for each figure you want to put on the screen. After that, you must effectively OR together their outputs before applying the signal to the GAME VID IN terminal of the Sourcebox unit.

Exactly how you should go about ORing together these signals depends on whether they emerge from the figure-generating systems as white on black or black on white. All of the multiplexers in this chapter generate inverted, black-on-white signals, but the inverter connected to the output of the multiplexer in Fig. 4-5 and the NAND gate at the outputs of the multiplexers in Fig. 4-17 reverse the image so that it is properly shown as a white matrix on a black background.

These "upright" signals can be ORed together (combined on the screen) by first running them to separate inputs of a common NOR IC. The output of that NOR gate can then be inverted to yield a composite video signal having the proper white-on-black format.

If you are confused about any procedure for combining complex figures on the screen, you ought to review the material in the section entitled COMBINING ANY NUMBER OF STATIC FIGURES ON THE SCREEN in Chapter 3. The procedures described there can be carried over to the circuits in this chapter.

MATRIX OPERATIONS FROM 64-CELL GENERATORS

The idea of expanding the cell-generating capacity by adding more multiplexer circuits can be extended indefinately. Each new

multiplexer provides 16 more basic cell locations. Most experimenters, however, begin questioning the feasibility of expanding the complex-figure generating system beyond a certain point.

Is the ability to create a large and highly detailed pattern worth the trouble of working with all the multiplexer hardware? Only you can answer that question. It depends on what you are trying to do and what it's worth to you in the long run.

This section deals with an especially useful 64-cell generator. It is built around four 16:1 multiplexer ICs, thus giving the experimenter 64 data-input programming terminals. As described earlier in this chapter, any basic multiplexer-type figure generator can be easily modified to double the number of matrix cells. In this particular case, the experimenter has access to 128 cells. Then the user might elect to use a foldover scheme, thereby extending the number of cells to 256.

It turns out that this 64-, 128-, 256-cell matrix system is adequate for generating some of the most popular figures found on commercial TV games: cowboys, baseball players, tanks, aircraft of all sorts, an endless variety, really. Just look at some of the patterns used as examples through the remainder of the section.

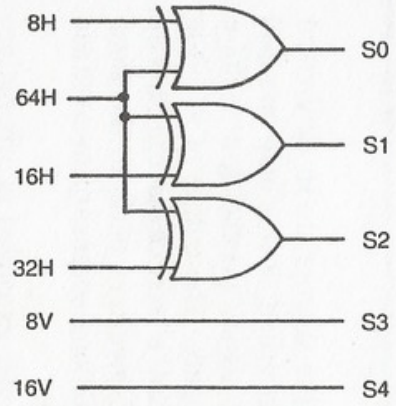
The 64-Cell Generator Circuit

The 64-cell generator circuit in Fig. 4-26 uses four 16:1 multiplexer ICs. The 4 lower-order select inputs to the system, S0 through S3, select one of the 16 data inputs on each multiplexer. And the demultiplexer circuit, IC9, is responsible for enabling one of the four multiplexers, depending on the status of the two higher-order select inputs, S4 and S5.

The system is windowed and the outputs of the multiplexers are NANDed in a fashion identical to the 32-cell generator in Fig. 4-17.

The purpose of the four inverters, IC1-A through IC1-D, require some special explanation. It is a fact of TTL technology that most ICs in that family are capable of driving up to 20 other TTL-type circuits. Normally a gate drives far less than 20 others, so the problem of overloading the output never becomes an important design factor.

When using the matrix-extension technique (expanding the cell count from 64 to 128 in Fig. 4-26) one of the H- or V-count outputs from the Sourcebox unit might have to drive 20 or 30 D inputs at the same time. Whenever a particular complex-pattern circuit calls for driving more than about 15 D inputs from the same source, that source ought to be buffered. And that's the purpose of IC1.



SELECT: S0 = 8H F 64H
S1 = 16H F 64H
S2 = 32H F 64H
S3 = 8V
S4 = 16V

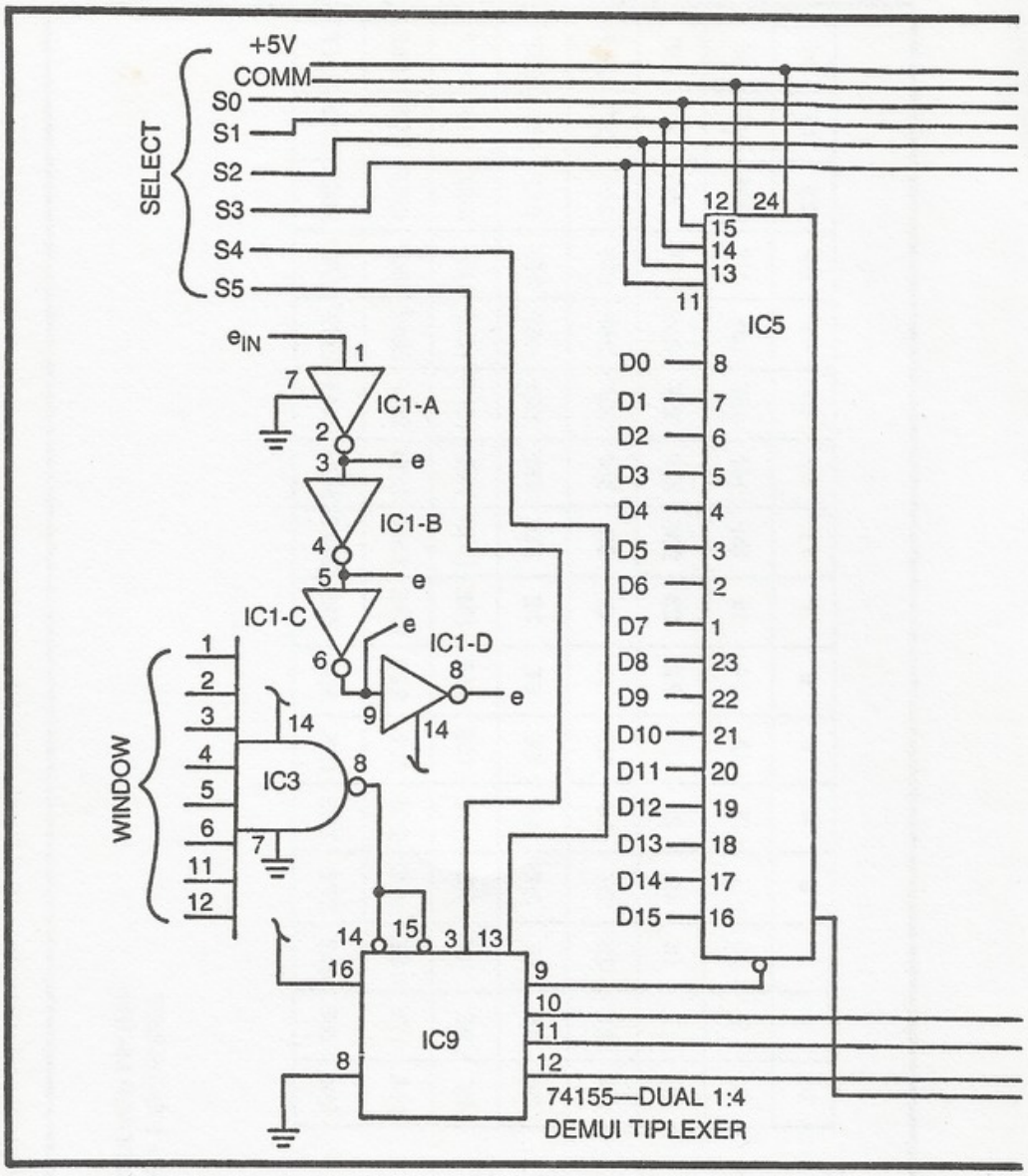
WINDOW: 256H
$\overline{128H}$
$\overline{128V}$
64V
32V

DATA IN: 0, 1, 36V, $\overline{32V}$, AS REQUIRED

16 × 8 EXTENDED FOLDOVER MATRIX

0	1	2	3	4	5	6	7	7F	6F	5F	4F	3F	2F	1F	0F
8	9	10	11	12	13	14	15	15F	14F	13F	12F	11F	10F	9F	8F
16	17	18	19	20	21	22	23	23F	22F	21F	20F	19F	18F	17F	16F
24	25	26	27	28	29	30	31	31F	30F	29F	28F	27F	26F	25F	24F
0E	1E	2E	3E	4E	5E	6E	7E	7EF	6EF	5EF	4EF	3EF	2EF	1EF	0EF
8E	9E	10E	11E	12E	13E	14E	15E	15EF	14EF	13EF	12EF	11EF	10EF	9EF	8EF
16E	17E	18E	14E	20E	21E	22E	23E	23EF	22EF	21EF	20EF	19EF	18EF	17EF	16EF
24E	25E	26E	27E	28E	29E	30E	31E	31EF	30EF	29EF	28EF	27EF	26EF	25EF	24EF

Fig. 4-25. Configuration, circuit, and sample specifications for a 16 × 8 extended fold over matrix.



The e_{in} connection comes from the appropriate H- or V-count source. As far as that source is concerned, it has only one load connected to it. IC1-A inverts that signal, producing a useful $\bar{e}$ source that can drive up to 20 D inputs. The output of IC1-A is inverted by IC1-B to yield a noninverted "e" source that can likewise drive up to 20 D inputs. The last 2 inverters perform the same function as the first 2, providing 20 additional sources of inverted and noninverted figure-expansion signals.

Suppose, for example, a particular expanded matrix pattern calls for 25 32H inputs and 22 $\overline{32H}$ signals. The 32H signal from the

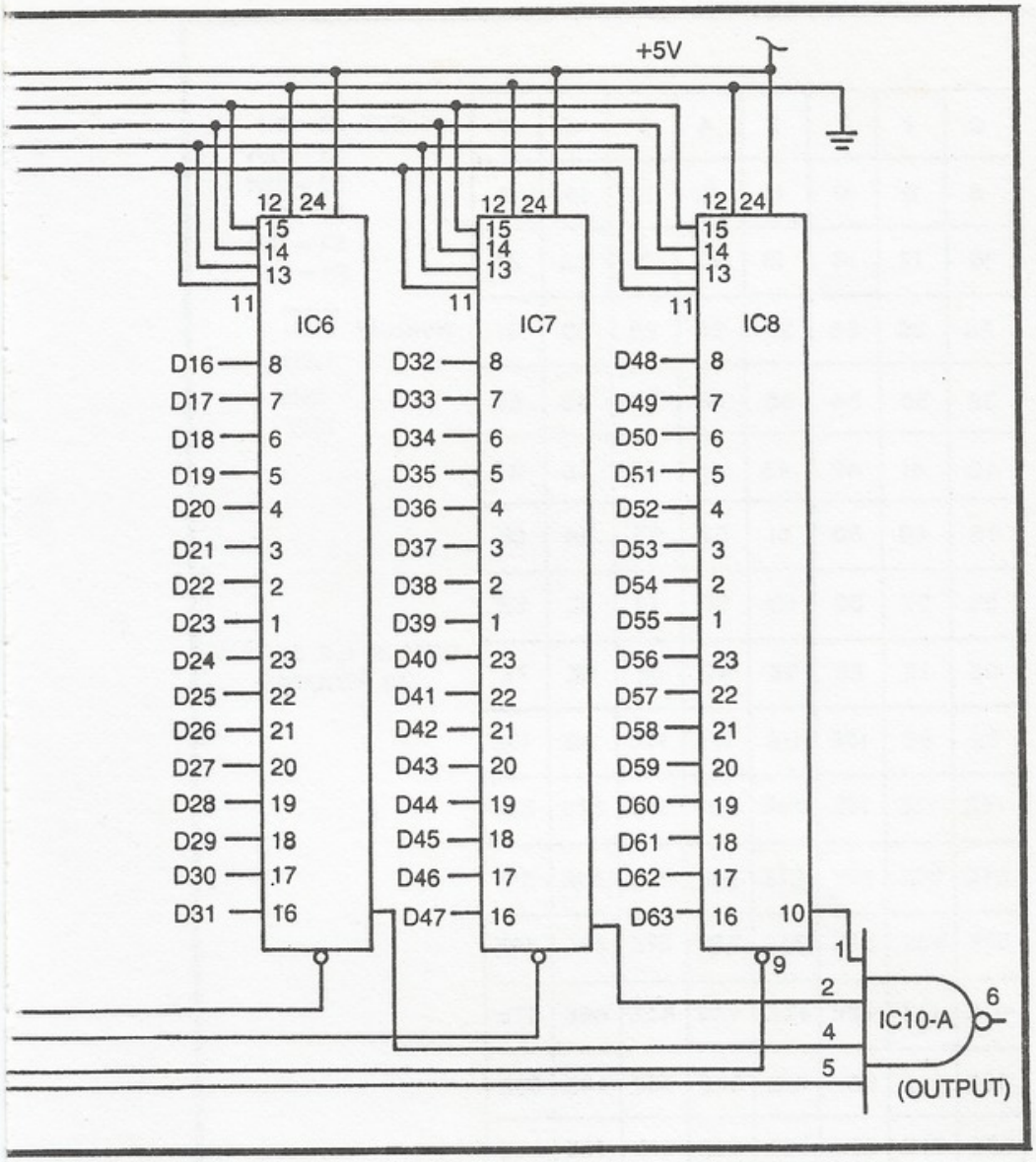


Fig. 4-26. Circuit for a basic 64-cell matrix generator.

Sourcebox unit can be connected to the e_{in} terminal. Then 11 of the 22 $\overline{32H}$ signals can be tapped off the output of IC1-A, 12 or 13 of the 32H signals can be tapped from the output of IC1-B, and the remainder of the $\overline{32H}$ and 32H signals can be taken from the outputs of IC1-C and IC1-D respectively.

Some 8x16 Expanded Matrices

Figure 4-27 shows a vertically oriented, 128-cell matrix that can be easily generated by the circuit in Fig. 4-26. If you build this circuit and use the SELECT and WINDOW data prescribed here, you will

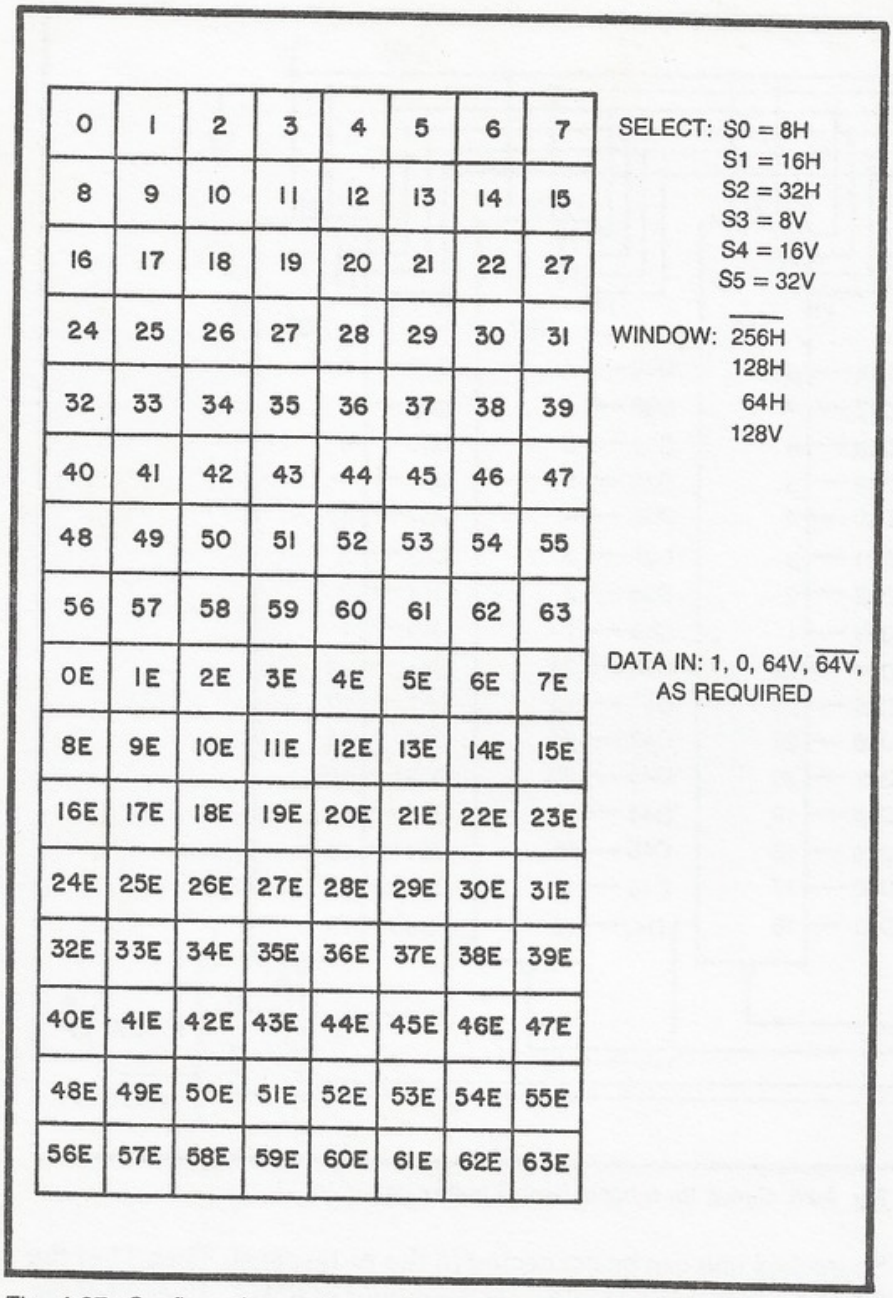


Fig. 4-27. Configuration and sample specifications for an 8 × 16 extended matrix.

find a rather large white rectangle occupying the lower right-hand quadrant of the screen. Connecting any of the D inputs to logic 0 creates a black cell in the corresponding matrix position and in its extended location as well.

The two examples in Fig. 4-28 include a rather novel footprint and the profile of a cowboy. While the footprint might not be very

useful for building a TV game, it certainly illustrates some of the fun you can have creating video graphics with this system.

Figure 4-24 illustrated a way to build the image of a cowboy. The circuit was much simpler in that case, but it was using a foldover technique that demanded a symmetrical pattern. The profile in Fig. 4-28b, however, is a nonsymmetrical version of the same hombre.

It is possible to spend many, many hours playing with this matrix-generating scheme. If you want to make the figures smaller, simply scale down the SELECT, WINDOW, and D INPUT specifications one or two orders of magnitude. Cutting each of the specifications in half, for instance, cuts the size of the image in half.

Play with this system as long as you want. Just because you are having fun with it doesn't mean you aren't learning anything. Keep track of your experiments, noting both your failures and successes. That information will prove invaluable later on.

The extended matrix can be adjusted to form a horizontally oriented, 16×8 pattern. See the suggested specifications and pattern in Fig. 4-29.

The display in Fig. 4-30 merely shows one example of how the 16×8 extended matrix can be used. The image in this instance is a battleship or destroyer. Of course it can be modified slightly to transform it into a submarine or aircraft carrier.

This matrix is good for any sort of relatively complex, horizontal, nonsymmetrical figure.

A 256-Cell, Extended Foldover Matrix

The foldover scheme applied to the 128-cell matrix just described yields a 256-cell matrix. The example in Fig. 4-31 shows the rather simple additional circuitry and the left-hand half of an ever-popular tank image.

When you build the circuit and wire it to the specifications shown in Fig. 4-31, you will find a full tank image, with its right-hand half being a mirror image of the portion shown here. It is often advisable to show only the nonfolded half of such patterns, but for no reason other than drawing pictures with 256 cells becomes rather tedious. Besides, all the necessary information is contained in the original portion.

MULTIPLYING THE NUMBER OF IDENTICAL IMAGES ON THE SCREEN

While a single complex figure can play a vital role in most kinds of TV games, it is often desirable to display a number of identical complex figures on the screen. Take for example the mines for a minefield game or cars in an auto racing game.

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63
0E	1E	2E	3E	4E	5E	6E	7E
8E	9E	10E	11E	12E	13E	14E	15E
16E	17E	18E	19E	20E	21E	22E	23E
24E	25E	26E	27E	28E	29E	30E	31E
32E	33E	34E	35E	36E	37E	38E	39E
40E	41E	42E	43E	44E	45E	46E	47E
48E	49E	50E	51E	52E	53E	54E	55E
56E	57E	58E	59E	60E	61E	62E	63E

SELECT: WINDOW:

S0 = 8H $\overline{256H}$
S1 = 16H 128H
S2 = 32H 64H
S3 = 8V 128V
S4 = 16V
S5 = 32V

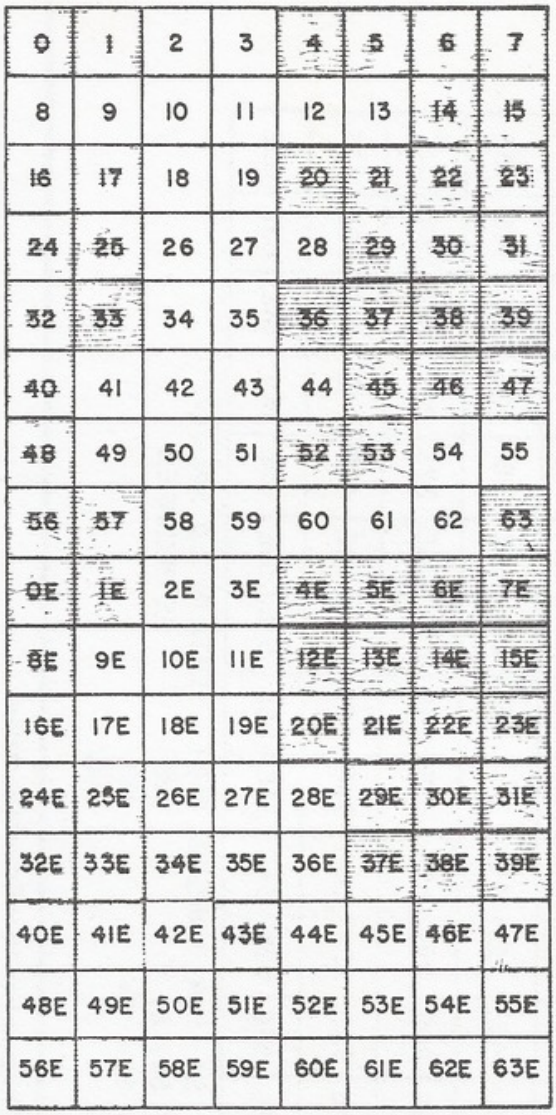
D INPUTS:

D0 = 0	D33 = 64V
D1 = 0	D34 = 0
D2 = 64V	D35 = $\overline{64V}$
D3 = 64V	D37 = 64V
D4 = 64V	D38 = $\overline{64V}$
D5 = $\overline{64V}$	D39 = 0
D6 = $\overline{64V}$	D40 = 0
D7 = 0	D41 = 0
D8 = 0	D42 = 0
D9 = 0	D43 = $\overline{64V}$
D10 = 64V	D44 = $\overline{64V}$
D11 = 64V	D45 = $\overline{64V}$
D13 = $\overline{64V}$	D46 = $\overline{64V}$
D14 = $\overline{64V}$	D47 = 0
D15 = 0	D48 = 0
D16 = 0	D49 = $\overline{64V}$
D17 = 0	D50 = $\overline{64V}$
D18 = 64V	D51 = $\overline{64V}$
D20 = 64V	D52 = $\overline{64V}$
D21 = 0	D53 = 64V
D22 = 0	D54 = 0
D23 = 0	D55 = 0
D24 = 0	D56 = 0
D25 = 0	D57 = $\overline{64V}$
D27 = 64V	D58 = $\overline{64V}$
D28 = $\overline{64V}$	D59 = $\overline{64V}$
D29 = $\overline{64V}$	D60 = $\overline{64V}$
D30 = $\overline{64V}$	D61 = $\overline{64V}$
D31 = 0	D62 = 0
D32 = 0	D63 = 0

A

The procedure for generating a single complex figure can require quite a number of IC devices as clearly demonstrated throughout this chapter. There is no need, however, to build one complete matrix generator circuit for each figure that is to appear on the screen—at least not as long as the figures are all identical.

It turns out that one complex-figure generator is sufficient for creating any number of identical images on the screen. The general



SELECT: WINDOW:

S0 = 2H — $\overline{256H}$
S1 = 4H — 128H
S2 = 8H — 64H
S3 = 2V — 32H
S4 = 4V — 128V
S5 = 8V — 64V
32V

D INPUTS:

D0 = 0
D1 = 0
D4 = 0
D5 = 0
D6 = 0
D7 = 0
D8 = $\overline{16V}$
D12 = $\overline{16V}$
D13 = $\overline{16V}$
D14 = 0
D15 = 0
D16 = 0
D17 = 16V
D20 = 0
D21 = 0
D22 = 0
D23 = 0
D24 = 0
D25 = 0
D29 = 0
D30 = 0
D31 = 0
D32 = 0
D33 = 0
D34 = $\overline{16V}$
D36 = 16V
D37 = 0

D38 = 0
D39 = 0
D40 = 0
D41 = $\overline{16V}$
D43 = $\overline{16V}$
D45 = 16V
D46 = 0
D47 = 16V
D48 = 0
D51 = $\overline{16V}$
D52 = 0
D53 = 16V
D55 = $\overline{16V}$
D56 = 0
D57 = 0
D60 = $\overline{16V}$
D62 = $\overline{16V}$
D63 = 0

B

Fig. 4-28. Sample complex figures in the 8 × 16 extended matrix. (a) Left footprint. Folding this figure by means of 64H creates a pair of footprints—left and right. See circuit in Fig. 4-31. (b) Profile of a cowboy.

idea is to first build the circuit for generating the desired figure, using the techniques already described in this chapter. Then some rather simple logic circuits can be added to the window inputs to create any number of the figures on the screen and in any desired pattern.

To get a first-hand, practical view of this procedure, build up any of the simpler complex-figure circuits and display the image on

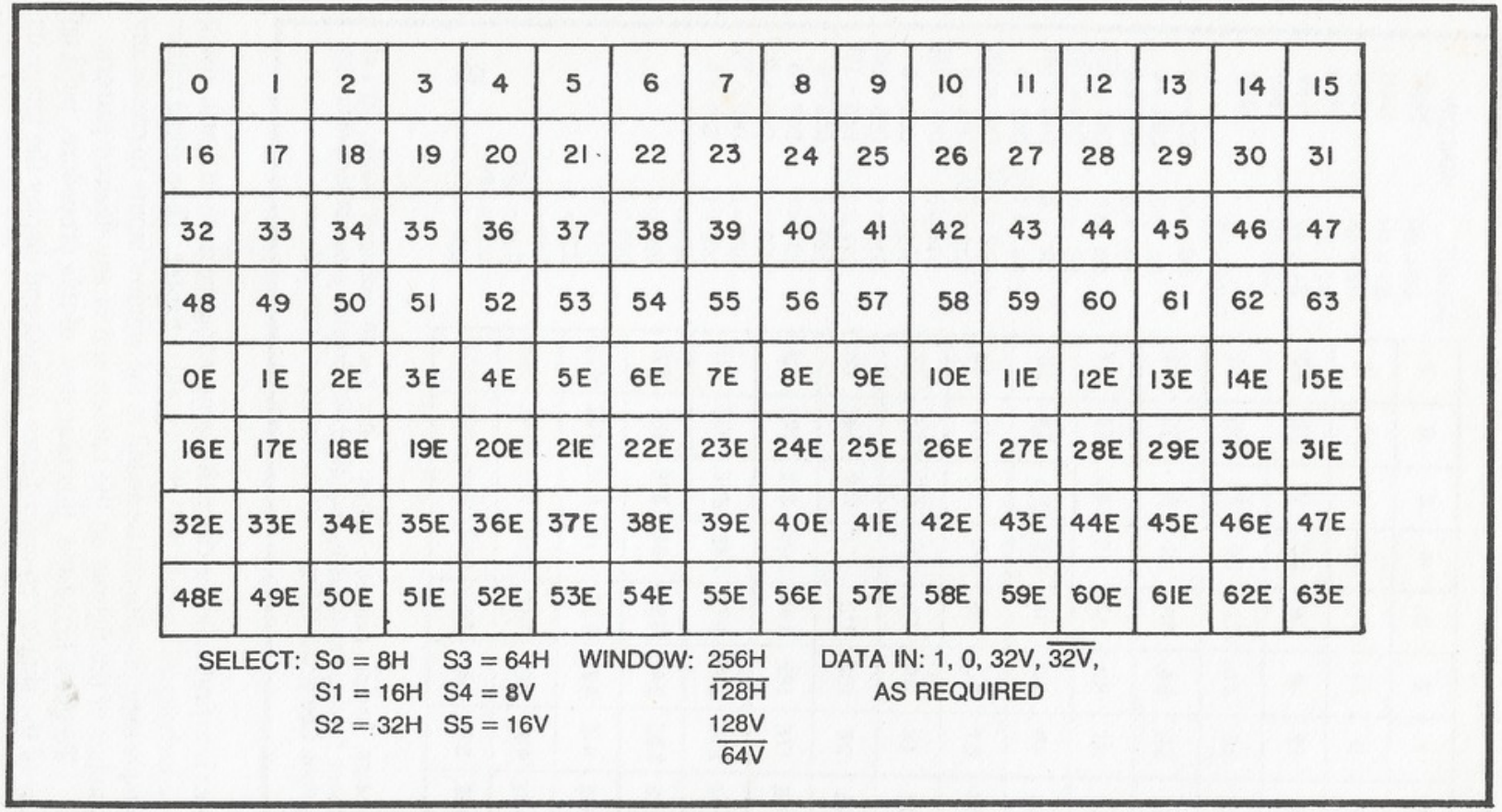


Fig. 4-29. Configuration and sample specifications for a 16 × 8 extended matrix.

the screen *without* any windowing. You will find that your figure is repeated a number of times in closely spaced rows and columns.

The pattern of images on the screen might be rather interesting without the benefit of windowing, but it is hardly useful in the context of a TV game. What remains to be done is selectively window some of those images, leaving some in place and eliminating the others.

As is so often the case in this TV-game business, there are several different approaches to selecting the figures that are to appear on the screen. Study all the approaches presented here, doing as much hands-on experimenting as possible.

Bar and Rectangle Windowing

Figure 4-32 shows in a block diagram fashion how the image-windowing circuitry is interfaced with a complex-figure matrix generator. The 8-input NAND gate normally shown as the window input for the complex-figure generators in this circuit is simply replaced with a different sort of windowing circuit.

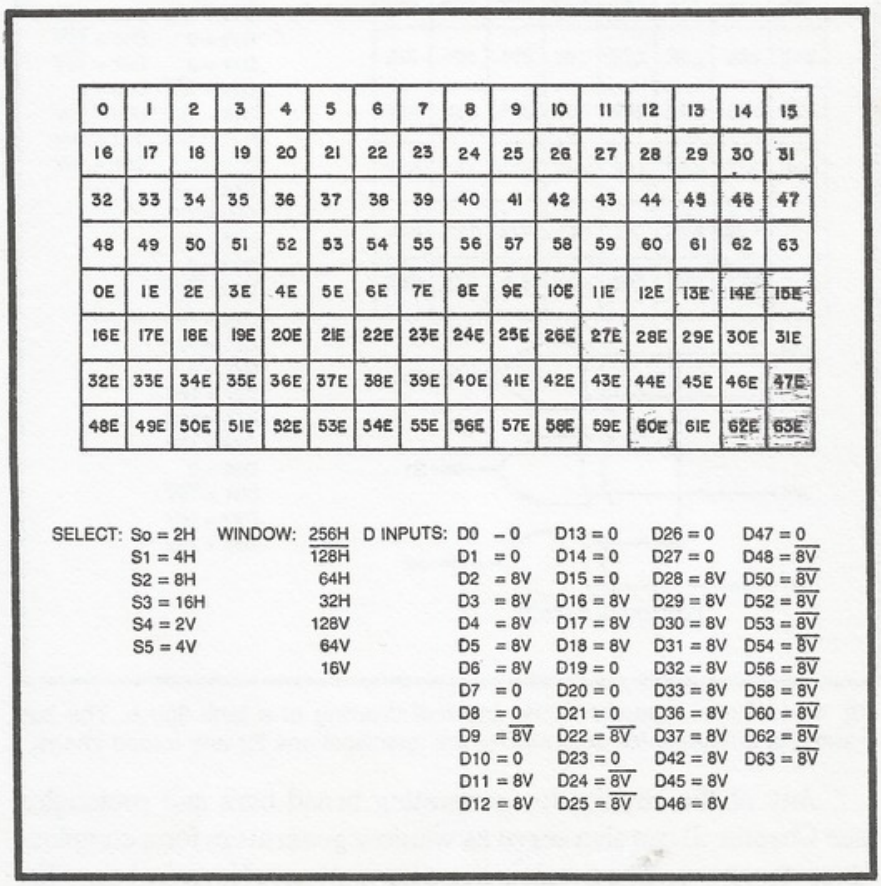


Fig. 4-30. Figure of a battleship built in a 16 × 8 extended matrix.

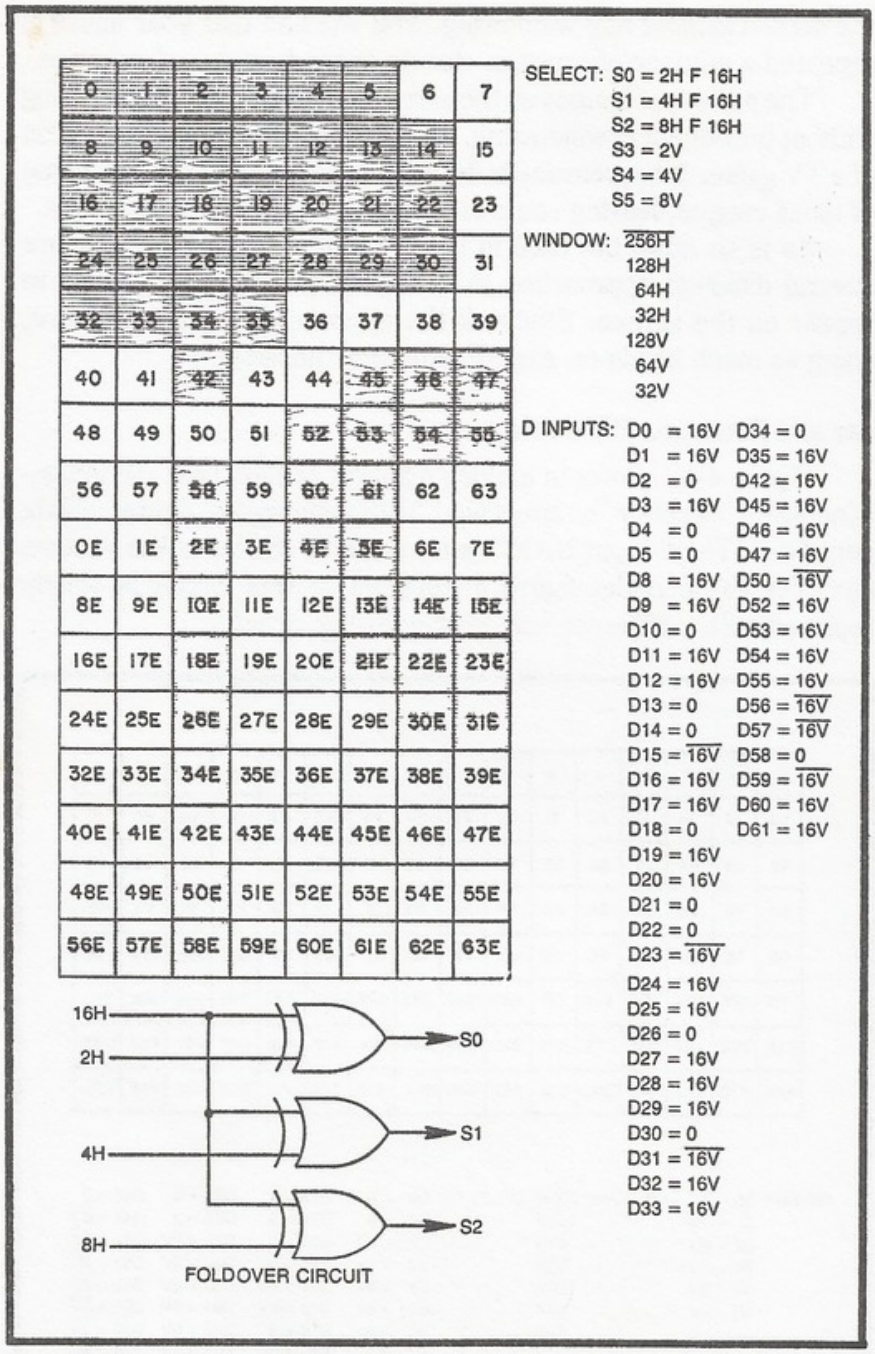


Fig. 4-31. Circuit, specifications, and half drawing of a tank figure. The half drawing is adequate for determining the specifications for any folded image.

Any of the circuits for generating broad bars and rectangles (see Chapter 3) can also serve as window generators for a complex-figure generator. The result is a regular pattern of identical complex figures on the screen. Use any of the bar or rectangle generators

described in Chapter 3, but make certain they deliver a black-on-white signal to the $\overline{CE}$ input of your multiplexer in the complex-figure generator.

Let's look at that last statement a little closer. The multiplexer (complex-figure generator) is windowed ON whenever its $\overline{CE}$ input is pulled down to logic 0. This principle has been used in all of the examples thus far in this chapter. The multiplexer is thus enabled whenever the windowing circuit generates a logic 0. Or in other words, the complex figure will appear on the screen any time the windowing circuit shows a black bar or rectangle.

Of course the windowing black bar or rectangle must have a size equal to or greater than that of the figure it is controlling. Making the windows too small cuts off part of the figure, while making the windows too large allows two or more of the figures to appear hooked together. (The latter condition might be something of an advantage in certain instances, however).

You ought to be anxious to experiment with this idea now; so start by building the matrix generator circuit in Fig. 4-5, making *no*

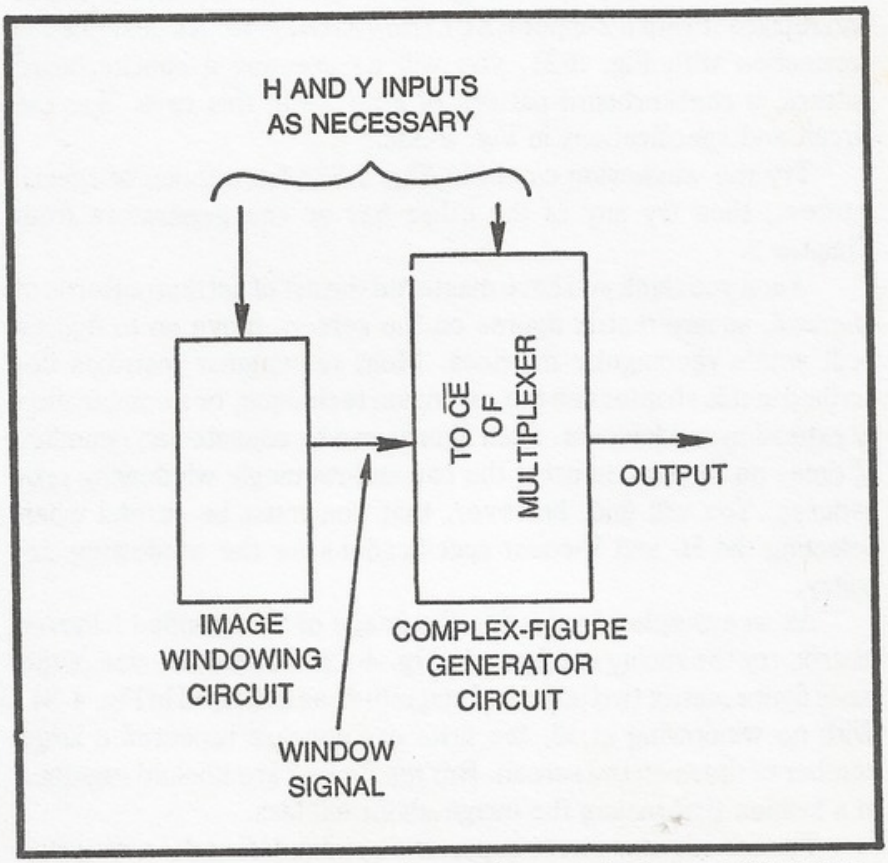


Fig. 4-32. Block diagram of circuitry for repeating complex figures on the screen.

connections to the window inputs for the time being. If you use the select and D-input specifications in Fig. 4-33a, you will find the screen filled with rows and columns of hooked-together Xs. (It gives the visual impression of a small checkerboard pattern, however).

If you are having trouble visualizing the Xs, window down to a single figure with 256H, 128H, 64H, 32H, 128V, 64V, and 32V. Now you should see a single X figure as shown in Fig. 4-33a. Remove all these window inputs before going on to the next step of this experiment.

Now the idea is to selectively eliminate some of the Xs, leaving behind a distinct pattern of them. First try applying 32H and 32V to the windowing NAND gate. You should find that half the Xs are eliminated, getting rid of the confusing, hooked-together feature. What is left is a regular pattern of horizontal and vertical X lines.

Including 64H and 64V with the 32H and 32V sources already connected to the window inputs increases the spacing between the rows and columns of Xs. See this particular set of window specifications in Fig. 4-33b.

Now remove the windowing NAND gate, IC3, from the circuit and replace it with a 2-input EXCLUSIVE-OR gate. As described in connection with Fig. 3-21, you will be creating a checkerboard pattern, a checkerboard pattern of little Xs in this case. See the circuit and specifications in Fig. 4-33c.

Try the windowing circuit in Fig. 4-33d for a touch of special interest, then try any of the other bar or line generators from Chapter 3.

When you think you have mastered the art of setting patterns of identical, square matrix figures on the screen, move on to figures built within rectangular matrices. Most rectangular matrices described in this chapter use the extension technique, or a combination of extension and foldover. Such figures can be repeated any number of times on the screen using the bar-and-rectangle windowing procedures. You will find, however, that you must be careful when selecting the H- and V-count specifications for the windowing circuitry.

As an example of repeating the image of an extended foldover matrix, try the racing car figure in Fig. 4-12. Reduce the size of the basic figure matrix two orders of magnitude as indicated in Fig. 4-34. With no windowing at all, the little car appears repeated a large number of times on the screen. But the figures are hooked together in a fashion that makes the image about useless.

The little figures can be separated by eliminating alternate rows and columns of them. The circuit in Fig. 4-34 shows the specifica-

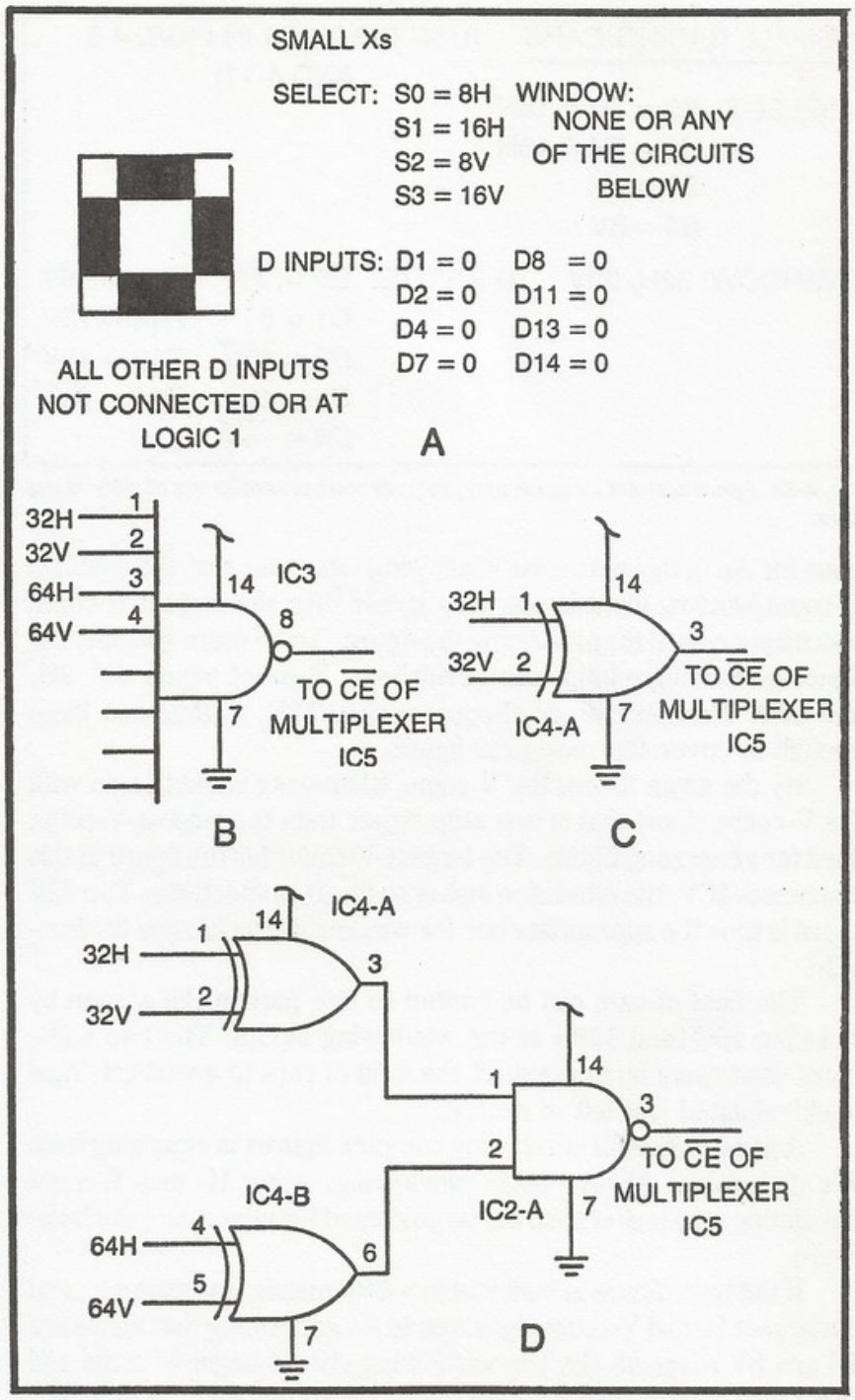


Fig. 4-33. Initial experiments with the line/rectangle pattern-repeating technique. (a) Matrix configuration and sample specifications for an X figure. (b) Windowing circuit for creating rows and columns of figures. (c) Circuit for generating checkerboard patterns of figures. (d) Circuit for generating an interesting pattern of figures.

SMALL RACING CARS	(USE CIRCUITS IN FIGS. 4-5 AND 4-11)		
SELECT: S0 = 4H F 16H			
S1 = 8H F 16H			
S2 = 4V			
S3 = 8V			
WINDOW: 32H, 32V	D INPUTS:	D0 = $\overline{16V}$	D9 = 16V
		D1 = 0	D12 = 16V
		D2 = $\overline{16V}$	D13 = 16V
		D3 = 16V	D14 = $\overline{16V}$
		D6 = $\overline{16V}$	

Fig. 4-34. Specifications for generating regular rows and columns of little racing cars.

tions for doing this particular windowing job. Note that the smallest H-count window input is one step larger than the largest H-count specification used for generating the figure. To be more specific, the figure-generating multiplexer circuit uses H-count inputs 4H, 8H, and 16H. The next larger H-count signal, 32H, is thus just large enough to cover the racing car figure.

By the same token, the V-count windowing should begin with the V-count signal that is one step larger than the largest V-count used for generating figure. The largest V-count for the figure in this instance is 16V, the extension inputs to the D connections. The 32V signal is thus the appropriate one for windowing the images horizontally.

The field of cars can be limited to one part of the screen by including $\overline{256H}$ and 128H at the windowing inputs. The two additional windowing inputs restrict the field of cars to a vertical "race track" situated just left of center.

A general rule for windowing complex figures is emerging from this discussion. Always begin windowing, using H- and V-count signals *one* step higher than the largest used for generating the basic figure.

If the basic figure is built within a 4×8 matrix, for example, and the largest H- and V-count signals used for generating that figure are 4H and 8V respectively, the windowing should begin with 8H and 16V.

Windowing for Irregular Patterns of Identical Figures

While the notion of windowing a basic complex figure to get a particular pattern of rows and columns serves some useful purposes

for TV games, many other games call for patterns of identical figures that are not in regular rows and columns. It is often more desirable to create irregular, or even random, patterns of identical figures, something that cannot be done with the bar-and-rectangle windowing approach.

It is possible to generate a complex pattern of complex, but identical, figures by windowing a figure-generating multiplexer circuit with yet another multiplexer circuit. In other words, the window generator is, itself, a complex figure generator.

Figure 4-35 illustrates this particular approach to generating some complex patterns. IC5 and IC6 are both 16:1 multiplexer circuits. IC6 generates the basic complex figure as described in the first sections of this chapter. The inputs to S0 through S3 determine the dimensions of the complex figure, while information at the D inputs determine what the figure will be.

IC5 is a similar kind of circuit that generates a matrix of its own. The dimensions of this pattern matrix are determined by H- and V-count signals applied to pattern-select inputs WS0 through WS3, while the pattern, itself, is determined by the data applied to the field pattern data inputs W0 through W15.

Electrically speaking, the two multiplexer circuits are virtually identical. They play two entirely different roles in the pattern-generating process, however. IC6 generates the basic figure, while IC5 determines where and how many identical figures appear on the screen.

To see how this scheme works, program IC6 to generate the simple X pattern in Fig. 4-33a. As long as there are no connections to the inputs of IC5 and the pattern window NAND gate, you will see the basic X pattern repeated all over the screen.

Now connect the pattern select inputs to IC5 as specified in Fig. 4-35: WS0 = 32H, WS1 = 64H, WS2 = 32V, and WS3 = 64V. There will be no obvious change in the pattern on the screen, however, until one or more of the W inputs to IC5 is connected to logic 0. Every time you connect one of those W inputs to logic 0, you will find some of the figures eliminated from the screen.

IC5, itself, is serving the function of a complex figure generator. The "figure" in this instance is the desired pattern of cars on the screen. A logic 1, or no connection, to a W input allows the basic figure to appear at the corresponding matrix location. Setting a W input to logic 0, on the other hand, eliminates a figure at the matrix location. Using inverted or noninverted versions of 128H creates an extended window matrix. Figure 4-36 shows the specifications for

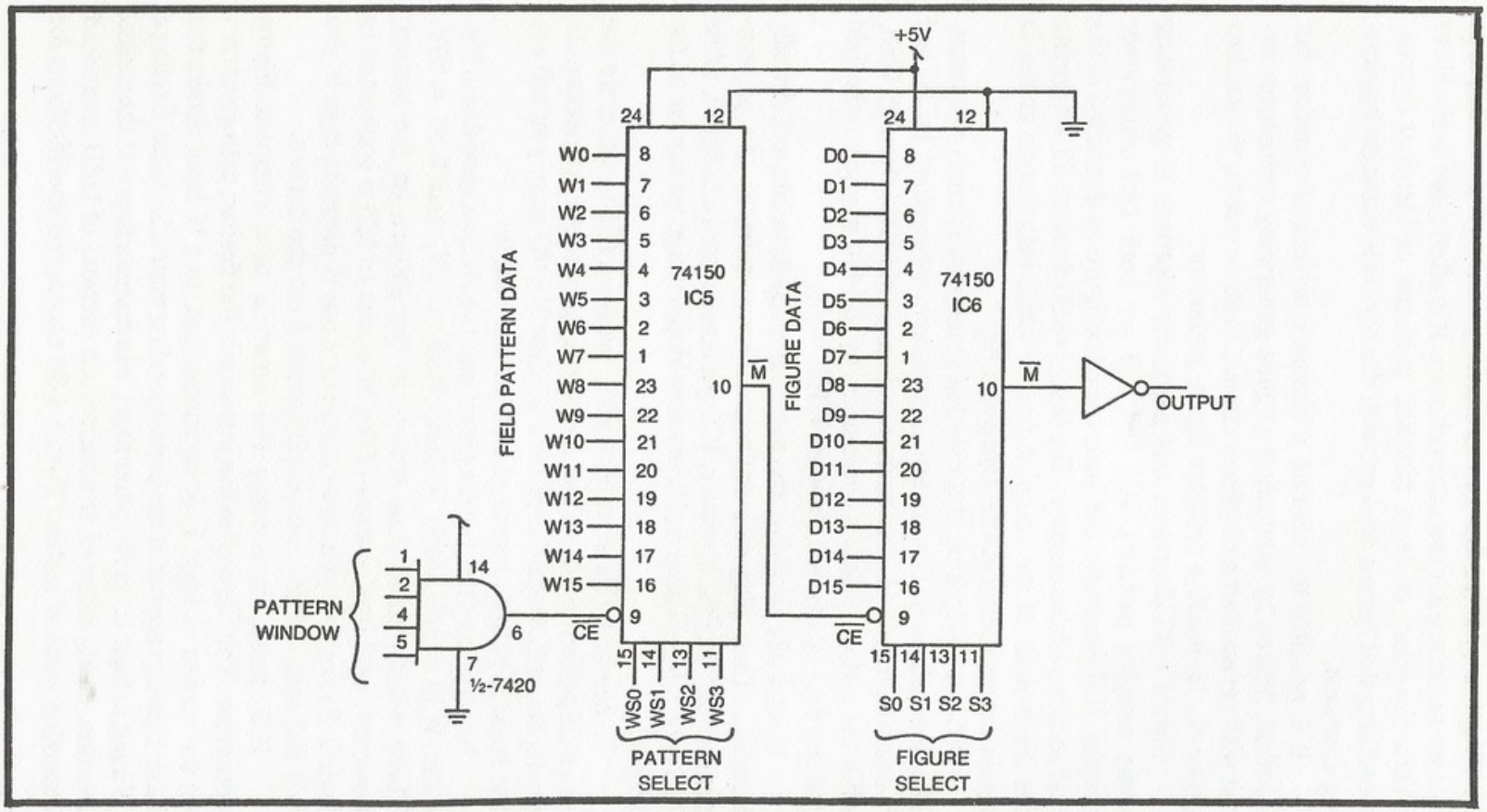


Fig. 4-35. Basic circuit for arranging multiple images into any complex pattern.

generating a large X figure that is, itself, made up of smaller X figures. Using the circuit in Fig. 4-35, IC6 generates the little Xs, and IC5 determines the big X pattern. The designated pattern window inputs restrict the pattern to a single large X on the screen.

If you understand how the basic complex-figure generators work, you can apply that knowledge to the complex-pattern generator. Any of the complex-pattern generators described in the first part of this chapter can be used as window generators. It is possible, if not altogether practical, to create a 256-cell figure matrix using the circuit in Fig. 4-26, and then repeat that figure anywhere within another 256-cell windowing matrix. The result would be a full-screen, complex pattern of identical complex figures.

SMALL X FIGURE (USE CIRCUIT IN FIG. 4-35)

SELECT: S0 = 8H WINDOW: FROM IC5
S1 = 16H
S2 = 8V
S3 = 16V

D INPUTS: D1 = 0 D8 = 0
D2 = 0 D11 = 0
D4 = 0 D13 = 0
D7 = 0 D14 = 0

LARGE X PATTERN

SELECT: WS0 = 32H WINDOW: $\overline{256H}$
WS1 = 64H $\overline{128H}$
WS2 = 32V $\overline{128V}$
WS3 = 64V

W INPUTS: W1 = 0 W8 = 0
W2 = 0 W11 = 0
W4 = 0 W13 = 0
W7 = 0 W14 = 0

Fig. 4-36. Example of arranging a complex figure into a complex pattern, a large X built from smaller Xs.

You must play with this scheme for a while to get a real understanding of how it works. Keep it simple at first, starting with the circuit in Fig. 4-35, then move on to one that uses two multiplexers for generating the basic figure and one for determining the repeated pattern.

It is important to master this technique now. Later chapters dealing with complete games, and the motion of figures assume an understanding of the complex-figure and pattern-generating schemes.