Chapter 5

Building Motion-Control Circuits

Motion makes TV games. Without some player-controlled and automatic motion, today's video game business would not exist. Given the choice between a system that only generates complex static figures and one that generates very simple figures that move, most people would choose the system that includes the motion feature.

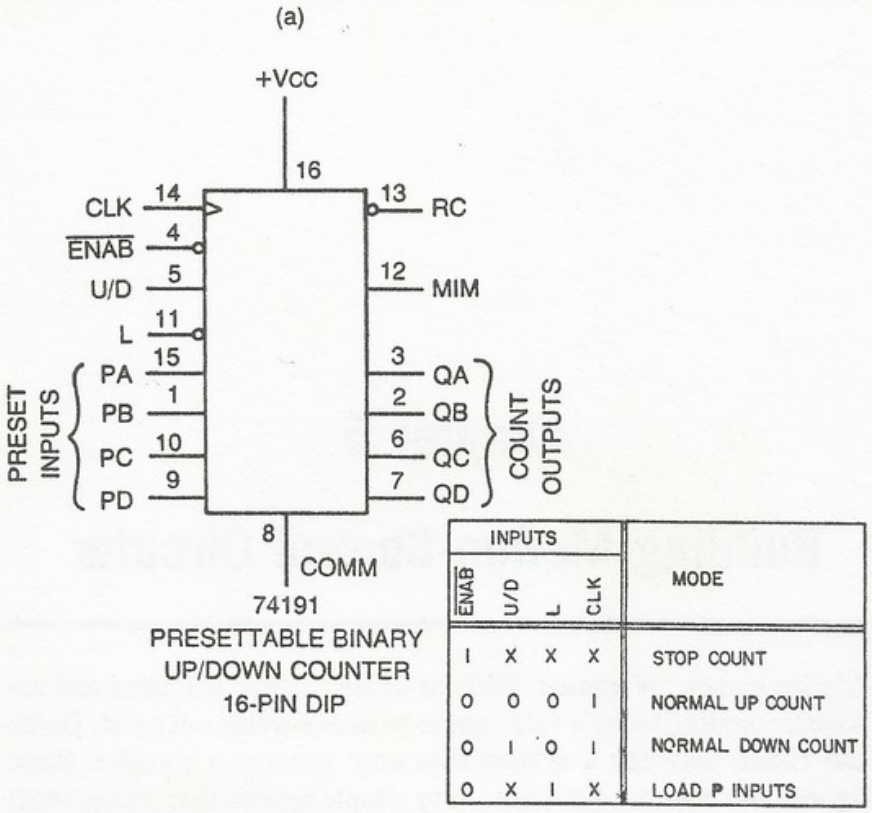
The figures for the original table tennis, squash, and hockey games were exceedingly simple—merely rectangles and lines. But the player-controlled paddle motion and the automatic ball motion gives these games their real appeal. No one seems to care whether the ball is round or square. Motion makes the game.

You can build some good video games around the player-controlled and automatic motion circuits described in this chapter. There is no real need to generate the complex figures featured in Chapter 4.

At some point in your experience with custom video games, however, you will most likely find that adding one or two complex figures makes the games more fun and interesting.

Work through the descriptions and experiments in this chapter, carefully noting the special features of each one. There are a number of options offered here, and that implies that some motion-control circuits are better than others under certain circumstances.

Get a good understanding of all the motion circuits now, and you will be able to design more effective and efficient game circuits later on.


(a)
+Vcc
16
CLK
14
ENAB
4
U/D
5
L
11
PA
15
PB
1
PC
10
PD
9
PRESET INPUTS
13
RC
12
MIM
3
QA
2
QB
6
QC
7
QD
COUNT OUTPUTS
8
COMM
74191
PRESETTABLE BINARY
UP/DOWN COUNTER
16-PIN DIP
INPUTS
ENAB
U/D
L
CLK
MODE
I X X X STOP COUNT
0 0 0 I NORMAL UP COUNT
0 I 0 I NORMAL DOWN COUNT
0 X I X LOAD P INPUTS
(b)
TIMER A
1 DIS
2 THR
3 CV
4 RST
5 OUT
6 TRIG
7 COMM
Vcc 14
DIS 13
THR 12
CV 11
RST 10
OUT 9
TRIG 8
TIMER B
556 DUAL TIMER
14-PIN DIP
1 COMM
2 TRIG
3 OUT
4 RST
Vcc 8
DIS 7
THR 6
CV 5
555 TIMER
8-PIN DIP

SPECIFIED	SUBSTITUTE	
556	FIRST 555	SECOND 555
1 2 3 4 5 6	7 6 5 4 3 2	N. A.
7	1	1
8 9 10 11 12 13	N.A.	2 3 4 5 6 7
14	8	8

(c)

Fig. 5-1. Some ICs used for the first time. (a) Pinout and function table for the 74191 counter. (b) Pinouts for the 556 and 555 timers. (c) A chart showing how to substitute two 555 timers for a single 556 dual time. Numbers indicate pin numbers.

MOTION-CONTROL TINKERBOX

If you have been performing the experiments outlined in Chapters 3 and 4, you should have at hand a good assortment of NAND, NOR, and invert gates. You will need them for the motion-control Tinkerbox.

In addition, you will need some 555 monostable multivibrators (timers or the 556 dual timers), a few more 7493 counters such as those used in the Sourcebox, and some 74191 presettable up-down binary counters. Study the diagrams in this chapter, making up a list of ICs you might have to order now.

You will need the timers for performing the experiments in the first part of this chapter, but fortunately, they aren't difficult to find these days. You might have to send away for the 74191 counters, but here you have some lead time because they aren't required for a while.

Figure 5-1 shows the pinouts for the 555 and 556 timers and the 74191 counter. Most fo the circuits described in this call for using 556 dual timers. You can, however, substitute two 555s by making the changes in pin numbers as indicated on the chart in Fig. 5-1.

The counter in Fig. 5-1a is in its normal counting mode whenever the $\overline{\text{ENAB}}$ input is at logic 0, the load input, L, is at logic 1,

and clock pulses are applied to the CLK input. The counter counts up or down, depending on the logic level applied at the U/D terminal. Setting U/D to logic 0 lets the counter count up, and setting it to logic 1 makes it count down, or backwards.

The counter can be preset to any desired count by applying the desired binary count at the preset inputs, PA through PD, and pulling the L input down to logic 0.

The counting operation can be stopped and held at any desired count by raising the $\overline{\text{ENAB}}$ input to logic 1. Counting then resumes from that point as soon as ENAB is returned to 0.

The ripple-clock, RC, and maximum/minimum, M/M, outputs perform special cascading and output control functions that will be described in detail when considering the actual circuits that use them.

SIMPLE PLAYER-CONTROLLED MOTION

Let's get some motion on the screen as quickly and simply as possible. To do this, tinker together the circuit shown in Fig. 5-2.

This circuit generates a narrow horizontal or vertical bar on the screen. The horizontal version can be moved vertically anywhere on the screen by means of the 500 kΩ potentiometer control, R1. And when you build the vertical version, you will find you can move that bar horizontally across the screen.

The heart of this system is a 555 timer. The circuit in Fig. 5-2 calls for using one-half of a 556 dual timer which, in essence, performs the job of a single 555 device.

The monostable timing is initiated by the HRST or VRST pulse from the Sourcebox unit, and the actual amount of timing is determined by R1, R2, and C2. When generating a movable horizontal bar, trigger the circuit from VRST and fix the values of R2 and C2 to 3.3 kΩ and 0.047 μF. To generate a movable vertical bar, initiate the timer from HRST and fix the values of R2 and C2 at 33 kΩ and 100 pF. Note that potentiometer R1, the motion control adjustment, is 500 kΩ in either case.

So here is what happens: A vertical or horizontal reset pulse initiates the monostable timing. It sets the pin-5 output terminal from logic 0 to logic 1, where it remains until the timing interval is over. If the circuit is triggered by HRST, this timing interval takes place with every horizontal scan line. Triggering the circuit from VRST causes the timing to take place once each vertical frame.

The NAND gate (IC3-A), inverter, and associated RC components make up a pulse-shortening circuit that functions exactly as

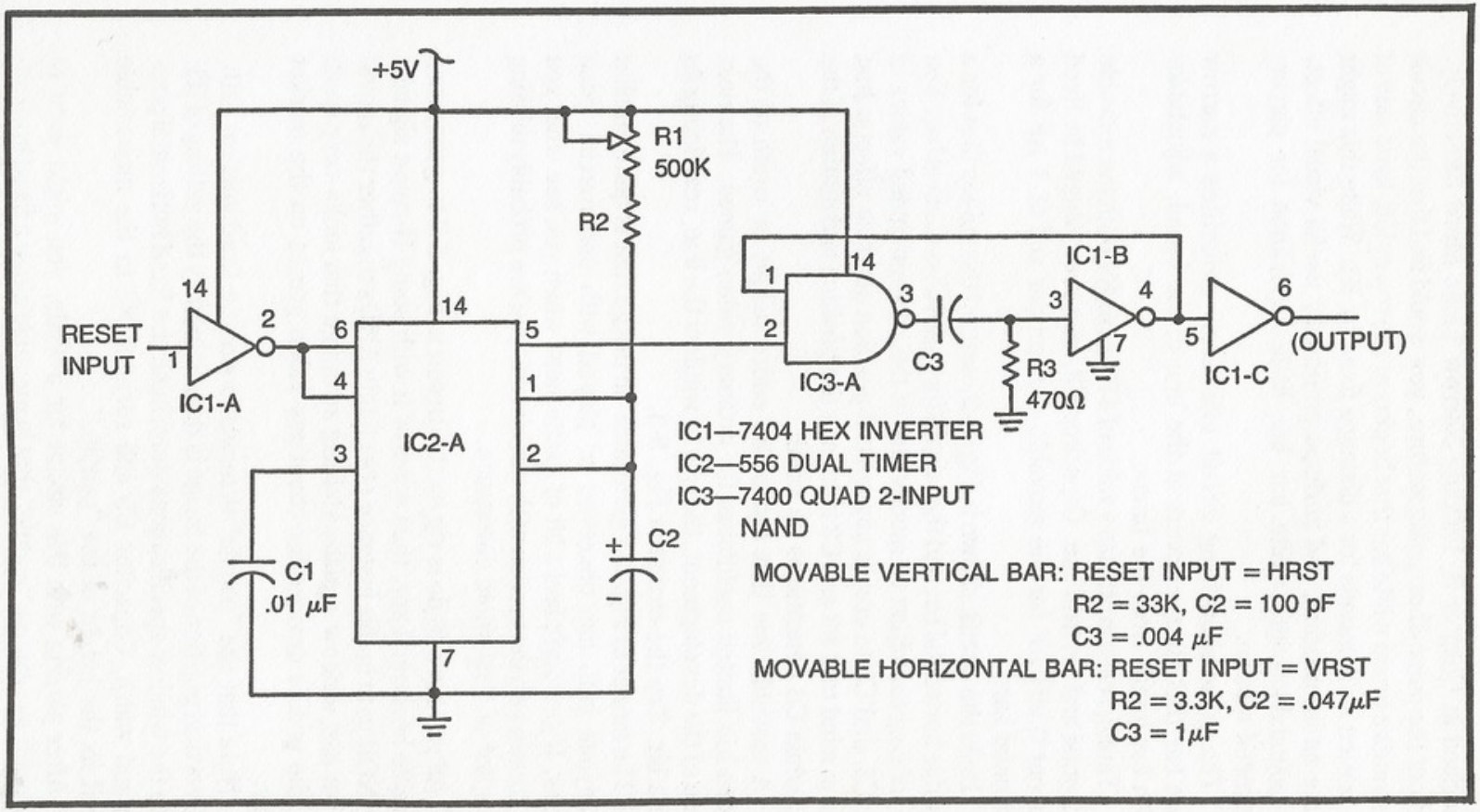


Fig. 5-2. Circuit for letting the user adjust the horizontal or vertical position of a line on the screen.

described in Chapter 3, Building Narrow Lines More Effectively. Without the pulse-shortening feature, you would find that the circuit generates a broad white bar that begins at the top or left-hand side of the screen and extends for a distance fixed by R1. While this might create an interesting, and perhaps potentially useful visual effect, the adjustable-width white bar has little application for player-controlled motion.

The pulse-shortening circuit effectively generates a narrow white bar that always occurs at the end of the broad, adjustable-width bar image from the timer.

Timing resistor R3 has a value of 470 ohms for both the movable horizontal and vertical line. Capacitor C3, however, should be fixed at about 0.004 μF for the movable vertical bar and at 1 μF for a horizontal bar.

Build the circuit shown in Fig. 5-2, setting the values first for a movable horizontal bar and then again for a movable vertical bar. You can, of course, adjust or modify any of the recommended values of R2, C2, and C3 to create any special position and size effects. Just bear in mind that R2 and C2 influence the position and motion of the bar, while C3 determines its width.

A movable line that spans the entire height or width of the screen has limited usefulness for ordinary video games. The next phase of the development, then, is to window the line, restricting its size a bit. Try the circuit in Fig. 5-3.

The circuit in Fig. 5-3 generates an image that is quite familiar to anyone who has observed or played with commercial video games. If you don't feel a bit of excitement when you see what you are creating on the screen with this circuit, you are probably missing the spirit of this whole enterprise.

All you have to do is rig up the circuit in Fig. 5-2 to generate a movable horizontal bar, then window it with some H-count signals. The NOR gate in this instance effectively ANDs together the movable bar and window signals, yielding an image that looks very much like the paddle devices for countless video games on the market today.

Note that the "paddle" is movable along a fixed vertical path. The vertical position of the image is determined by the setting of R1, while the window specifications determine the fixed horizontal position and width. Capacitor C3 and resistor R3 in the monostable circuit fix the height of the "paddle."

After playing with this circuit for a while, you might want to reorient it to create a "paddle" that is movable along a fixed horizon-

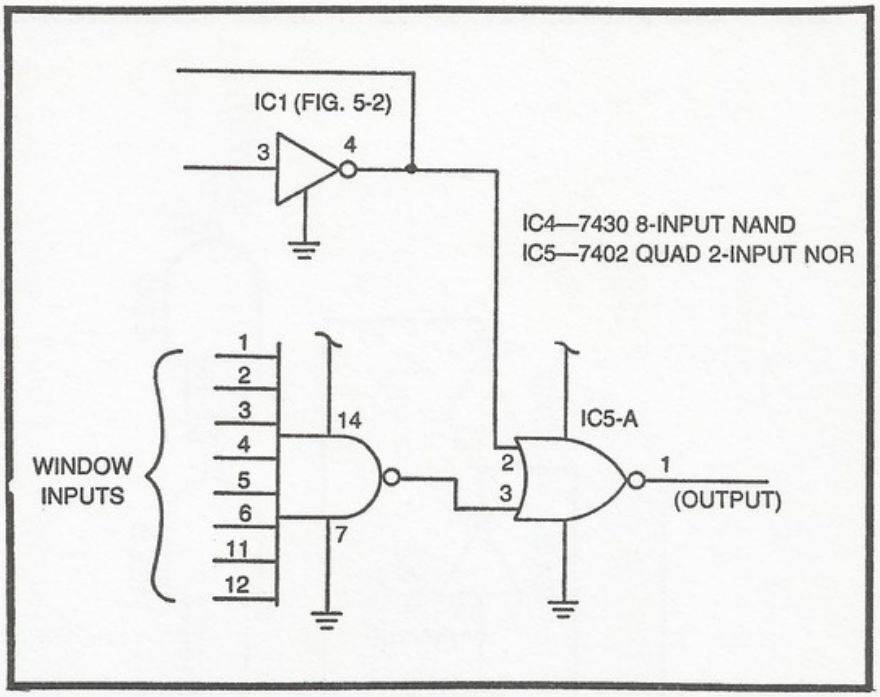


Fig. 5-3. Circuit for windowing the line generated by the circuit in Fig. 5-2.

tal path. What's involved in making that modification? Simply modify the circuit in Fig. 5-2 to create a movable vertical line, then window it with V-count signals from the Sourcebox Unit.

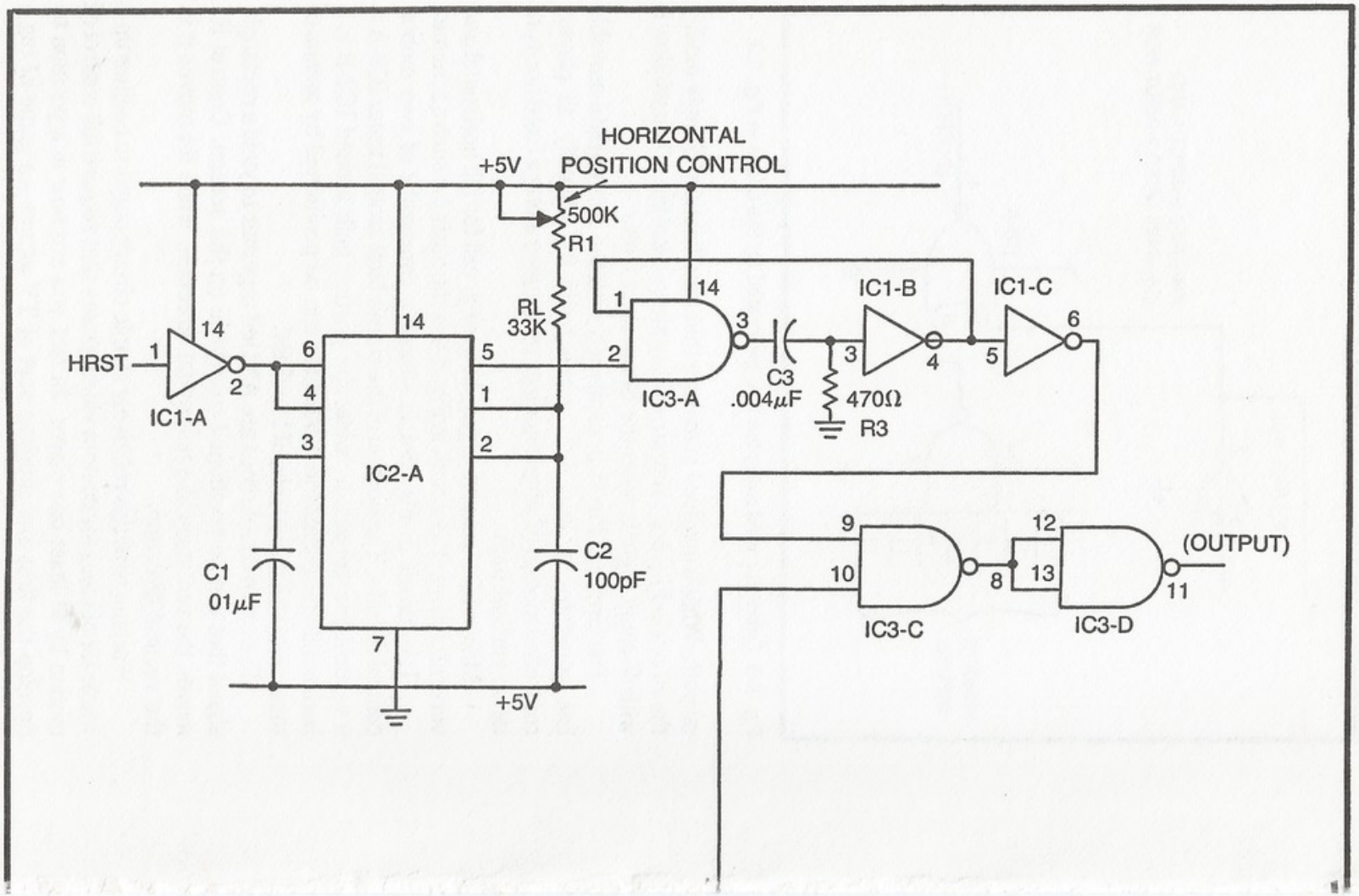
The circuit in Fig. 5-3, combined with the appropriate movable line generator in Fig. 5-2, is the basis for virtually all player-controlled motion of simple lines or rectangles along a fixed horizontal or vertical path.

Many video game designs, however, call for full horizontal *and* vertical control. The circuit in Fig. 5-4 is the basis for such a scheme.

The circuit in Fig. 5-4 is simply a composite of two motion control circuits. The portion of the circuit built around timer IC2-A is a vertical-line generator, while the portion built around IC2-B is a horizontal-line generator. Both lines can be positioned by means of their respective controls, R1 and R4.

The two movable lines are ANDed together to yield a rectangle signal that can be positioned anywhere on the screen. Control R1 moves the rectangle in a horizontal direction, while R4 moves it in the vertical direction.

Working with this relatively simple circuit ought to conjure up a whole lot of ideas for custom video games that require full control of motion by at least one player. In fact you are now in a position to develop the first and simplest sort of TV games—a game of tag.


HORIZONTAL
POSITION CONTROL
+5V
500K
R1
RL
33K
HRST
IC1-A
IC2-A
IC3-A
IC1-B
IC1-C
C3
.004μF
470Ω
R3
C1
01μF
C2
100pF
+5V
IC3-C
IC3-D
(OUTPUT)

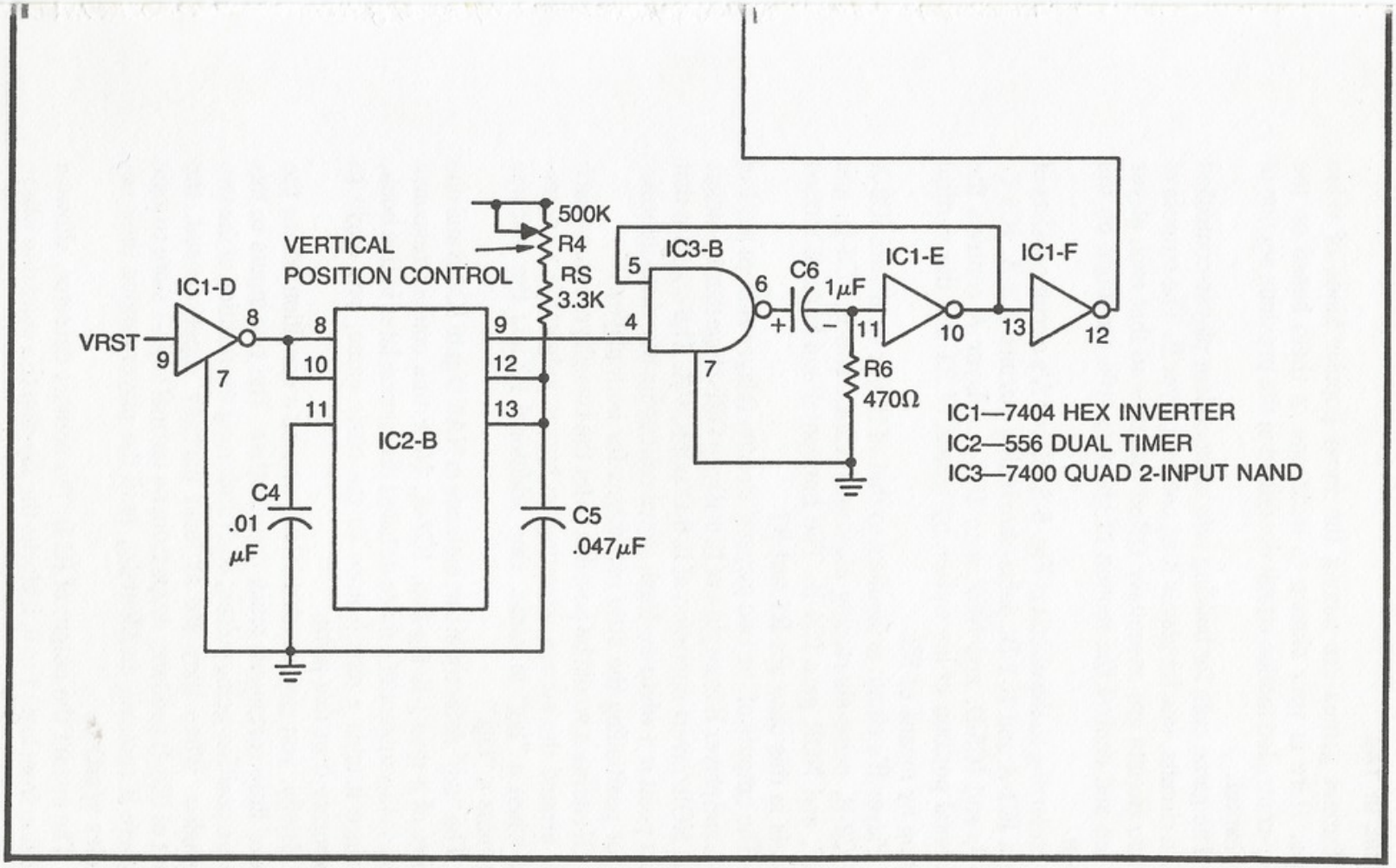


Fig. 5-4. A circuit that allows the user to move a rectangle in both the horizontal and vertical directions.

A GAME OF TAG

Pursuit games are among the most popular kinds of video games. Here is your chance to build one of them based on the player-controlled motion circuit described in the previous section of this chapter.

The game calls for building two independent player-controlled motion circuits, one for player A an one for player B. The outputs of the two circuits are essentially ORed together so that each player can see and control the motion of his own little rectangle on the screen.

Referring to the circuit in Fig. 5-5, player A's circuit consists of timers IC1-A and IC1-B, pulse-shortening circuits IC3-A, IC5-C, IC3-B, and IC5-D, and NOR gate IC6-A. Player A controls the horizontal position of his square by means of R1 and the vertical position by means of R3.

Player B's circuit is identical to that of Player A: timers IC2-A and IC2-B, pulse-shortening elements IC3-C, IC5-E, IC3-D, and IC5-F, and NOR gate IC6-B. The horizontal and vertical motion controls in this case are R5 and R7.

The outputs of the two players' circuits (player A from pin 1 of IC6-A and player B from pin 4 of IC6-B) are ORed together by means of the NOR/invert operation of IC6-C and IC4-A. The signal at that output point is a white-on-black signal containing the video information for positioning the little rectangles for both players.

Of course it would be possible to let the two players chase each other around the screen, counting on human judgement to determine when a "tag" is made. One additional IC lets the machine determine a "tag."

The "tag" detector in this instance is NAND gate IC4-B and one section of a dual J-K flip-flop, IC7-A. Now this contact-response circuit is the subject of a more detailed discussion later in this book, but since it adds a nice feature to the basic game, you ought to incorporate it at this point.

Briefly, you can see that IC4-B senses a condition where the players' little rectangles touch one another. The two inputs to this gate are positive-going pulses, each indicating the position of the two rectangles. When they are at least partially superimposed, the output of IC4-B suddenly drops from its normal logic-1 state to logic 0 (where it remains, incidentally, until the players move their rectangles apart).

The instant the output of IC-B, the contact detector, shows a transition from logic 1 to 0, it clocks the flip-flop to a condition where

its Q output is set to logic 0. This Q output from IC7-A is connected to the reset inputs of all four timer circuits. And when Q is set to zero by the contact detector, it disables all four timers, sweeping the rectangles into the systems horizontal and vertical blanking regions.

Whenever the two rectangles touch, then, they both disappear from the screen. That is a clear indication that contact has been made. Resetting the game is a matter of first adjusting at least one of the motion controls to a different position, and then striking the RESET switch.

Striking the RESET pushbutton sets the Q output of IC7-A back to logic 1. This enables the timers to allow another "tag" play to begin. One of the controls must be changed before hitting the RESET button, however. Without changing one of the controls, the system would remain in its "tag" mode and the rectangles could not appear on the screen.

A somewhat refined and more challenging version of this tag game is presented in a later chapter.

The circuit in Fig. 5-5 uses only seven ICs. These ICs and the associated components (except the player controls) can be mounted on the standard plug-in board. (Radio Shack 276-153). Using the pin numbers specified in parentheses, the board can be plugged directly into the Sourcebox unit.

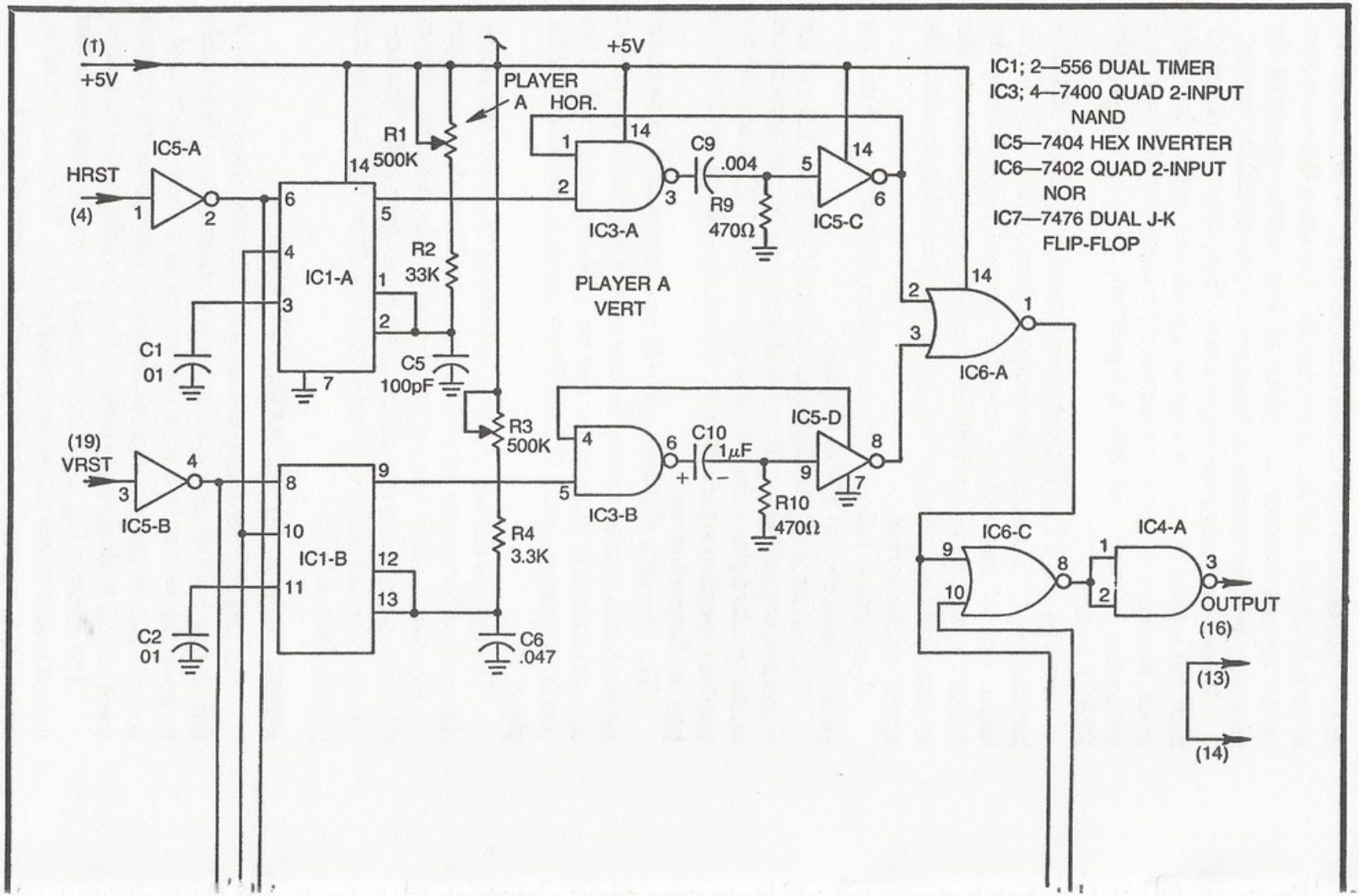
The two sets of player controls can be mounted in separate project boxes. Cables running between the circuit board and project boxes require only three wires: one for +5V and one to each of the two fixed timing resistors (R2 and R4 for player A, and R6 and R8 for player B).

The players can toss a coin to see who is "it," then that player chases the other around the screen until a "tag" is made. The players can then switch roles. Obstacle and timing circuits described later in this book can enhance the quality of the game quite a bit. This one, though, represents the simplest sort of TV game that has any real playing interest.

ADDING "INERTIA" TO THE PLAYER CONTROLS

The player-controlled positioning circuits described thus far in this chapter have an unrealistic quickness about them. It is possible to drive the rectangle across the screen in a mere fraction of a second. The simple modification described here gives the controls some inertia, lending a more realistic kind of motion to the rectangle.

The circuit in Fig. 5-6 is built around the monostable timer that is used in all previous player-controlled circuits. The timing interval, however, is adjusted in this instance by R7 instead of R1.


(1)
+5V
+5V
PLAYER A HOR.
R1 500K
IC5-A
HRST
(4)
IC1-A
R2 33K
C1 01
C5 100pF
IC3-A
C9 .004
R9 470Ω
IC5-C
PLAYER A VERT
IC1; 2—556 DUAL TIMER
IC3; 4—7400 QUAD 2-INPUT NAND
IC5—7404 HEX INVERTER
IC6—7402 QUAD 2-INPUT NOR
IC7—7476 DUAL J-K FLIP-FLOP
IC6-A
(19)
VRST
IC5-B
IC1-B
R3 500K
R4 3.3K
C2 01
C6 .047
IC3-B
C10 1μF
IC5-D
R10 470Ω
IC6-C
IC4-A
OUTPUT
(16)
(13)
(14)

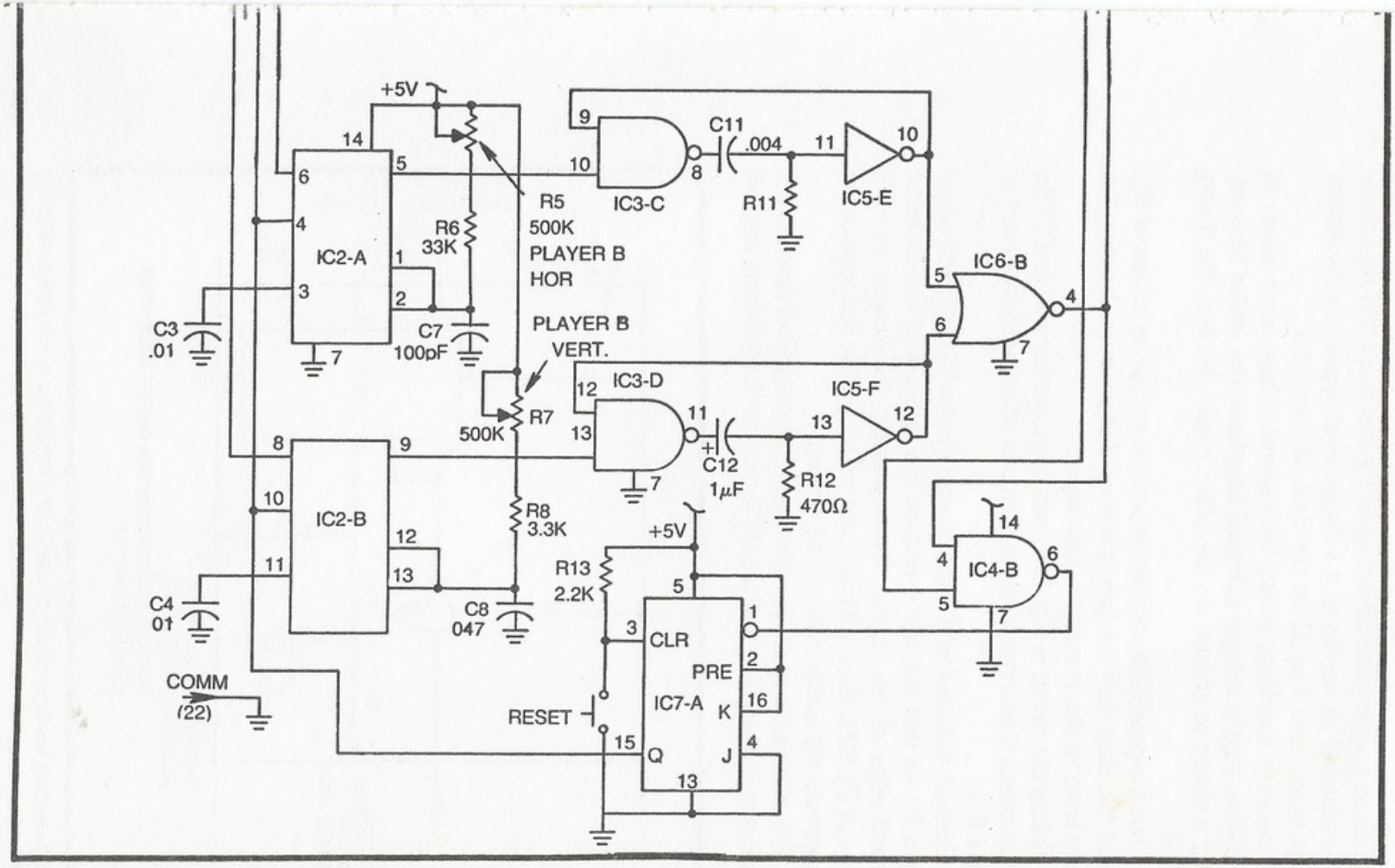


Fig. 5-5. Complete circuit for a simple game of tag.

This technique takes advantage of the fact that a 555-type timer can be adjusted by means of a voltage level applied to its control voltage input (pins 3 or 11 on the dual 556 version).

Generally speaking, pulling the control voltage input closer to the positive supply voltage potential lengthens the timing interval. Pulling it closer to ground, on the other hand, shortens the timing interval.

Thus it is possible to set the basic timing range by means of R1, R2, and C2, then vary the timing around that point by changing the voltage level to the control voltage input.

Set up the circuit in Fig. 5-6, omitting capacitors C7 and C8 for the time being. Connect a temporary jumper wire to the wiper arm of control R7.

Connect the loose end of the jumper wire to COMM, and adjust trimpot R1 so that the white vertical line appears at the extreme left-hand edge of the screen. Then connect the jumper to +5V instead of COMM. Adjust R1 until the vertical white line appears on the extreme right-hand side of the screen.

You might have to repeat this operation a couple of times to get a perfect response. The line should appear at the left-hand edge of

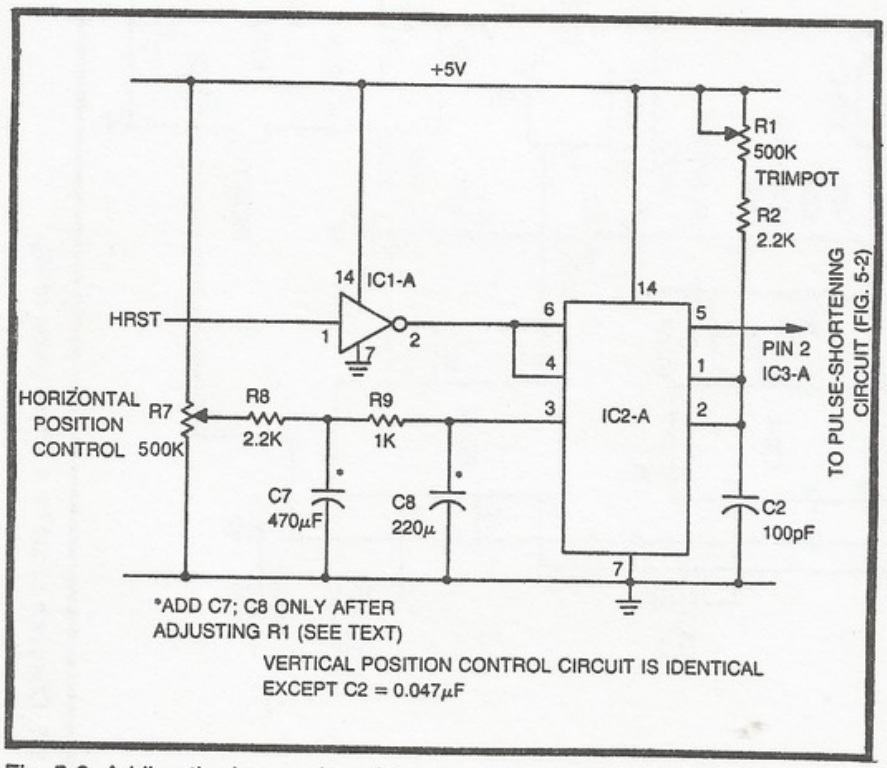


Fig. 5-6. Adding the impression of inertia to a horizontal or vertical positioning control.

the screen when the wiper arm of R7 is to COMM, and it should move to the extreme right-hand side when the wiper arm of R7 is to +5V.

Once you have set the position of R1 to your satisfaction, don't move it again. Remove the jumper from R7 and position the line on the screen by means of that control.

With this preliminary alignment job out of the way, insert capacitors C7 and C8 as shown in Fig. 5-6. As you adjust R7 now, you will find that the line responds as through it has some "slop" or inertia. The line's response, in other words, is not immediately coupled to changes at the position control.

While the values of "inertia" resistors R8 and R9 are critical to the alignment of the timer, the values of C7 and C8 are not. You can change the values of those capacitors to get the amount of inertia you want. The larger the values of C7 and C8, the more inertia the line seems to have.

The vertical positioning circuits can be modified in a similar fashion, triggering with VRST and changing the value of C2 from 100 pF to 0.047 μF. The initial alignment procedure is the same one already described for the "sloppy" horizontal position control circuit.

MANUAL CONTROL OF COMPLEX FIGURES

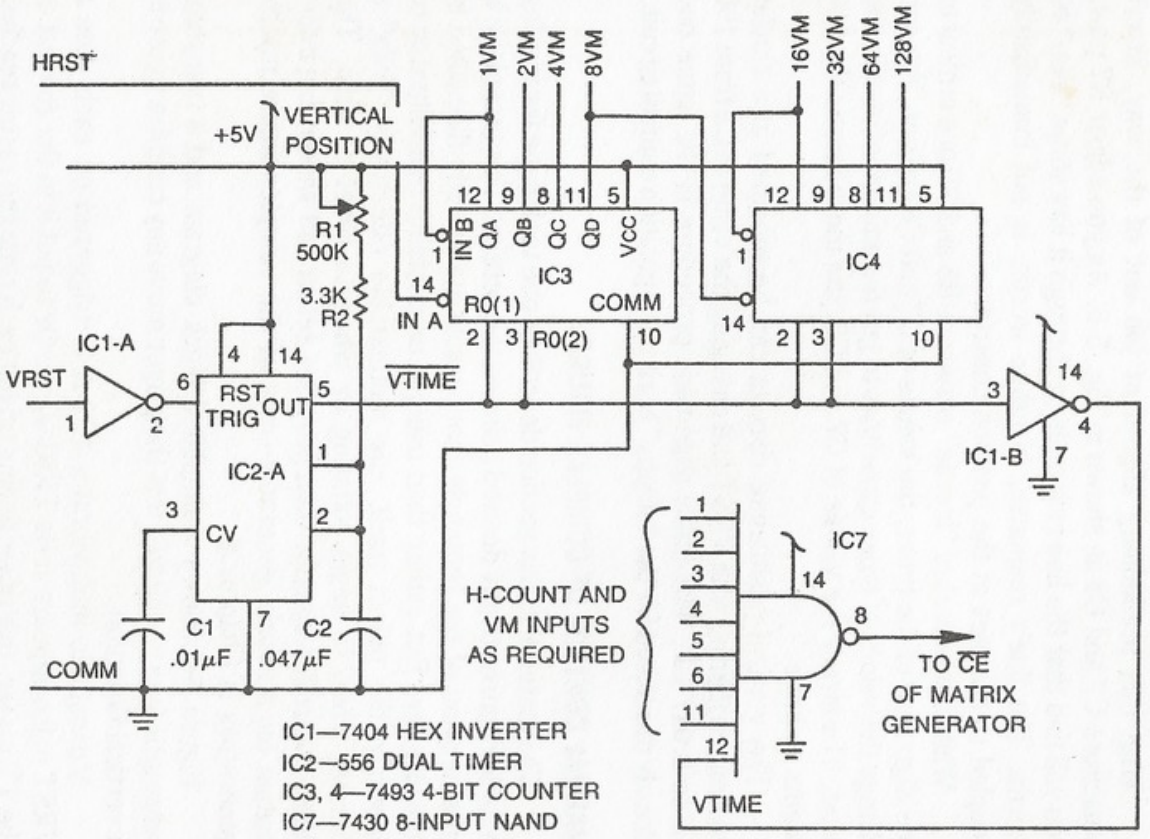
The material in this section describes one technique for moving complex figures to any desired point on the screen. This technique is an extension of the rectangle-motion scheme already discussed in this chapter. But rather than using a monostable multivibrator to generate the object itself, the monostable effectively "tells" a counter when to begin counting or when to reset to zero. The timer-controlled counter then provides select and windowing information for a matrix generator—any of the complex figure matrix generators in Chapter 4.

Figure 5-7 shows a simplified block diagram and a complete schematic for a circuit that lets the player move any complex figure in a vertical direction.

Monostable multivibrator IC2-A is triggered on each time a VRST pulse appears at its TRIG input. Or to put it in the context of the TV raster, this timer begins timing each time the raster reaches the bottom of the screen.

The timer's output remains at logic 1 through the vertical-blanking interval and into some portion of the vertical-scanning interval determined by the values of C2, R2, and R1. A very similar timing circuit in Fig. 5-2 generates a horizontal line that can be

A


HRST
+5V
VERTICAL POSITION
R1 500K
3.3K R2
1VM
2VM
4VM
8VM
16VM
32VM
64VM
128VM
IC3
IC4
IN B
QA
QB
QC
QD
VCC
IN A
R0(1)
R0(2)
COMM
VTIME
IC1-A
IC1-B
VRST
RST
TRIG
OUT
IC2-A
CV
C1 .01μF
C2 .047μF
COMM
H-COUNT AND VM INPUTS AS REQUIRED
IC7
TO CE OF MATRIX GENERATOR
VTIME
IC1—7404 HEX INVERTER
IC2—556 DUAL TIMER
IC3, 4—7493 4-BIT COUNTER
IC7—7430 8-INPUT NAND

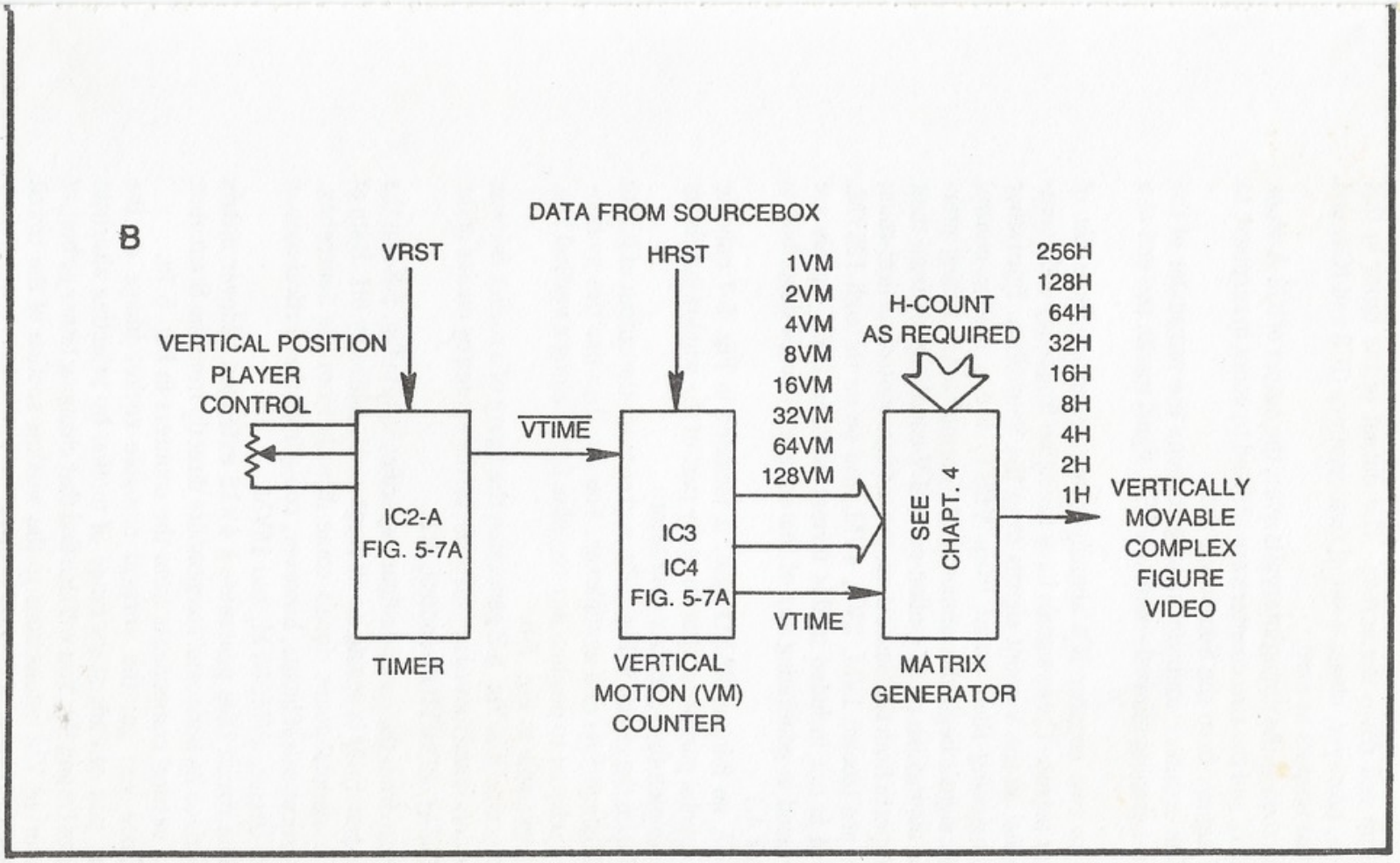


Fig. 5-7. Manual control of complex figures. (a) Basic circuit diagram for vertical motion control. (b) General block diagram for building vertically movable complex figures.

moved up and down the screen. The output of the timer in this instance, however, clears a set of two counters (IC3 and IC4) and holds the outputs at zero.

As soon as the timing interval is over, the output of IC2-A drops to logic 0, and the two counters are allowed to count in response to HRST signals from the Sourcebox unit.

The counters continue running through the remainder of the vertical-scanning interval—until a VRST signal marks the end of a frame.

The two counter ICs actually generate an alternate set of V-count pulses. These signals have the same frequency and counting format as the V-count signals from the Sourcebox. But rather than beginning their count from VRST, these motion-control V-count signals begin the moment IC2-A completes its timing interval. To distinguish these motion-control V-count signals from their counterparts from the Sourcebox, the motion-control vertical-count signals are labeled 1VM, 2VV, 4VM, and so on through 128VM. (256VM is not included in this format because 256V signals are rarely used in generating any of the complex figures described in Chapter 4.)

IC7, an 8-input NAND gate, is included in Fig. 5-7 only for experimental purposes. It is normally part of the windowing circuit for a complex-figure matrix generator.

Set up the circuit in Fig. 5-7a, and connect the output of IC7 to the $\overline{CE}$ connection of a multiplexer. You can then use this motion-control scheme to position any complex figure along a vertical line. See an example in Fig. 5-8.

The circuit in Fig. 5-8 generates the figure of a rocket that can be positioned and moved up and down on the screen by means of the VERTICAL POSITION control, R1.

Note from the complex-figure specifications in Fig. 5-8 that the matrix generator is seeing its S0 input as 4H folded by 8H. Both of these horizontal-count signals come directly from the Sourcebox. The vertical-select inputs, however, come from the vertical-motion control circuit, 4VM, 8VM, and 16VM.

The circuit thus generates a 4×16 extended foldover matrix that derives its horizontal components directly from the Sourcebox and its vertical components from the counters in Fig. 5-7a.

Once you get the vertical movable rocket image on the screen, you can adjust its range of motion by selecting alternate values of C2 and R2. You will also find that changing either or both of the inverted VM connections to the window section of the circuit modifies the range of vertical control.

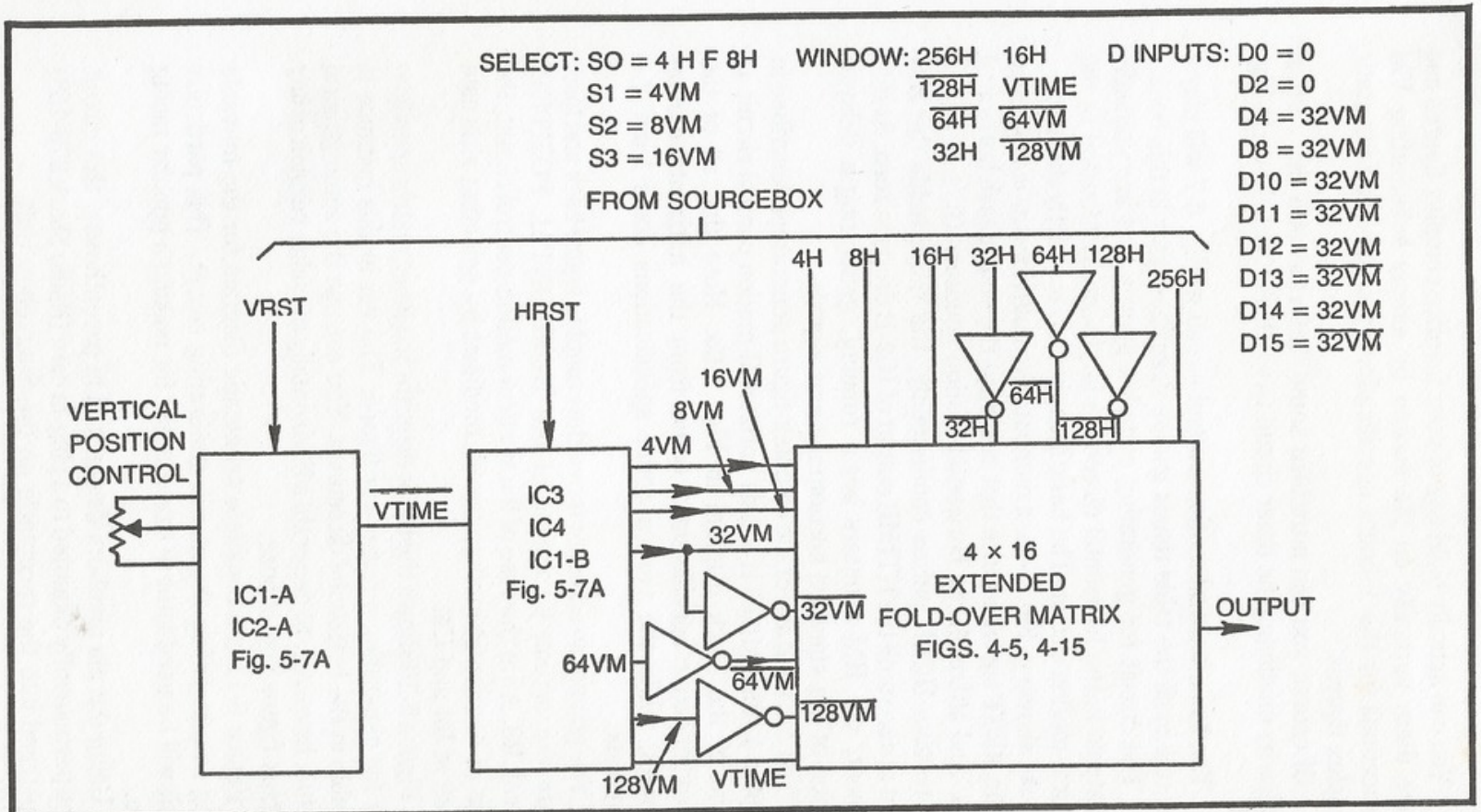


Fig. 5-8. Specifications and block diagram for making a vertically movable rocket figure.

You can actually build any of your favorite complex figures and move them vertically on the screen by simply substituting VM connections for the V-count specifications you specified for static complex figures.

Of course you can introduce some "intertia" into the vertical motion by modifying the timer circuit to work like that shown in Fig. 5-6.

The basic vertical-motion control circuit in Fig. 5-7 will play a vital role in all the table tennis games described later in this book.

The circuit for generating complex figures that are manually adjustable in the horizontal directions is practically identical to the vertical-motion circuit. The basic idea, in fact, is exactly the same.

As shown in Fig. 5-9a, a monostable multivibrator is triggered by the HRST signal, a pulse that occurs at the end of each horizontal scan. And although the horizontal-motion counters (IC's 5, 6, and 8A) receive HCLK pulses continuously, the timer holds the HM count at zero until the HTIME output of IC2-B drops to zero. At that moment, the HM counters begin running, generating a delayed version of the standard horizontal-count signals.

You can build any of the complex-figure generators described in Chapter 4, then attach it to this horizontal-motion control circuit as shown in the block diagram in Fig. 5-9b. Note that all of the horizontal-count specifications come from the horizontal-motion counters, while the vertical-count specifications come from the Sourcebox.

The player can adjust or move the complex figure back and forth across the screen by means of the HORIZONTAL POSITION control, R3. As in the case of the vertical-motion-control circuit, the range of horizontal motion can be modified by selecting alternate values of R4 and C4.

Figure 5-10 shows the block diagram and basic matrix specifications for generating a racing-car figure. The car in this instance is movable in the horizontal directions. You can use the same general scheme, however, to generate a horizontally movable version of any complex figure you choose.

Figure 5-11 is a complete schematic diagram for the movable racing car, including the matrix-generating circuit. This particular circuit will be used later in this chapter for making a popular racing game.

Using the pin numbers designated in parentheses, this circuit can be permanently mounted to a plug-in card (Radio Shack 276-153) and plugged into the receptacle on the Sourcebox Unit.

The HORIZONTAL POSITION control, R1, should be mounted to a small project box and wired to the main circuit at pins 5 and 6. If touching this little box causes distortion of the displayed figure, run an additional wire from the box, itself, to circuit COMM.

The figure can be viewed alone by connecting the inverted output of the multiplexer to card pin 16 (GAME VID IN). When this circuit is used as part of a larger game format, however, you will take the output from pin 15.

After mastering the technique for building circuits that allow vertical or horizontal motion of any complex figure, you should be able to generalize the procedure to build circuits having both vertically and horizontally controlled motion of any complex figure.

To achieve manual control over both vertical and horizontal positioning, simply build the two circuits in Figs. 5-7a and 5-9a, using their VM and HM outputs for the V- and H-count specifications. Any figure generated by the procedures outline in Chapter 4 can be moved in this fashion.

It is important to experiment with this motion-control scheme until you grasp some of its more subtle features and master them. Proper windowing and range of horizontal and vertical motion can cause some headaches for anyone who has not done their homework with this system.

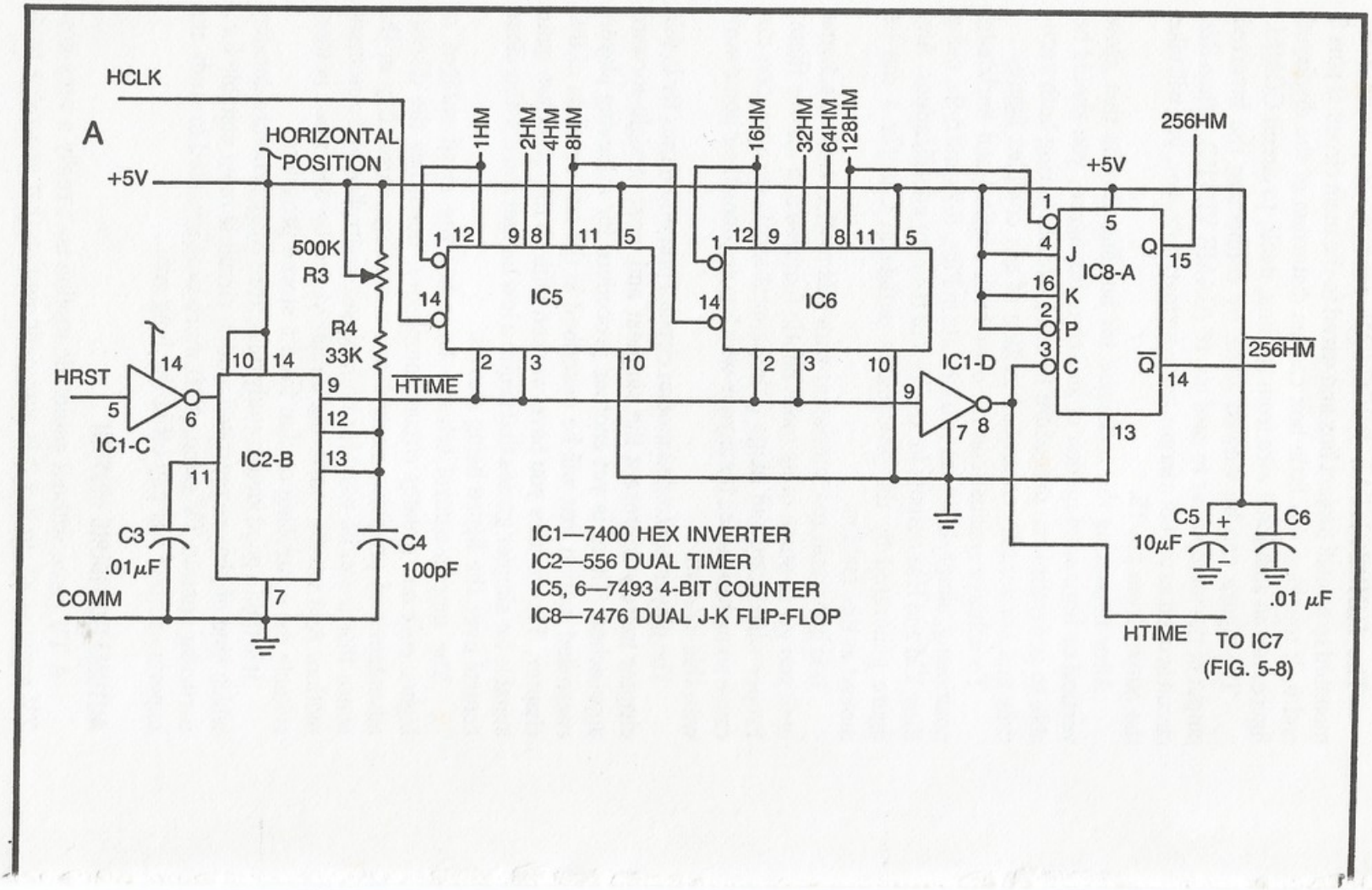
The player-controlled motion circuits presented thus far in this chapter merely represent the simplest and most straight-forward approaches. There is yet another procedure for achieving player-controlled motion that will be described in the last sections of this chapter. The circuits you have worked with to this point are quite suitable for simpler games that reqire a low budget and not-too-close control over the figure being moved.

The motion-control scheme later has the disadvantages of higher cost and greater circuit complexity, but it has the distinct advantages of precision control and versatility, versatility in the sense that it can be used for both player-controlled and automatic motion. And what's more, the circuit yet to be described is most suitable for controlling motion from a stored program.

It is thus a good idea to study this entire chapter before deciding which kind of player-controlled motion circuit is most suitable for a particular custom TV game. A bit more time invested in study and experimentation will pay off in the long run.

AUTOMATIC FIGURE MOTION

A TV game without automatic motion isn't really a very good TV game at all. In fact it is automatic motion that separates tradi-


HCLK
A
HORIZONTAL POSITION
+5V
500K
R3
R4
33K
HRST
IC1-C
IC2-B
C3
.01μF
COMM
C4
100pF
HTIME
1HM
2HM
4HM
8HM
16HM
32HM
64HM
128HM
IC5
IC6
IC1-D
IC8-A
256HM
C5
10μF
C6
.01 μF
IC1—7400 HEX INVERTER
IC2—556 DUAL TIMER
IC5, 6—7493 4-BIT COUNTER
IC8—7476 DUAL J-K FLIP-FLOP
HTIME
TO IC7
(FIG. 5-8)

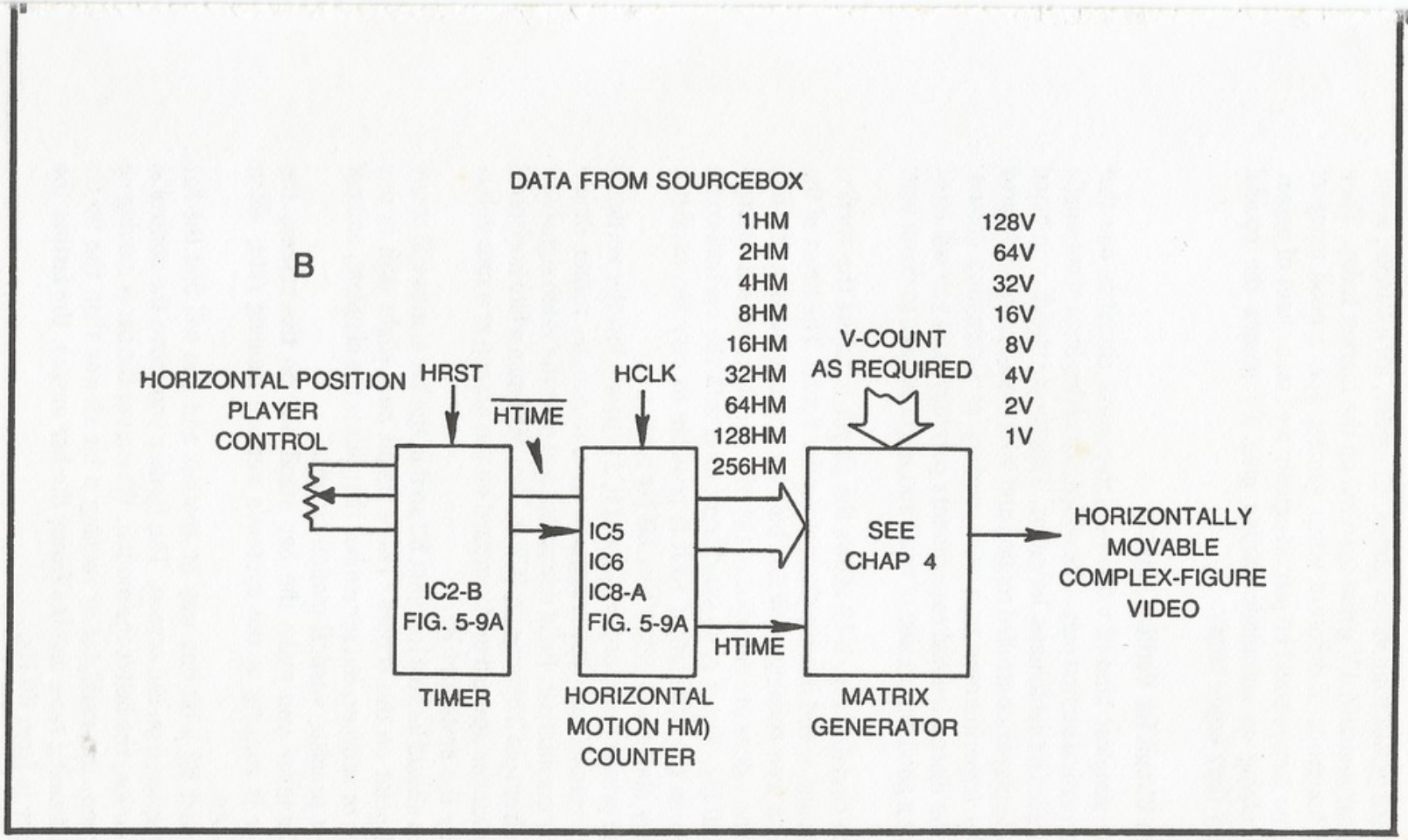


Fig. 5-9. Manual control of complex figures. (a) Basic circuit for horizontal motion control. (b) General block diagram for building horizontally movable complex figures

tional board games from video games. Consider, for example, some of the programmable TV game systems on the market today. They boast of hundreds of different video games; yet, a good many of those same games could be played equally well on a sheet of paper. Games relying on automatic motion give TV games the special popularity they enjoy today.

A Simple Circuit for Vertical Motion

The simplest kind of video-motion-control circuit is one that involves vertical motion only. The simplicity of this form of automatic motion belies its usefulness, however. This simple circuit, combined with some player-controlled motion and interesting complex figures leads the experimenter to a wide variety of interesting games. Follow the discussion and experiments carefully, and you will most likely get a good impression of what you might be able to do on your own.

The circuit in Fig. 5-12 allows the player to adjust the vertical direction and speed of a simple rectangular figure. The heart of the circuit is a free-running oscillator built around one section of a 556 timer. The values are selected so that the oscillator runs at approximately 60 Hz. There is no synchronization with the vertical-count sequence in the Sourcebox, so it is possible to vary the oscillator frequency above and below that 60-Hz rate.

The primary purpose of IC2 and IC3 is to window the vertically moving rectangle so that it appears as a rectangle, rather than a narrow horizontal bar. Build this circuit and note the video signal as it is taken from pin 2 of inverter IC2-A. You will find a white horizontal bar that moves upward or downward on the screen at a rate determined by the setting of R2.

You should be able to adjust R2 so that the bar stands still at any desired point on the screen. There might be a slight drift in one direction or another, but remember that this is the simplest, and not the most precise, vertical-motion circuit.

Whenever you make the bar stand still on the screen, the oscillator is running at the system's vertical framing rate, about 60-Hz rate.

Adjust R2 a bit one way or another and you will find the bar moving upward on the screen. The farther you move the control in that direction, the faster the motion. Whenever the bar is moving up the screen, the oscillator is running a bit slower than the 60-Hz vertical framing rate; and the faster the bar moves, the farther the frequency is from 60-Hz.

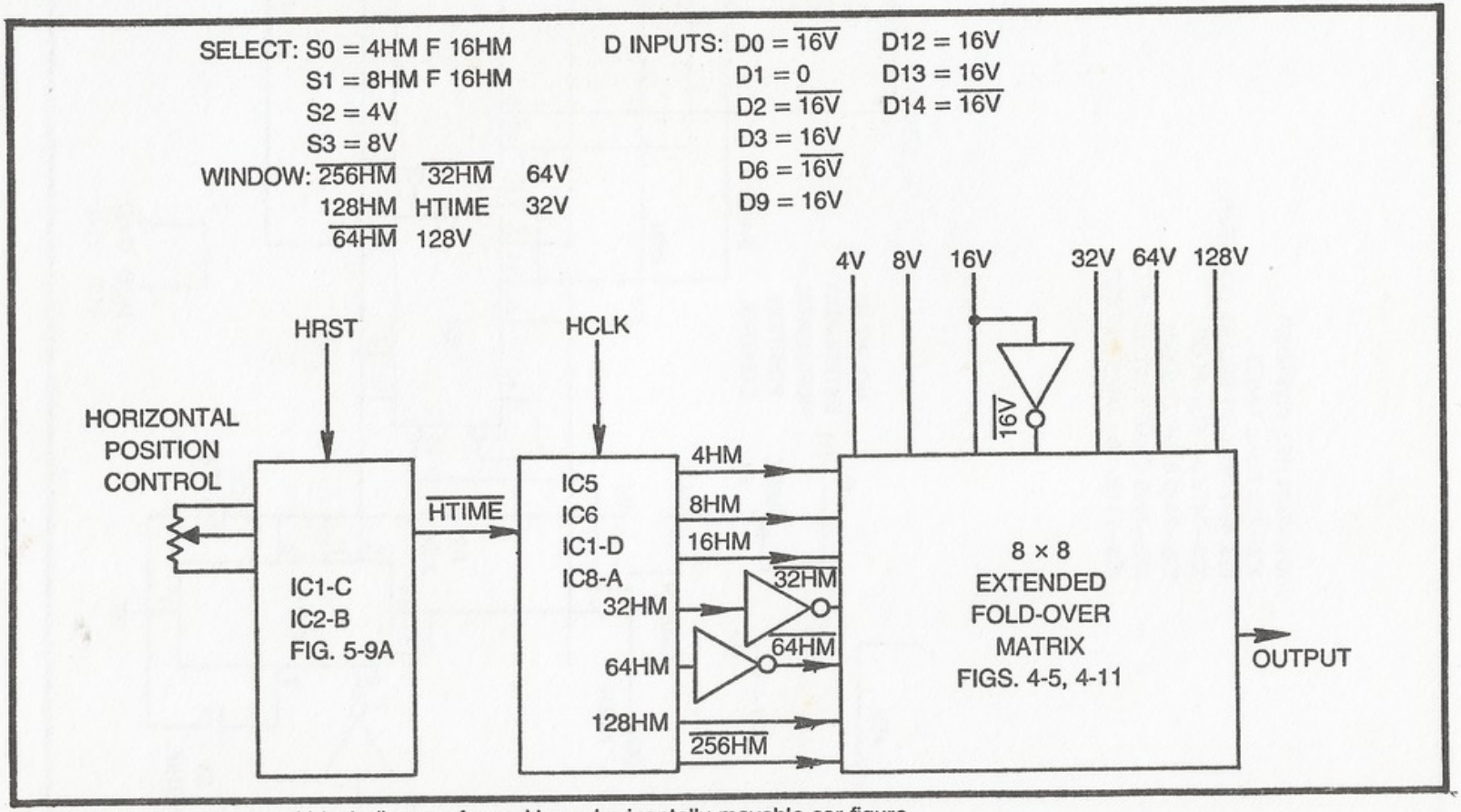
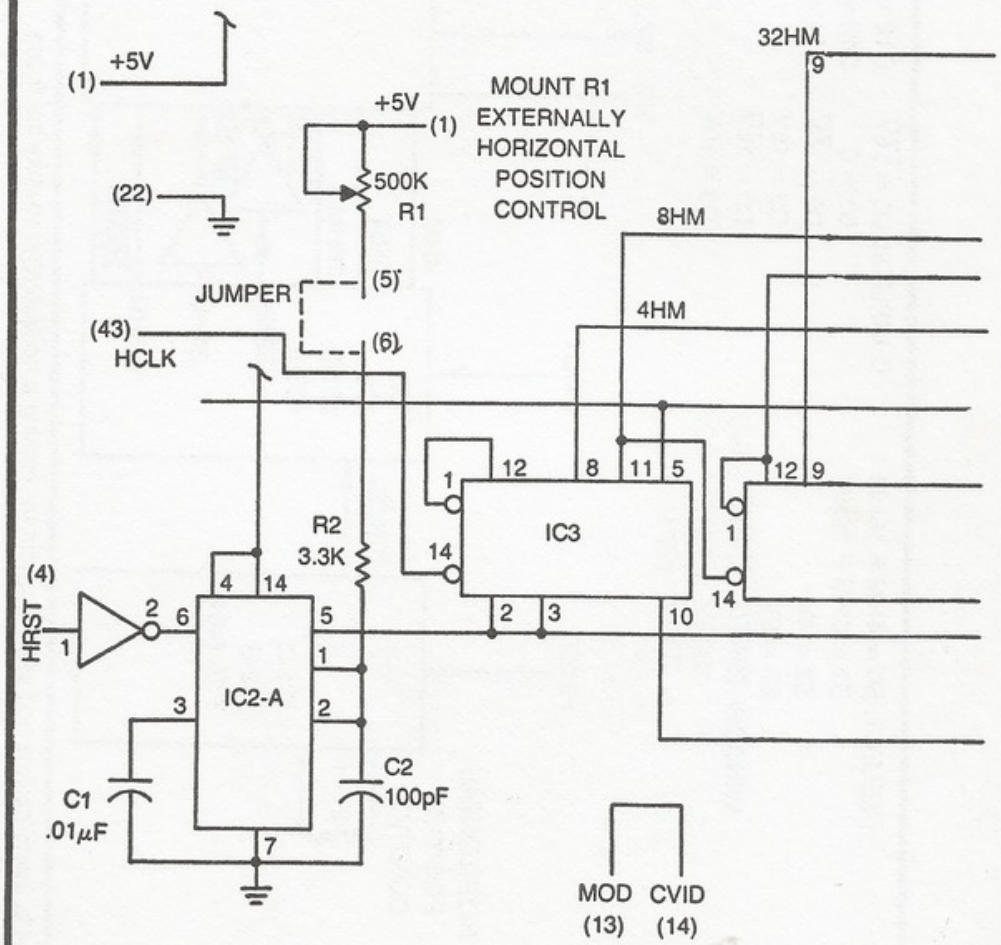
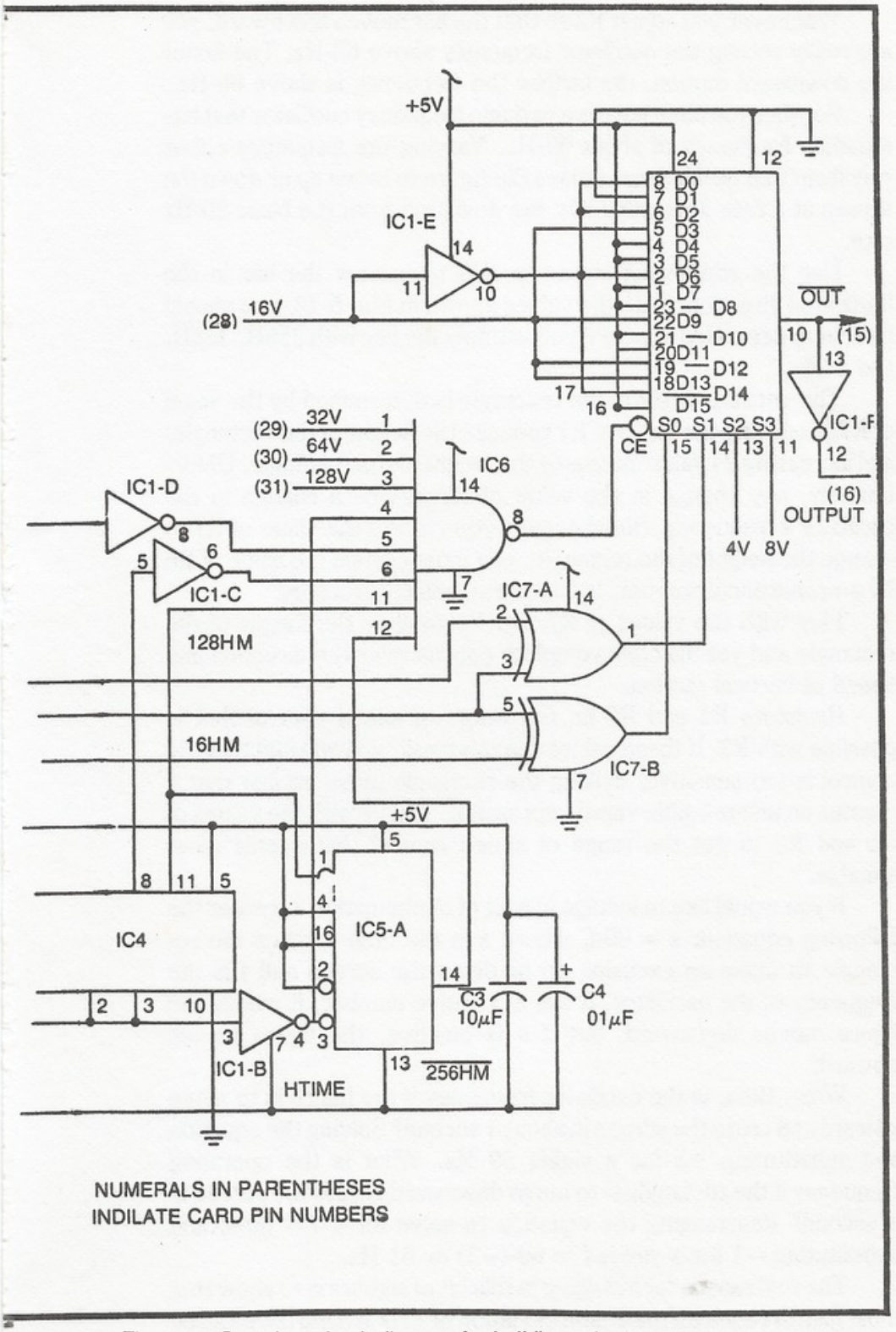


Fig. 5-10. Specifications and block diagram for making a horizontally movable car figure.

IC1—7404 HEX INVERTER
IC2—556 DUAL TIMER
IC3, 4—7493 4-BIT BINARY COUNTER
IC5—7476 J-K FLIP-FLOP
IC6—7430 8-INPUT NAND
IC7—7486 QUAD EXCLUSIVE-OR
IC8—74150 16:1 MULTIPLEXER





NUMERALS IN PARENTHESES
INDILATE CARD PIN NUMBERS

Fig. 5-11. Complete circuit diagram for building a horizontally movable car

Whenever you adjust R2 so that the bar moves downward, you are really setting the oscillator frequency above 60-Hz. The faster the downward motion, the farther the frequency is above 60-Hz.

So what you have here is a variable frequency oscillator that has a middle frequency of about 60-Hz. Varying the frequency either way from that 60-Hz point causes the figure to move up or down the screen at a rate determined by the deviation from the basic 60-Hz rate.

Use the windowing inputs to IC3 to narrow the bar in the horizontal direction. With the values shown in Fig. 5-12, you should be able to see a nice square if you window the bar with 256H, $\overline{128H}$, and 64H.

The vertical height of the rectangle is determined by the value of R7. Reducing the value of R7 reduces the height of the rectangle, and increasing its value increases the height of the rectangle. Unfortunately, any change in the value of R7 causes a change in the oscillator's frequency. So whenever you change the value of R7 to change the height of the rectangle, you must change the value of R6 by a proportional amount, but in the opposite direction.

Play with the values of R7 and R6 to alter the height of the rectangle and yet maintain complete control over the direction and speed of vertical motion.

Resistors R1 and R2 fix the range of speed control that is possible with R2. If these values are too small, you will find that the control is too sensitive, making the rectangle move so fast that it creates an unintelligible visual impression. Tinker with the values of R1 and R2 to get the range of speed control that seems most suitable.

If you would like to indulge in a bit of mathematics, consider the following equation: s = 06-f, where s is the time it takes the rectangle to make an excusion up or down the screen and f is the frequency of the oscillator. If s is a negative number, it means the figure moves downward, but if s is positive, the figure moves upward.

What, then, is the oscillator frequency if the figure is to move upward and cross the screen in about 1 second? Solving the equation and substituting +1 for s yields 59 Hz. What is the operating frequency if the rectangle is to move downward across the screen in 1 second? Rearranging the equation to solve for f: f = 60-s; and substituting −1 for s yields f = 60-(−1) or 61 Hz.

The real reason for indulging in this bit of algebra is to show that most games require a maximum deviation of 1 Hz around the basic 60

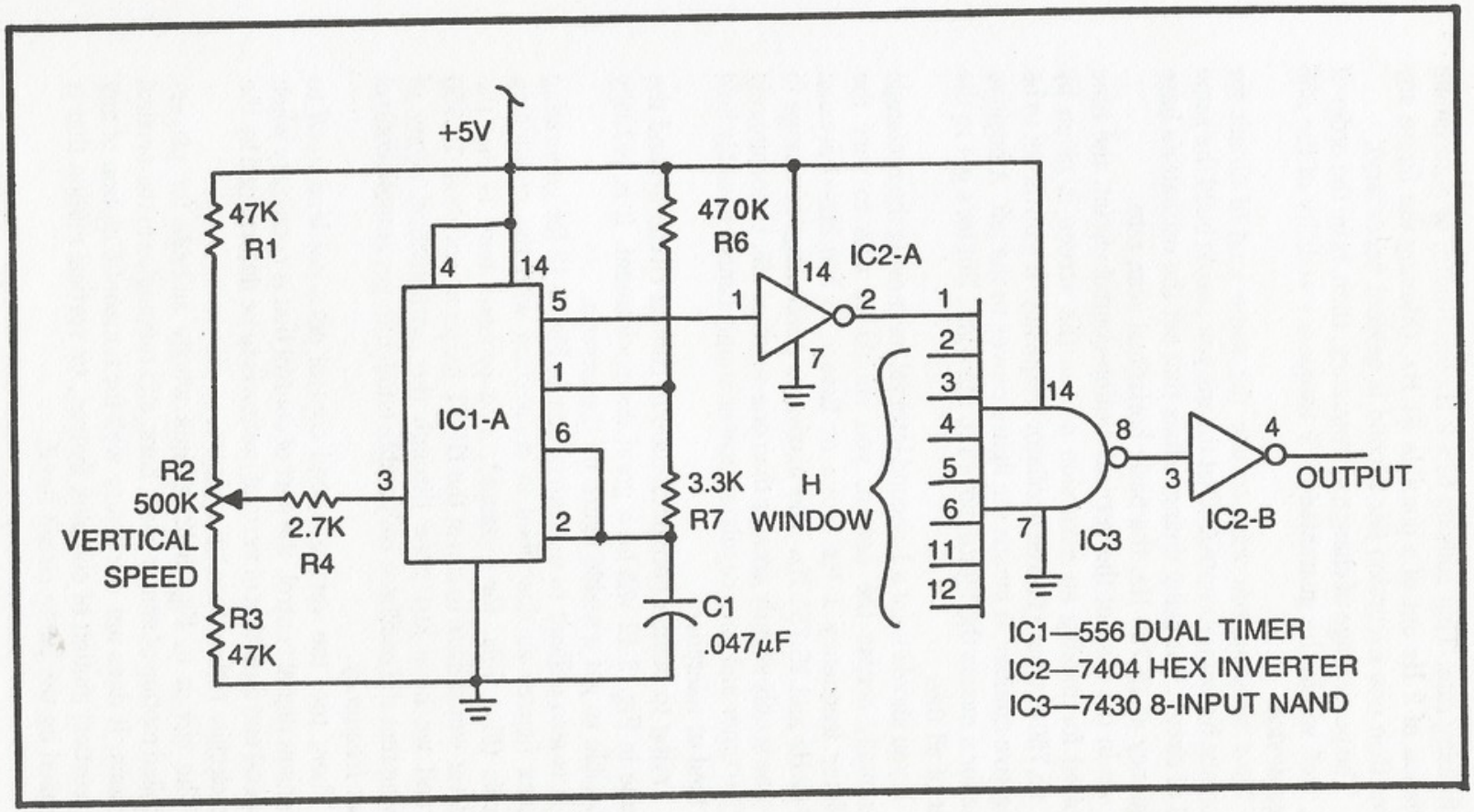


Fig. 5-12. A simple vertical speed and direction control.

Hz framing rate. That adds up to a 2 Hz bandwidth, or a maximum deviation of 2 Hz out of a possible 61 Hz. (Moving the figure any faster than one excursion per second is seldom necessary).

The percentage of change of frequency, then, is on the order of 3%, well within the guaranteed 2% frequency stability of the 556 timer device.

Now suppose you want to use the same kind of circuit for automatic horizontal control. That means you should build the same kind of circuit, but using timing values that set the oscillator's base frequency at 15,750 Hz, the basic horizontal scan rate.

As in the case of the vertical-motion-control circuit, the time required for making an excursion across the screen is given by $s = 15{,}750\text{-}f$, where f is the oscillator frequency. If s turns out to be a negative number, it means the figure moves to the left. A positive value for s means the figure moves to the right. But let's get to the point of all this.

If you should want a 1-second interval for moving the rectangle horizontally across the screen, you should be able to vary the oscillator frequency 1 Hz above or below 15,750 Hz—between 15,749 Hz and 15,751 Hz. The maximum percentage of change in this case is only slightly greater than one-tenth of 1%. Unfortunately the 556 timer and your regulated power supply cannot possibly hold that kind of tolerance.

Trying to build a horizontal-motion-control circuit around the scheme in Fig. 5-12 will be a great disappointment. It is virtually impossible to get a stable figure on the screen.

It is also difficult to use the circuit in Fig. 5-12 for generating complex figures as described in connection with the positioning controls (Fig. 5-9a, for instance). The problem here is that the oscillator would have to run at the HRST frequency of about 15,750 Hz, and we have just gone through the mathematical agony of showing that the oscillator isn't stable enough to get smooth control at that frequency.

Then, too, the vertical speed control does not lend itself to convenient digital control, the sort of control that is necessary when the speed and direction of vertical motion is to be determined by the game circuit, rather than by a player.

The circuit in Fig. 5-12 is thus wholly suitable for player-controlled motion of simple lines, bars, and rectangles in the vertical directions. It does not work very well for horizontal motion of any kind, vertical motion of complex figures, or vertical motion that is controlled by the game circuit itself.

Fortunately there is an alternate scheme that overcomes all three of these disadvantages. The principle involved here is commonly called *slipping-counter motion*. The idea is to build a horizontal- or vertical-counter circuit that is practically identical to those in the Sourcebox unit. The second counter circuit, however, runs out of sync with those in the Sourcebox. In effect, the images created by this slipping counter moves across the screen by virtue of the fact they are out of sync with the Sourcebox counters creating the raster pattern.

Slipping-Counter Vertical Motion Control

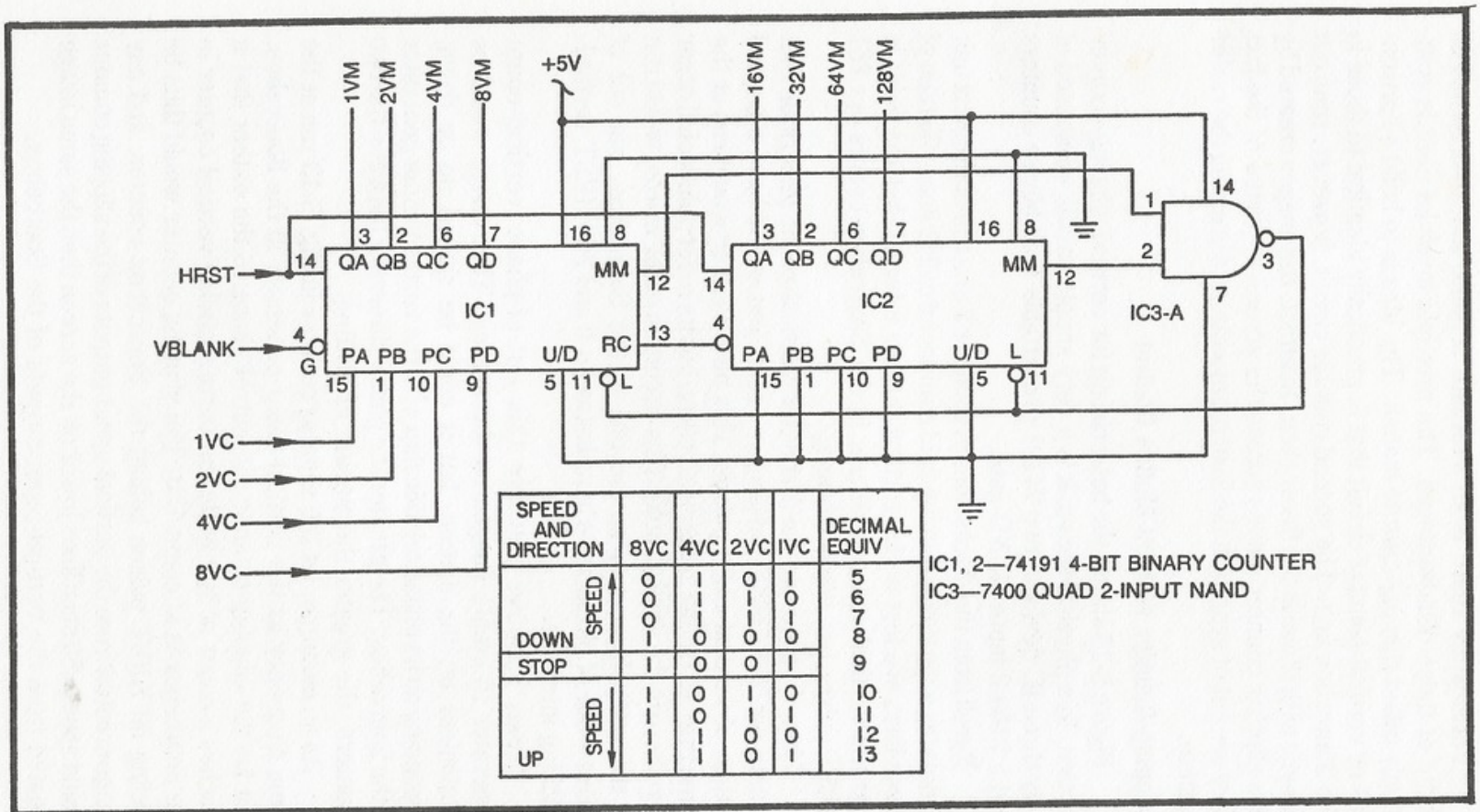
Figure 5-13 shows the basic circuit for vertical-slipping-counter motion. In a physical sense, it is a very simple circuit, composed of only three IC devices: two 74191 presettable 4-bit binary counters and a 7400 2-input NAND gate.

Recall from the discussions in Chapter 2 that the Sourcebox unit generates a vertical-scanning field composed of 261 lines. Sixteen of these lines are lost in the vertical retrace interval, but the point is that the vertical-count generator in the Sourcebox counts out 261 HRST pulses per vertical frame.

Now if you could build another vertical-count generator that counts out 261 HRST pulses per cycle, you would have a second source of vertical-count pulses. This new counter would run at the same rate as the one in the Sourcebox, but its reset point could occur anywhere in the vertical field. This circuit, in other words, would run at the same frequency as the one in the Sourcebox, but out of phase—out of phase anywhere between 0 and 260 HRST vertical-clocking intervals.

Next, suppose you use this out-of-phase vertical-count generator to create images on the screen. That image would be motionless on the screen, but it could be shifted up or down, depending on its phase relationship with the vertical-count generator in the Sourcebox. The greater the phase difference between the two counters, the greater the amount of shifting.

As an example, let the slipping counter in Fig. 5-13 run at the same frequency as the vertical-count generator in the Sourcebox. But let the slipping counter run out of phase to the extent that it reaches a count of 100 while the corresponding vertical counter in the Sourcebox is at count 150. The slipping counter would thus be cycling 50 HRST pulses behind the Sourcebox counter, and any image created from the vertical-count outputs of the slipping counter would appear 50 scan lines lower on the screen than the same image created from the vertical-count outputs of the Sourcebox.



SPEED AND DIRECTION		8VC	4VC	2VC	IVC	DECIMAL EQUIV
DOWN	SPEED	0	1	0	1	5
		0	1	1	0	6
		0	1	1	1	7
		0	1	1	0	8
STOP		1	0	0	1	9
UP	SPEED	1	0	1	0	10
		1	0	1	1	11
		1	1	0	0	12
		1	1	0	1	13

Fig. 5-13. A slipping-counter control for vertical speed and direction.

Alter the phasing so that the slipping counter reaches a count of 100 while the corresponding Sourcebox counter is at 50, and you will find the image shifted 50 scan lines higher on the screen. The slipping counter in this case is running 50 HRST pulses ahead of the Sourcebox vertical-count generator.

An image created by the VM outputs of the vertical slipping counter will be motionless on the screen as long as the frequency is identical to that of the vertical-count generator in the Sourcebox, but let the frequencies be different, and you will find the image moving up or down the screen.

Suppose the slipping counter has some provision for altering the number of HRST pulses it includes in one vertical frame. If this counter has a counting capacity of 262 instead of the usual 261, it resets one scan line later per frame. The overall effect is that any image created from the slipping-counter outputs moves down the screen at a slow but steady rate.

There are two ways to consider this kind of motion effect. Both lead to the same conclusion, an it is left to the reader to use the point of view that suits his own way of thinking.

One way to explain the steady downward motion just described is to consider that the slipping counter is running one HRST pulse farther out of phase each time one vertical frame is completed. And since the position of the image is determined by phase relationship between the slipping counter and Sourcebox, a continuous change in that phase relationship produces the effect of a steady motion.

Another way to look at the situation is to consider that the slipping counter runs at a slightly different frequency than its counterpart in the Sourcebox. If the slipping counter is cycling at 262 pulses per frame instead of 261, that means it is running at a slightly lower frequency. The slipping counter, in other words, is running out of sync with the Sourcebox counter. And since it is running at a slightly lower frequency, the image "rolls" downward on the screen.

Setting the slipping counter to count out 260 HRST pulses, instead of the 261 pulse-interval from the Sourcebox, the image will appear to move upward on the screen. The idea here is that the slipping counter is running at a higher frequency.

The direction of motion is thus determined by whether the slipping counter is cycling at a higher or lower frequency than the corresponding counter in the Sourcebox. If the slipping counter is running at a higher frequency (short counting) the image moves upward, and if the slipping counter is running at a lower frequency (long counting) the image moved downward . Of course the image is motionless as long as the two frequencies are the same.

Now the greater the frequency difference, the faster the apparent motion becomes. If the slipping counter is set so that it short-counts four pulses per frame, it moves upward about four times as fast as it does when short counting just one pulse per frame. Long counting four pulses per frame, by the same line of reasoning, makes the image move downward rather rapidly.

The table in Fig. 5-13 shows five 4-bit words that can be loaded into the vertical-motion-slipping-counter circuit. When that number is 1001 (decimal 9) the slipping counter runs at the same frequency as the vertical-count generator in the Sourcebox unit, and the result is a motionless image on the screen.

Loading decimal 8, 7, 6, or 5, however, forces the slipping counter to long-count by 1, 2, 3, or 4 HRST pulses per frame, yielding an image that moves downward at the rate of 1, 2, 3, or 4 scan lines per frame. (Since there are 60 frames completed each second, it is possible to calculate the time it takes the image to move down the screen.)

According to the table in Fig. 5-13, loading the decimal equivalents of 10, 11, 12, or 13 causes the image to move upward. The counter actually short-counts by 1, 2, 3, or 4 scan lines per frame in this instance.

All of this discussion merely indicates *what* the circuit in Fig. 5-13 does. Now it is time to investigate exactly *how* it does the job.

Anyone familiar with the fundamentals of digital electronics ought to recognize the two-counter portion of the circuit as an 8-bit synchronous binary counter. The counters are clocked by the HRST input to pin 14 of both IC's. Note, however, that the counters are disabled thrugh the VBLANK interval by means of the VBLANK signal applied to the G (enable) input of IC1. The circuit is thus allowed to count at the HRST rate only as long as VBLANK is at logic 0—at all times except through the VBLANK interval.

The 2-input NAND gate normally shows a logic-1 output, dropping to logic 0 only when the counter outputs reach a maximum count of 11111111. At that instant, the 2-input NAND gate sees a pair of logic-1 inputs, and its output drops to 0.

Whenever the output of IC3-A drops to logic 0, the two counter ICs are loaded with a certain set of binary numbers. IC2 is always loaded with 0000 virtue of the fact that its preset inputs are permanently tied to logic-0 common. IC1, however, is loaded with whatever 4-bit number appears at its preset inputs, PA through PD, and that number is the motion code described in the table accompanying the diagram in Fig. 5-13.

So what happens here is that the counter advances to its maximum count (11111111, or decimal 256) where it is immediately reloaded to the number appearing at the preset inputs of IC1, anywhere between 5 and 13. If the VC inputs show 1001, for instance, the counter replaces 11111111 with 00001001, or decimal 9. It then counts up to 11111111, with a 16-count pause whenever the VBLANK interval occurs. The total number of counting intervals in 1 complete cycle is thus 261, exactly equal to the number of counting intervals generated by the Sourcebox vertical-count circuit. An image using the VM outputs stands still on the screen.

If the VC inputs are altered to show a binary number other than 9, the slipping counter has more or fewer counts per cycle. Loading binary 8, for example, forces the slipping counter to work with one additional HRST pulse per cycle. In effect, this increases the cycle time, or in other words, decreases the frequency. The result in this instance is that images generated by the VM outputs appear to move upward on the screen.

Loading numbers larger than 9 shortens the counting cycle of the slipping counter, thereby making it cycle at a higher frequency. Any image using the VM outputs appears to move down the screen.

Construct the circuit in Fig. 5-13, and make your initial tests of taking the video from one of the higher-order VM outputs, 32VM or 64VM for instance. The horizontal bars on the screen should stand still when loading 1001 at the VC inputs. Then they should move upward rather slowly when loading 1010 or 1011. The bars move at the same speed, but upward, when loading 1000 or 0111 at the VC inputs.

You are now in a good position to generate bars, lines, rectangles, and complex figures that show continuous vertical motion. Use any of the procedures outlined in Chapters 3 and 4, substituting the VM signals where you would normally specify the Sourcebox V-count signals. Use the Sourcebox H-count signals in the usual fashion until you have built the horizontal slipping counter described in the section that follows.

If you have been conducting the vertical-motion experiments as recommended thus far in this chapter, you are probably acutely aware of the fact that the procedure for changing the speed and direction of motion for the slipping counter is somewhat more awkward that of the oscillator-controlled motion circuits.

The slipping-counter technique does not lend itself to direct control by means of a simple variable resistor, whereas the simpler control circuits do. The closing section of this chapter describes a

rather simple circuit for achieving potentiometer control over slipping counters. But in instances where a player is to have manual control over vertical speed and direction, the control circuit in Fig. 5-12 is the better choice, assuming, of course, the image is a simple line, bar, or rectangle.

The digital speed and direction control feature of the slipping-counter circuit, on the other hand, is the better choice in games calling for automatic or machine control of vertical speed and direction. The slipping-counter technique is also the only option open when the figure is more complex than a simple line, bar, or rectangle.

Master the fundamentals of both vertical-control circuits and you will be fully prepared to handle any game-designing situation calling for vertical motion.

Slipping-Counter-Horizontal-Motion Control

While the experimenter might have several options when it comes to selecting vertical-motion-control circuits, no such options exist for controlling the speed and direction of horizontal motion.

One might think the vertical-motion circuits described in this chapter could be modified to suit the needs for horizontal motion. The idea might be to simply substitute HCLK for HRST and HBLANK for VBLANK. This is not the case at all. Merely substituting H-count parameters for V-count parameters creates an image on the screen that is, first, quite confusing and, second, practically useless.

The primary feature of horizontal counting is that the slipping counter must not be permitted to change its count length with every horizontal line. The horizontal-slipping counter must be loaded with its stop code for every visible scan line on the screen. If this is not done, the experimenter sees a series of diagonal lines, rather than straight vertical lines, moving across the screen.

The horizontal-slipping counter must see its stop code at all times except during one particular scan line, preferably one that occurs during vertical blanking.

If the horizontal-slipping counter is then loaded with a number that retards the horizontal timing, the vertical line will appear to move to the right. That retarded count, however, must occur only once during the scanning field.

On the other hand, shortening the count during one particular scan line makes the vertical line appear to move to the left. Again, the counter must see its stop code at any other time.

The circuit in Fig. 5-14 represents the basic horizontal-slipping counter. It is, indeed, the only horizontal-motion circuit used throughout this book.

ICs 1, 2, 3 and 5 in Fig. 5-14 perform the same general function as the vertical-slippng counter in Fig. 5-13. They are responsible for counting out a horizontal-count cycle of nine bits, four bits each for IC1 and IC2, and a ninth bit from the J-K flip-flop, IC5.

The 3-input NAND gate, IC5-A, senses the counter overflow of 111111111, or decimal 511. The starting point for the stop code in this case is binary 10010010, or decimal 137. Take the difference between these 2 figures and you end up with a counting cycle that is 374 pulses long. That is the number of pulses required for making a horizontally movable figures appear motionless on the screen.

An astute reader, however, might note that this number is quite different from the counting cycle of the horizontal-count generator in the Sourcebox unit. Recall that the Sourcebox generates a 454-pulse horizontal cycle.

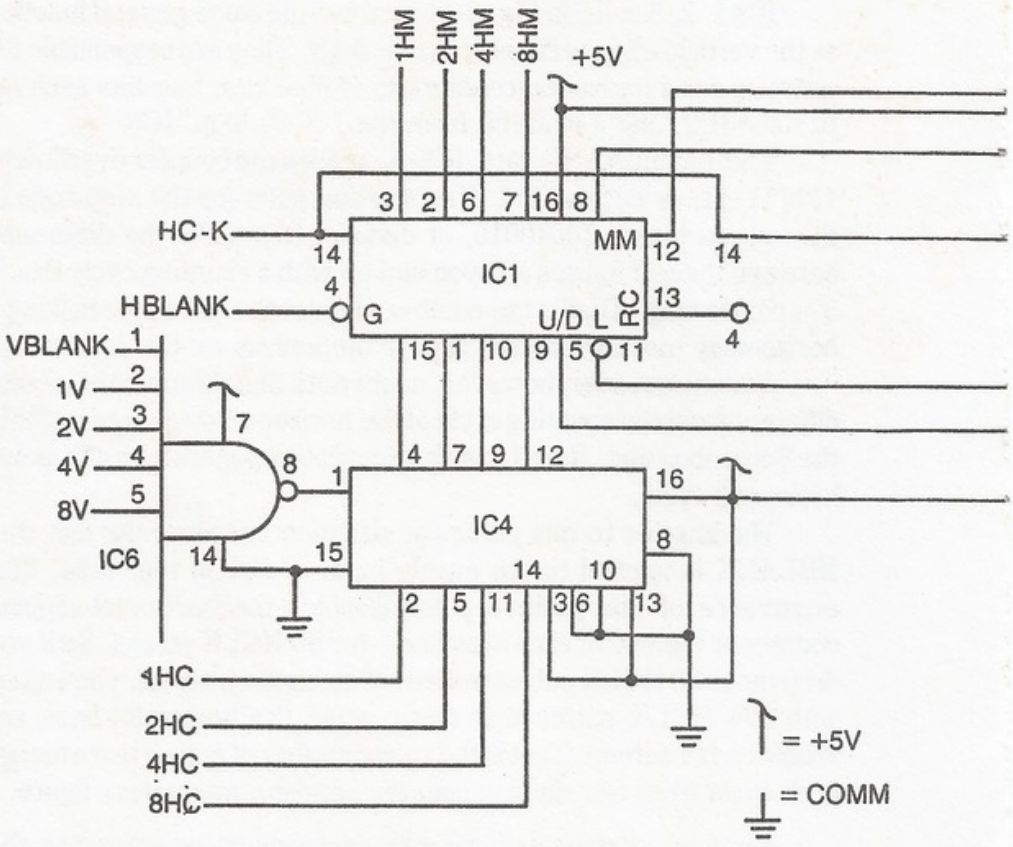
The answer to this particular situation concerns the fact that HBLANK is applied to the enable input of IC1 in Fig. 5-14. The occurrence of this positive pulse disables the horizontal-slipping counter at the end of each scan line—for 80 HCLK pulses. So if you drop those 80 HCLK pulses from the Sourcebox interval, you end up with 374 HCLK pulses that occur while the horizontal lines are visible on the screen. That is the same number of pulses that emerge each cycle from the slipping counter set for a motionless figure.

Any figure having its horizontal components generated by the horizontal-slipping counter thus appears motionless on the screen as long as that counter is being loaded with a 374 HCLK cycle.

Now notice from Fig. 5-14 that the five higher-order bits are always loaded as 01000. The clearing input of IC3-A and the preset inputs of IC2 are always fixed for loading those values, no matter what the motion code might be.

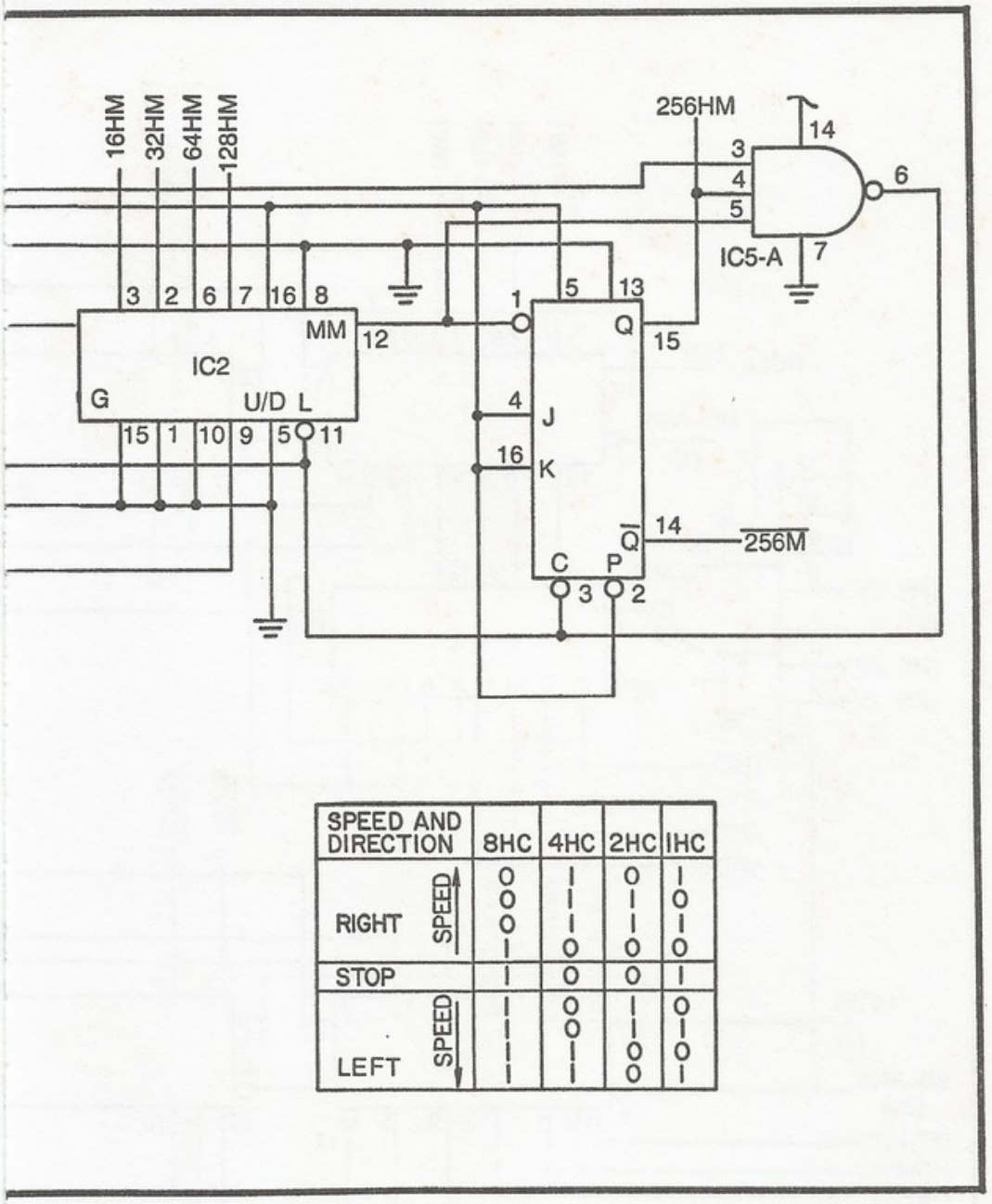
The peculiar feature of the horizontal-slipping counter is that the stop code—the four lower order bits loaded at the preset inputs of IC1—must be loaded at the end of each horizontal cycle. If something other than the stop code is loaded each cycle, the circuit generates moving diagonal lines rather than moving, straight vertical lines.

A motion code other than the stop code is loaded only during one particular scan line that occurs during vertical retrace on the screen. IC4 and IC6 in Fig. 5-14 take care of this situation.



IC1, 2—74191 4-BIT COUNTER
IC3—7476 DUAL J-K FLIP-FLOP
IC4—74157 QUAD 2:1 DATA SELECTOR
IC5—7410 TRIPLE 3-INPUT NAND
IC6—7430 8-INPUT NAND

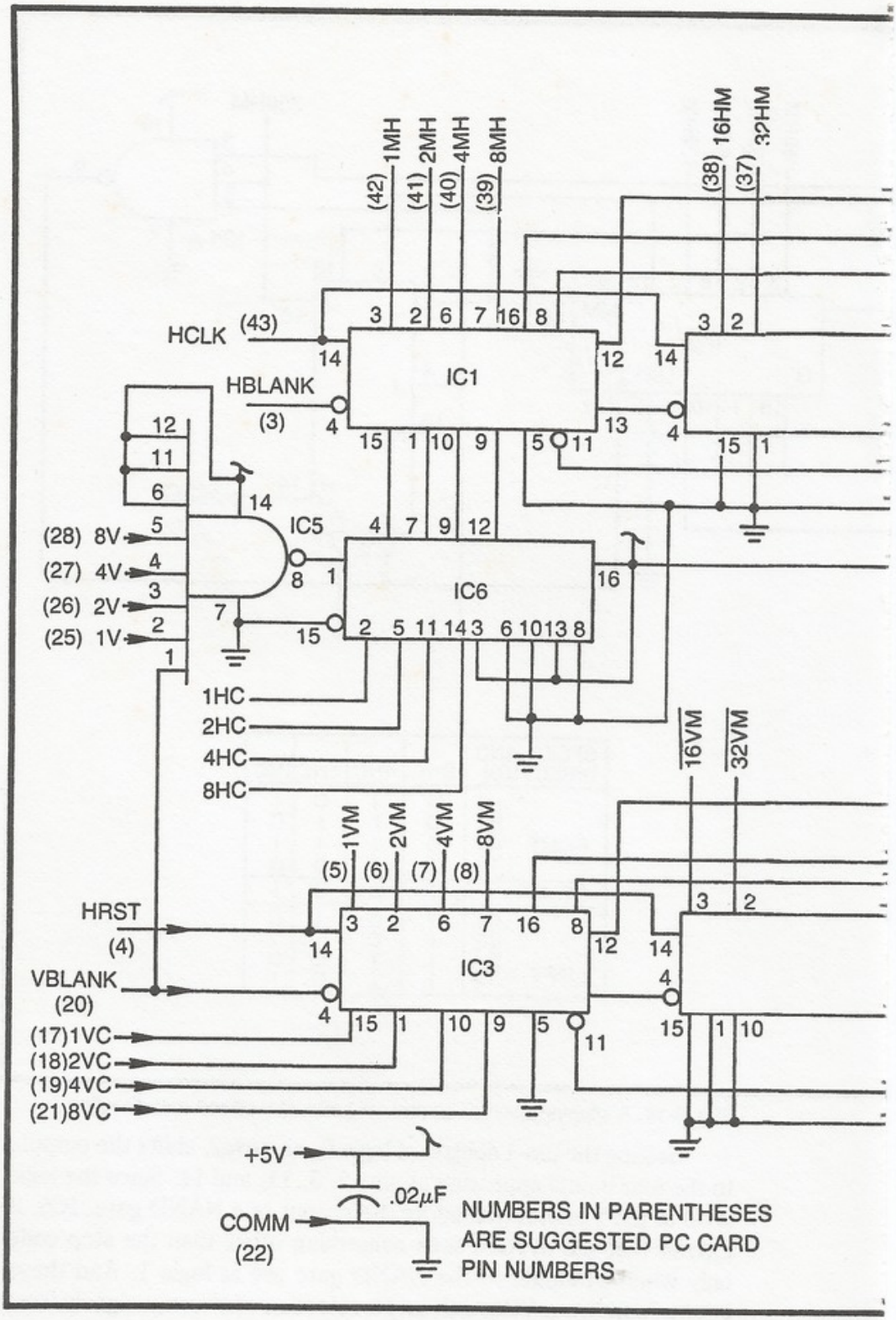
IC4 is a quad 2:1 multiplexer. It is a circuit that works very much like a 4-pole double-throw selector switch. When its input at pin 1 is at logic 1, the inputs at pins 3, 6, 10, and 13 appear at its outputs (pins 4, 7, 9, and 12 respectively). Note that these inputs are fixed at the circuit's stop code for the four lower-order bits: 1001. So as long as the signal at pin 1 of IC4 is at logic 1, the slipping counter is loaded with its stop code.



SPEED AND DIRECTION		8HC	4HC	2HC	1HC
RIGHT	SPEED	0 0 0 1	1 1 1 0	0 1 1 0	1 0 1 0
STOP		1	0	0	1
LEFT	SPEED	1 1 1 1	0 0 1 1	1 1 0 0	0 1 0 1

Fig. 5-14. A slipping-counter control for horizontal speed and direction.

Setting the pin-1 control at logic 0, however, shifts the outputs to the four inputs appearing at pins 2, 5, 11, and 14. Since the logic level at pin 1 is determined by the output of a NAND gate, IC6, it follows that the system sees something other than the stop code only when all inputs to the NAND gate are at logic 1. And these inputs come from VBLANK and a selection of V-count signals from the Sourcebox.


1MH
2MH
4MH
8MH
16HM
32HM
(42)
(41)
(40)
(39)
(38)
(37)
HCLK (43)
HBLANK
(3)
IC1
IC5
IC6
(28) 8V
(27) 4V
(26) 2V
(25) 1V
1HC
2HC
4HC
8HC
16VM
32VM
1VM
2VM
4VM
8VM
(5)
(6)
(7)
(8)
HRST
(4)
VBLANK
(20)
IC3
(17)1VC
(18)2VC
(19)4VC
(21)8VC
+5V
.02μF
COMM
(22)
NUMBERS IN PARENTHESES
ARE SUGGESTED PC CARD
PIN NUMBERS

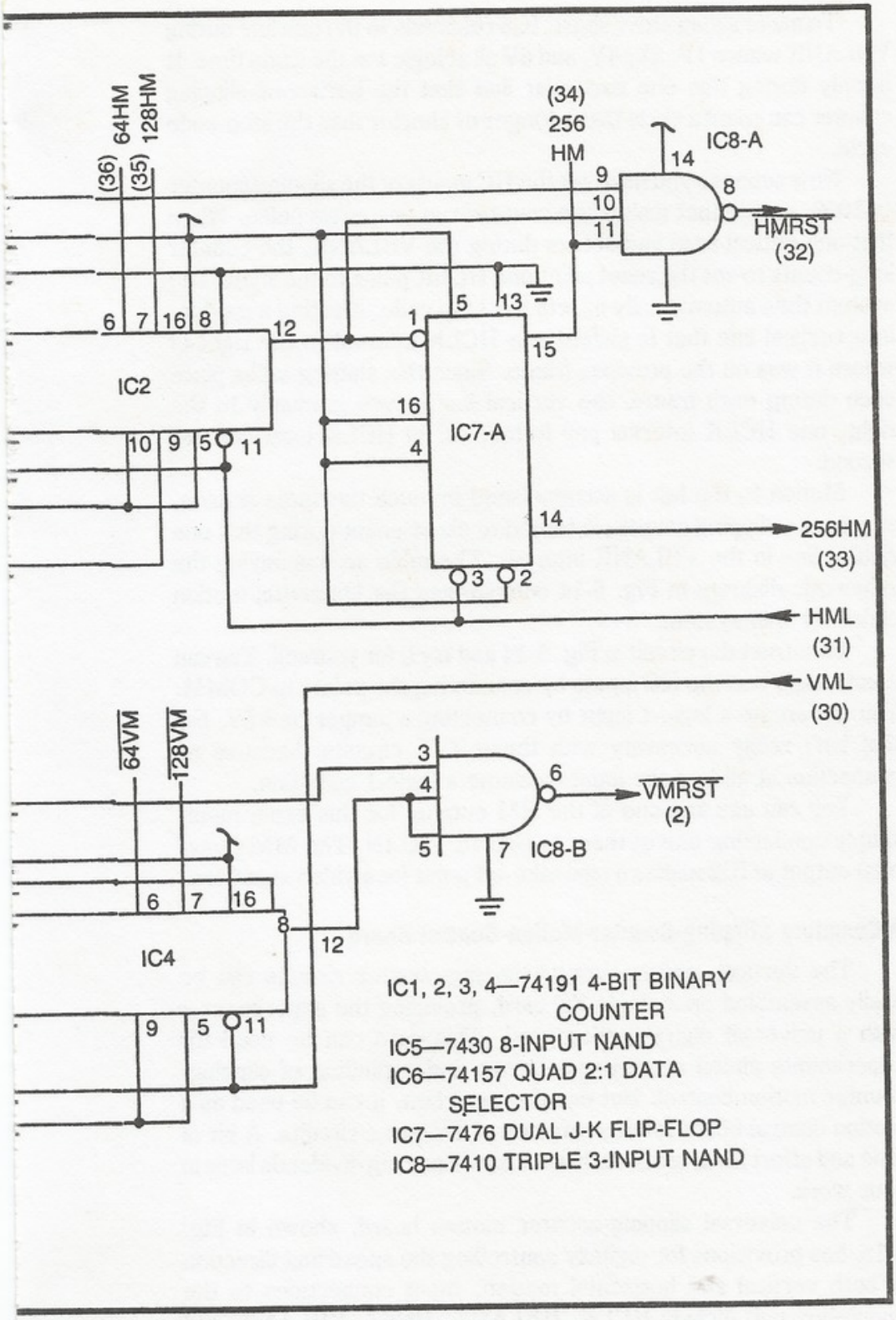


Fig. 5-15. Schematic diagram for a slipping-counter motion control circuit board

To make a long story short, IC6 responds to the one line during VBLANK where 1V, 2V, 4V, and 8V all at logic 1 at the same time. It is only during this one particular line that the horizontal-slipping counter can count a cycle that is longer or shorter than the stop-code cycle.

Now suppose you have set the HC inputs of the slipping counter to 1000, a code that makes the counter run one extra pulse. When that one critical scan line occurs during the VBLANK, the counter long-counts to set the reset point one HCLK pulse to the right. The system then automatically injects the stop code, creating a motionless vertical line that is shifted one HCLK interval to the right of where it was on the previous frame. Since this shifting takes place once during each frame, the vertical bar moves gradually to the right, one HCLK interval per frame, or 60 HCLK intervals per second.

Motion to the left is accomplished in much the same fashion, except the slipping counter is forced to short-count during that one critical line in the VBLANK interval. The table accompanying the schematic diagram in Fig. 5-14 summarizes the horizontal-motion codes for this system.

Construct the circuit in Fig. 5-14 and try it for yourself. You can create logic-0 at the HC inputs by connecting the points to COMM. You can create a logic-1 input by connecting a jumper to +5V, but that isn't really necessary with these TTL circuits, because no connection at all lets the inputs assume a logic-1 condition.

You can use any one of the HM outputs for this experiment, simply connecting one of them to GAME VID IN. The MM (max/min) output of IC2 makes a nice take-off point for a video signal too.

A Complete Slipping-Counter-Motion-Control Board

The vertical- and horizontal-slipping-counter circuits can be easily assembled on a single PC card, providing the experimenter with a universal digital-motion card. This card can be used for experiments aimed at getting a better understanding of slipping-counter motion control. But equally important, it can be used as a motion control board for any number of TV game circuits. A bit of time and effort invested in such a board will pay big dividends later in your work.

The universal slipping-counter motion board, shown in Fig. 5-15, has provisions for digitally controlling the speed and direction of both vertical and horizontal motion. Input connections to the Sourcebox unit include HCLK, HBLANK, HRST, VBLANK, and

vertical-count signals 1V, 2V, 4V, and 8V. Of course +5V and COMM should be included on this list.

The control inputs are 1HC, 2HC, 4HC, and 8HC for horizontal motion, and 1VC, 2VC, 4VC, and 8VC for vertical motion. The motion-control codes for these two sets of inputs are included in Figs. 5-14 and 5-13 respectively.

The slipping-counter outputs are designated 1HM through 256HM for horizontal counting, and 1VM through 128VM for vertical counting. These outputs can be used for generating lines, bars, rectangles, and complex figures, as described in Chapters 3 and 4. Merely substitute the HM signals for H-count specifications and VM signals for the corresponding V-count specifications.

The scheme is left programmable to some extent. The idea here is to give the experimenter the greatest possible amount of flexibility with this one circuit board. To operate the system in the normal fashion, merely connect the $\overline{\text{HMRST}}$ output to $\overline{\text{HML}}$, and $\overline{\text{VMRST}}$ to $\overline{\text{VML}}$. (Leaving these points "programmable" allows the experimenter to insert other kinds of reset circuits that initialize the position of the movable figure.)