Chapter 8

Programmable Position and Motion Controls

It is possible to realize some overall savings of time and money by using programmable figure-generating and motion-control circuits. The initial investment is larger because programmable circuits are generally more complicated than those designed for specific game applications. Being able to use the same circuitry for a number of different video game formats, however, soon lets the circuit pay for itself a number of times over.

The circuits described here are close cousins of the fully programmable TV game computer systems on the market today. The experimenter isn't bound to a set of fixed game formats; yet, the scheme goes a long way toward simplifying game design procedures and reducing the amount of new hardware for each game. It also turns out that these little programmable circuits can do some things that are terribly difficult to do with the motion-control circuits described thus far.

The digital device at the heart of this programmable-figure scheme is the 7485 4-bit magnitude comparator shown in Fig. 8-1. Basically, the circuit accepts two 4-bit binary words or numbers, compares their magnitudes, and generates an output specifying whether one is equal, greater or less than the other.

One of the two input numbers is designated number A, and is composed of bits A0 through A3, with A0 being the least-significant bit. The second input number is designated number B, and is composed of bits B0 through B3, with B3, with B0 being the least-significant bit.

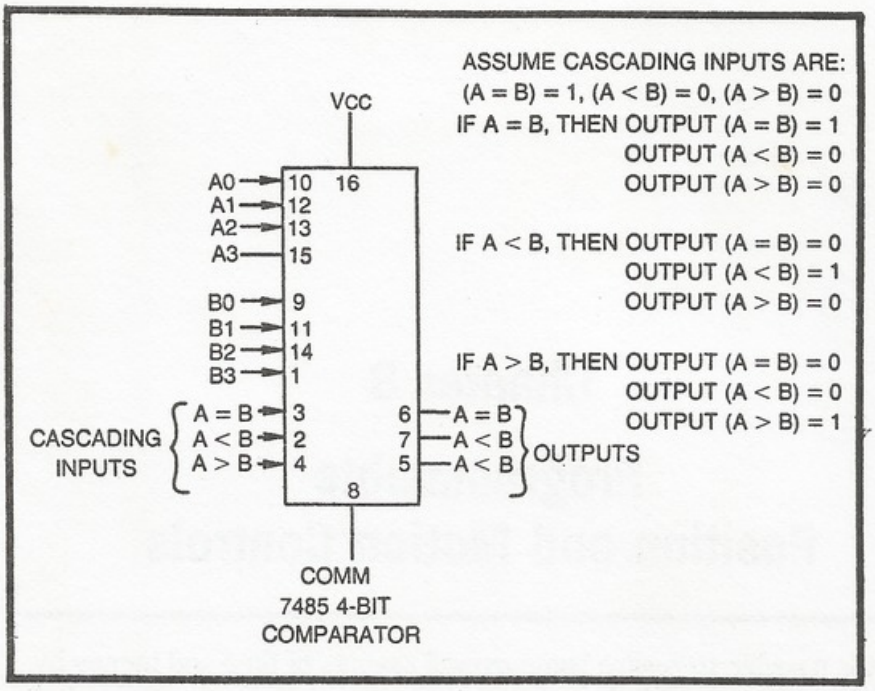


Fig. 8-1. Pinout and operating features of the 7485 4-Bit magnitude comparator.

When these two 4-bit numbers are applied to their respective inputs of the comparator, one of the three outputs switches to a logic-1 level. If the two numbers are exactly equal, output A=B goes to logic 1. If number A happens to be greater than B (A=1001 and B=0101, for example), output A > B goes to logic 1. And finally, if input A is less than B, output A < B goes to logic 1.

The cascading inputs are used only when the comparator IC is being used with an identical unit to compare words having eight or more bits. Otherwise, cascading input A=B should be connected to logic 1, and the inequality cascading inputs should both be grounded. Circuits in the following sections of this chapter illustrate all these operating modes.

A PROGRAMMABLE FIGURE POSITION CONTROL

Chapter 3 deals with the basic circuitry for generating lines, bars, and rectangles on the screen. The logic-circuit designs in those instances determine both the size and position of the figures. Such figures can be placed anywhere on the screen by using the appropriate set of H- and V-count inputs from the Sourcebox, but once they are fixed, it is difficult to change them on a finished circuit board.

The circuit described here is generally more complicated than any in Chapter 3, but it is rather easy to change the parameters

determining the size and position of the figure. In a manner of speaking, it is a programmable-figure size and position control.

The circuit in Fig. 8-2 shows a complete figure-programming circuit for either the horizontal or vertical parameters for fixing the position and size of a line/bar figure on the screen.

The circuit has two sets of inputs. The inputs labeled 1, 2, 4..., 256 go to their respective connections from the H- or V-count terminals of the Sourcebox unit. If the circuit is being used to generate horizontal parameters, for example, input 1 goes to 1H, input 2 goes to 2H, and so on.

A second set of nine inputs labeled 1P through 256P generally go to fixed 1 or 0 logic levels.

These inputs are continuously compared in IC1, IC2, and IC4-A. IC1 compares the four lower-order bits, and when they are equal, it delivers an A=B logic level to IC2 where the 16, 32, 64, and 128 bits are compared. If the first eight pairs of inputs are equal, IC2 then generates an A=B output from its pin-6 connection.

IC1 and IC2 are 4-bit magnitude comparators that are cascaded to perform 8-bit comparison. The two IC's compare the eight lower-order H- or V-count signals with the eight corresponding logic levels at the P inputs. IC4-A, in conjunction with an inverter function at IC4-B, make up a 1-bit magnitude comparator for the 256 bit. If the 256 bit from the Sourcebox is the same as the 1 or 0 logic level at the 256P input, the output of IC4-B goes to logic 1.

IC3-A thus sees two comparison signals. If all eight of the lower-order bits are equal and the 265-bit inputs are equal, IC3-A generates a logic-0 level as long as that condition exists.

Putting this information all together, the circuit in Fig. 8-2a works as a 9-bit magnitude comparator, generating a logic-0 output only when the signals from the Sourcebox have logic levels that are identical to those set at the nine P inputs.

Use the circuit in Fig. 8-2a as the basis of an experiment with the magnitude comparator scheme. Connect the Sourcebox inputs to the H-count signals and make provisions for either grounding or connecting the P inputs to +5V. If the PP output is connected directly to the GAME VID IN connection on the Sourcebox, you will be working with a fine, black horizontal line on a white background. Running PP through another inverter before applying the signal to Sourcebox will generate a white line on a black field.

In any event, you will find you are working with a vertical line that is 1H wide. To get the experiment started, connect the P inputs to the following pattern of 1s (+5V or no connection at all) and 0s (ground connection): 100001010, where the bit on the left is the

256P input and the one on the right is the 1P bit. You should find a 1H line running down the center of the screen.

What is happening here? By programming the P inputs to 100001010, you are asking the circuit to look for that particular pattern of 1s and 0s from the H-count outputs of the Sourcebox. And since that particular H-count occurs at the center of the screen, it follows that the comparator circuit generates its output at that particular moment. See the master counting table in Chapter 2 for other program patterns.

There are two conditions that will not show a line on the screen. If the program inputs specify an H-count in the horizontal blanking region, the comparator generates a line figure, but it is lost in the blanking region. In the other case, you can program counts that are larger than the number of H-counts in a line—larger than binary 111000110 or decimal 454. In this instance, you are asking the circuit to look for a number that Sourcebox never generates, and as a result, the line is never generated at all.

The same basic ideas apply when using this comparator scheme with V-count inputs. Here the circuit generates a horizontal line that is 1V wide and in a position determined by the pattern of logic levels at the P inputs. As in the case of the horizontal-comparison experiment, programs calling for a line in the vertical-blanking region or any calling for a line at 100000101 (decimal 261) or more do not generate visible lines.

So if the circuit in Fig. 8-2a is wired for H-count programming, you can fix the position of a 1H vertical line anywhere on the screen between 000000000 (extreme left side) and 11100101 (extreme right side). The pattern of 0s and 1s at the P inputs correspond exactly to the inputs to a NAND gate as specified in Chapter 3. In essence, this comparator circuit works as a 9-input NAND gate for fixing the size and position of a line on the screen. The only difference here is that the programming can be changed much easier than for a hard-wired NAND gate.

Once you have built this comparator circuit and you are certain you understand how to program the P inputs to set the position of the 1H-wide vertical line, set the position to some convenient viewing place on the screen, and connect the 1P input to the 1H input. Now input 1 and 1P are both operating from 1H. They are always equal. The result is a line that is 2H-clock-pulses wide. You can still adjust the position of the line from program inputs 2P through 256P, but now the line is wider.

Then connect 2P to input 2 and 2H. With the two lower-order program inputs thus connected to their respective H-count inputs,

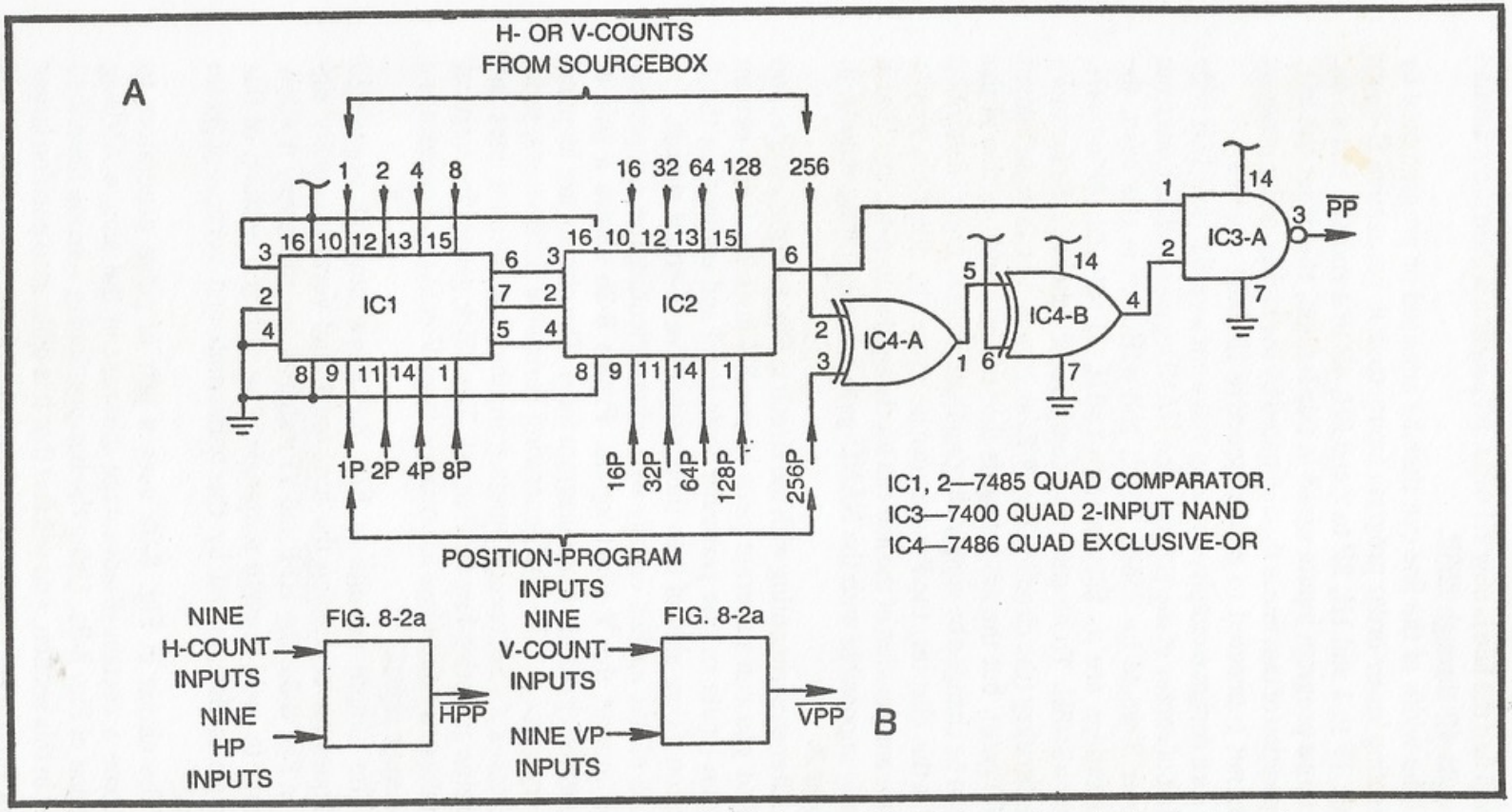


Fig. 8-2. A figure programming circuit. (a) General circuit schematic. (b) Input and output designations for horizontal and vertical figure programming.

you will find the line is now 4H wide. Its position is then programmed by inputs 4P through 256P.

The width of the line can thus be adjusted or programmed by connecting lower-order program inputs to their respective H-count inputs: 1P to 1 and 1H, 2P to 2 and 2H, 4P to 4 and 4H, and so on. The more program inputs wired in this fashion, the wider the line. The position of that line is then adjusted by the higher-order program inputs not connected to their respective H-count sources.

As a design example, suppose you want a vertical line 16H wide just left of center of the screen. The NAND-gate specifications from Chapter 3 would be $\overline{256H}$, 128H, and 64H, but in this case, the specifications are as follows: 1P=1=1H, 2P=2=2H, 4P=4=4H, and 8P=8=8H. To be sure, the comparator scheme calls for more more circuitry (the circuit in Fig. 8-2a as opposed to a simple 3-input NAND gate), but the advantage is that the size and position of the line can be changed by simply altering the P inputs. And as described later in this chapter, the P inputs can be altered by other circuitry—the size and position of the line can be changed automatically. That is virtually impossible with the NAND-gate line-generating scheme in Chapter 3.

After experimenting with the circuit in Fig. 8-2a for a while, you will find you can generate sets of parallel lines by fixing certain program inputs to one particular logic level and connecting the P inputs on either side of it to their respective H-count signals.

All of this applies equally well when orienting the comparator circuit around the V-count inputs. Figure 8-2b shows a pair of comparator circuits in block diagram form. The circuits are identical in every respect. The one generating horizontal parameters, however, takes its Sourcebox inputs from the nine H-count lines and generates an active-low $\overline{HPP}$ signal. The block generating vertical data takes its Sourcebox information from V-count and generates a $\overline{VPP}$ output signal.

The simple circuits in Fig. 8-3 show several suggested techniques for combining the horizontal- and vertical-equality signals. In Fig. 8-3a, the $\overline{HPP}$ and $\overline{VPP}$ signals are effectively ANDed together to yield a white rectangle. The size and position of this rectangle is determined by the horizontal- and vertical-program inputs.

The circuit in Fig. 8-3b uses a pair of pulse generators to overcome a certain disadvantage inherent in the simple ANDing operation of Fig. 8-3a. Using the bar-positioning scheme described earlier in this section, you will find that it is difficult to position larger bars or rectangles exactly where you want them. A bar that is 64H

wide, for example, can appear only in one of six different positions, the same six positions indicated for black and white 64H bars in Fig. 3-1.

The circuit in Fig. 8-3b, on the other hand, allows maximum precision and flexibility as far as positioning a rectangle of any size is concerned. The whole business of determining the horizontal and vertical dimensions of the rectangle is taken from the programming of the comparators and shifted to the values of the capacitors, C1 and C2. The programming of the comparators fixes the position of the rectangle with 1H and 1V precision.

While it is difficult to alter the size of a rectangle generated by the output circuit in Fig. 8-3b, the experimenter (or the game circuit

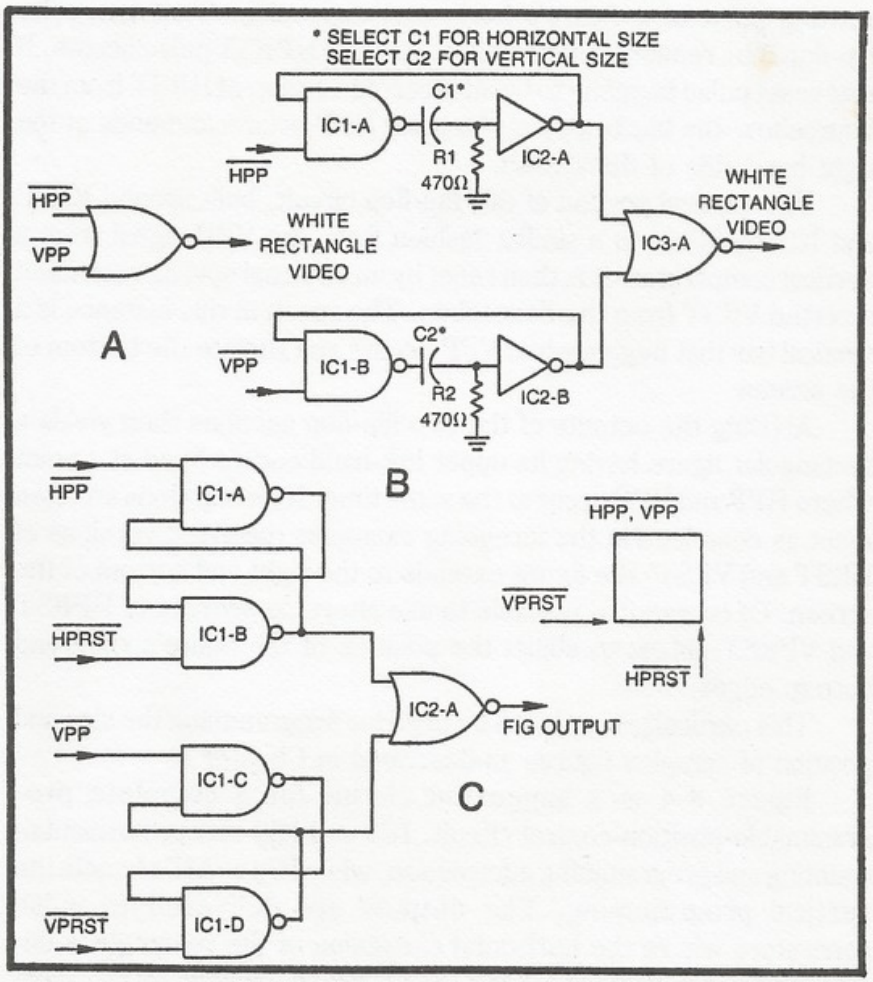


Fig. 8-3. Combining outputs from horizontal and vertical position programming circuits. (a) Forming a rather small rectangle figure by effectively ANDing active-low inputs to a NOR gate. (b) Adjusting the dimensions of a rectangle by selecting capacitor values in a pair of pulse generators. (c) Using R-S flip-flops to extend the dimensions of a comparator-programmed rectangle.

itself) can have precise control over the position via the comparators' program inputs. This feature will become especially valuable when working with ultra-slow-motion controls later in this chapter.

The purpose of the most complex comparator output circuit in Fig. 8-3c might seem rather obscure at this point, but it is shown here for the sake of completeness. The circuit is composed of two $\overline{R}$-$\overline{S}$ flip-flops, each having one input taken from a comparator circuit. The flip-flops are reset by PRST pulses from an external source, and their outputs are effectively ANDed together by IC2-A.

The flip-flop composed of IC1-A and IC1-B is set to its active state whenever it sees an HPP pulse from the horizontal-program comparator. Presumably, the occurrence of this pulse indicates the starting point of a figure's horizontal dimension. That particular flip-flop then remains active until an external HPRST pulse occurs. If that reset pulse happens to be an inverted version of HRST from the Sourcebox, the bar begins at the point HPP occurs and ends at the right-hand side of the screen.

The vertical portion of this flip-flop circuit, built around IC1-C and IC1-D, is set in a similar fashion from the VPP signal from a vertical comparator. It is then reset by an external source such as an inverted VRST from the Sourcebox. The result in this instance is a vertical bar that begins when VPP occurs and runs to the bottom of the screen.

ANDing the outputs of the two flip-flop sections then yields a rectangular figure having its upper left-hand corner fixed at a point where HPP and VPP occur at the same time. If the flip-flops are then reset as described in the foregoing examples (inverted versions of HRST and VRST) the figure extends to the right and bottom of the screen. Of course it is possible to use alternate sources of HPRST and VPRST pulses to adjust the position of the figure's right and bottom edges.

This particular circuit will be used for programming the size and position of complex figures as described in Chapter 4.

Figure 8-4 is a suggested circuit for a complete programmable-position-control circuit. IC1 and IC2 accept horizontal-counting and programming information, while IC3 and IC4 handle the vertical programming. The outputs are delivered to pulse generators where the horizontal dimension of the rectangle is determined by the value of C1 and the vertical dimension by the value of C2.

The circuit can be assembled on a 40-pin board (Radio Shack 276-153) with plenty of room to spare. The arrangement calls for using all 40 pins, however.

The program inputs can be selected by means of 18 different toggle switches, one to each of the P inputs. Certainly this would be a rather awkward scheme for programming the position of the figure, but it is an alternative that is suitable in some circumstances. See the suggested input switch circuit in Fig. 8-6.

Figure 8-5 shows an alternative to the comparator positioning board in Fig. 8-4. The primary advantage of this alternate circuit is that it requires only 23 pins as opposed to the full complement of 40 pins for the circuit in Fig. 8-4. The trick is to eliminate virtually all of the H- and V-count inputs by building a set of counters on the position-control board, itself.

The horizontal counters are IC5 and IC6, and since they are clocked by 1H and reset by HRST from the Sourcebox, they follow the basic counting pattern of the H-count system in the Sourcebox. IC7 and IC8 perform the same function for the vertical-positioning circuit, being clocked by HRST and reset by VRST from the Sourcebox.

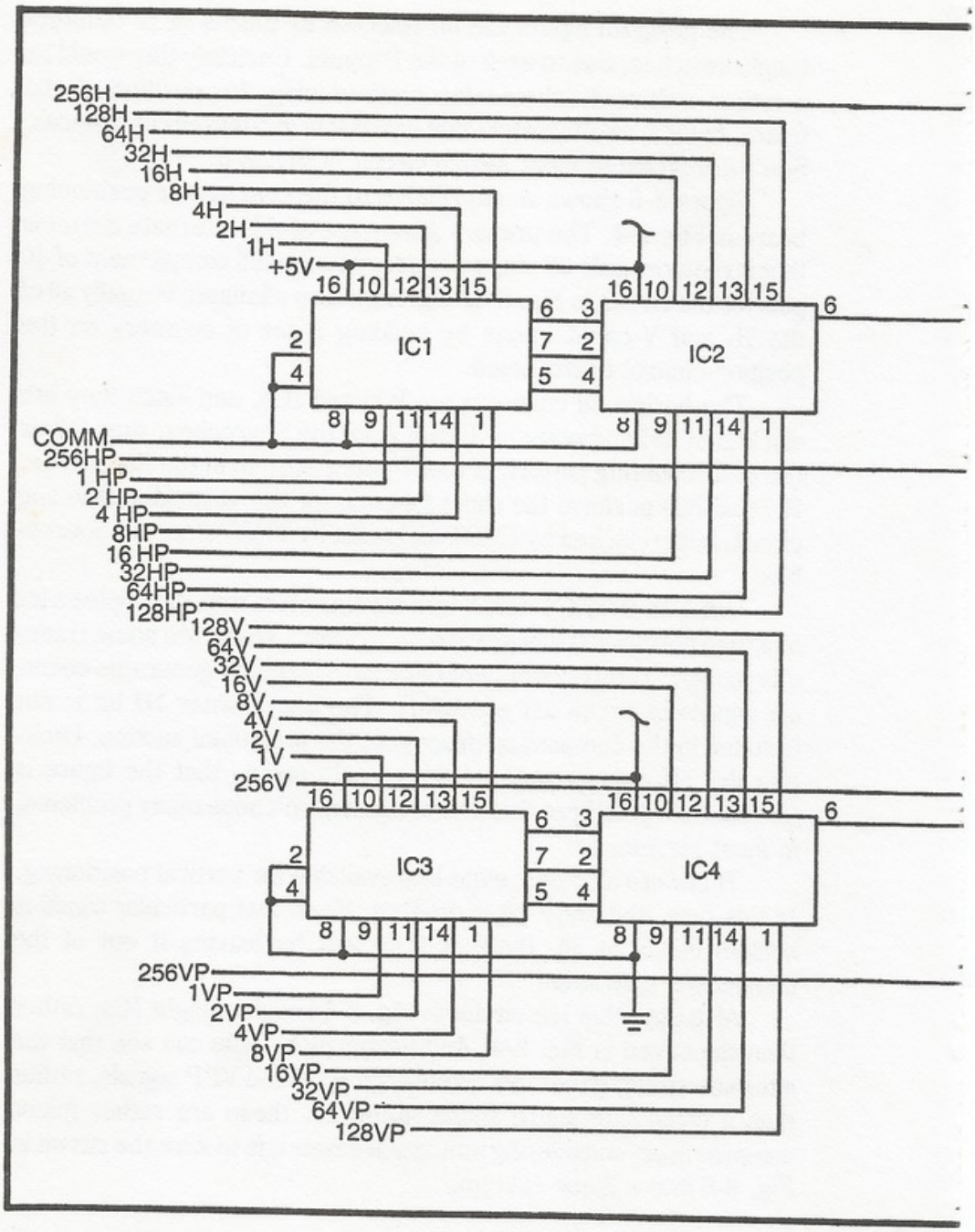
Using on-board H- and V-count generators thus eliminates a lot of wiring between the board and Sourcebox. There are some trade-offs though. The H-count generator (IC5 and IC6) generates counting signals between 2H and 256H. The lower-order 1H bit is not included in the comparison process in the horizontal section. Omitting the 1H comparison, however, only means that the figure is positioned with 2H precision. And that doesn't pose many problems, in most instances.

There are also only eight bits available for vertical positioning. In this case, the 256V bit is omitted. Since that particular signal is seldom useful at all, there is little lost by leaving it out of the comparison operation.

Also note that the circuit in Fig. 8-5 requires eight ICs, rather than the seven in Fig. 8-4. And furthermore, you can see that the alternate circuit generates active-high HPP and VPP signals, rather than a composite-figure signal. Even so, these are rather minor compromises, considering how much easier it is to wire the circuit in Fig. 8-5 into a game system.

Switch Inputs for Figure Positioning

Figure 8-6 is a sketch of a circuit that can be used for entering as many as 18 position-control bits into either of the circuit boards in Figs. 8-4 and 8-5. The circuit is drawn specifically for the circuit in Fig. 8-4, but it can be interfaced with the simpler circuit in Fig. 8-5 by omitting the 1HP position (S9 and R9) and the 256VP position (S10 and R10).



Position Programming From Counters

In the context of the comparator positioning circuits described thus far in this chapter, it is possible to specify the positioning information from a switch panel (Fig. 8-6, for example) or from any other source of logic levels. Some of those "other sources" include

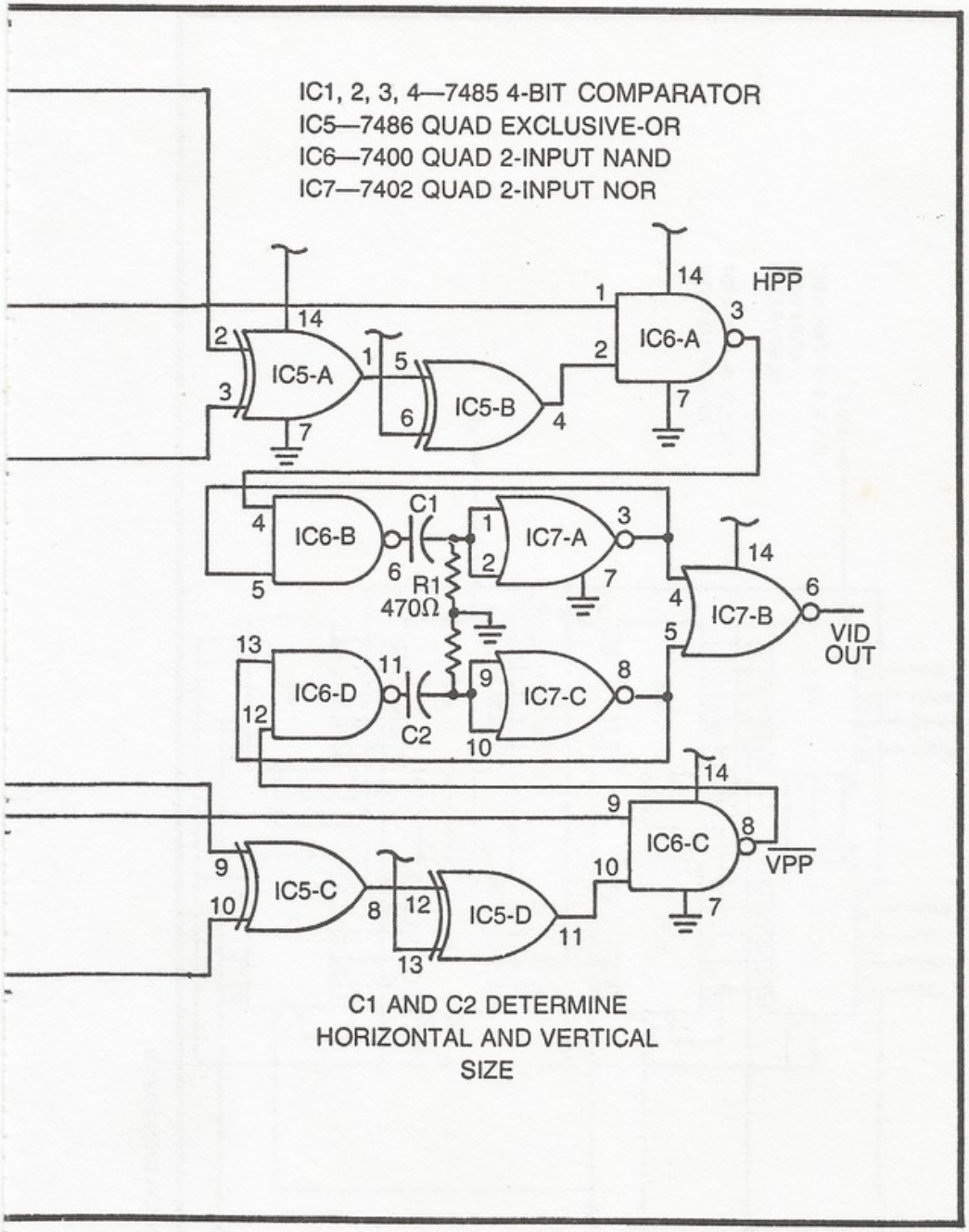


Fig. 8-4. Circuit diagram for a complete position programming system.

counters, data latches, multiplexer pattern generators, or random access memories.

Figure 8-7 shows how a set of binary counters can be used for specifying position data. The essential idea in this case is to vary the position of a figure each time the circuit is clocked. One phase of the

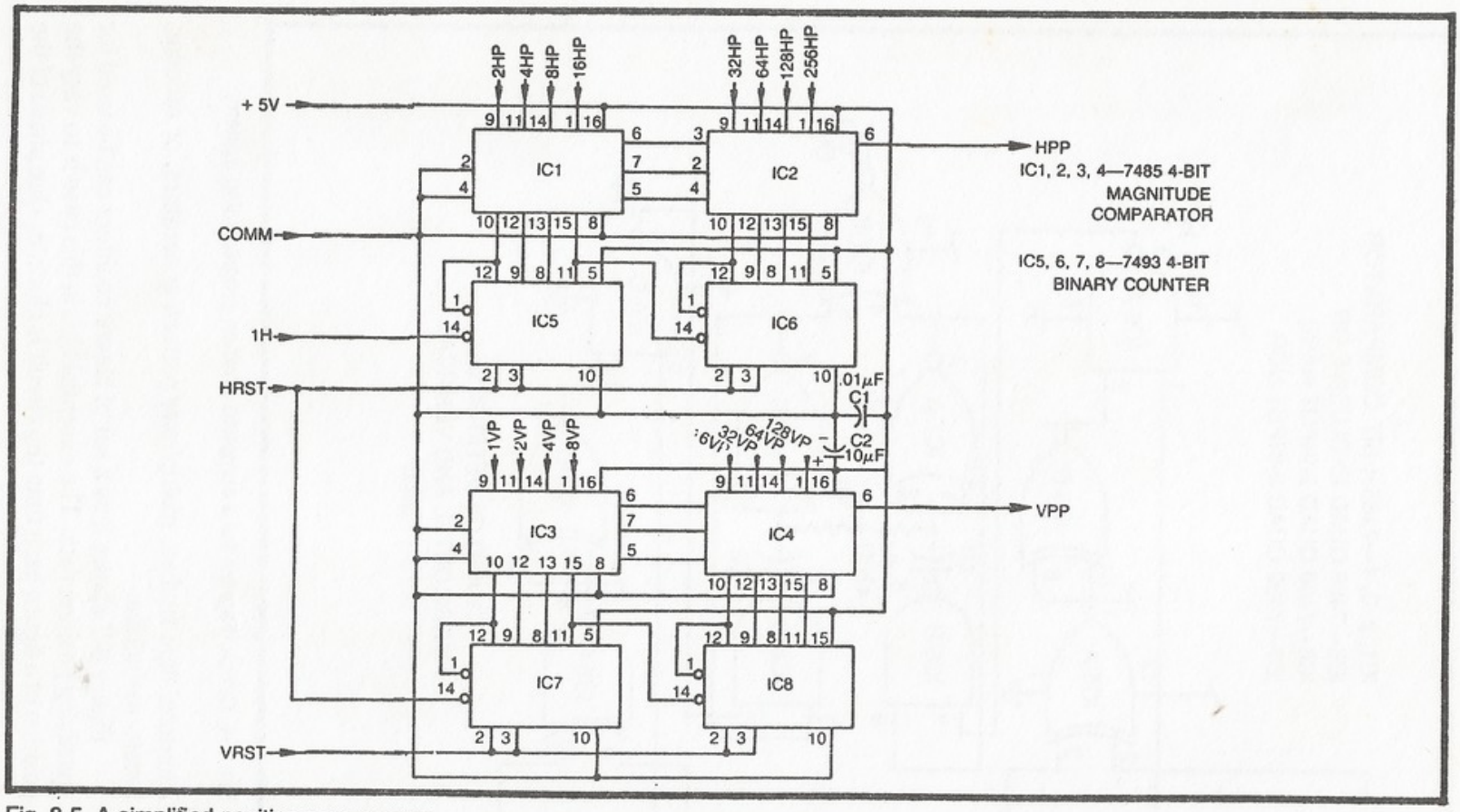


Fig. 8-5. A simplified position programmer.

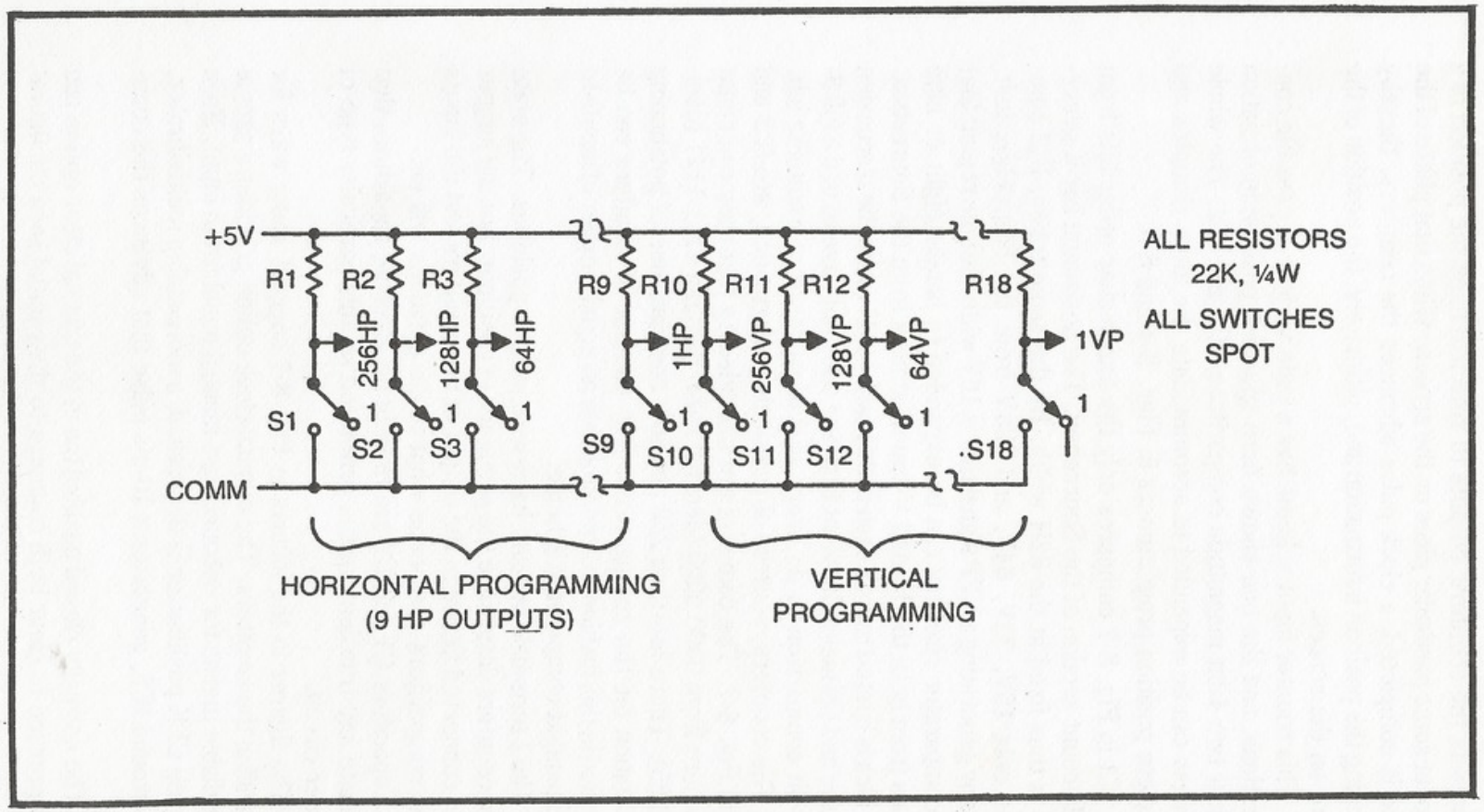


Fig. 8-6. A suggested switch panel for programming the position of a figure on the screen.

game sets the counters' outputs to zero and fixes the position of a figure at some particular place on the screen. When that phase of the game is completed, a clock pulse advances the counters, thereby changing the position information and, ultimately, the position of the figure on the screen.

This counter input scheme has a wide variety of possible configurations, and the one shown here specifies position information for just two 4-bit magnitude comparitors, IC1 and IC2. The whole business can be expanded to accommodate the more complex and complete position programmers in Figs. 8-4 and 8-5.

IC1 in Fig. 8-7 compares only the four higher-order bits from the H-count section of the Sourcebox. The horizontal bar it generates is thus fixed at the 32H width. By the same token, IC2 compares only 16V, 32V, 64V, and 128V from the Sourcebox unit, thereby generating a VPP signal that is 16V wide. Let me repeat that the comparator circuitry can be expanded to handle eight or nine signals from both the H- and V-count outputs from the Sourcebox. The idea is limited to four each in this case to (1) make the discussion simpler and (2) make the point that one does not have to use a full 8- or 9-bit comparison to do every sort of pattern-programming job.

The counters, sources of positioning information, are IC3 and IC4 in Fig. 8-7. The counters are cascaded so that they can count anywhere from 0000 0000 (decimal zero) through 1111 1111 (decimal 255). There are thus 256 possible combinations of positioning information for the comparators. Any number of counters can be cascaded in this fashion to accommodate an equal number of inputs to an expanded comparator scheme.

The J terminals indicate hard-wire jumper positions. The state of the counters determine the sequence of positions, but the jumper wires connected between the output of the counters and the inputs of the comparitors determine what those positions will be.

Capacitors C1 and C2 are merely de-glitching capacitors that eliminate any transient spikes generated by this particular type of counter circuit.

The figures in the inserts in Fig. 8-7 suggest some ways for controlling the counters. The manual clock option provides a simple and reliable means for debouncing a manual pushbutton input. Each time the CLK pushbutton is depressed, a monostable multivibrator, built around IC5, generates a 10-ms pulse that advances the counters.

The normally closed pushbutton in the manual clear option can be depressed to clear both counters to their initial zero configuration.

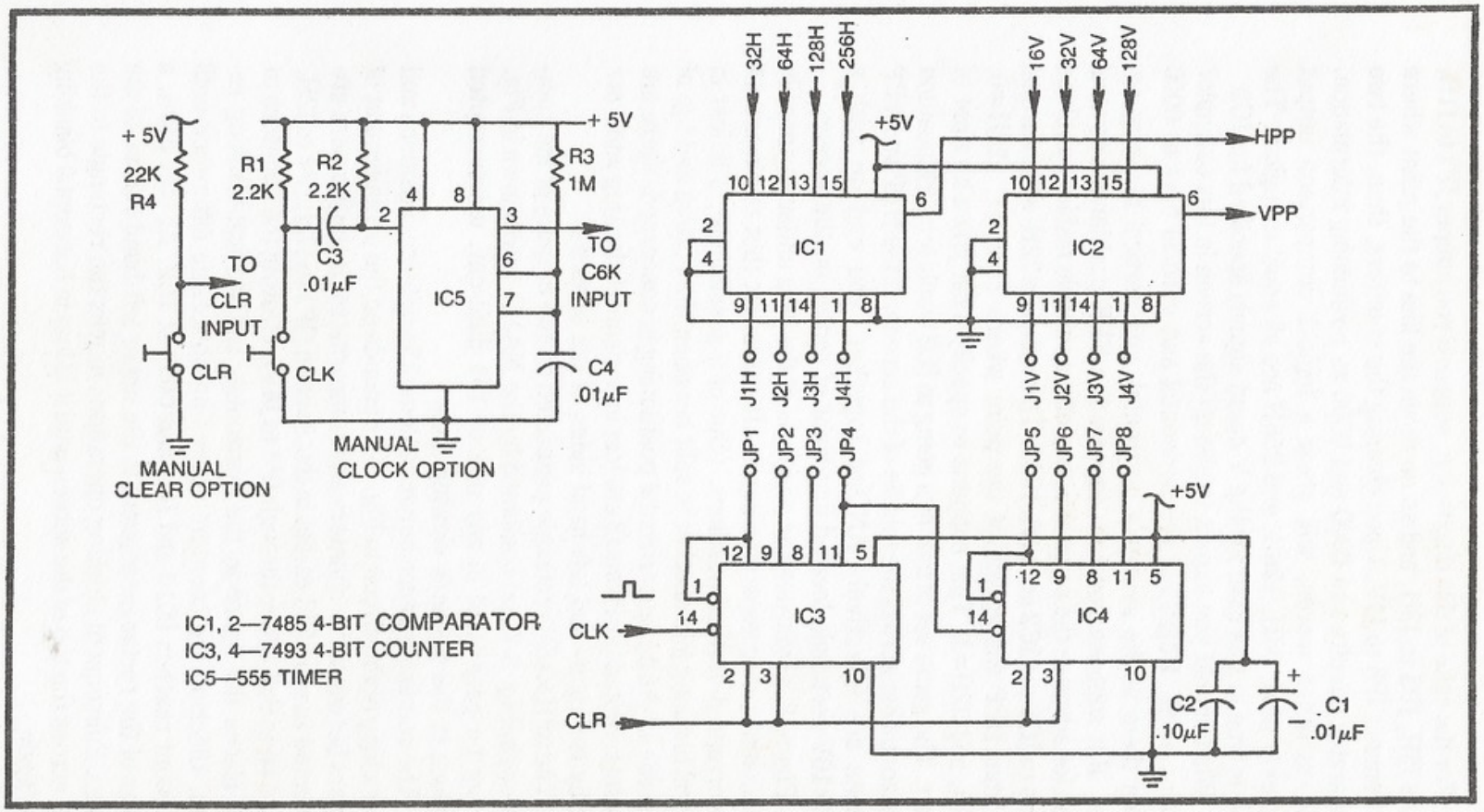


Fig. 8-7. Position programming from counter circuits—schematic and suggested input circuits.

For the sake of this discussion, suppose you jumper JP1 to J1H, JP2 to J2H, JP3 to J3H, and so on down the line to the point where you connect JP8 to J4V. Upon clearing the counters, then, the two comparator circuits see 0000 and 0000 as positioning information. HPP, in other words, will show a logic-1 comparison output whenever 32H, 64H, 128H, and 256H are all equal to logic 0. The same is true in this case for the V-count signals specified for IC2.

What should you expect to see on the screen in this example? Nothing at all. Beginning the horizontal and vertical bars at 0000 places them in the system's horizontal- and vertical blanking regions. But suppose you now depress the CLK pushbutton several times (or advance the counter in any other suitable fashion) until the inputs to IC1 and IC2 are both 1001. This means a 32H-wide vertical bar from HPP will appear at the point where 256H=1, 128H=0, 64H=, and 32H=1. That happens to specify a bar just a bit right of center. The same set of four bits going to IC2 would let VPP position a horizontal bar across the middle of the screen. If the HPP and VPP outputs are effectively ANDed together, you end up with a 32H×16V rectangle located just right of center on the screen.

Figure 8-8 indicates the positions of the bars under all possible combinations of comparator inputs. It turns out that there are 14 uninterrupted vertical positions. (Out of a possible 16, 1 is lost in vertical blanking and another is split between the bottom and top of the screen.) As far as horizontal positioning is concerned, there are 11 uninterrupted positions (3 are lost in horizontal blanking and 2 are split by the right- and left-hand sides of the screen).

The 32H×16V rectangle generated by the outputs of the comparators in Fig. 8-7 and combined by the NAND gate shown in Fig. 8-8 can be positioned in any one of 154 different, uninterrupted positions in the screen's viewing area.

The correspondence between these 154 different positions and the clocking of the circuit in Fig. 8-7 depends on the arrangement of jumpers between the counters and comparators. If the jumpers are connected as specified earlier in this section (JP1 to J1H, JP2 to J2H, and so on in that order through JP8 to J4V), clearing the counters to zero places the figure in the horizontal- and vertical-blanking regions. Clocking the counters causes no noticeable difference until the count reaches 0010 and IC1 and 0001 at IC2. At that point, a corner of the rectangle appears in the upper left-hand corner of the screen. Subsequent clocking operations moves the rectangle to the right across the top of the screen until it is lost in horizontal blanking once again.

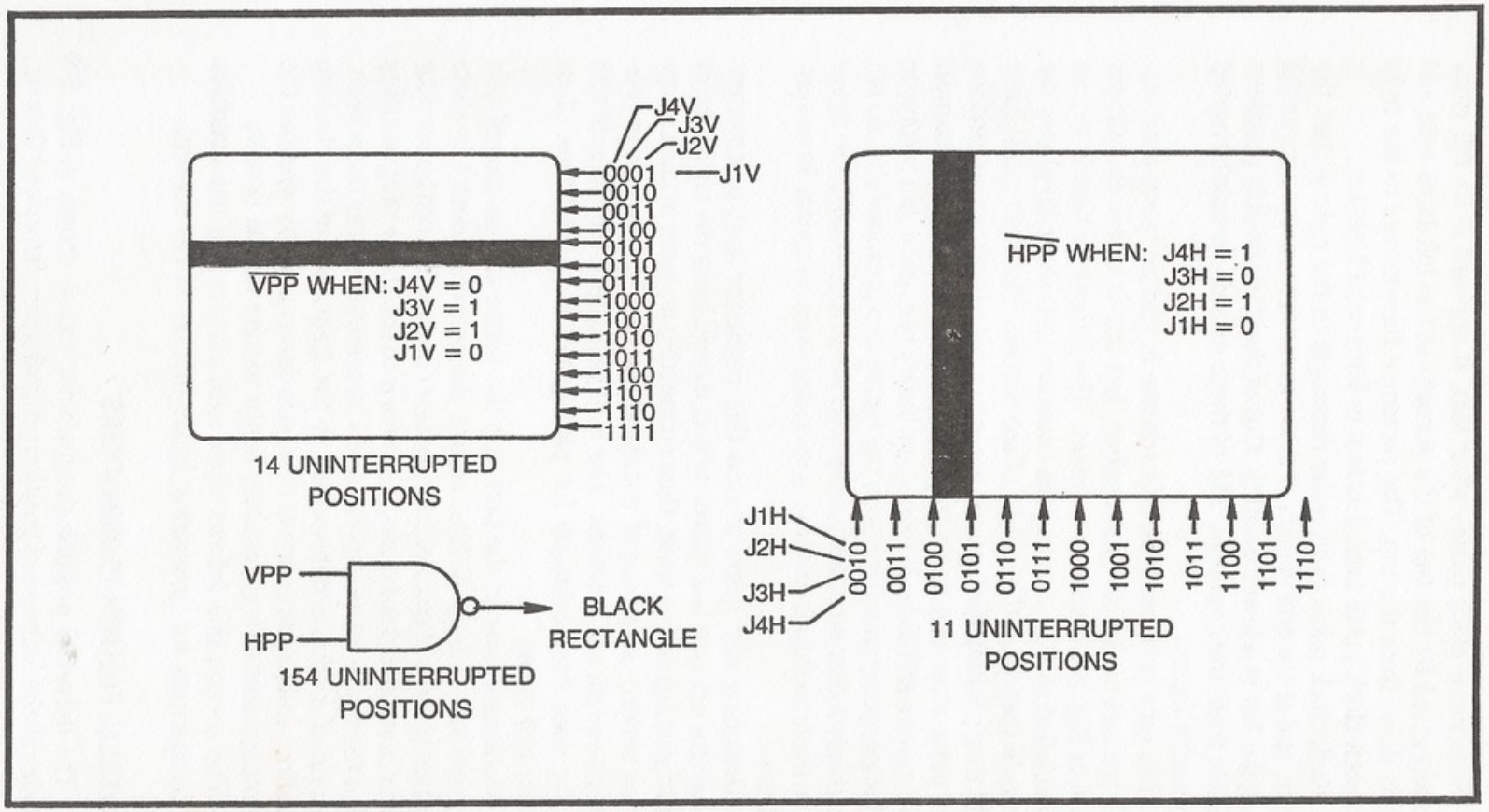


Fig. 8-8. Relationships between position-programming input parameters and actual bar positions on the screen.

Four more clock pulses after that, it appears in its full glory (uninterrupted by the top of the screen) at the left-hand side and slightly down from the top. The rectangle then moves to the right with each clock pulse until it is lost in horizontal blanking.

Each clock pulse carries the rectangle to the right across the screen, and at the end of each of these excursions, it appears at the left again, but in a lower position. Out of the 256 possible positions available from the counters, 154 of them expose an uninterrupted rectangular figure.

The only problem with this scheme is that so many combinations are lost in the blanking regions, but this is where the jumper option in Fig. 8-7 becomes an asset. The jumper sequence can be scrambled such that clearing the counters to 0000 0000 places the rectangle very near the middle of the screen. The next clock pulse might then position the figure in the upper right-hand corner, and the clock pulse after that might take it to the lower right-hand corner. The counters still count in their usual binary sequence, but mixing up the jumper programming places the figure into positions that do not necessarily follow an orderly, stepwise pattern. In effect, the figure can be made to skip around the screen and even disappear from view at times.

Designing any game around this particular sort of counter/comparator scheme is a matter of first determining the sequence of comparator inputs you want, then connecting jumpers in such a way that an orderly sequence of counts is transformed into your prescribed sequence of positions. The Golf game described later in this chapter uses this technique for placing the holes and tees of an 18-hole golf game.

Returning now to the fact that the figure can be moved in a stepwise and orderly fashion across the screen, doesn't it seem possible this sort of counter/comparator combination can be used for motion control? Indeed it can. The idea is to set up the programming so that the figure moves in very small increments across the screen. The rate of motion is determined by the frequency of the clocking operation, and the direction of motion is determined by whether the and comparators for generating figure motion on the screen.

The section that follows deals with circuits that use counters and comparitors for generating figure motion on the screen.

UNIVERSAL POSITION PROGRAMMERS

The universal position programmer circuit shown in Fig. 8-9 includes all the features of position programmers described thus far,

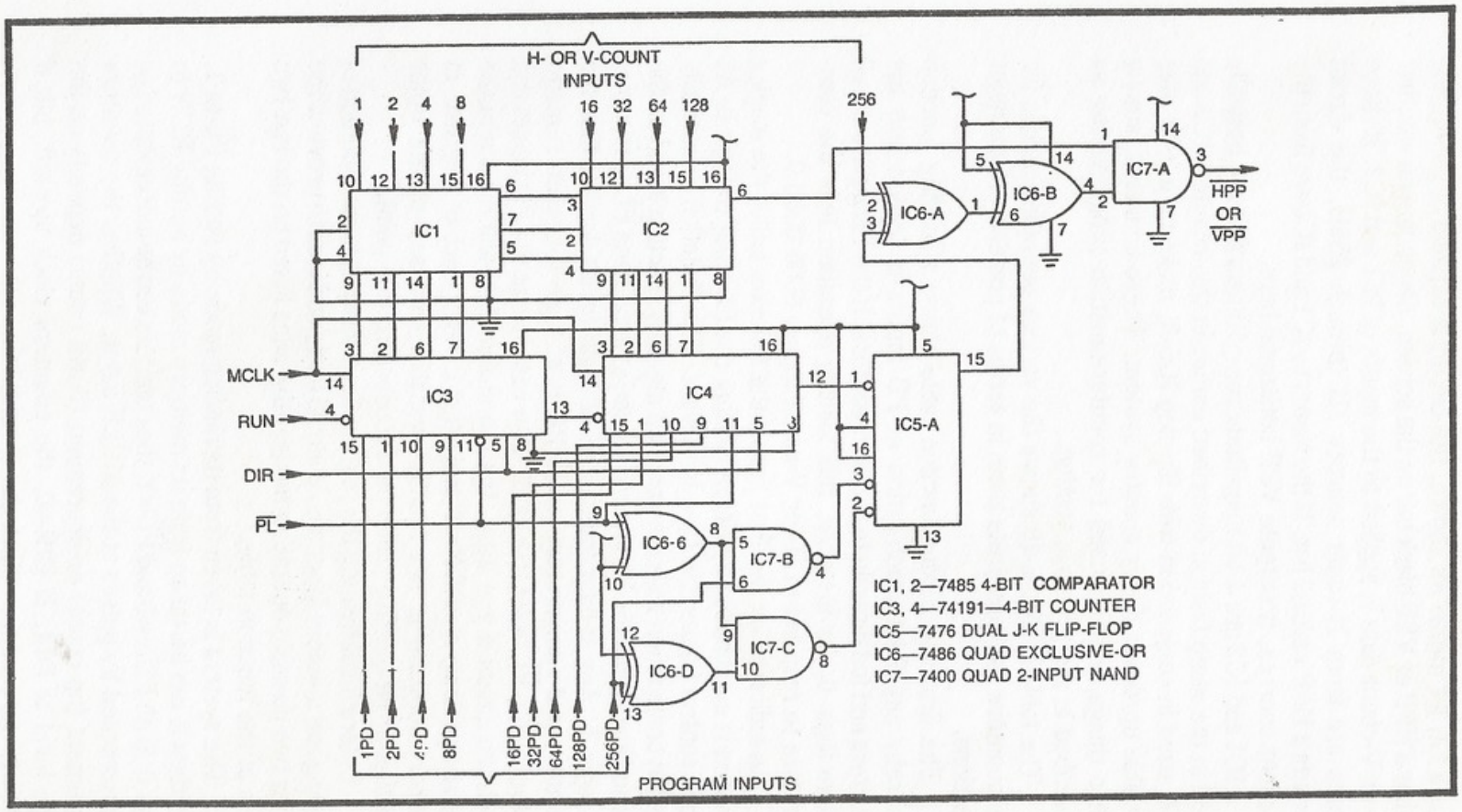


Fig. 8-9. Circuit for a universal position programmer.

plus a lot, lot more. As shown, the circuit is capable of generating either a HPP or VPP black bar on the screen, depending on whether H- or V-count data is applied to the inputs of IC1 and IC2. If these inputs are from H-count sources 1H through 256H, the circuit outputs a HPP vertical bar. If the inputs to IC1 and IC2 are from the V-count sources, it outputs VPP horizontal bars.

IC1 and IC2 are 4-bit magnitude comparators that are basically used in the same fashion described earlier in this chapter. IC3 and IC4, used in conjunction with flip-flop IC5-A, make up a 9-bit presettable up/down binary counter system. These counters, among other things, can be used for counter-position programming as described in the previous section.

The table in Fig. 8-10 shows the various operating modes for the counters and interprets them in terms of position-programmer functions.

The first line of the function table shows a loading function whereby any 9-bit combination of PD inputs is loaded into the counters and fed directly to the comparators. If the PD inputs are all set to logic 0, for instance, this loading operation tells the comparators to respond to H- or V-count inputs 0000 0000 0.

As indicated on the first line of the function table, the loading function is established by simply setting the PL input to logic 0. All other control inputs are then irrelevant. This function is normally used in conjunction with the program memory function listed on the second line. Here the RUN input is fixed at logic 1 and $\overline{PL}$ is returned to logic 1. Any 9-bit combination of logic levels loaded into the counters and comparators while PL is at logic 0 is then remembered" when $\overline{PL}$ is set to logic 1. You can change the PD inputs and clock the circuit if you want, but the stored data will remain in the system as long as RUN=1 and $\overline{PL}$=1. This sort of register of memory function is not readily available with any of the other position-programming schemes described to this point.

Before investigating the real implications of this combination of loading and memory functions, it would be a good idea to preview the other two possible operating modes, the ones listed on the last two lines of the function table.

The system is taken from its loading mode by setting $\overline{PL}$ to 1, and then it can be taken from its memory mode by setting RUN to logic 0. With RUN=0 and $\overline{PL}$=1, the counters can be incremented or decremented by pulses at the MCLK input. Whether the counters increment (up count) or decrement (down count) depends on the logic level at DIR. If DIR=0, the counters clock upward, but if DIR=1, the counters clock downward.

CONTROL INPUTS				OPERATING MODE
RUN	$\overline{PL}$	DIR	MCLK	
X	0	X	X	LOAD PD INPUTS
1	1	X	X	PROGRAM MEMORY
X	1	0	Ω	INCREMENT (UP COUNT)
0	1	1	Ω	DECREMENT (DOWN COUNT)

X = NOT RELEVANT
LOAD 0000 00000 TO CLEAR

Fig. 8-10. Function table for the universal position programmer.

A particular count can be stopped and "remembered" by simply setting RUN to logic 1 at the appropriate time. If desired, the count can be resumed from that point by simply setting RUN to logic 0 again.

Perhaps the best way to get a good feeling for how the universal position programmer works is by building the circuit in Fig. 8-9 onto a circuit board and interfacing it with the input controls shown in Fig. 8-11. Building the position programmer onto a circuit board will not be a waste of time and effort because it will prove useful in some TV-game designs later on. The input circuits can be built in a breadboard fashion so that the parts can be used for other game purposes.

For the sake of a preliminary set of experiments, connect the comparator inputs 1 through 128 to their respective H-count signals from the Sourcebox unit. This will provide an HPP output signal that ultimately appears as a fine, black vertical line on the screen. The simple circuit in Fig. 8-12 shows a technique for widening the bar and changing it to white on black for easier viewing.

Experiments With a Position Programmer

To begin the experiments, set the STOP/RUN switch to the logic-1 STOP position. Then set the nine PD input switches for some desired bar position. In the context of the presentation in Chapter 3, a logic-1 PD input is tantamount to a noninverted H-count signal, while a logic-0 input yields the effect of using an inverted H-count input.

Note there is no response on the screen while adjusting the settings of the PD switches. This feature allows the experimenter to

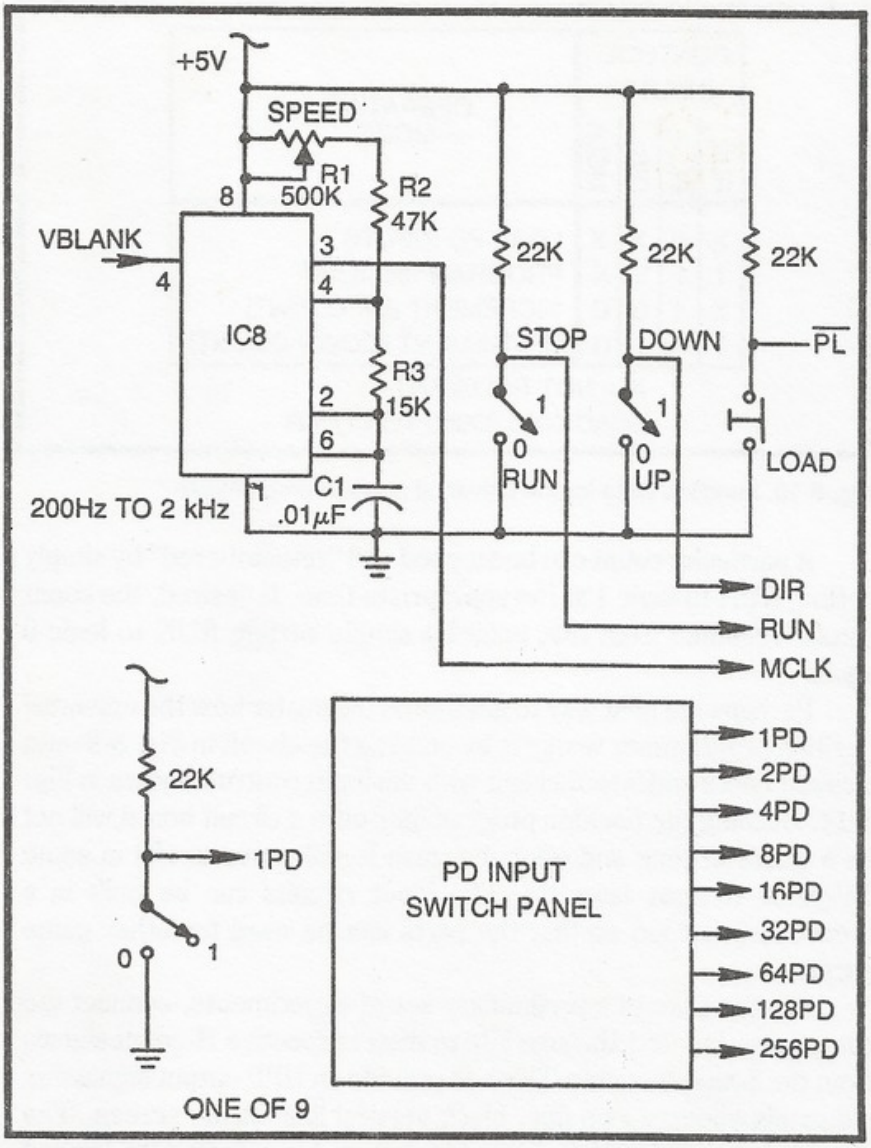


Fig. 8-11. Suggested input interface for experiments with the universal position programmer.

set up the position-programming codes without having the figure jumping all over the screen. With the PD switches now set to a desired program combination, momentarily depress the LOAD pushbutton. You will see the bar on the screen jump immediately to the position you specified on the PD input switches.

Try a few more positions, first setting the codes on the PD switch panel and then depressing the LOAD pushbutton. The bar can indeed be set to any desired position on the screen with 1H precision.

Once you are satisfied you understand the function of the load and memory scheme, position the bar close to the middle of the screen, set the UP/DOWN switch to its UP position, and then set the RUN/STOP switch to RUN. IC8 in Fig. 8-11 is connected as a free-running multivibrator, and if it is operating properly, you should see the bar moving to the right across the screen.

The speed of motion depends mainly on the value of R1. If the multivibrator is running at its lower frequency limit (about 200 Hz) the bar drifts across the screen rather slowly. If, on the other hand, R1 is set so that the multivibrator runs in the neighborhood of 2 kHz, the bar makes a complete cycle across the screen in about 4 seconds.

IC8, itself, is enabled only through the VBLANK interval. Note the VBLANK connection to pin 4 of IC8. While this multivibrator is enabled, it produces clocking pulses for the counters in Fig. 8-9, effectively changing the position information for the bar. The changes take place in an orderly fashion, giving the impression of rather smooth and continuous motion across the screen. The faster the multivibrator runs through the VBLANK interval, the faster the bar advances across the screen.

The direction of motion is determined by the UP/DOWN switch, a switch that ultimately sets the DIR control input to logic 1 or 0. The bar thus moves to the right across the screen when UP/DOWN is set to UP, and then the bar moves to the left when that switch is set to DOWN.

The motion can be stopped, holding the bar at some particular point on the screen, by setting the RUN/STOP switch to STOP. The multivibrator continues producing clocking pulses through every VBLANK interval, but the bar remains motionless on the screen because the counters are disabled.

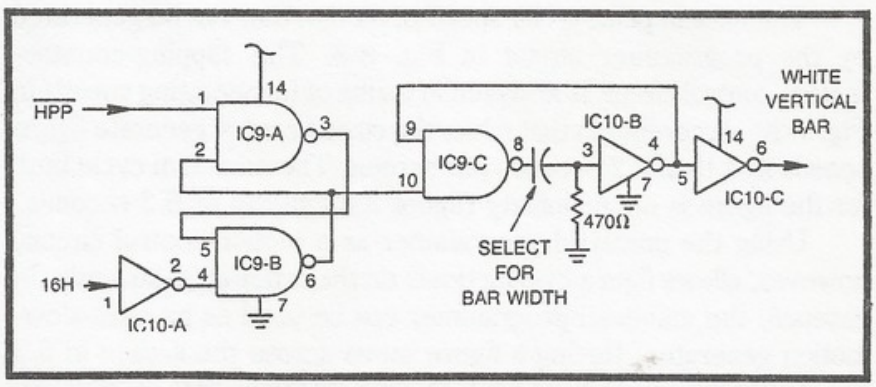


Fig. 8-12. A circuit for widening the bar generated by the universal position programmer.

Figure 8-13 shows a set of curves that translate the multivibrator's operating frequency into numbers indicating the speed of the bar. Those numbers indicate the length of time required for making one complete cycle of motion across the screen. If, for example, the system is operating from H-count signals from the Sourcebox, and the multivibrator is operating at 1 kHz, the Horizontal Motion curve shows that the bar will complete one cycle of horizontal motion across the screen in about 7.6 seconds.

If the inputs to the comparators in Fig. 8-9 are taken from the V-count signals from the Sourcebox, the bar moves vertically at a rate shown by the Vertical Motion curve. In the case of vertical motion, setting the UP/DOWN switch to UP causes the bar to move downward, while setting that switch to DOWN causes the bar to move upward. (If this seems to cause some confusion, simply bear in mind that the UP and DOWN designations on the direction switch indicate the direction of the counter, and not the direction of motion.

A Slow-Motion Figure Generator

A thoughtful reader ought to be asking some relevant questions at this pont. Why should I ever build a figure-motion-control circuit around a set of counter/comparator circuits when the slipping counters are simpler? What can this universal position programmer do that cannot be done with the simpler positioning circuits described earlier in this chapter?

Both questions can be answered in the following terms. The universal position programmer is, indeed, a more complicated system than any slipping counter and position programmer. The fact of the matter is, however, that this universal programmer can perform some operations that are virtually impossible with the simpler circuits.

The case in point is the speed of motion that can be generated by the programmer circuit in Fig. 8-9. The slipping-counter-motion-control circuit is analyzed in terms of its operating speeds in Fig. 7-22. According to that table, the circuit cannot generate figure speeds less than 6.2 seconds per screen. The minimum cycle time for the figure is either infinity (figure motionless) or 6.2 seconds.

Using the universal programmer as a motion-control circuit, however, allows figure-cycling times on the order of 20 seconds. In essence, the universal programmer can be used as an ultra-slow-motion generator. Having a figure move across the screen in 6.2 seconds (the greatest amount of time possible with a slipping-counter arrangement) might be too fast for many game applications.

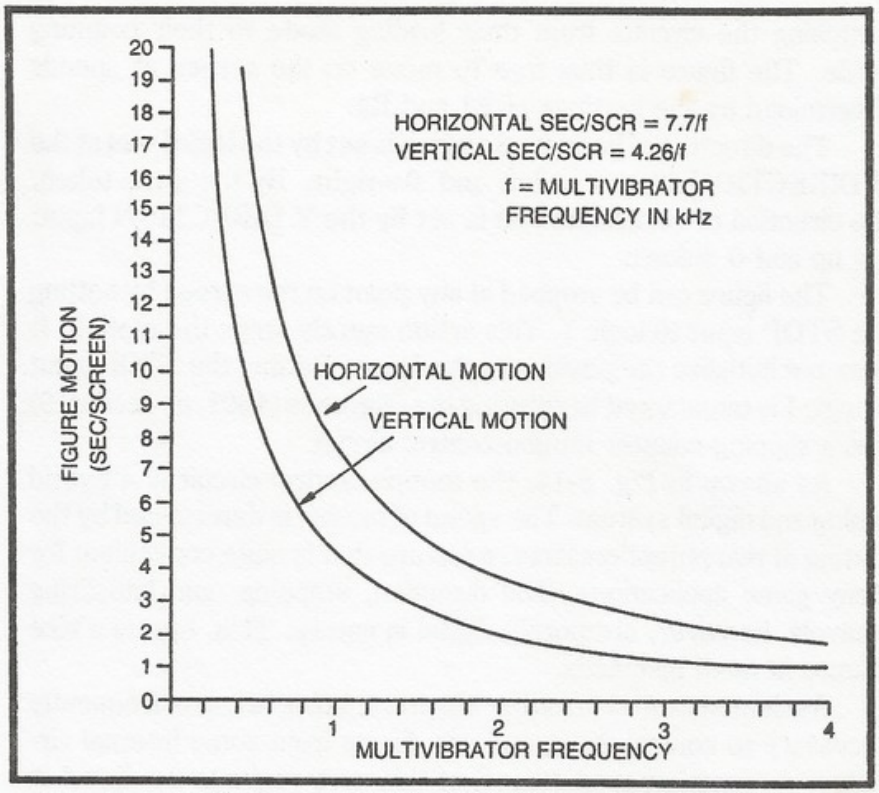


Fig. 8-13. Curves and equations for determining the screen cycle time of a slow motion figure generator.

Where this is the case, the universal programmer comes to the rescue.

The circuit in Fig. 8-14 shows a practical motion-control circuit built around a pair of universal-position programmers. Two of the circuits in Fig. 8-9 are required in this case, one for controlling vertical motion and another for controlling the horizontal component of figure motion.

Both motion circuits are clocked from their own multivibrators: IC1-A for generating vertical-speed pulses and IC1-B for generating MCLK pulses for the horizontal-position programmer. The two position programmers share a common position-initializing circuit built around IC2-A and IC2-B. The idea here is to load some prescribed position codes into the position programmers whenever a negative-going pulse occurs at the INT input, pin 1 of IC2-A. All counting action stops as long as the system remains in this initializing mode, and the position programmers take their positioning information directly from their respective PD inputs.

A negative-going pulse at the system's MOVE input, however, sets the $\overline{PL}$ inputs of the two programmers to logic 1, thereby

switching the circuits from their loading mode to their counting mode. The figure is thus free to move on the screen at speeds determined by the settings of R1 and R4.

The direction of horizontal motion is set by the logic level at the H DIRECTION input: 1= left and 0=right. By the same token, the direction of vertical motion is set by the V DIRECTION input: 1= up and 0 =down.

The figure can be stopped at any point on the screen by setting the STOP input to logic 1. This action merely stops the motion. It does not initialize the position of the figure. Setting the STOP input to logic 1 is tantamount to entering the stop code (1001, or decimal 9) into a slipping-counter motion-control circuit.

As shown in Fig. 8-14, the motion-control circuit is a hybrid analog and digital system. The speed of motion is determined by the setting of two potentiometers, a feature that is quite convenient for many game applications. The direction, stopping, and initializing controls, however, are purely digital in nature. This, too, is a nice feature in most instances.

As demonstrated in earlier chapters, however, it is frequently necessary to control the speed of a figure from some internal circuitry. Automatic speed control of this sort ought to be digital in nature. The circuit should be able to accept a binary word that determines the figure speed.

Compare this situation with the speed-control scheme for a slipping counter. In the case of a slipping counter, the most natural way to control the speed of the figure is by entering a 4-bit binary word. The circuit is basically a digital one. Converting the slipping-counter motion-control system to one having potentiometer control is a tricky process calling for an analog-to-digital (A/D) converter circuit.

The situation is just reversed for the universal position programmer. This circuit most naturally accepts potentiometer speed controls. And where it is necessary to control the speed in a digital fashion, it is necessary to add a digital-to-analog (D/A) converter. The circuit in Fig. 8-15 shows a simple D/A converter scheme that is most appropriate for setting the motion speed.

The D/A converter in this case is a binary ladder network composed of resistors R7 through R12. The PC inputs are eight possible combinations of 1s and 0s, as shown in the table accompanying the circuit. IC1-A is the same multivibrator having that designation in Fig. 8-14. Two such circuits are thus required for full vertical- and horizontal-motion control.

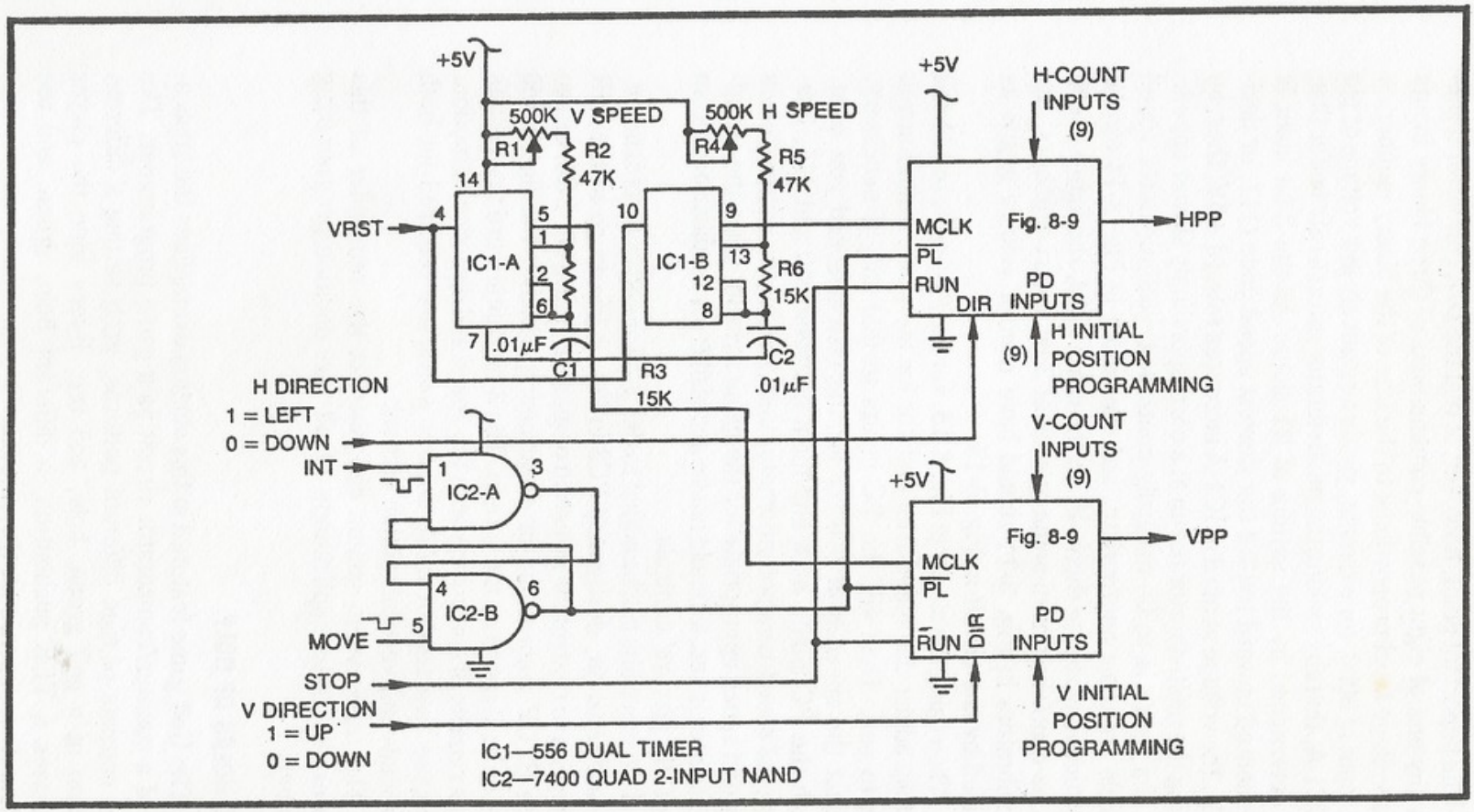


Fig. 8-14. A practical motion control circuit built around comparator position programmers.

The important point, however, is that the D/A converter translates any one of eight possible combinations of three binary inputs into a voltage level proportional to the size of the binary number. A 3-bit input of 0000, for example, yields a relatively low voltage at pin 3 of IC1-A, thereby making the multivibrator run at a rate faster than that determined by the setting of R1 alone. As the 3-bit input is incremented upward toward the slowest speed code (111, or decimal 1), the voltage at pin 3 of IC1-A increases toward +5V, thereby causing the multivibrator to run at a correspondingly slower speed.

If a figure is to be manually controlled, potentiometer speed controls are most appropriate, and the circuit in Fig. 8-14 can be constructed as shown there. If, on the other hand, the figure speed is to be controlled from some source of 3-bit binary numbers, the multivibrators in Fig. 8-14 should have control voltage inputs as modified by the circuit in Fig. 8-15.

When using the circuit in Fig. 8-15, set the PC inputs to 111 and carefully adjust the speed-trim resistor for the slowest motion you want to see. Then set the PC inputs to 000 and, if necessary, readjust the speed-trim resistor for the fastest speed you want. Switch the PC inputs back and forth between 000 and 111, fine-tuning the speed-trim potentiometer until you get the two extremes of motion speed appropriate for the game at hand. Once the speed-trim resistor is set, it should not be accessible for readjustment from any of the players' controls.

Two more bits can be added to this 3-bit speed-control scheme. One additional bit, designated PC3 in Fig. 8-15 can go to the DIR connection to control the direction of motion. A fifth bit, PC4, can go to the STOP connection for determining whether the figure will move or be stopped on the screen. The 5-bit digital word that results is thus capable of controlling both the speed and direction of motion. Of course two sets of these control words are required for both horizontal- and vertical-motion control.

The three video games described in the remainder of this chapter illustrate applications of all the position-programming schemes.

NINE HOLES OF GOLF

The Golf game featured in this section exemplifies the application of a counter/comparator circuit as a game programmer. The game consists of nine different patterns, each setting a different position of a golf green, hole, and tee. Every time the player depresses a TEE pushbutton, a different hole, green, and tee configuration appears on the screen.

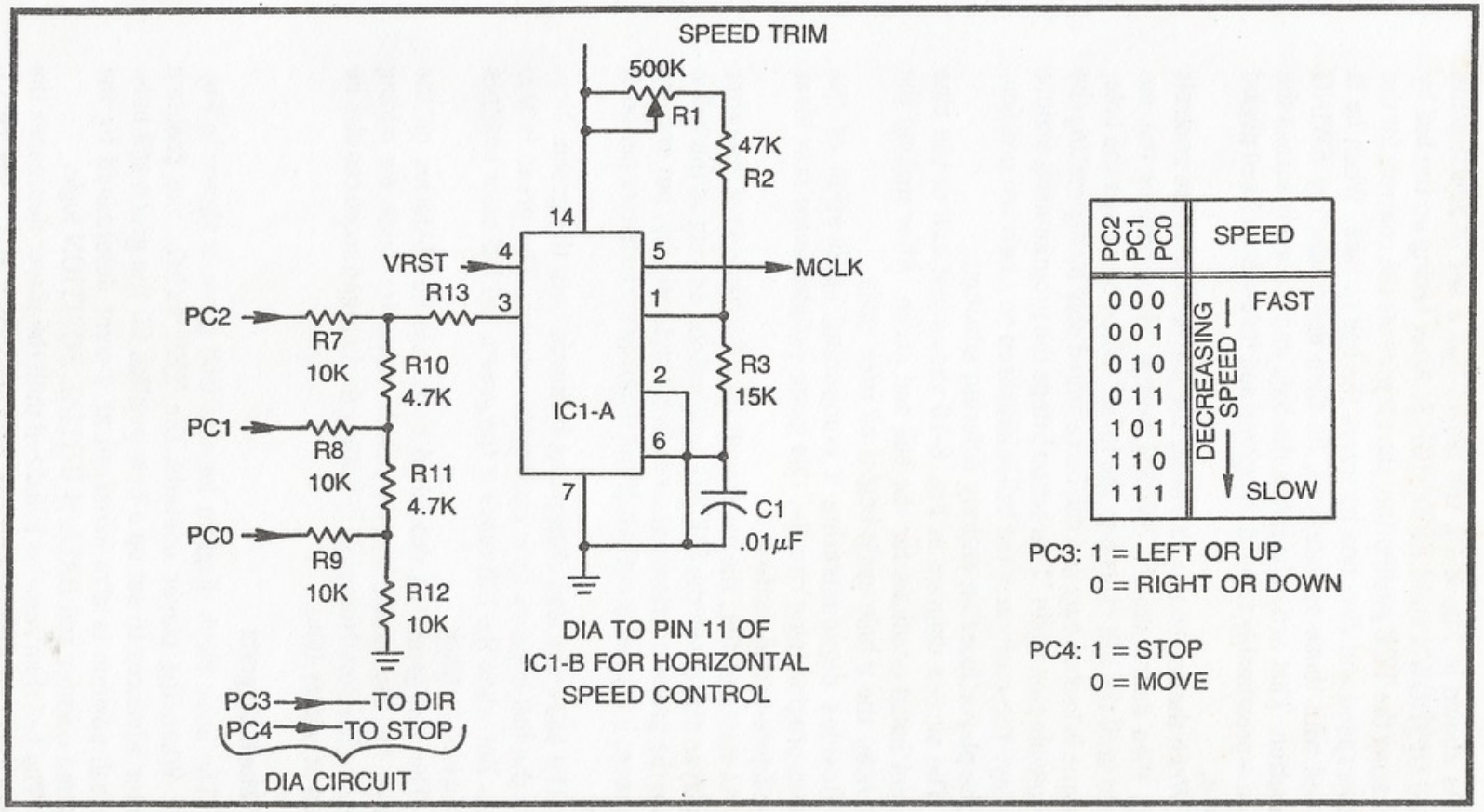


Fig. 8-15. A D/A converter circuit and input table for controlling speed from digital words.

As shown in Fig. 8-16, the player has a set of adjustments labeled UP/DOWN and LEFT/RIGHT. After teeing up the ball by depressing the TEE pushbutton, the player sets the controls for the relative speed and direction he wants the ball to take. When he is satisfied with those adjustments, he then depresses the SWING pushbutton. That action launches the ball, and it travels across the screen—presumably toward the green and hole—for a fixed period of time.

When the ball comes to a stop, the player adjusts the controls again, then depresses the SWING button. He continues this sequence until the ball "falls into the hole." When he makes the hole, the game is locked out, and can be resumed only by depressing the TEE pushbutton again. This action brings the programming feature into play: the position of the ball is initialized on a new tee position, and the player faces an entirely different situation.

The screen diagram in Fig. 8-16 shows just one of the nine different initial positions for the ball and green. After making the ninth hole, the 9-hole cycle begins all over again.

Besides demonstrating a compelling application of the position-programming circuits, this game includes some new ideas about figure-speed control.

As stated earlier, the ball travels for a fixed period of time after the player depresses the SWING pushbutton. As long as the ball is not on the green, it travels at a relatively high velocity, but once on the green, its speed is cut in half and it moves for a shorter period of time.

The ball color also changes as it moves onto the green. Normally the ball appears as a small white square. The green is also white. But when the ball reaches the green, the ball color changes from white to black.

The golf game, as described here, has no obstacles on the course, but the circuit boards include some terminals for adding obstacles later on. Numerals indicating the hole and score can also be added at a later time.

Golf Block Diagram

The basic block diagram for this Golf game is shown in Fig. 8-17. When the player operates the TEE switch, the program counter advances to set up a new position for the green and hole. The ball position is also initialized at a point determined by the program counter and BALL H INITIAL POSITION logic.

The ball then remains initialized until the player depresses the SWING switch. Depressing the SWING switch starts the BALL

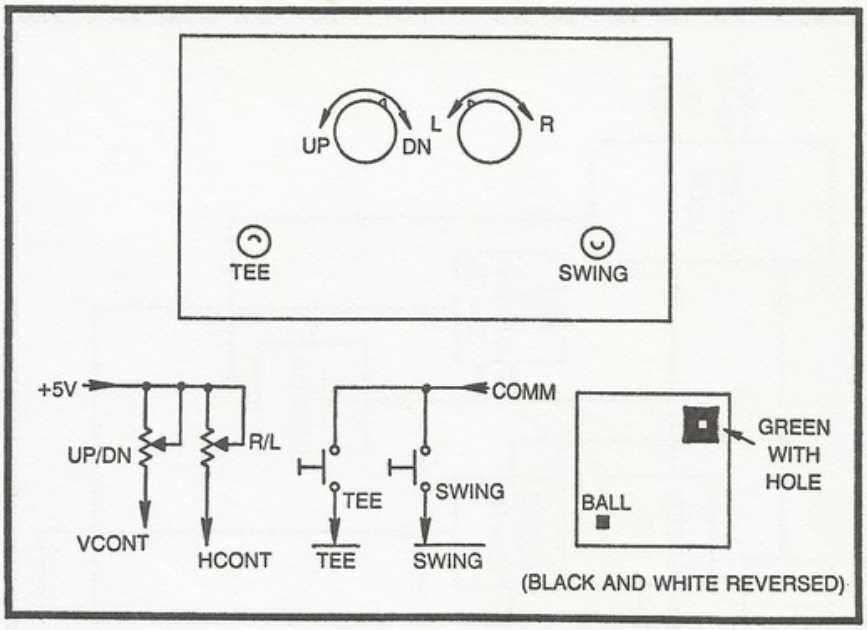


Fig. 8-16. Control panel diagram and schematic, and screen diagram for Golf.

TRAVEL TIMER and allows the slipping counter to move the ball in a direction and at a speed determined by the settings of the player's controls. When the ball timing interval is over, the BALL SPEED CONTROL sets the speed to zero, effectively stopping the ball on the screen. The ball then moves from that point when the player depresses the SWING button.

When the ball reaches the green, the BALL ON GREEN circuit generates logic levels that both change the color of the ball from white to black and slow down the ball speed and travel time.

The ball can be putted on the green, presumably toward the hole. And once the SCORE block senses contact between the ball and hole, it resets the BALL TRAVEL TIMER to stop ball motion and lock out the SWING switch. The ball cannot be moved then until the whole system is reinitialized by depressing the TEE switch.

Circuit Boards for Golf

The golf game described in this chapter requires four circuit boards: those shown in Figs. 8-18, 8-19, 8-20, and a slipping-counter board from Fig. 5-15. The functional block diagram in Fig. 8-17 and the wiring diagram in Fig. 8-22 can be quite helpful for learning about the circuit boards shown here.

The logic board in Fig. 8-18 includes the program counter, ball initialization control, ball travel timer, and A/D converter blocks shown on the functional block diagram.

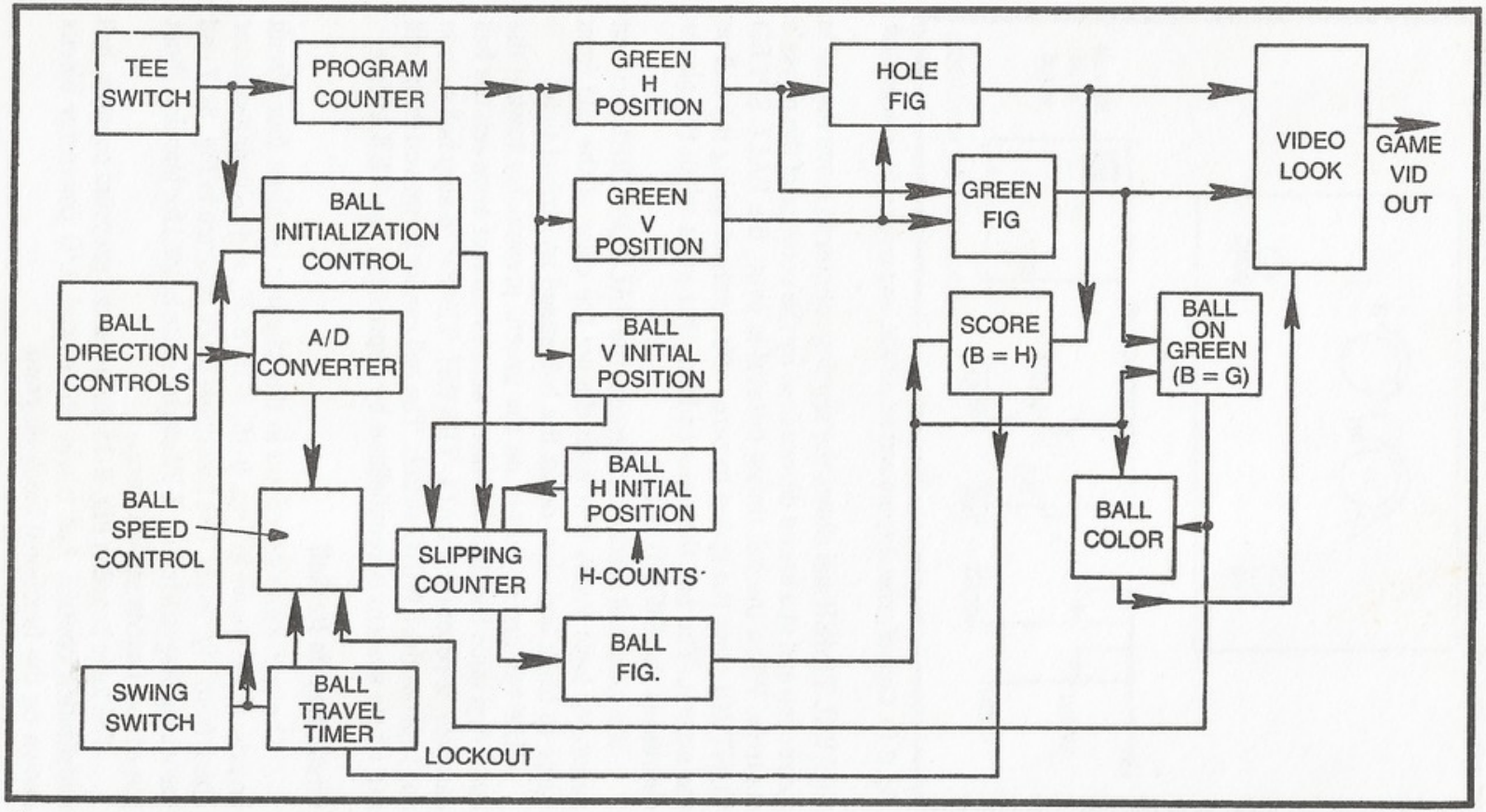


Fig. 8-17. Function block diagram for Golf.

Depressing the TEE pushbutton on the control panel generates a logic-0 level that is wired to the TEE input of this logic board. This action initiates a 10-ms monostable multivibrator action from IC5-A. The brief pulse from that IC increments the count of the program counter, IC9.

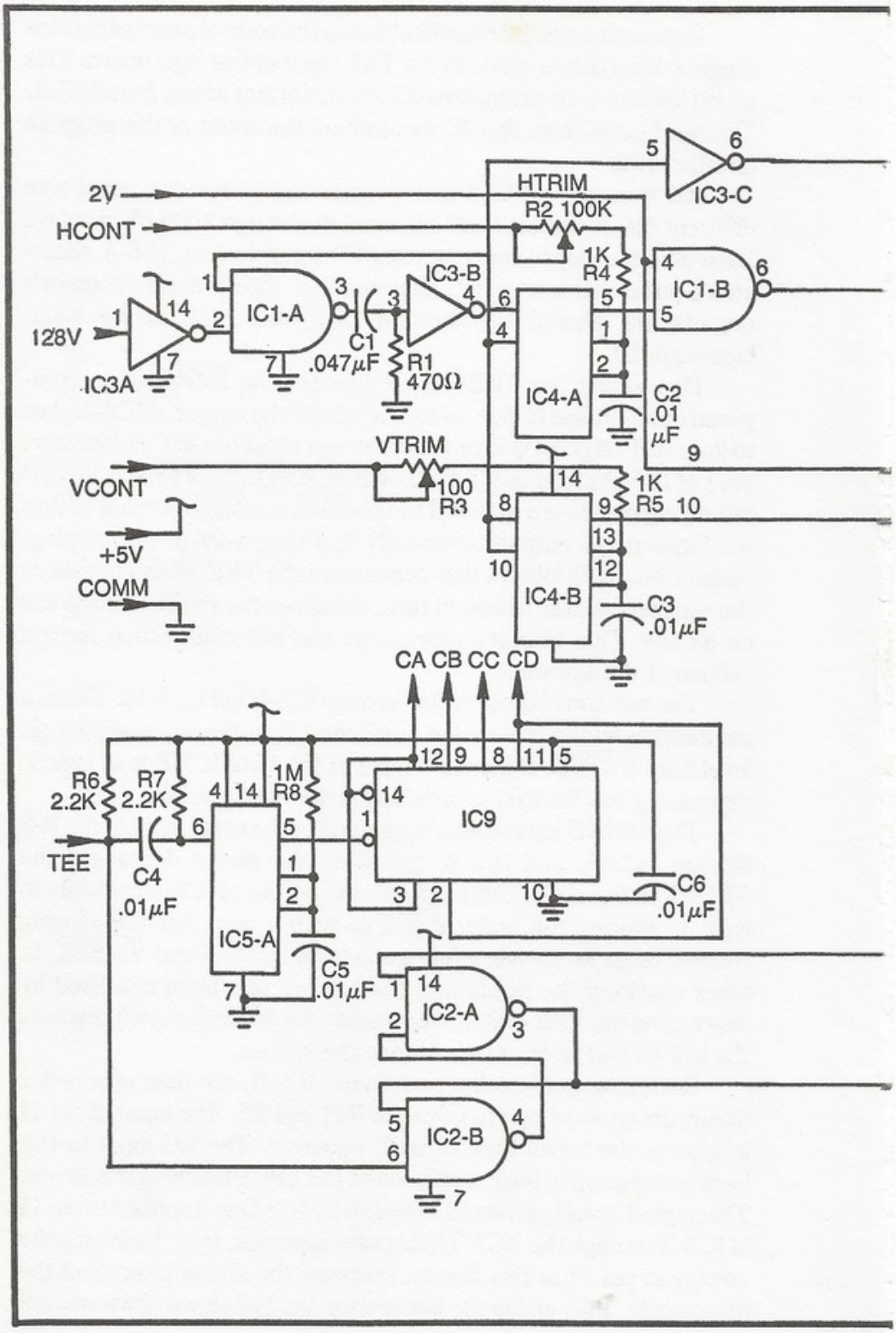
IC9 is simply a 4-bit binary counter that is wired to count nine different binary states, 0000 (decimal 0) through 1000 (decimal 8). Each time the player depresses his TEE pushbutton, IC5-A generates a pulse that increments that counter. The program counter's outputs are labeled CA through CD, where CA is the least-significant bit.

Depressing the TEE switch also sets an $\overline{R}$-$\overline{S}$ flip-flop, composed of IC2-A and IC2-B, to a state where the output of IC2-A goes to logic 0. This point is wired to the select input of a 2:1 multiplexer, pin 1 of IC6. As long as pin 1 of IC6 is at logic 0, $\overline{HINTP}$ and $\overline{VINTP}$ initializing pulses are directed to the $\overline{HML}$ and $\overline{VML}$ outputs of that IC. Since these outputs eventually find their way to the slipping-counter board, it follows that depressing the TEE switch initializes the slipping counter which, in turn, initializes the position of the ball on its tee. This circuitry represents the ball initialization control section of the system.

The ball-travel timer is built around IC5-B in Fig. 8-18. This is a monostable multivibrator that is initiated by a negative-going logic level from $\overline{SWING}$. If the $\overline{RST}$ input at pin 10 of IC5-B is at logic 1, depressing the SWING switch starts the timing action.

The SWING input is also connected to the reset input of the R-S flip-flop, IC2-A, and IC2-B. So when the player depresses the SWING button on his control panel, the output of IC2-A switches to logic 1, altering the state of IC6 in such a way that the slipping counter takes its loading information from $\overline{HMRST}$ and $\overline{VMRST}$. In other words, if the position of the ball has just been initialized by depressing the TEE switch, depressing the SWING switch releases the ball so that it can move across the screen.

Returning to the ball-travel timer, IC5-B, the time of travel is mainly determined by the values of R11 and C8. The input at pin 11 influences the travel time as well, however. The BG input to this logic board goes to logic 1 only when the ball is touching the green. That logic-1 level is inverted to logic 0 by IC3-D and applied to pin 11 of IC5-B through the BGT TRIM potentiometer, R12. Lowering the voltage at pin 11 in this fashion shortens the timing interval of the monostable, thus giving the impression the ball slows down when it touches the green.


2V
HCONT
128V
IC3A
IC1-A
C1
.047μF
R1
470Ω
IC3-B
HTRIM
R2 100K
1K
R4
IC3-C
IC1-B
IC4-A
C2
.01
μF
VTRIM
100
R3
VCONT
+5V
COMM
1K
R5
IC4-B
C3
.01μF
CA CB CC CD
R6
2.2K
R7
2.2K
1M
R8
TEE
C4
.01μF
IC5-A
C5
.01μF
IC9
C6
.01μF
IC2-A
IC2-B

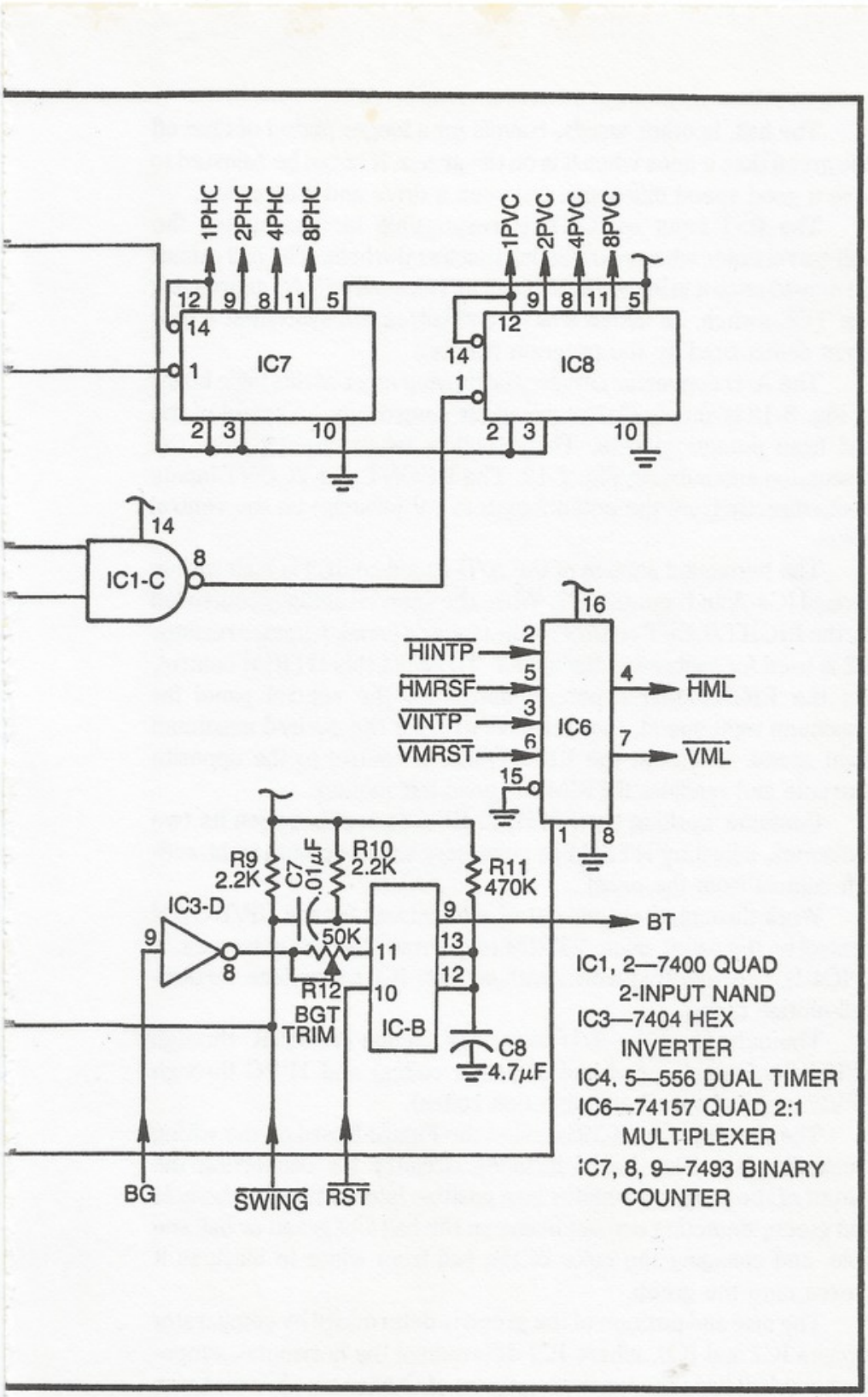


Fig. 8-18. Logic board schematic for Golf

The ball, in other words, travels for a longer period of time off the green than it does when it is on the green. R12 can be adjusted to give a good speed difference between a drive and putt swing.

The $\overline{RST}$ input to IC5-B is responsible for locking out the ball-travel timer whenever the ball touches the hole. The ball cannot be moved once it is in the hole. It can be released only by depressing the TEE switch, an action which reinitializes the system at a new point determined by the program counter.

The A/D converter circuitry occupying most of this logic board in Fig. 8-18 is simply that required for controlling the speed of the ball from potentiometers. The circuit is taken directly from the discussion surrounding Fig. 7-12. The HCONT and VCONT inputs come directly from the potentiometers (or joystick) on the control panel.

The horizontal portion of the A/D speed control is built mainly around IC4-A and counter IC7. While the speed is actually controlled by the RIGHT/LEFT control on the player's panel, trimmer resistor R2 is used for calibrating the speed. To adjust this HTRIM control, set the RIGHT/LEFT potentiometer on the control panel for maximum right speed, then trim HTRIM for the desired maximum right speed. Then set the RIGHT/LEFT control to the opposite extreme and readjust HTRIM for good left motion.

Continue working the RIGHT/LEFT control between its two extremes, adjusting HTRIM as necessary to get smooth right-and-left control from the panel.

Work through the same set of adjustments for the UP/DOWN control on the panel, using VTRIM to calibrate the two extremes. It is IC4-B, you see, that works with counter IC8 to produce vertical-ball-motion codes.

The outputs of the A/D converter section are 1PHC through 8PHC (horizontal speed and direction codes) and 1PVC through 8PVC (vertical speed and direction codes).

The circuit in Fig. 8-19 is called the Figure Board on the wiring block diagram. This board includes circuitry for converting the output of the program counter into position information for the hole and green, detecting contact between the ball and green or ball and hole, and changing the color of the ball from white to black as it moves onto the green.

The size and position of the green is determined by comparator circuits IC7 and IC9, where IC7 determines the horizontal component and IC9 fixes the vertical component. These two ICs compare board inputs CA through CD with horizontal- and vertical-count

signals from the Sourcebox unit. Recall that the "C" inputs come from the program counter on the logic board in Fig. 8-18.

These two comparators are programmed so that the horizontal positions of the green can be in any one of seven positions on the right-hand side of the screen. The vertical positions can be any one of four possible between the top and bottom. That all figures out to 28 possible positions, but the program counter allows only nine of them. It is left to the reader to analyze the programming as shown in Fig. 8-19, and perhaps alter it to suit his own notions of where the green should appear.

IC8 uses the "C" states from the program counter to position the ball when it is first set on the tee. There are seven possible positions between the top and bottom of the screen, but only one horizontal position that is determined by the H-count inputs at IC5-C.

The initial position information for the ball is transformed into a brief pulse by sets of pulse generators, and then leaves the board as slipping-counter initializing pulse BHINTP and BVINTP.

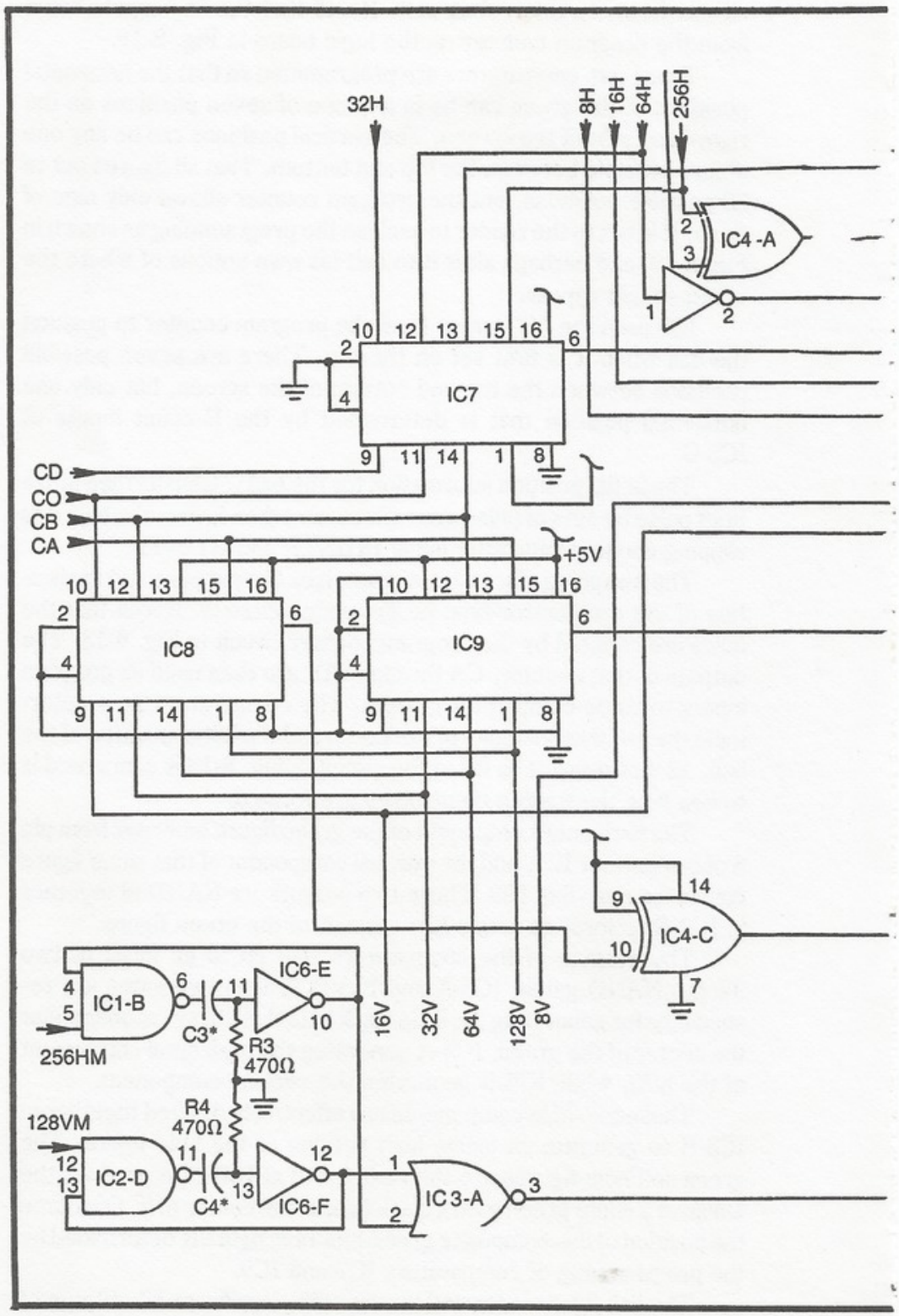
The foregoing discussion summarizes the purpose and application of the comparator-type programming circuits. Recall that the holes are counted by the program counter circuit in Fig. 8-18. The outputs of that counter, CA through CD, are then used as program inputs to three comparator circuits. The comparators then determine the relative positions of the green and the initial position of the ball. All that remains as far as the circuit in Fig. 8-19 is concerned is to see how the figures themselves are formed.

The horizontal component of the green figure emerges from pin 6 of comparator IC7, and the vertical component of that same figure comes from pin 6 of IC9. These two outputs are NANDed together in IC2-B to form an active-low version of the green figure.

The outputs of the comparators also go to an input of two 3-input NAND gates, IC5-A and IC5-B. These two gates are responsible for generating the little black hole that always appears near the center of the green. IC5-A generates the horizontal component of the hole, while IC5-B generates the vertical component.

These two hole components are effectively ANDed together in IC3-B to generate an active-high version of the hole figure. The green and hole figures are then combined at IC3-C to produce the image of a white green with a black hole in the center of it. Of course the position of this composite green-and-hole figure is determined by the programming of comparators IC7 and IC9.

The ball figure is formed by the most-significant-bit outputs of the slipping-counter board. Each time the horizontal-slipping


32H
8H
16H
64H
256H
IC4 -A
IC7
CD
CO
CB
CA
+5V
IC8
IC9
IC4-C
IC1-B
IC6-E
256HM
C3*
R3
470Ω
R4
470Ω
128VM
IC2-D
C4*
IC6-F
IC 3-A
16V
32V
64V
128V
8V

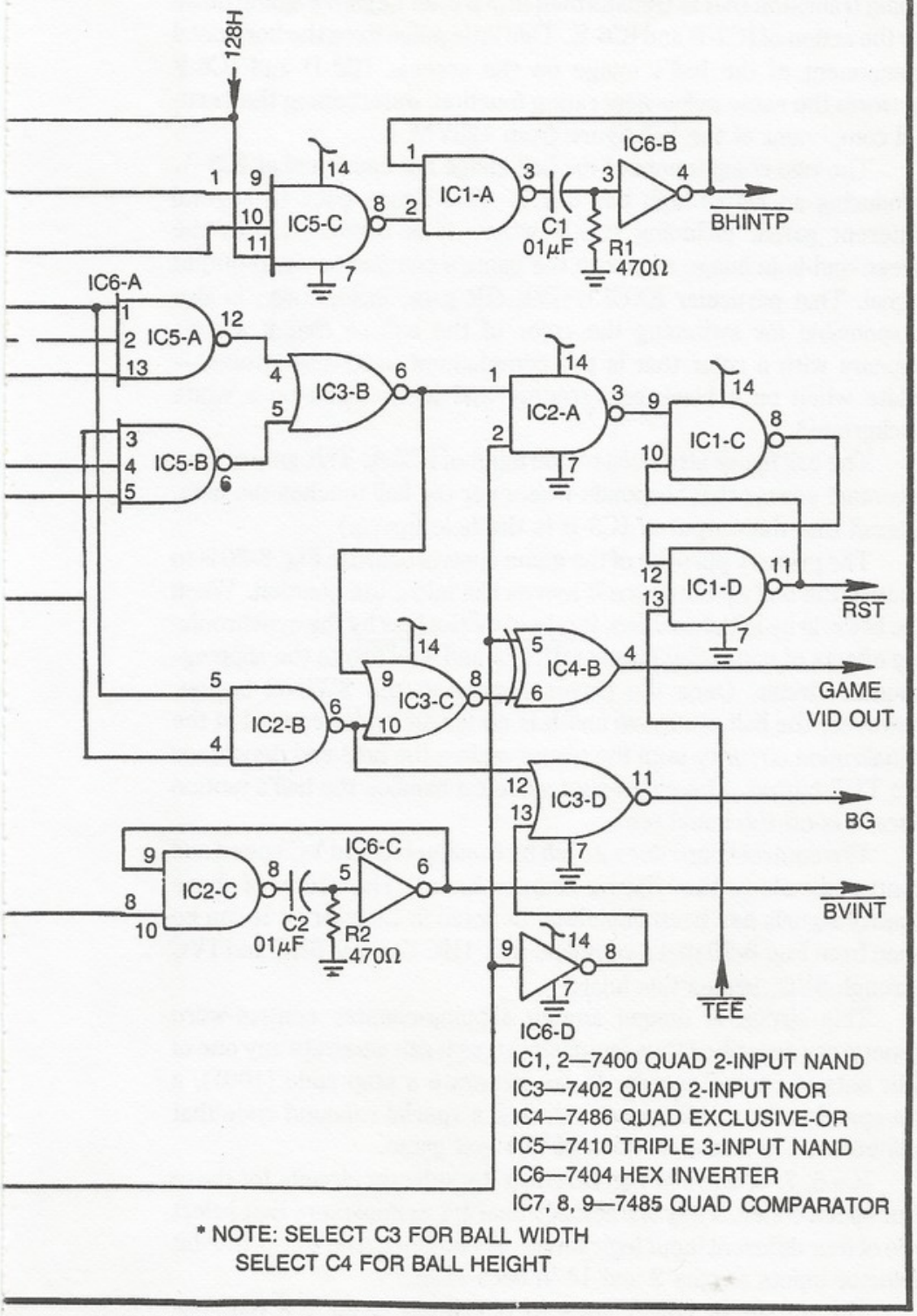


Fig. 8-19. Figure board schematic for Golf

counter is reset, for instance, the 256HM signal makes a negative-going transition that is transformed into a brief negative-going pulse by the action of IC1-B and IC6-E. This little pulse fixes the horizontal component of the ball's image on the screen. IC2-D and IC6-F perform the same pulse-generating function, determining the vertical component of the ball figure from 128VM.

The two components of the ball image are combined at IC3-A, producing an active-high ball figure. This figure goes to several different gates, including IC6-B where it is combined with the green-and-hole image to create the game's composite video-output signal. That particular EXCLUSIVE-OR gate, incidentally, is also responsible for switching the color of the ball so that it always appears with a color that is the complement of its background—white when on a black background, and black when on a white background.

The ball figure also goes to one input of IC2-A. This gate acts as a contact sensor that responds whenever the ball touches the hole. (Recall that the output of IC3-B is the hole figure.)

The primary purpose of the game control board in Fig. 8-20 is to control the ball motion once it leaves the initial tee position. When the ball is in its initial position, it is held motionless by the synchronizing effects of initializing pulses ($\overline{\text{HINTP}}$ and $\overline{\text{VINTP}}$) to the slipping-counter circuit. Once the player depresses the SWING button, however, the ball can move and it is no longer under control of the initialization circuitry until the player makes the hole and depresses the TEE button. The game-control board handles the ball's motion once it is off the initial tee.

The control board does its job by feeding HC and VC speed and control signals to the slipping-counter board. The action of these control signals has been described in detail in Chapter 5. It can be seen from Fig. 8-20 that a complete set, 1HC through 8HC and 1VC through 8VC, leaves this board.

This circuit is unique among slipping-counter control-word generators described thus far inasmuch as it can generate any one of four sets of control words. It can generate a stop code (1001), a full-speed code, a half-speed code and a special rebound code that will be used in a later version of this golf game.

ICs 6, 7, 8, and 9 in Fig. 8-20 are the selector circuits for these four speed codes. They are actually dual 4:1 multiplexers that select one of four different input logic levels, according to the status of 2-bit selector inputs at pins 2 and 14 in each case.

The output of IC2-A goes to one input of an $\overline{\text{R}}$-$\overline{\text{S}}$ flip-flop composed of IC1-C and IC1-D. The $\overline{\text{RST}}$ output of that flip-flop is set

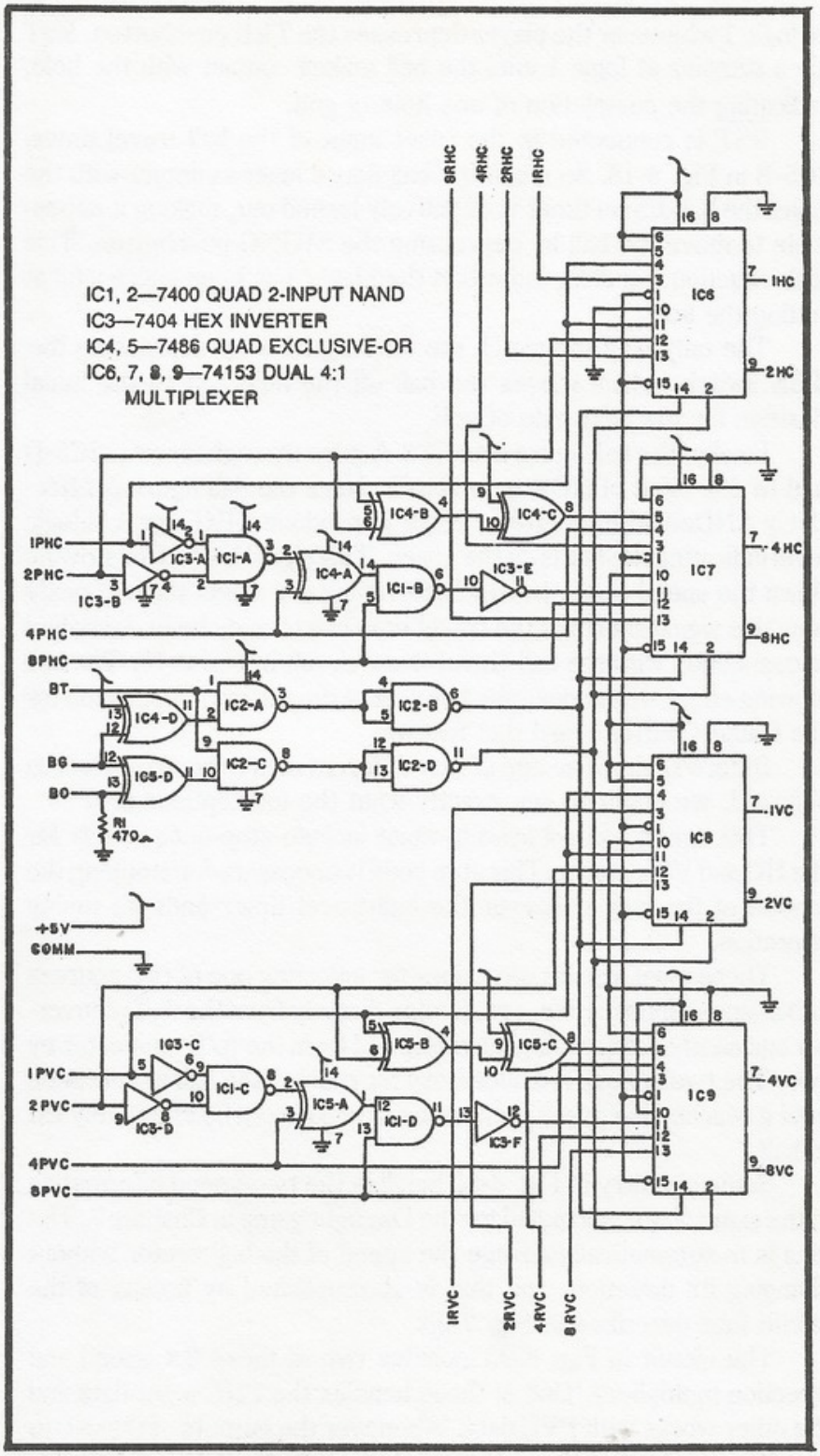


Fig. 8-20. Game Control board schematic for Golf.

to logic 1 whenever the player depresses the TEE pushbutton. $\overline{RST}$ then remains at logic 1 until the ball makes contact with the hole, indicating the completion of one hole of golf.

$\overline{RST}$ is connected to the reset input of the ball travel timer, IC5-B in Fig. 8-18. So when the ball figure makes contact with the hole, the ball-travel timer is effectively locked out, making it impossible to move the ball by depressing the SWING pushbutton. This lack of action is a clear indication the player has been successful at hitting the hole.

The only way to unlatch the $\overline{RST}$ signal is by depressing the TEE switch, which moves the ball off the hole and to the initial position for the next hole of golf.

Finally, the ball figure from IC3-A goes through inverter IC6-D and to one input of NOR gate IC3-D. Here the ball figure is effectively ANDed with the green figure to produce a BG signal, a logic level indicating the ball is on the green. This signal is used for slowing down the speed of the ball and shortening the travel time. Exactly how this signal shortens the travel time has already been described in connection with the ball-travel timer circuit in Fig. 8-18. The ball slowing effect will be described as part of the theory of operation for the Game Control board that follows.

Before seeing *how* one of four different sets of control words is selected, we ought to see exactly what the four options are.

This speed control system must include stop-code words for the HC and VC outputs. This stop code is necessary for stopping the motion of the ball whenever the ball-travel timer ends its timing operation.

There must also be provisions for selecting one of two sources of ball-speed information, one coming directly from the A/D converter and another that multiplies the speed from the A/D converter by two. The two speeds are necessary for running the ball at full speed until it reaches the green, where the speed should be effectively cut in half.

Some circuitry in Fig. 8-20 handles the two-speed information in the same way it is handled for the Dogfight game in Chapter 7. The idea is to automatically change the speed of the ball vector without changing its direction, and that is accomplished by means of the circuit first described in Fig. 7-23.

The circuit in Fig. 8-20 includes two of these 2× speed and direction multipliers. One of these handles the PHC input data and the other works with PVC data. Whenever the outputs of these two circuits are directed through the selectors to the HC and VC out-

puts, the ball travels twice as fast as it does when these circuits are not selected.

In this particular Golf game, the ball's speed data is taken from the 2× vector multiplier circuits until it touches the green. As soon as the ball touches the green, the speed and direction data is selected directly from the PHC and PVC inputs, giving the impression the ball speed is cut in half.

These three sources of speed and direction information—the stop code, 2× vector speed, and normal speed—are the only ones used in this particular game. There are provisions on the board, however, for a fourth option.

The fourth option is a rebound effect that is called up whenever the ball strikes an obstacle on the course. Since the obstacles are not included in this game format, the rebound circuitry is not used. A later chapter, however, takes up the subject of rebound effects, and the obstacle feature will be added to the Golf game at that time.

For the time being, it is sufficient to say that the ball speed and direction selectors can take their information from the RHC and RVC inputs. The obstacle contact input, BO, is normally pulled down to logic 0 by R1; so as long as no connection is made to that input, the obstacle rebound inputs (the RHC and RVC inputs) are never selected.

The circuit in Fig. 8-20 thus has provisions for selecting one of four sources of speed and direction information, but only three are used in this version of the Golf game.

The sources of ball speed and direction information that reach the HC and VC outputs are determined by the BT and BG inputs. For the present time, the special rebound-selecting input, BO, will be ignored. The function table in Fig. 8-21 shows the relationship between the BT and BG inputs and the ball-speed and direction information selected at the HC and VC outputs.

Recall that the BT signal is an active-high logic level that stands at logic 1 as long as the ball-travel timer is activated. This logic-1 level indicates the ball is supposed to be in motion. The ball is not supposed to be moving as long as BT=0.

The BG input to the control board is a logic level that indicates whether or not the ball is on the green. When the ball is NOT on the green, BG=0, but as soon as the ball makes contact with the green, BG switches to logic 1.

According to the function table in Fig. 8-21, then, the circuit selects the slipping-counter stop code (1001) whenever BT=0. The ball, in other words, should stop moving whenever the ball-travel timer is not timing a SWING operation.

BT	BG	HC OUTPUTS	VC OUTPUTS	BALL MOTION
0	0	STOP CODE	STOP CODE	STOP
0	1	STOP CODE	STOP CODE	STOP
1	0	2X PHC CODE	2X PVC CODE	FAST—BALL NOT ON GREEN
1	1	PHC CODE	PVC CODE	SLOW—BALL ON GREEN

NOTE: THESE FUNCTIONS ASSUME THE REBOUND SELECT INPUT, BO, IS FIXED AT LOGIC 0

Fig. 8-21. Functions for Golf speed control.

The circuit then selects the 2× vector-multiplier function whenever BT=1 and BG=0. Interpreting this in terms of game operations, that means the ball is in flight, but has not yet touched the green. The ball travels in a direction, but at twice the speed, dictated by the outputs of the A/D converter.

The selector circuit then takes its data directly from the A/D converter whenever BT=1 and BG=1. This status occurs whenever the ball is moving and touching the green. The overall effect is that it travels in the same direction, but at half the speed, it has during free flight off the green.

Golf Wiring Block Diagram

Figure 8-22 shows the wiring block diagram for this particular Golf game. The system requires four circuit boards of the type specified throughout this book (Radio Shack's 276-153 40-pin plug-in boards).

The power-supply loading can be balanced reasonably well by operating the slipping-counter board from one supply and the three remaining boards and control panel from another 1-A supply.

The obstacle inputs are not shown here because they should be left unconnected for this version of golf. A later chapter will show how to use these RHC and RVC connections on the control board.

AMBUSH

Ambush is a one-player game that takes advantage of position programming in a rather unique way. The game is a variation of the old arcade game where the player is supposed to shoot at bad guys

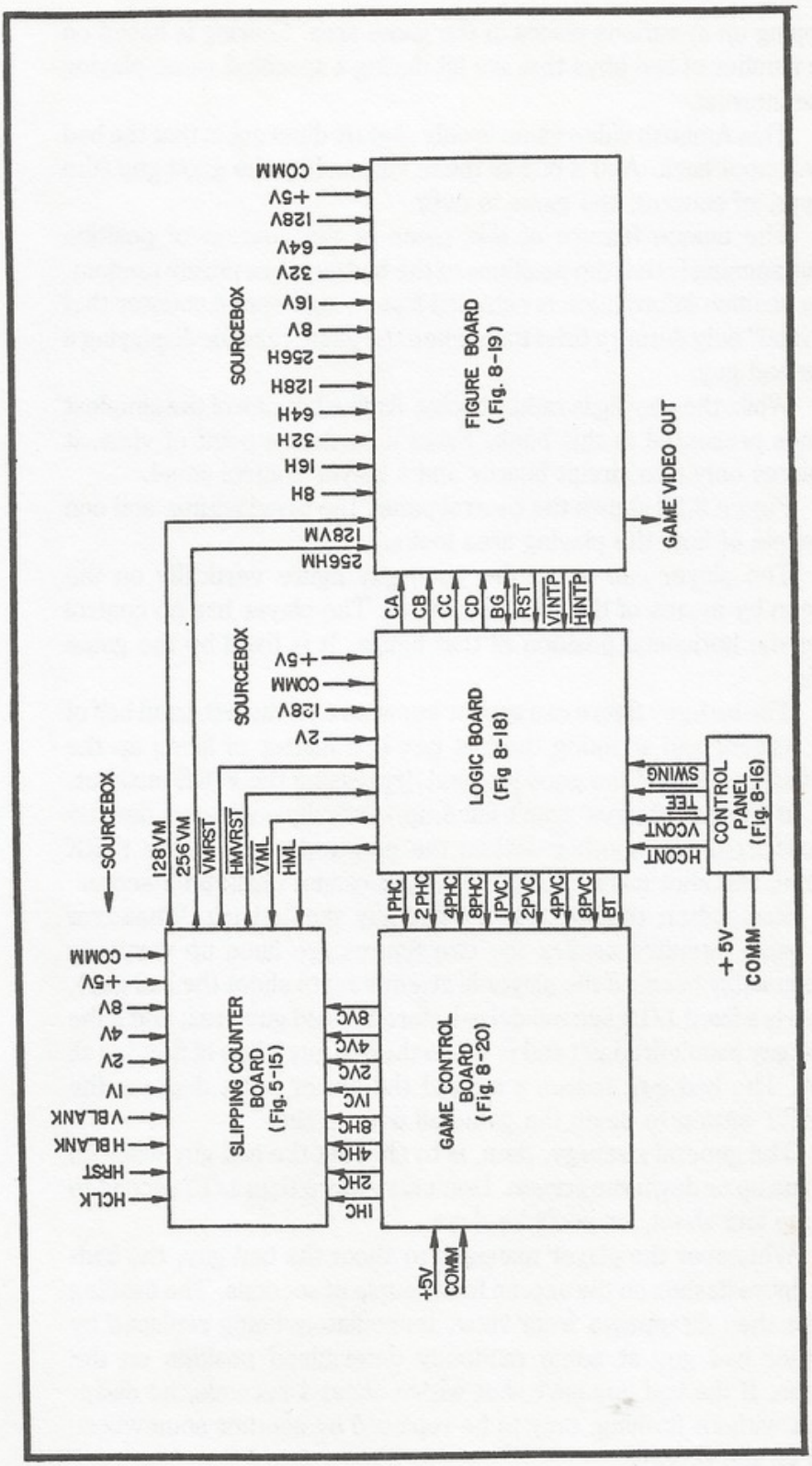


Fig. 8-22. Wiring block diagram for Golf.

popping up at various places in the game area. Scoring is based on the number of bad guys that are hit during a specified game-playing time interval.

This Ambush video game is only slightly different in that the bad guys shoot back. And if one of these villians hits the good guy (the player, of course), the game is over.

The unique feature of this game in the context of position programming is that the positions of the bad guys are purely random. The position information is selected from a high-speed counter that is "read" only during a brief time when the game calls for displaying a new bad guy.

While the playing is rather tricky, Ambush is one of the simplest games presented in this book. From a hardware point of view, it requires only two circuit boards and a player control panel.

Figure 8-23 shows the control panel, the panel wiring, and one example of how the playing area looks.

The player can move the good-guy figure vertically on the screen by means of the MOVE control. The player has no control over the horizontal position of that figure. It is fixed by the game logic.

The bad-guy figure can appear anywhere on the left-hand half of the screen, and shooting the bad guy is a matter of lining up the vertical position of the good guy and depressing the FIRE pushbutton. If the two figures aren't lined up vertically—situated directly across from one another—when the player depressed the FIRE button, the shot misses, and the player cannot chalk up a score.

Remember, though, that the bad guy shoots back. Whenever the game circuitry senses the two figures are lined up vertically (presumably because the player is attempting to shoot the bad guy), there is a fixed 1/10 second delay before the bad guy fires; and if the good guy hasn't fired yet and is still in the bad guy's line of fire, it's all over. The bad guy scores a hit and the player must depress the START button to begin the game all over again.

The general strategy, then, is to shoot at the bad guy while on the run up or down the screen. Don't take more than 1/10 second to line up and shoot, or you'll be dead.

Whenever the player manages to shoot the bad guy, the bad-guy figure flashes on the screen for a couple of seconds. The flashing figure then disappears from view, immediately being replaced by another bad guy at some randomly determined position on the screen. If the bad guy isn't shot within about 4 seconds, he disappears without flashing, only to be replaced by another somewhere else on the screen.

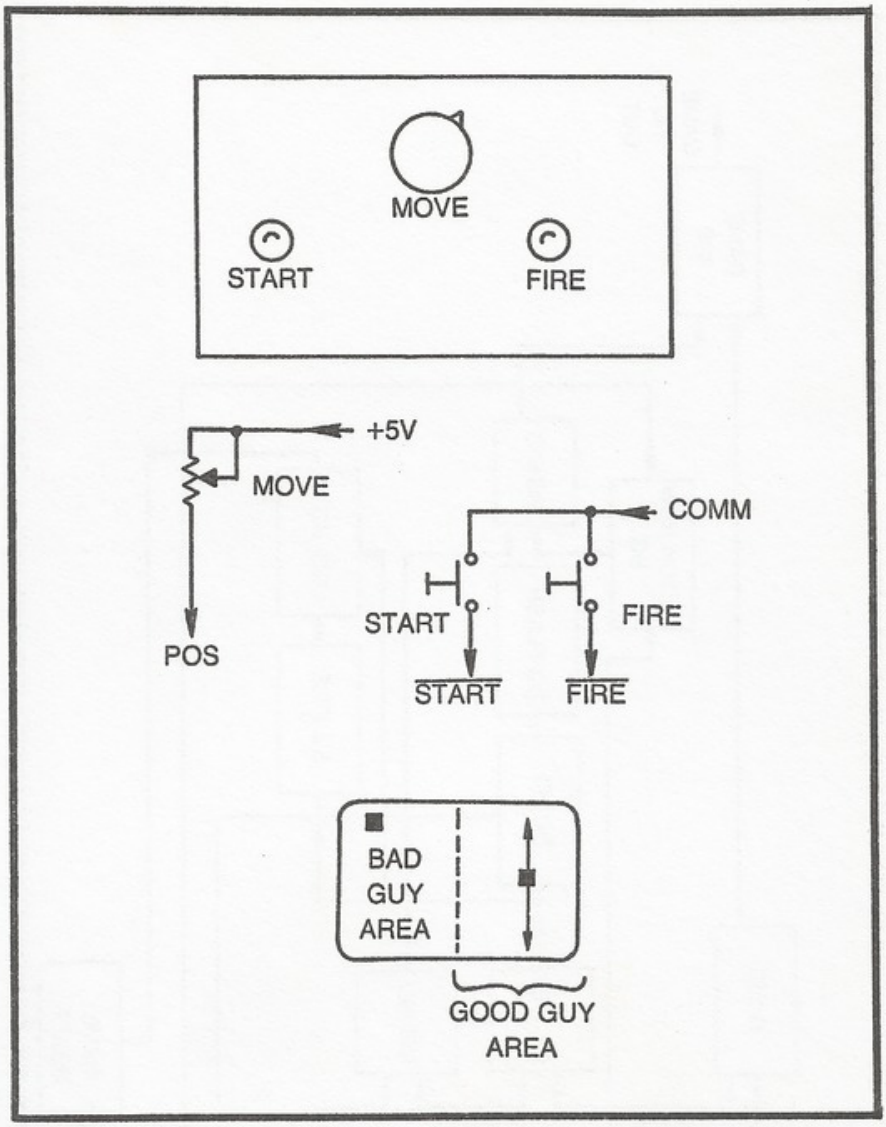


Fig. 8-23. Control panel diagram and schematic, and screen diagram for Ambush.

Providing the good guy isn't hit, the game runs for about 3 minutes before it is reset. The player then totals up the number of successful shots, and depresses the START button to begin another game.

Ambush Block Diagram and Schematics

The block diagram and schematics for the two special circuit boards are shown in Figs. 8-24 through 8-26. The wiring block diagram is in Fig. 8-27. Use all of these figures when studying the theory of operation of this particular video game.

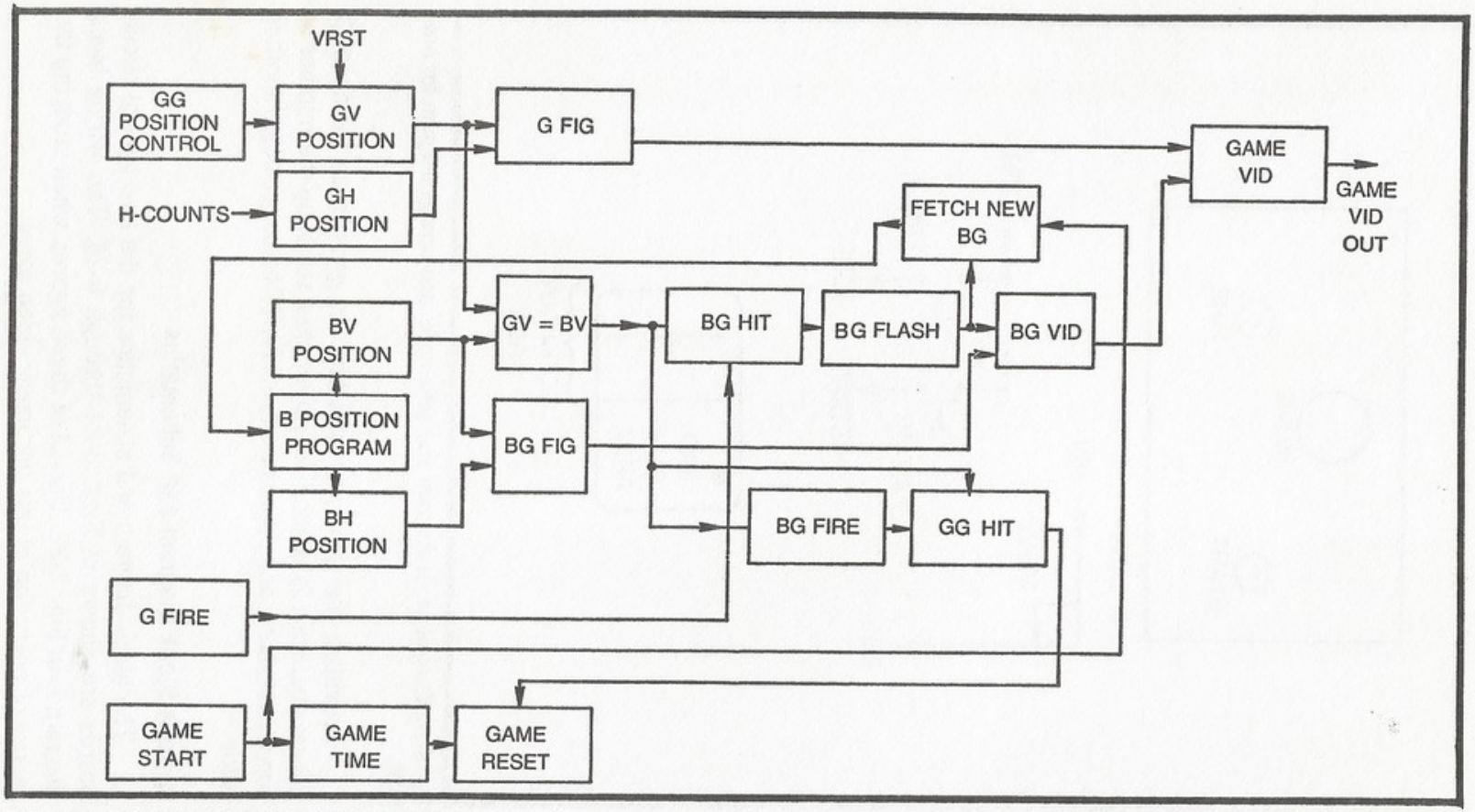


Fig. 8-24. Function block diagram for Ambush.

Whenever the player depresses the START button on the control panel, a negative-going logic level at pin 5 of IC1-B in Fig. 8-25 sets the state of a $\overline{R}$-$\overline{S}$ flip-flop. The flip-flop responds by showing a negative-going edge at pin 3 of IC1-A. That edge-triggers a pulse-generator circuit composed of IC1-C and IC4-A. The brief negative-going pulse from that pulse generator triggers IC6-A, the game's 3-minute master timer circuit. Depressing the START button thus initiates a 3-minute monostable timing interval that can be interrupted only by a logic-0 level to pin 4 of that timer device. Of course that $\overline{GHIT}$ input indicates the good guy has been hit.

Now note that the output of the master game timer (IC6-A, pin 5) goes to a pulse generator and then to the resetting input of the $\overline{R}$-$\overline{S}$ flip-flop (pin 1 of IC1-A). The game is thus reset whenever the output of IC6-A makes a transition from logic 1 to 0. And that occurs when (1) the 3-minute game time expires or (2) the good guy is shot.

Depressing the START button also sends a logic 1 level to pin 8 of IC6-B. This enables the monostable so that the good guy can fire a shot by depressing the FIRE button. The shot is timed by IC6-B so that it lasts only about 1/10 second, but if GV=BV (good-guy vertical equals bad-guy vertical), the output of IC1-D registers a hit in favor of the good guy.

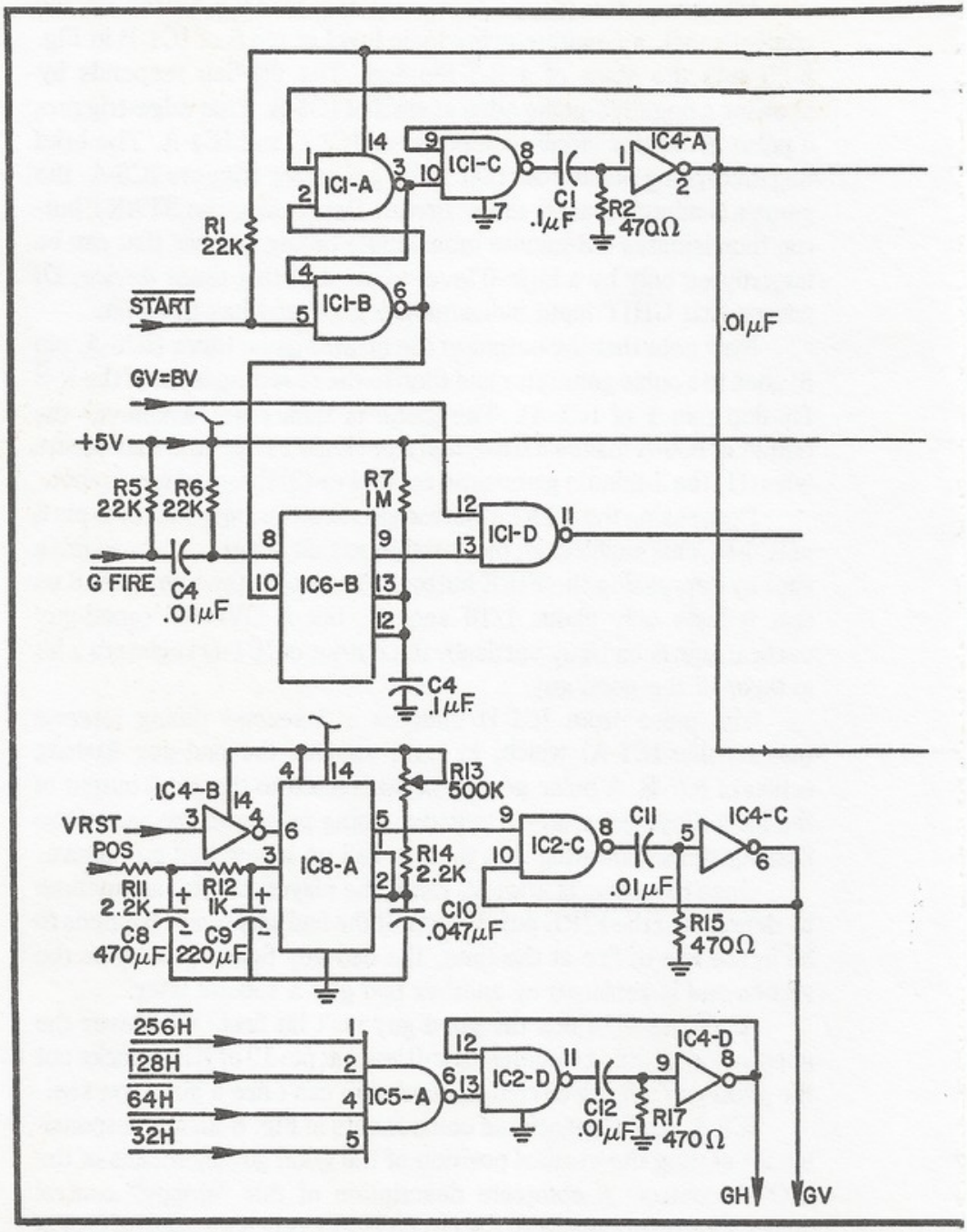
The pulse from IC1-D initiates a 1-second timing interval (monostable IC7-A) which, in turn, enables the bad-guy flashing action of IC7-B. A pulse generator connected to the pin-5 output of the flash timer generates a negative-going pulse and the *end* of the flashing time, indicating it is time to call up a new bad guy figure.

Once the game is started, then, the player can fire at any time by depressing the FIRE pushbutton. If the bad-guy figure happens to be in the line of fire at the time, the bad-guy figure flashes on the screen and is replaced by another bad guy a second later.

All of this assumes the good guy isn't hit first. Whenever the good guy is hit, the resulting logic-0 level at pin 10 of IC6-B locks out the good guy's firing circuit. A dead man can't fire a gun, you see.

IC8-A and its associated components in Fig. 8-25 are responsible for setting the vertical position of the good guy by means of the MOVE control. A complete description of this "sloppy" control appears in conjunction with the circuit in Fig. 5-6. The value of C11 in this particular circuit can be selected to fix the vertical size of the good-guy figure.

IC5-A fixes the horizontal part of the good-guy figure. Capacitor C12 can be selected to scale the width of that figure. The vertical and horizontal components of the good-guy figure are thus available from pin 6 and pin 8 of IC4.



IC5-B in Fig. 8-25 determines when a new bad-guy figure is to be selected. The input connections to this IC show the new figure is selected under any one of three game conditions: when the game is first started, after a dead bad guy goes on to the hereafter, and after a bad guy has managed to survive on the screen for about 5 seconds.

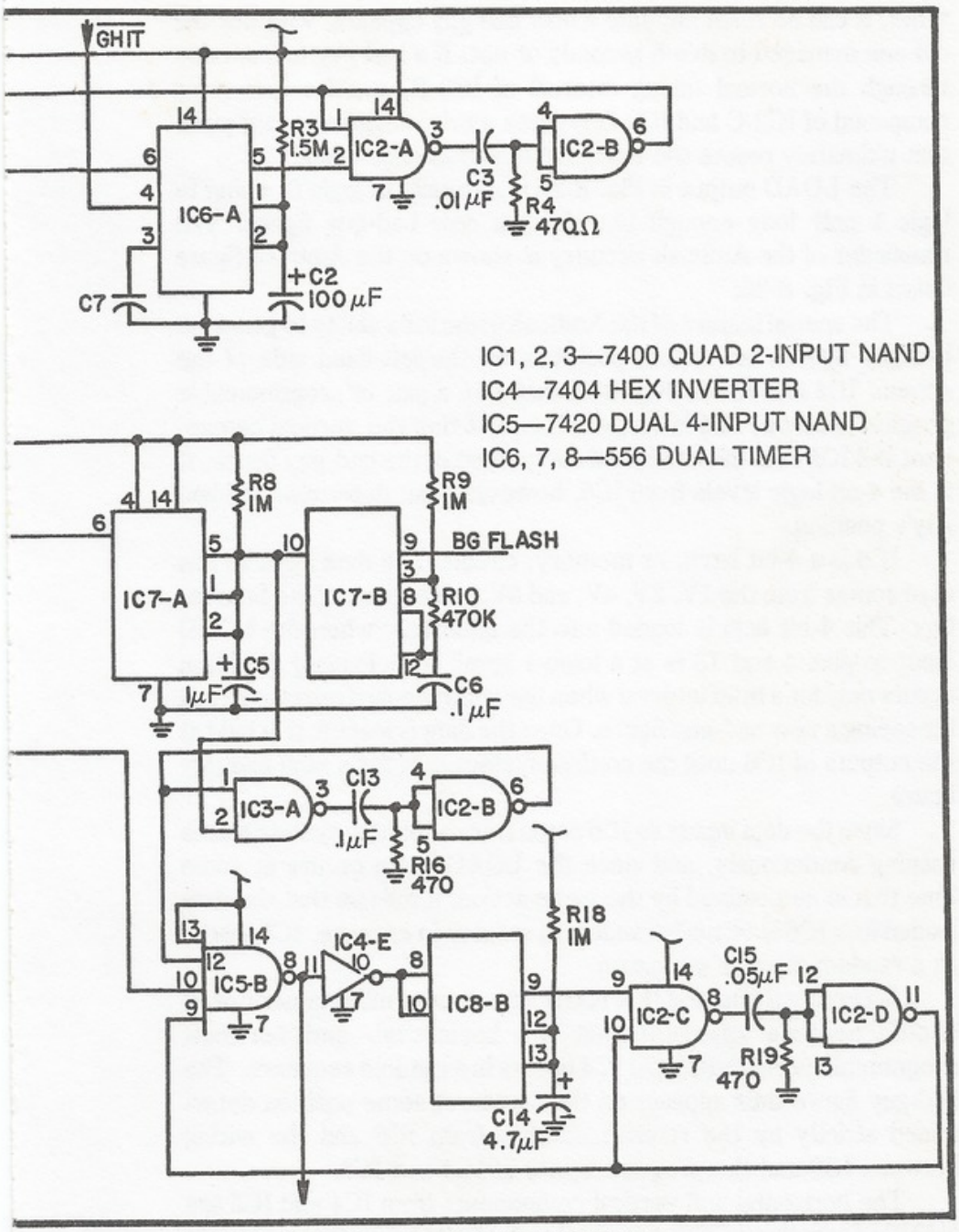


Fig. 8-25. Control board schematic for Ambush.

The circuits for indicating the start of a new game and the end of a bad guy's death scene have been described already.

IC8-B is responsible for keeping a bad guy on the screen for no more than 5 seconds. This timer is initiated whenever a new bad guy appears on the screen, but since it is connected as an interruptable

timer, it can be reset anytime a new bad guy appears, whether the old one managed to live 5 seconds or not. If a bad guy can survive through the normal timing interval of IC8-B, a pulse generator composed of IC2-C and IC2-D creates a brief negative-going pulse that ultimately resets the monostable action once again.

The LOAD output in Fig. 8-25 is normally at logic 0, rising to logic 1 only long enough to call up a new bad-guy figure. The remainder of the Ambush circuitry is shown on the Ambush-figure board in Fig. 8-26.

The special feature of the Ambush game is its ability to generate bad-guy figures at random positions on the left-hand side of the screen. IC4 and IC5 in Fig. 8-26 make up a pair of programmable position controls. IC4 is responsible for fixing the vertical component and IC5 sets the horizontal component of the bad-guy figure. It is the 4-bit logic levels from IC6, however, that determine the bad guy's position.

IC6 is a 4-bit latch, or memory, circuit. The data input in this case comes from the 1V, 2V, 4V, and 8V connections of the Source-box. This 4-bit data is loaded into the latch only when the LOAD input to pins 4 and 13 is at a logic-1 level. This loading condition occurs only for a brief interval when the control board operations call for seeing a new bad-guy figure. Once the data is loaded, it is held at the outputs of IC6 until the control system calls for a new bad-guy figure.

Since the data inputs to IC6 come from a counter system that is running continuously, and since the LOAD pulse occurs at some time that is determined by the game action, it follows that the data loaded into IC6 is virtually random in nature. In essence, IC6 works as a random-number generator.

Accepting the notion that IC6 is a random-number generator, it should become apparent that the horizontal- and vertical-programming data to IC5 and IC4 occurs in a random sequence. The bad-guy figure thus appears on the screen at some position determined strictly by the random number from IC6 and the wiring between IC6 and the program inputs of IC4 and IC5.

The horizontal and vertical components from IC4 and IC5 are NANDed together in IC1-A to create a bad-guy figure that measures 16V×16H. This image is passed through IC2-A, where it can be flashed whenever the good guy scores a hit. The horizontal and vertical components of the good-guy figure are combined in IC2-C. And after that, the two figures are effectively ORed together by IC2-B and IC1-B to create the game's video output.

All that remains to be discussed is the way the bad guy knows he is lined up with the good guy so he can fire a shot. IC1-C senses the alignment of the two opposing figures and generates a negative-going pulse that initiates monostable timer IC7-A. This particular timer is set for 0.1 second. And at the end of that time, a second 0.1-second timer (IC7-B) is initiated.

IC1-C senses alignment of the two figures on the screen, IC7-A inserts a 0.1-second delay, and IC7-B is responsible for making the bad guy fire a shot. IC1-D in Fig. 8-26 normally shows a fixed logic-1 output, but if the bad guy fires while he is lined up with the good guy, IC1-D senses this fact and generates a $\overline{\text{GHIT}}$ output.

Recall that a $\overline{\text{GHIT}}$ pulse resets the entire game system, making it impossible for the good guy to fire another round until the player depresses the START pushbutton. Without the 0.1-second delay inserted by IC7-A, the good guy wouldn't stand a chance of shooting anyone.

Ambush Wiring Diagram

The wiring diagram in Fig. 8-27 indicates this is a rather simple and inexpensive game to set up. The only calibration adjustment is fixing the value of R13 in Fig. 8-25. This trimpot works with the MOVE control to set the range of motion of the good-guy figure.

R13 should be adjusted so that moving the MOVE control between its two extremes moves the good-guy figure to the top and bottom of the screen.

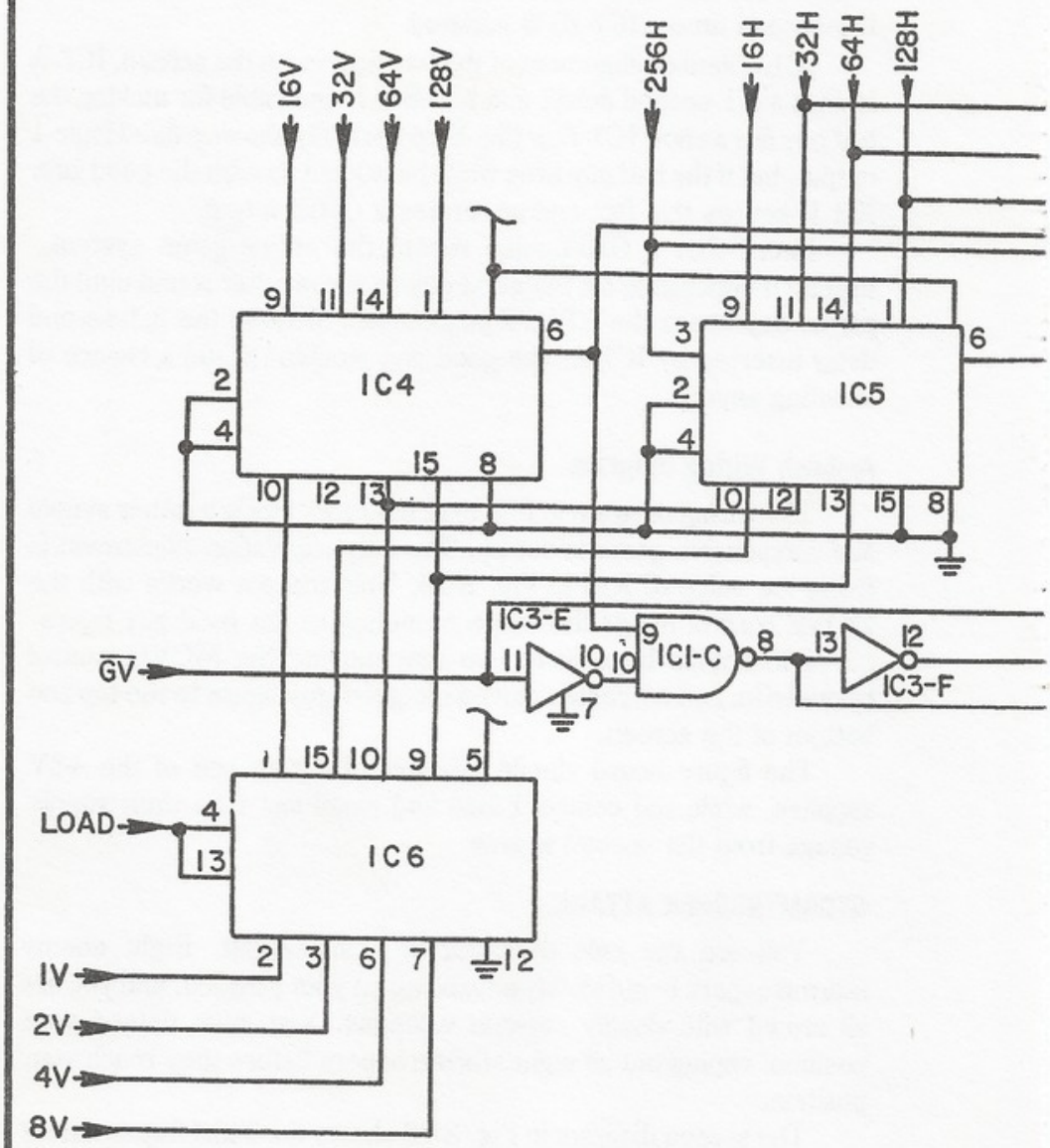
The figure board should be operated from one of the +5V supplies, while the control board and panel can take their supply voltage from the second source.

STORMTROOPER ATTACK

You are the sole defender of a spacecraft. Eight enemy stormtroopers begin slowly advancing on your position, and you are all armed with deadly ray-gun weapons. You must defend your position, wiping out all eight stormtroopers before they reach your position.

The screen diagram in Fig. 8-28 shows the initial formation for this Stormtrooper game. You, the good guy, are located at the bottom of the screen, and your motion is limited to that part of the screen. The eight stormtroopers move straight down the screen toward you at the rate of about one-screen-per-71 seconds. In other words, you have only about 60 seconds to kill them off.

The control panel includes a MOVE control that lets the player move the good guy back and forth along the bottom of the screen, a



IC1—7400 QUAD 2-INPUT NAND
IC2—7402 QUAD 2-INPUT NOR
IC3—7404 HEX INVERTER

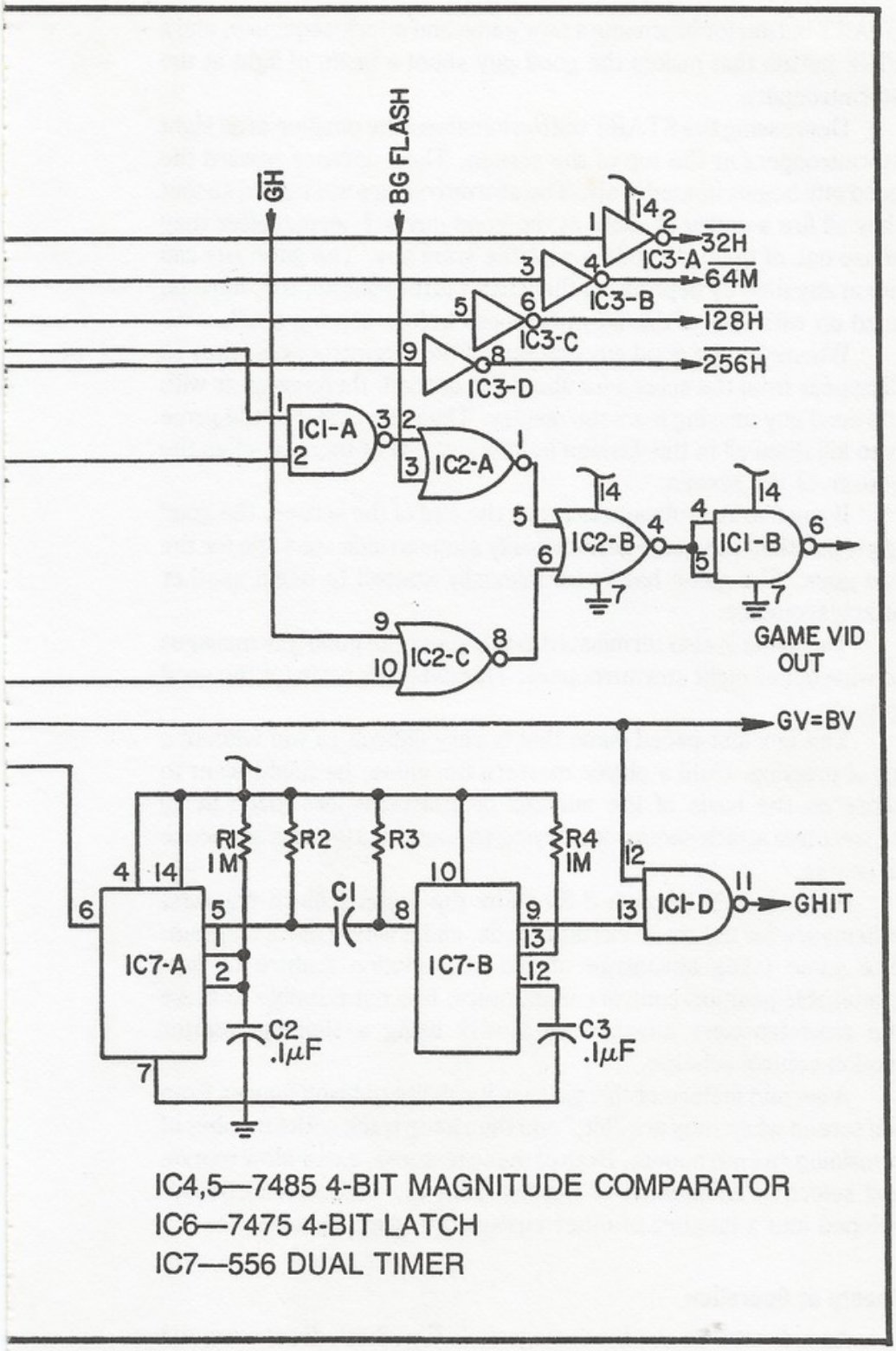


Fig. 8-26. Figure board schematic for Ambush

START button for beginning a new game and attack sequence, and a FIRE button that makes the good guy shoot a beam of light at the stormtroopers.

Depressing the START button initializes the position of all eight stormtroopers at the top of the screen. Their advance toward the good guy begins immediately. The stormtroopers are rigged so that they all fire a volley of shots at the good guy 0.1 second after they sense one of them is lined up with the good guy. The good guy can fire at any time by depressing the FIRE button, but he, too, must be lined up with one of the stormtroopers before scoring a kill.

Whenever the good guy hits one of the stormtroopers, they all disappear from the screen for about 0.5 second, then reappear with the dead guy missing from the display. The main object of the game is to kill them all in this fashion before any one of them reaches the bottom of the screen.

If any live stormtroopers reach the end of the screen, the good guy's position, the game automatically stops to indicate a win for the bad guys. The game has to be manually started to begin another attack sequence.

The game is also terminated in the event the good guy manages to wipe out all eight stormtroopers. This indicates a win for the good guy.

This is a fast-paced game that is very difficult to win without a lot of practice. Until a player masters the game, he might want to score on the basis of the number of stormtroopers killed in 10 consecutive attack sequences, trying to beat this 10-game sequence of scores.

Figures 8-29 through 8-33 show the general block diagram, schematics for the three circuit boards, and a wiring block diagram. The game takes advantage of the slow-motion feature of programmable position-control comparators. It is not possible to make the stormtroopers advance so slowly using a slipping-counter motion-control scheme.

A second feature of this game is its ability to blank figures from the screen when they are "hit," and then keep track of the number of remaining stormtroopers. Both of these features, extra-slow motion and selective elimination of figures from the screen, can be developed into a number of other custom TV-game systems.

Theory of Operation

As indicated on the block diagram in Fig. 8-29, depressing the START button both starts S MOTION and resurrects all previously

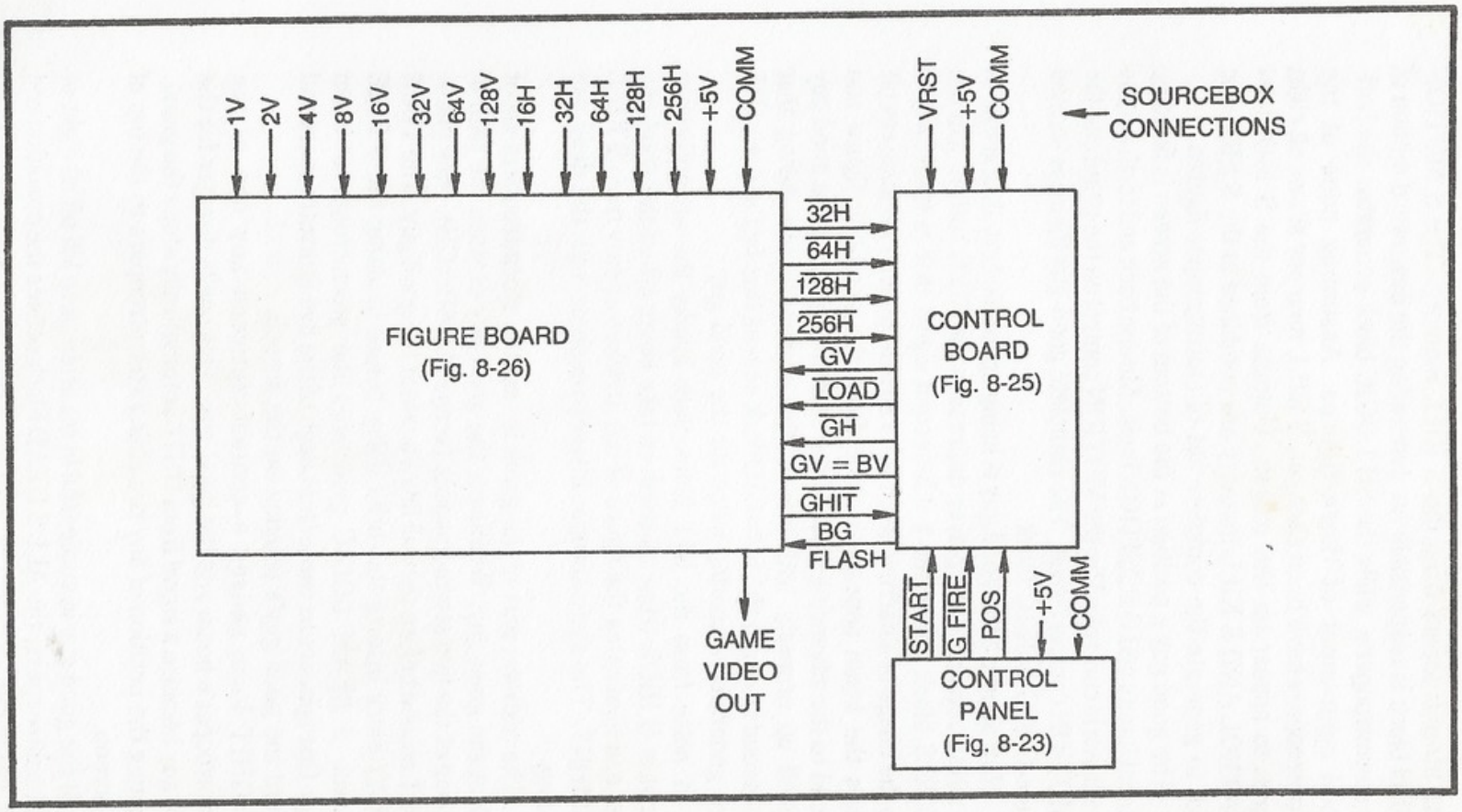


Fig. 8-27. Wiring block diagram for Ambush.

killed stormtroopers from the S KILL memory. The S MOTION control block is responsible for generating the downward motion of the stormtroopers, while the SH LOGIC block generates the horizontal component of those figures. Assuming none of the stormtroopers have been shot yet, S KILL memory allows all eight figures to appear on the screen. Outputs from the S MOTION CONTROL AND S KILL memory are combined in the S FIG logic block to generate the complete set of stormtrooper figures.

The good guy's position at the bottom of the screen is fixed by V-count inputs and G POSITION logic. Motion back and forth across the screen is controlled by the MOTION control on the panel and the G MOTION control circuit. The complete good-guy figure is formed by the G FIG LOGIC block.

Now suppose the good guy is lining up for his first shot at one of the stormtroopers. The player depresses the FIRE button, and the G FIRES block generates a 0.5-second signal that is transformed into the image of a narrow beam by G BEAM logic. This logic circuit makes the beam appear to come from the good-guy figure and extend to the advancing front of the stormtroopers. If the good guy is lined up properly with one of the stormtroopers during that 0.5-second interval, the SH=GH block senses that fact and the HIT block generates a scoring pulse for the good guy.

A pulse from the HIT block both blanks the stormtrooper figures at S HIDE (they all seem to take cover when the good guy fires, and eliminates the figure of the stricken enemy from S KILL MEMORY. The stormtrooper images reappear with the dead one missing.

The trickiest part of the game is that the stormtroopers shoot back at the good guy. Whenever the good guy crosses the path of any one of the live stormtroopers (sensed by SH=GH), they pause for 0.1 second before they all fire a beam at the good guy. The TIME DELAY block inserts this brief delay before initiating the S FIRE interval. S BEAM LOGIC generates the stormtroopers' beam image that appears to come from each of the live stormtroopers and toward the good guy's position on the screen.

GHIT logic senses a contact between any one of the stormtrooper's beam and the good guy. This spells disaster for the good guy because a signal from GHIT automatically stops the game, resetting the position of any remaining stormtroopers to the top of the screen.

If the good guy is successful in his attempt to kill off all eight of the stormtroopers, the ALL S DEAD block senses the condition and ends the game.

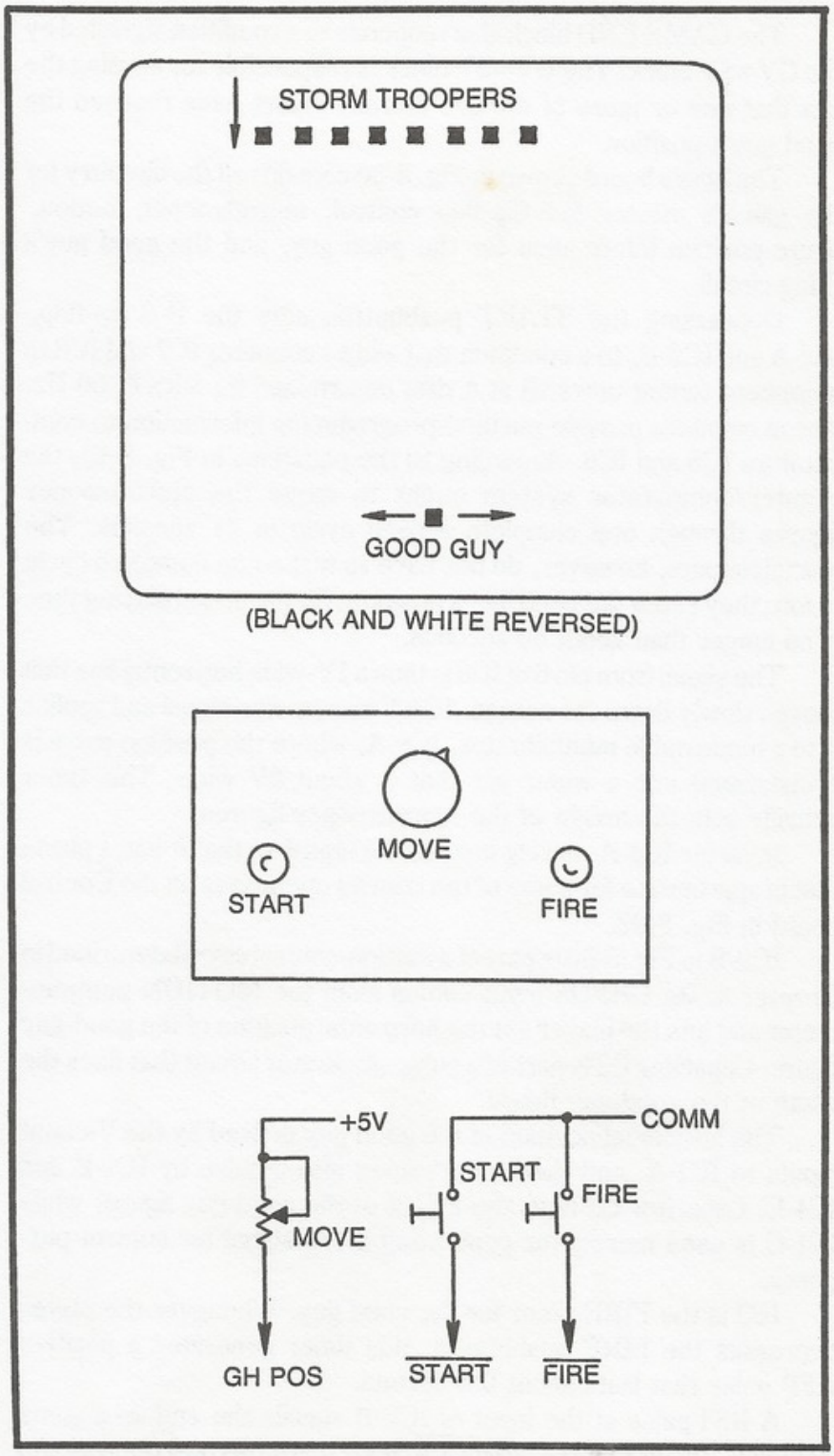


Fig. 8-28. Control panel diagram and schematic, and screen diagram for Stormtrooper Attack.

The GAME END block also responds to a condition signalled by the GV=SV block. The GV=SV block is responsible for sensing the fact that one or more of the live stormtroopers have reached the good guy's position.

The figure board shown in Fig. 8-30 contains all the circuitry for the game's master $\overline{R}$-$\overline{S}$ flip-flop control, stormtrooper motion, figure-position information for the good guy, and the good guy's firing circuit.

Depressing the START pushbutton sets the $\overline{R}$-$\overline{S}$ flip-flop, IC2-A and IC2-B, to a condition that allows counters IC7 and IC8 to increment (count upward) at a rate determined by VRST, 60 Hz. These counters provide position-programming information to comparators IC5 and IC6. According to the equations in Fig. 8-13, the counter/comparator system ought to move the stormtrooper figures through one complete screen cycle in 71 seconds. The stormtroopers, however, do not have to make one complete cycle before they reach the good guy's position. So the actual playing time is no longer than about 60 seconds.

The signal from pin 6 of IC6 is thus a 1V-wide horizontal line that moves slowly down the screen. IC4-F inverts this signal and applies it to a monostable multivibrator, IC9-A, where the position pulse is transformed into a wider bar that is about 8V wide. This timer actually sets the height of the stormtrooper figures.

Inverter IC4-A merely inverts the signal so that it has a phase that is appropriate for some of the control operations on the Control board in Fig. 8-32.

IC9-B in Fig. 8-30 is part of a motion-control circuit described in Chapter 5. Its GHPOS input comes from the MOTION potentiometer and lets the player set the horizontal position of the good-guy figure. Capacitor C2 is part of a pulse-generator circuit that fixes the width of the good-guy figure.

The horizontal position of the good guy is fixed by the V-count inputs to IC1-A, and then transformed into a pulse by IC1-B and IC4-E. Capacitor C3 fixes the height of the good-guy figure, while IC1-C is used merely for generating a GV signal for control purposes.

IC3 is the FIRE timer for the good guy. Whenever the player depresses the FIRE pushbutton, this timer generates a positive GFP pulse that lasts about 0.5 second.

A $\overline{RST}$ pulse at the input of IC2-B signals the end of a game sequence, generating a logic-0 $\overline{CLR}$ level that stops the counters and initializes the position of the stormtroopers.

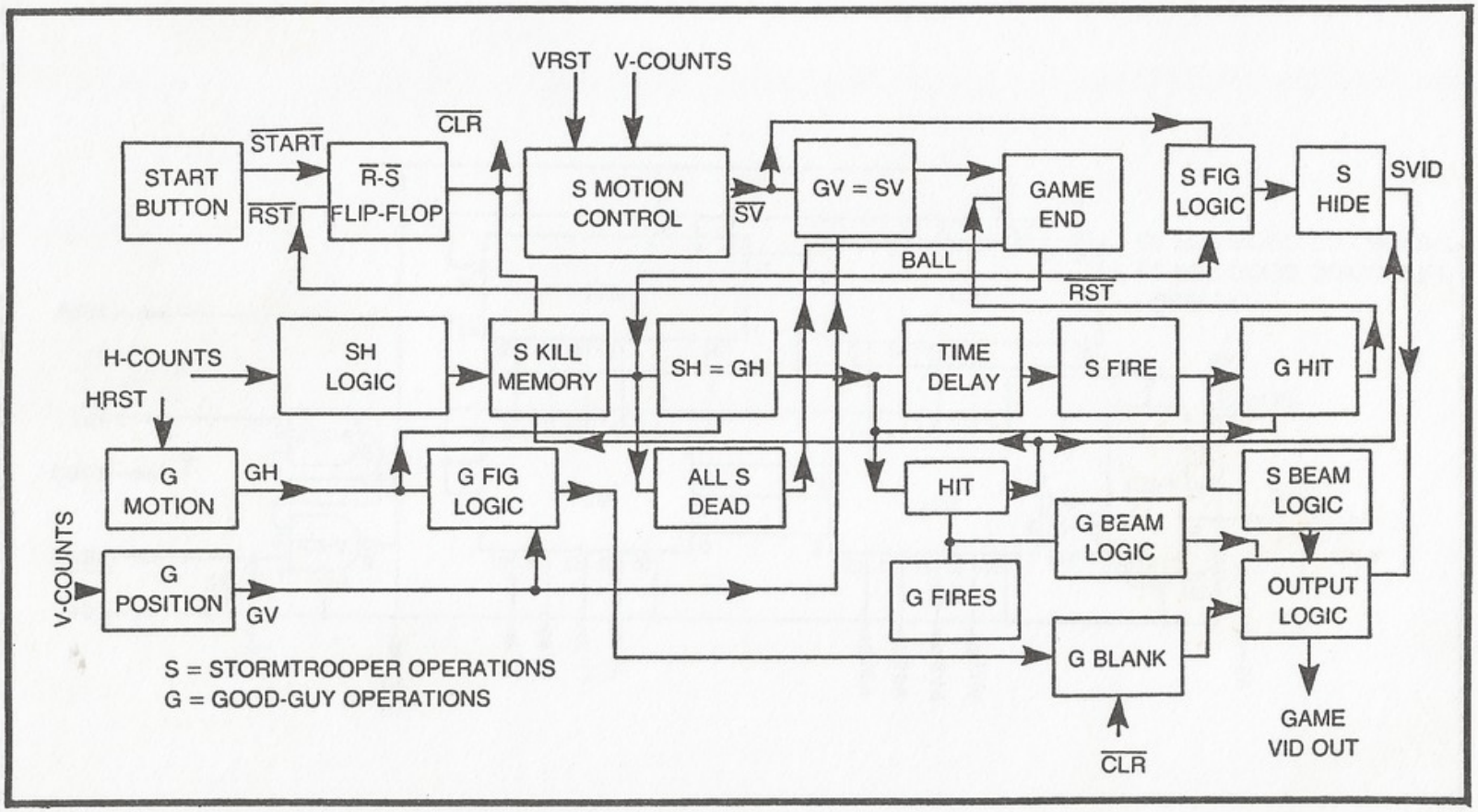
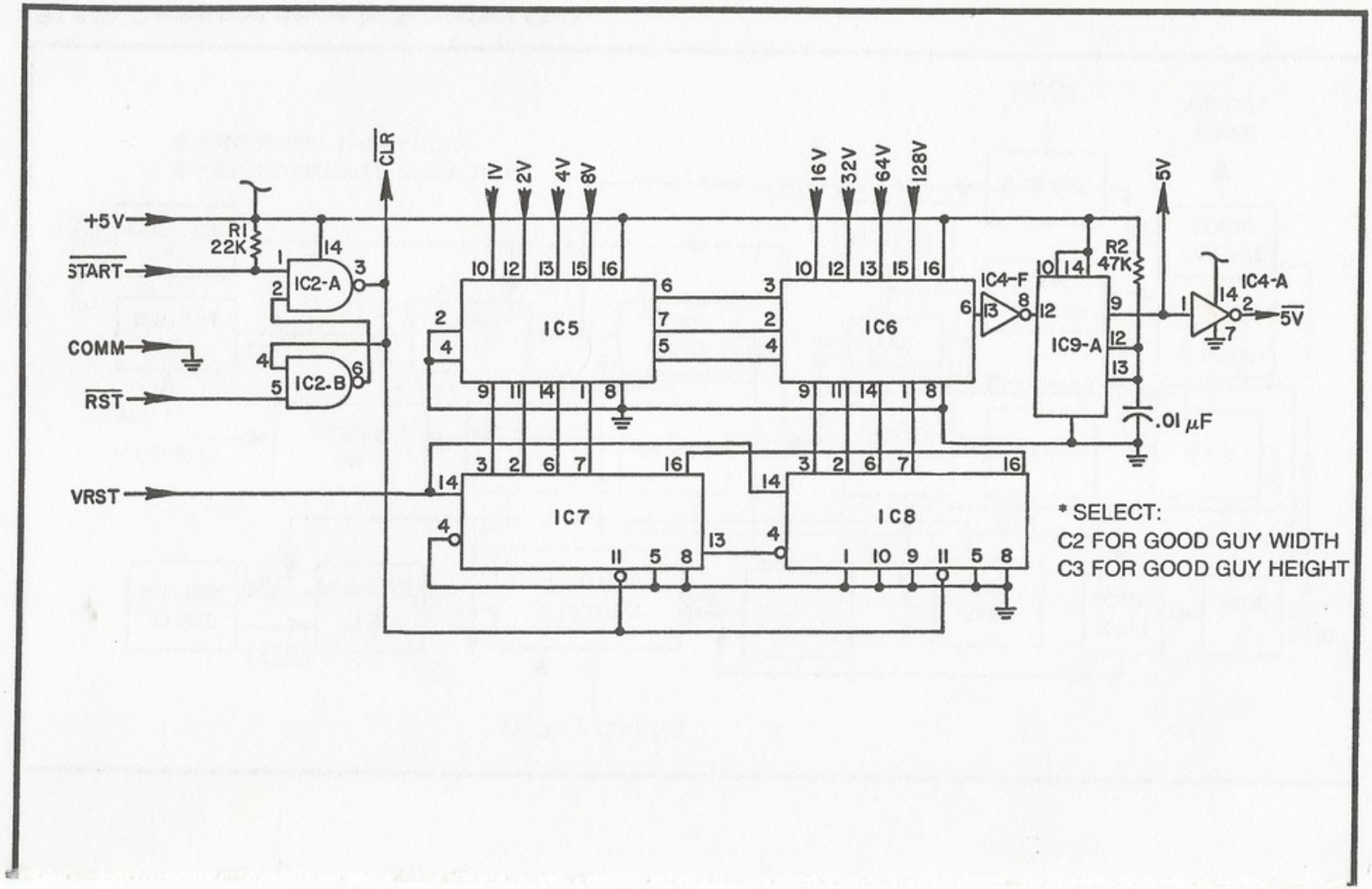


Fig. 8-29. Function block diagram for Stormtrooper Attack.


+5V
R1
22K
START
COMM
RST
VRST
IC2-A
IC2-B
CLR
1V
2V
4V
8V
16V
32V
64V
128V
IC5
IC6
IC7
IC8
IC4-F
IC9-A
R2
47K
5V
IC4-A
.01 μF
* SELECT:
C2 FOR GOOD GUY WIDTH
C3 FOR GOOD GUY HEIGHT

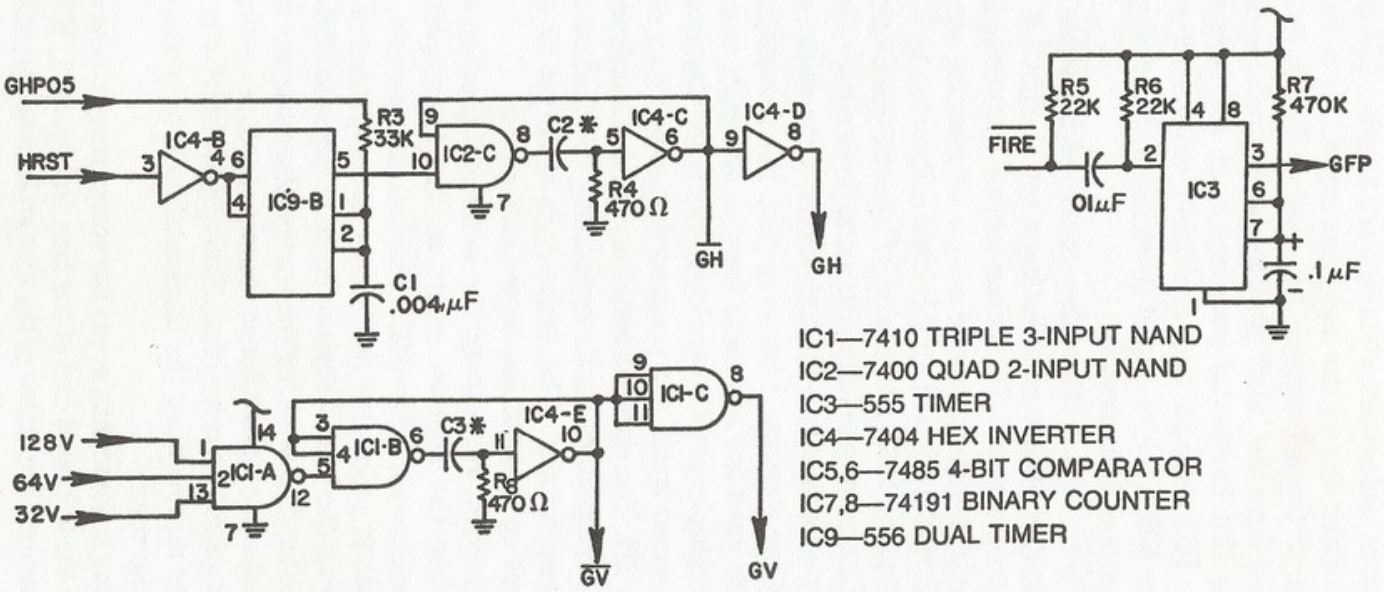


Fig. 8-30 Figure board schematic for Stormtrooper Attack

The Stormtrooper board in Fig. 8-31 keeps track of the live stormtroopers, generates the horizontal component of the stormtrooper figures, and senses when they are all killed. IC3 in Fig. 8-31 determines which one of the eight stormtroopers is hit and changes the state of the appropriate J-K flip-flop, IC6-A through IC9-B. Information from the "live" flip-flops is then recombined in IC5 to generate the horizontal component of the remaining stormtroopers. IC4 senses the condition where all eight stormtroopers are killed before any of them reach the good guy's position at the bottom of the screen.

IC3 and IC5 in Fig. 8-31 make up an addressable demultiplexer/multiplexer combination having a set of eight J-K flip-flops standing between them. The demultiplexer and multiplexer circuits are addressed from the same source of counting signals: 32H, 64H, and $\overline{128H}$. (The board's 128H input is inverted by IC2-B before it is applied as the $\overline{128H}$ address bit.) Since the demultiplexer and multiplexer ICs are operated from the same three address lines, it follows that they scan their data in precisely the same sequence.

These are 8-line devices. That is to say, the demultiplexer (IC3) takes a single input line and splits the data into eight scanned output lines. The multiplexer (IC5) works just the other way around, accepting eight scanned inputs and reassembling them into a single output line, SH or $\overline{SH}$. The eight outputs from the demultiplexer go to eight different J-K flip-flops, each representing the dead-or-alive status of each stormtrooper. The active-high Q output of each flip-flop then makes up an input line to the multiplexer.

Each address count represents the horizontal position of a stormtrooper. When the address inputs to the demultiplexer/multiplexer combination are $32H=64H=\overline{128H}=0$, for example, the beam on the screen is generating the horizontal component of the first stormtrooper on the left. When the count changes to $32H=1$, $64H=128H=0$, the beam is scanning the second stormtrooper from the left. This demultiplexer/multiplexer scanning process continues through the eighth address combination, $32H=64H=128H=1$, the point on the screen representing the horizontal position of the last stormtrooper on the right.

This addressing scheme thus scans the stormtroopers one at a time, beginning at the left-hand end of the line. In Fig. 8-31, IC6-A determines the dead-or-alive status of the first stormtrooper, IC6-B takes care of the second stormtrooper from the left, and so on through IC9-B that determines the status of the last stormtrooper on the right.

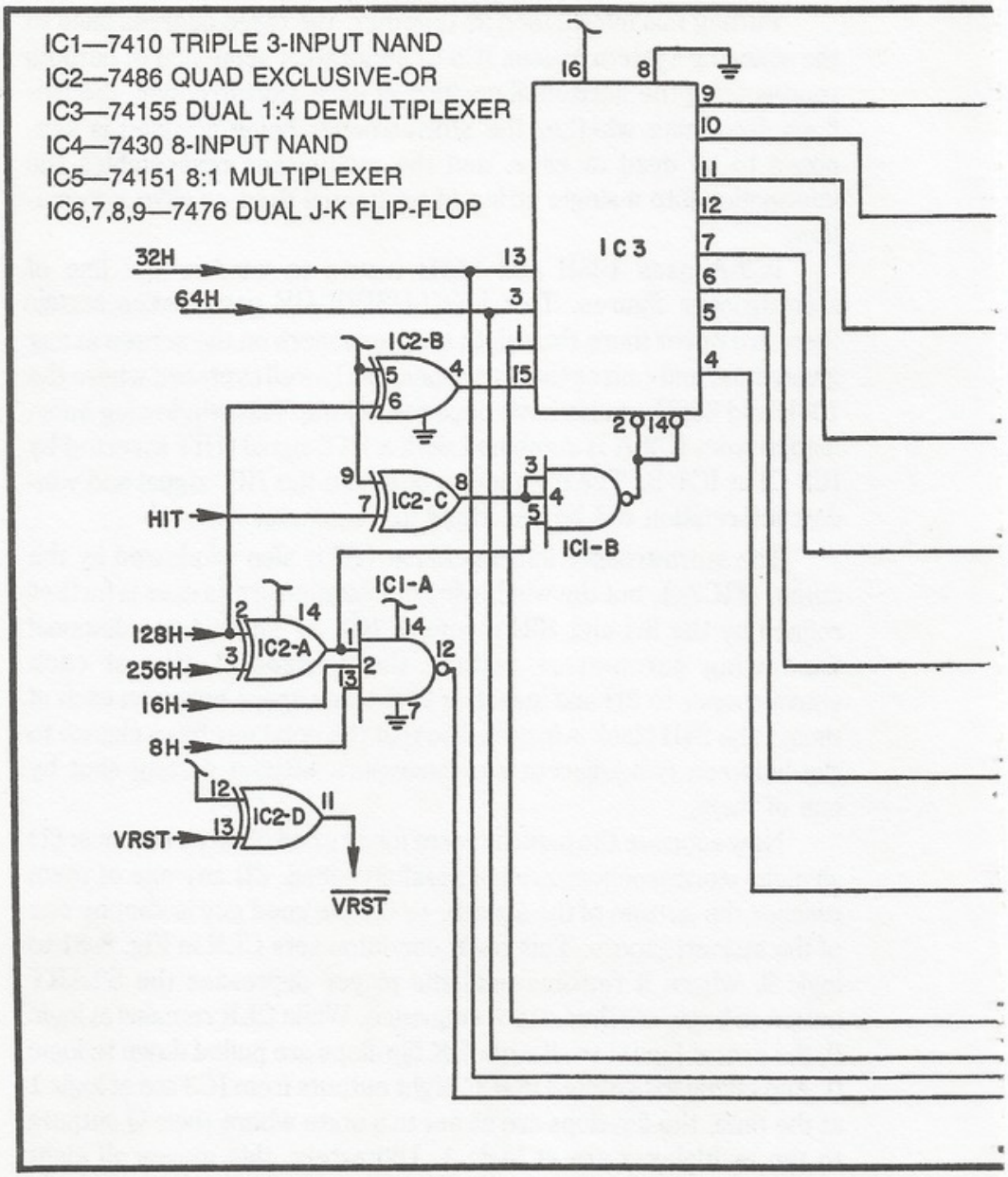
Putting this information all together, the three address lines to the scanning system causes IC3 to generate a sequence of outputs representing the horizontal position of each stormtrooper, the flip-flops determine whether the stormtrooper being scanned is supposed to be dead or alive, and the multiplexer reassembles the information into a single string of sequential dead-or-alive information.

IC2-A uses 128H and 256H inputs to window the line of stormtrooper figures. This EXCLUSIVE-OR gate makes certain there are never more than eight stormtroopers on the screen at any given time, and causes them to appear fairly well centered where the 128H and 256H signals have opposite colors. This windowing information from IC2-A is combined with a HIT signal (HIT inverted by IC2-C) at IC1-B. The interaction between the HIT signal and window information will be described in a moment.

The stormtrooper information at IC5 is also windowed by the output of IC2-A, but the windowing information in this case is further refined by the 8H and 16H inputs to IC1-A. These two additional windowing parameters reduce the horizontal size of each stormtrooper to 8H and inserts a 24H blank space between each of them. The 24H blank is inserted so that the good guy has a chance to slip between two adjacent stormtroopers without getting shot by one of them.

Now suppose the game is reset for any one of three reasons: (1) all eight stormtroopers are successfully killed, (2) any one of them reaches the bottom of the screen, or (3) the good guy is shot by one of the stormtroopers. This reset condition sets CLR in Fig. 8-31 to logic 0, where it remains until the player depresses the START button to begin another attack sequence. While CLR remains at logic 0, the preset inputs to all eight J-K flip-flops are pulled down to logic 0. And taking for granted that all eight outputs from IC3 are at logic 1 at the time, the flip-flops are all set to a state where their Q outputs to the multiplexer are at logic 1. Ultimately, this means all eight stormtrooper figures can appear on the screen. In fact any stormtrooper figure remains on the screen as long as the Q output of its corresponding flip-flop is at logic 1.

The flip-flops then remain in their "alive" logic-1 states until a successful HIT occurs. Whenever the good guy manages to shoot a given stormtrooper, a $\overline{\text{HIT}}$ pulse occurs at the windowing input (pins 2 and 14) of IC3. The addressing scheme for the demultiplexer works in such a way that its output is scanning the stormtrooper that is hit, thereby setting its flip-flop to a logic-0 state. This indicates a



"dead" stormtrooper, and the multiplexer now sees a logic-0 input at that position on the screen. The figure for that particular stormtrooper is thus eliminated until the whole game sequence is reset again.

As far as the player is concerned, he hopes to hit all eight stormtroopers before the game is automatically reset. If he is successful, the Q outputs of all eight flip-flops are finally set to logic 0, and no stormtrooper figures appear on the screen.

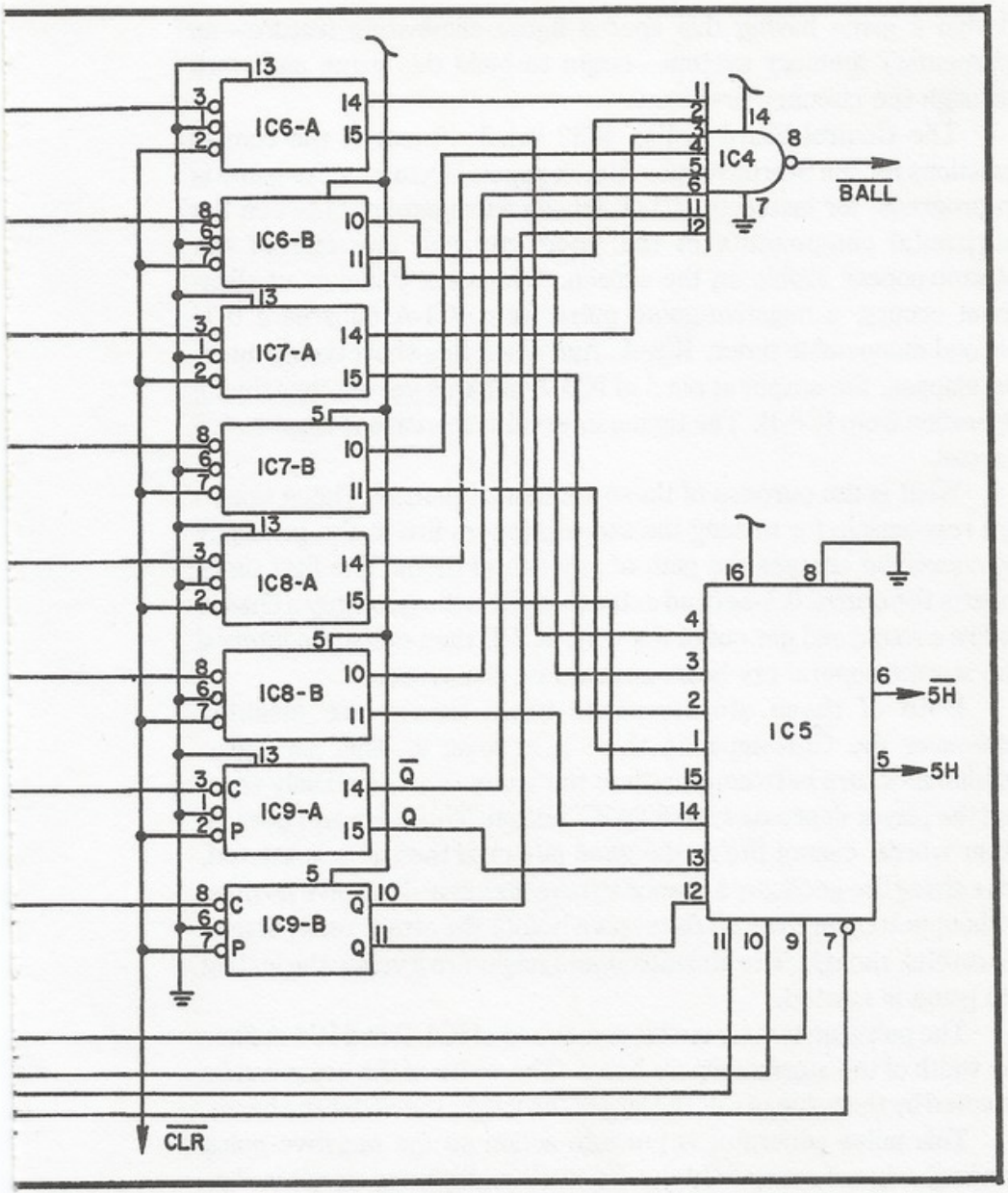


Fig. 8-31. Stormtrooper board schematic.

IC4 in Fig. 8-31 is responsible for sensing the condition where all eight stormtroopers are successfully eliminated. The inputs to this NAND gate come from the $\overline{Q}$ outputs of the flip-flops. When these active-low outputs all reach a logic-1 state, the output of IC4 drops to logic 0 to create the $\overline{BALL}$ signal, one of the three conditions required for resetting the game sequence.

The operation of the Stormtrooper board in Fig. 8-31 is far more difficult to explain than to use. Any experimenter hoping to

design a game having this special figure-eliminating feature—an elementary memory system—ought to build this game and work through the circuitry first hand.

The Control board in Fig. 8-32 handles most of the control functions for the Stormtrooper Attack game. Whenever the game is in progress, for instance, IC1-A senses an alignment between the horizontal components of the good guy and any one of the stormtroopers visible on the screen. Whenever this sort of alignment occurs, a negative-going pulse from IC1-A initiates a 0.1-second monostable timer, IC8-A. And when this short timing interval elapses, the output at pin 5 of IC8-A initiates yet another timing operation from IC8-B. The timing interval in this case is close to 0.5 second.

What is the purpose of these sequential timers? These timers are responsible for making the stormtroopers fire at the good guy whenever he crosses the path of any one of them. The first timer inserts the crucial 0.1-second delay that gives the good guy a chance to fire a round and get out of the way. IC8-B then times the interval the stormtroopers' ray beam appears on the screen.

Both of these stormtrooper beam timers are disabled whenever the $\overline{CLR}$ signal to them is at logic 0. This particular condition occurs between the time the game is automatically reset and the player depresses the START button. The stormtroopers, in other words, cannot fire at the good guy until the game is started, thus giving the good guy a chance to take his initial defensive position without getting blasted off the screen before the attack really starts. Be careful, though. The stormtroopers might fire a volley the instant the game is started.

The pulse-generator circuit composed of IC1-B and IC6-A fixes the width of the stormtroopers beam. The width of the beam can be selected by the value of C4, the larger the value, the wider the beam.

This pulse generator is put into action on the negative-going (leading) edge of every $\overline{SH}$ pulse from the multiplexer circuit in Fig. 8-31. The beam, in other words, always appears to come from the left-hand edge of an advancing stormtrooper figure. Thus the stormtroopers are all right-handed, a fact that might help the good guy's strategy.

The $\overline{R}$-$\overline{S}$ flip-flop made up of IC1-C and IC1-D determines where the stormtroopers' beams begin and end. The $\overline{SV}$ signal to one of the inputs to this flip-flop ensures that the beam always starts from the line of stormtroopers, and the VRST input makes certain the beams travel all the way to the bottom of the screen.

So there are three basic elements that make up the stormtroopers' beams. IC1-A and the two timers determine when the stormtroopers fire and the duration of their volley, IC1-B and IC6-A determine the horizontal position and width of each beam, while IC1-C and IC1-D fix the position and length of the beam in the vertical plane. All three of these stormtrooper beam parameters are NANDed together at IC4-A, resulting in the complete video information for that part of the game.

A dead stormtrooper, incidentally, cannot fire a round at the good guy simply because the $\overline{SH}$ pulse at IC1-B cannot occur. That particular pulse is eliminated at the Stormtrooper board.

Whenever the good guy depresses his FIRE button, the timer circuit on the Figure board (IC3 in Fig. 8-30) generates a GFP pulse that lasts about 0.5 second. This positive-going level is fed to pin 3 of IC4-B to determine the good guy's firing-time interval.

The R-S flip-flop built around IC2-B and IC2-C fixes the vertical length of the good guy's beam. The SV input to this flip-flop always starts the good guy's beam at the line of stormtroopers, and the $\overline{GV}$ input ends the beam at the good guy's position. The good guy's beam is actually drawn from the line of stormtroopers to the good guy, but the visual impression is that the good guy is firing upward on the screen, and it is a visual impression that is more important in this case.

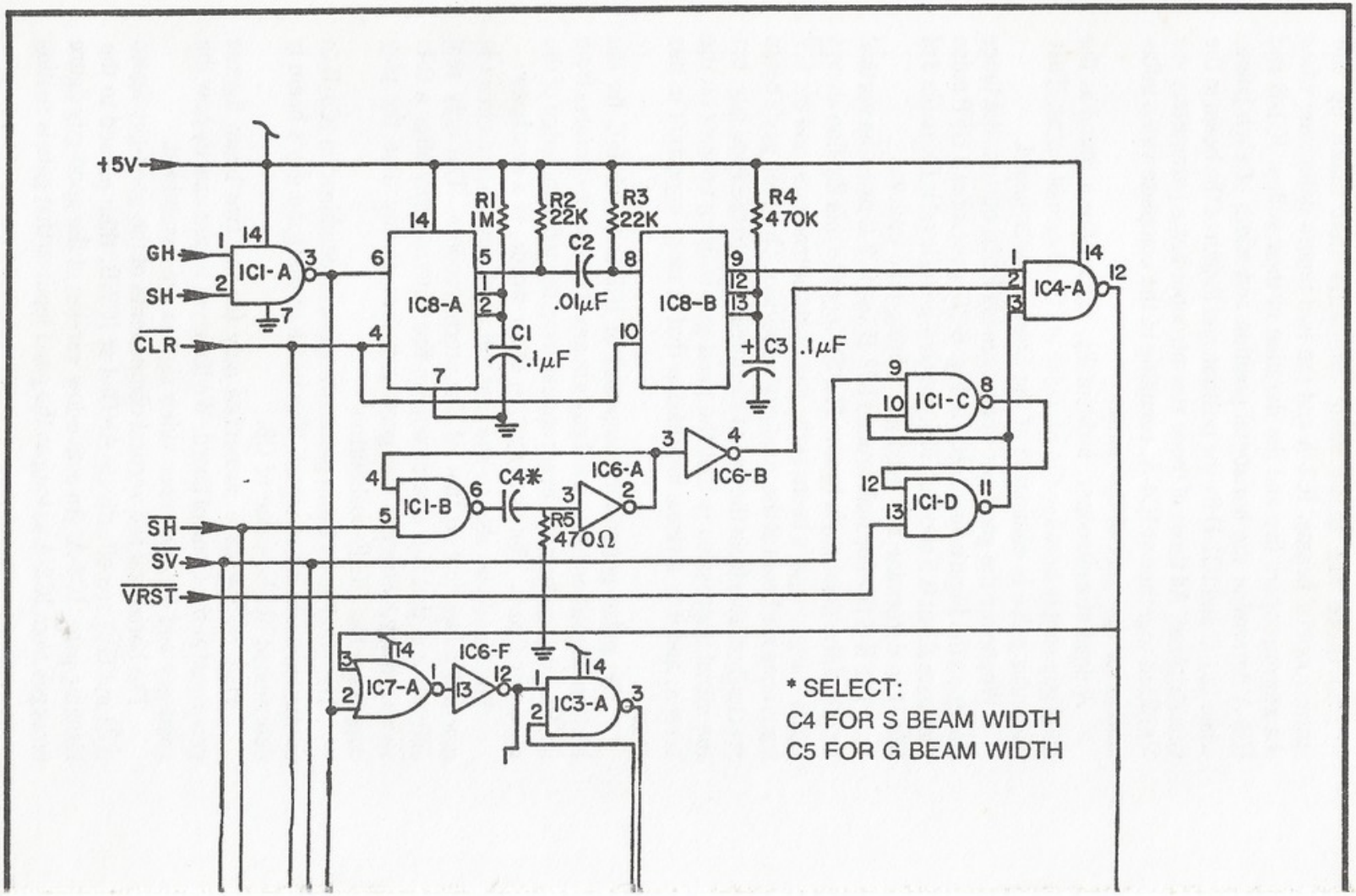
The pulse generator composed of IC2-D and IC6-C fix the horizontal position of the good guy's beam. The circuit operates from a $\overline{GH}$ signal, making the beam appear from the left-hand edge of the good-guy figure. The good guy, in other words, is a southpaw.

You should be able to see that the good guy's beam circuit is practically identical to that of the stormtroopers. The only real difference is that the stormtroopers fire automatically after a 0.1-second delay, whereas the good guy fires at any time the play depresses the FIRE pushbutton.

The good guy's beam parameters are all combined in IC4-B to create his beam video signal. The width of the good guy's beam is determined by the value of C5.

Thus far we have accounted only for the two beam figures generated at the Control board. We have yet to see exactly how the good-guy and stormtrooper video signals are assembled.

The horizontal and vertical components of the good-guy figure ($\overline{GH}$ and $\overline{GV}$) are effectively ANDed at IC7-B, then passed to the blanking gate, IC2-A. An active-low version of the good-guy figure emerges from IC2-A as long as the pin-1 input to that gate is resting


+5V
GH
SH
CLR
SH
SV
VRST
IC1-A
IC8-A
IC8-B
IC4-A
IC1-C
IC1-B
IC1-D
IC6-A
IC6-B
IC6-F
IC7-A
IC3-A
R1 1M
R2 22K
R3 22K
R4 470K
C1 .1μF
C2 .01μF
C3 .1μF
C4*
R5 470Ω
* SELECT:
C4 FOR S BEAM WIDTH
C5 FOR G BEAM WIDTH

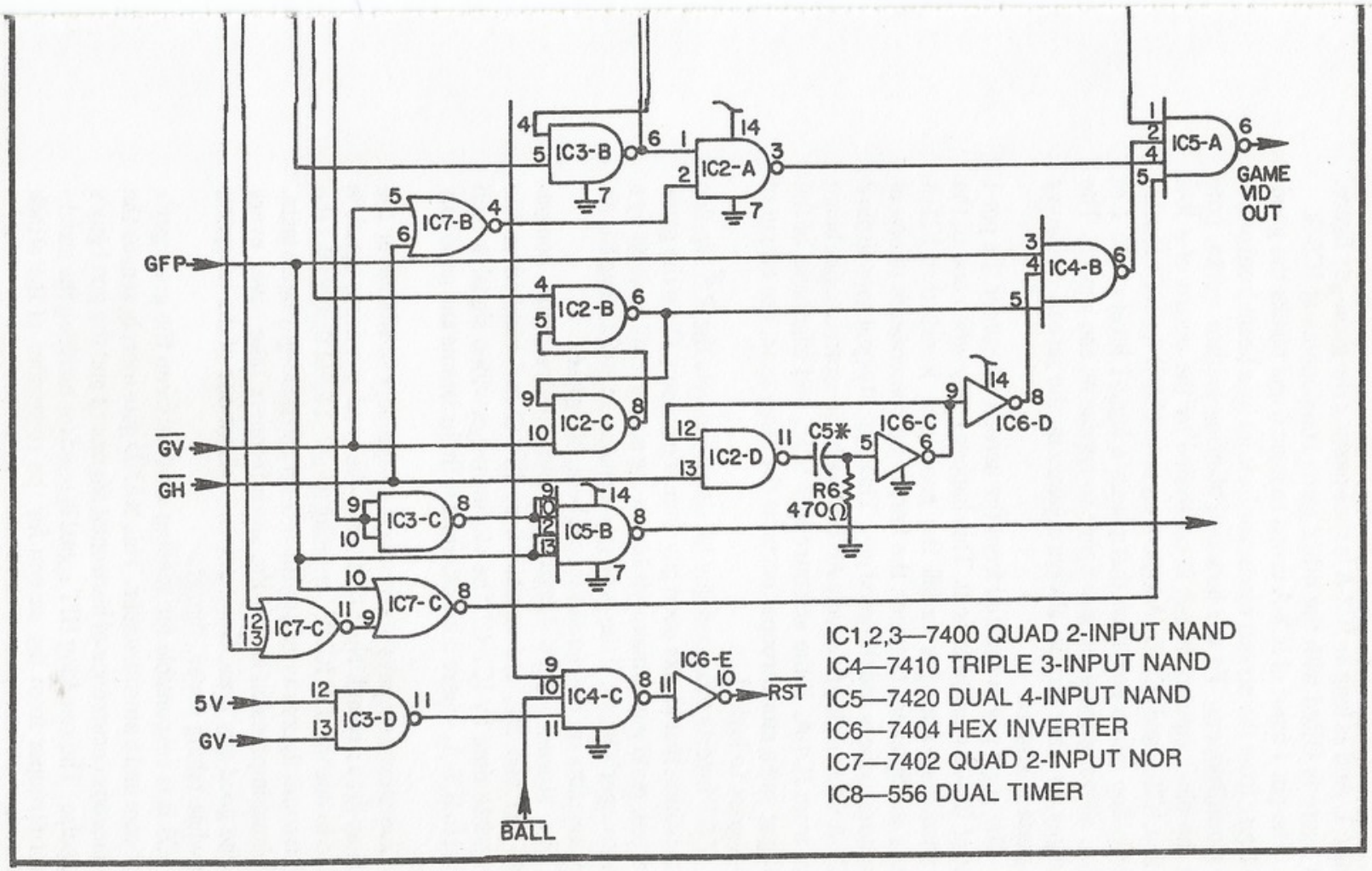


Fig. 8-32. Control board schematic for Stormtrooper Attack.

at logic 1. And as long as IC2-A is thus passing the good-guy figure, that figure is ORed with the other game components at IC5-A.

The pin-1 input of IC2-A drops to logic 0 and blanks the good-guy figure from the screen whenever he is hit by a beam from one of the stormtroopers. To see how this blanking feature works, note that the pin-1 input of IC2-A is controlled by the output of a $\overline{R}$-$\overline{S}$ flip-flop, IC3-A and IC3-B. A logic-0 input to IC3-B from $\overline{CLR}$ sets this flip-flop into a condition that places a logic-1 level at pin 1 of IC2-A, allowing the good guy figure to appear on the screen. The good guy, in other words, always appears on the screen whenever the game is started.

The good guy is blanked from the screen whenever the pin-1 input of IC3-A goes to logic 0. This happens only when one of the stormtroopers manages to kill the good guy. Recall that IC1-A senses an alignment between the horizontal components of one of the stormtroopers and the good guy. This logic-0 signal goes to pin 2 of IC7-A where it is effectively ANDed with the stormtrooper beam signal from IC4-A. If the stormtrooper beam and alignment of the good guy and a stormtrooper occur at the same time, the output of IC7-A goes to logic 1.

IC6-F inverts this good-guy-hit signal and sets the $\overline{R}$-$\overline{S}$ flip-flop to a condition that blanks good guy from the screen. Once the game is started, good guy remains visible on the screen until he is hit by a stormtrooper's beam. If the good guy manages to avoid being hit, his figure remains on the screen throughout the game.

The stormtroopers' horizontal- and vertical-figure components, SH and SV, are combined at IC7-C and blanked at the appropriate time by IC7-C. The stormtrooper video signal is then applied to IC5-A, where it is combined with the beams and good-guy figure.

Live stormtroopers are blanked off the screen whenever the good guy fires a round. Note that GFP, the good-guy firing signal, is applied to one input of IC7-C, creating a logic-1 level that blanks the stormtrooper figures as long as the 0.5-second good-guy beam lasts. The visual impression is that the stormtroopers take cover every time the good guy fires. Good guy can still hit one of the troopers, even while taking cover, though.

IC5-B is responsible for sensing a hit between the good guy's firing beam and a stormtrooper. This NAND gate merely senses the simultaneous occurrence of horizontal alignment and the good guy's firing pulse. The resulting HIT signal is used for blanking the stricken stormtrooper from the screen for the reaminder of the attack

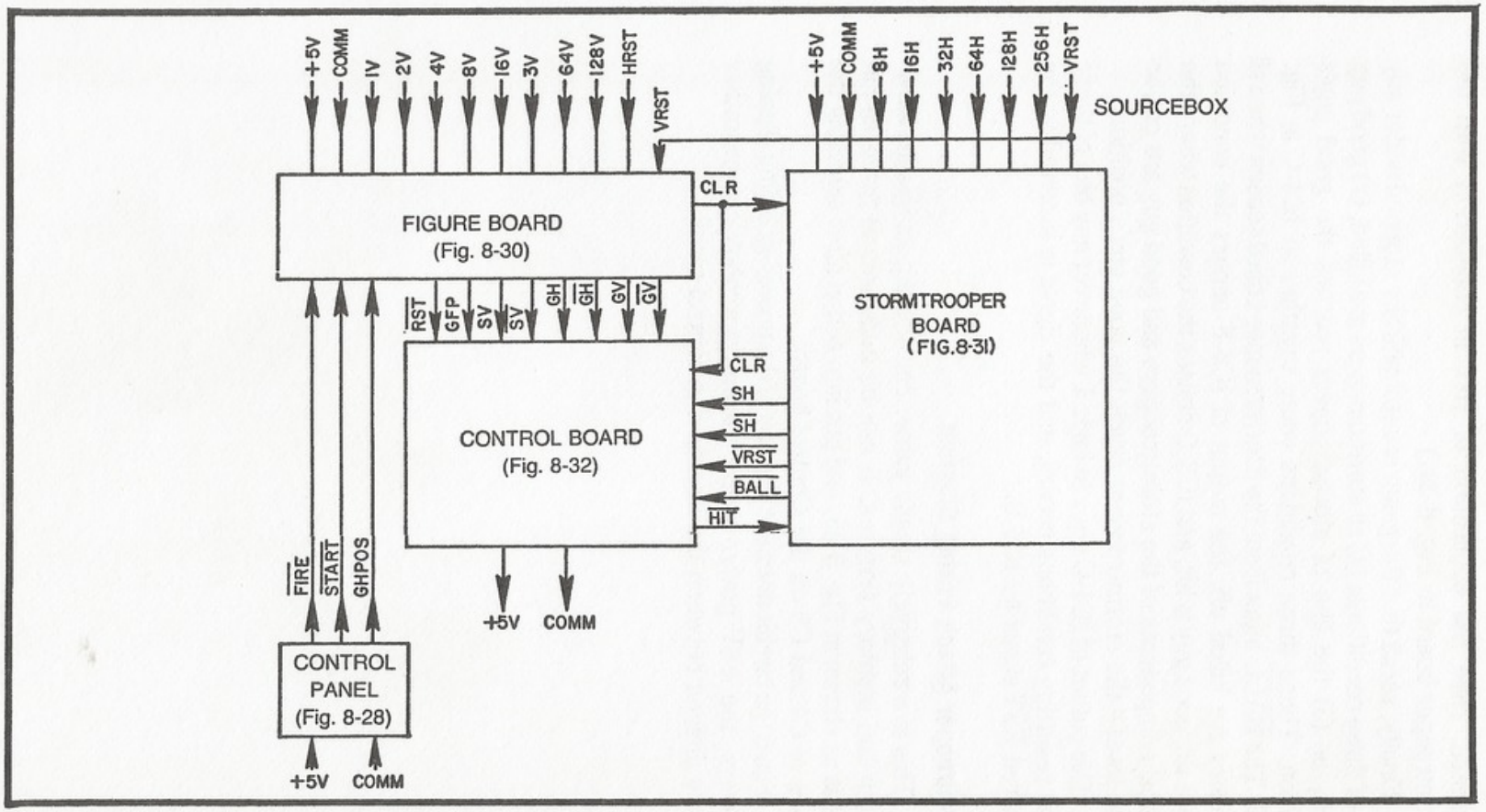


Fig. 8-33. Wiring block diagram for Stormtrooper Attack.

sequence. (See the application of HIT in connection with the Stormtrooper board in Fig. 8-31.)

Finally, recall that the game is automatically stopped under any one of three conditions: (1) all stormtroopers are killed, (2) good guy is hit, or (3) the line of stormtroopers reaches the good guy's position. These three conditions come together at IC4-C in Fig. 8-32. The $\overline{\text{BALL}}$ input from the Stormtrooper board senses when all troopers are killed off, the output of IC6-F senses the condition where the good guy is hit, and IC3-D senses the condition where the vertical components of the stormtroopers and good guy are on the same level—the stormtroopers reach the good guy's position.

The output of IC4-C goes to logic 1 whenever any one of these three resetting conditions occurs, and the signal is inverted to the required $\overline{\text{RST}}$ signal by IC6-E.

Stormtrooper Attack Wiring Diagram

This is a relatively simple game, considering its special slow-motion and memory features. It can be built around three circuit boards as shown in Fig. 8-33, and put into action after selecting the values of C4 and C5 on the Control board.

Since the circuit does not use any of the power-gobbling slipping counters, the +5V power sources from Sourcebox can be rather evenly divided between the boards and control panel.