Chapter 6

Some Useful Game Control Schemes

Automatic motion is indeed the hallmark of video games, but automatic game sequencing runs a close second. Most TV games employ automatic start-up and stop features, for example. Then there are other kinds of events and sequences of events that occur during a game, some manually controlled and others automatic.

Consider the control schemes for a basic table tennis game. The game sequence is usually initiated manually by depressing a RESTART pushbutton, but it generally ends automatically as one player reaches a certain score. In some instances, the ball-serving sequence is initiated manually, while in other versions of the same game, the ball is served manually. But in either case, the serving sequence is terminated automatically as one player misses the ball. These operations are examples of game control schemes.

Study the circuits in this chapter carefully. A proper understanding of them will make it easier for you to understand the control features of sample games presented later in this book, and it will certainly make the process of designing your own games more fun, easier, and more efficient.

GAME START/RESET CONTROLS

Most TV games begin with some kind of initial action somewhere on the screen, and by the same token, most games include a critical point where the system is to be reset to begin another cycle of one game or begin a new game altogether.

These start and reset operations can be wholly manual, fully automatic, or a combination of the two. In any case, start and reset controls are generally built around a flip-flop circuit.

The flip-flop circuits in Fig. 6-1 are properly classified as R-S flip-flops. To be more precise, the circuit in Fig. 6-1a is an $\overline{R}$-$\overline{S}$ flip-flop, while the one in Fig. 6-1b is an R-S flip-flop. The outputs in both examples are compliments of one another, one is always at logic 1, while the other is at logic 0.

The $\overline{R}$-$\overline{S}$ flip-flop in Fig. 6-1a is set and reset by means of negative-going (active-low) input pulses. Whenever the $\overline{START}$ input is pulled down to logic 0 and $\overline{STOP}$ is held at logic 1. the PLAY output goes to 1 and the $\overline{PLAY}$ output switches to logic 0. As long as $\overline{STOP}$ remains at logic 1, the outputs hold, or remember, this condition, even after $\overline{START}$ returns to logic 1.

The outputs can then be reversed only by pulling STOP down to logic 0, while holding $\overline{START}$ at logic 1.

The R-S flip-flop in Fig. 6-1b works exactly the same way, but in this case the inputs are active high. The outputs are set and reset by means of positive-going pulses at the START and STOP inputs. Whenever START is pulled up to logic 1, for instance, the PLAY output responds by going to logic 1. At that same time, $\overline{PLAY}$ goes to logic 0. The circuit then remains in that particular output state as long as STOP remains at logic 0. The START input can switch between 1 and 0 any number of times, but PLAY remains at logic 1 as long as STOP is held at 0.

You will find the simple arrangements in Fig. 6-1 appearing one or more times in just about every full-scale TV game presented in this book. The only real difference between the two circuits is the polarity of their input waveforms. One uses negative-going pulses, while the other uses positive-going pulses. The choice of using one circuit or the other depends largely on the polarity of the input pulses that are available from the circuits that operate them.

Manual Start Switch Circuits

Many games begin with a player depressing a start pushbutton, and in some instances, cycles within a game are initiated that way. The circuits in Fig. 6-2 show how to interface a START pushbutton with the flip-flop circuits in Fig. 6-1.

The circuit in Fig. 6-2a is the simplest of the three. In this case, resistor R1 keeps the $\overline{START}$ logic level normally pulled up to logic 1. Depressing the START button pulls that logic level down to logic 0, where it remains until the button is released. Any contact bounce

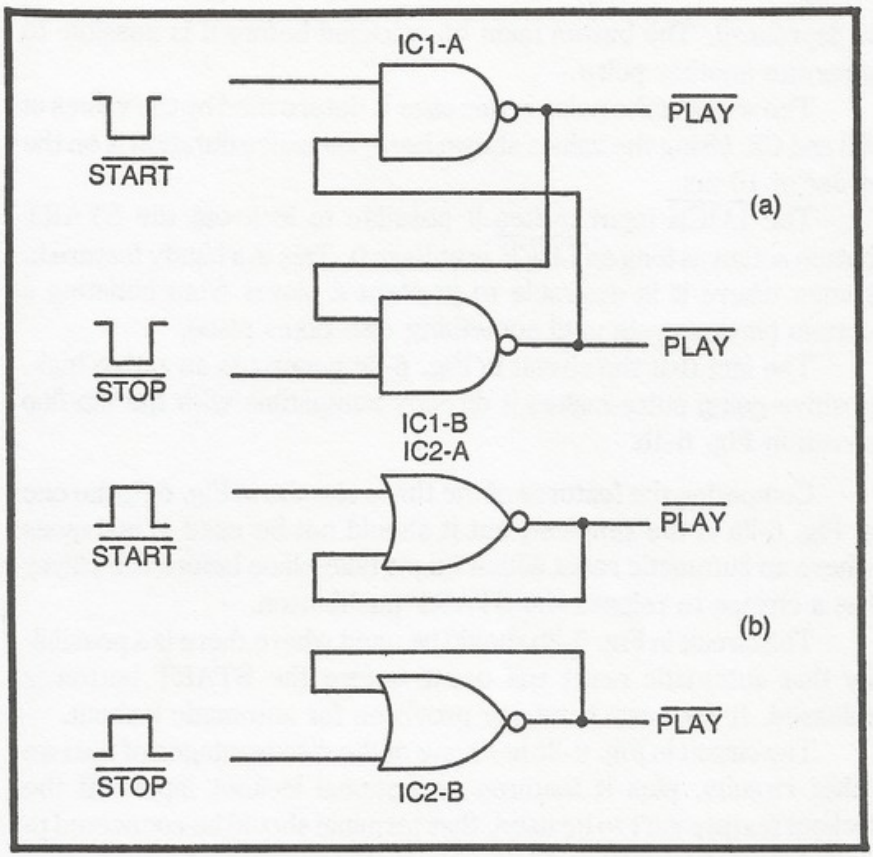


Fig. 6-1. Start/reset circuits. (a) A $\overline{R}$-$\overline{S}$ flip-flop triggered by negative-going pulses. (b) is R-S flip-flop triggered by positive-going pulses.

will appear at the output of this circuit, but it is effectively "filtered" by the flip-flop stage that follows it.

Since the signal from the circuit in Fig. 6-2a is an active-low, $\overline{\text{START}}$ level, it can be directly connected to the $\overline{\text{START}}$ input of the flip-flop in Fig. 6-1a.

The start circuit in Fig. 6-2b is a bit more complicated. In this case, however, depressing the START button generates a START pulse having a duration roughly equal to the time constant of R2 and C1. The button must be released before it is possible to generate another $\overline{\text{START}}$ pulse.

This circuit will show some contact-bouncing effects, but again, the flip-flop following that stage will eliminate these undesirable effects. The active-low nature of the pulse from this start circuit makes it directly compatible with the flip-flop in Fig. 6-1a.

The start circuit in Fig. 6-2c uses a 555-type timer to generate the START pulse. If the $\overline{\text{LOCK}}$ input is fixed at logic 1, this circuit generates a clean, positive-going pulse each time the START button

is depressed. The button must be released before it is possible to generate another pulse.

The width of the pulse in this case is determined by the values of R3 and C2. Using the values shown here, the pulse duration is on the order of 10 ms.

The $\overline{\text{LOCK}}$ input makes it possible to lock out the START button action as long as $\overline{\text{LOCK}}$ is at logic 0. This is a handy feature in games where it is desirable to prevent a player from initiating a certain playing cycle until something else takes place.

The fact that the circuit in Fig. 6-2c generates an active-high, positive-going pulse makes it directly compatible with the flip-flop circuit in Fig. 6-1b.

Comparing the features of the three circuits in Fig. 6-2, the one in Fig. 6-2a is the simplest, but it should not be used in instances where an automatic reset action might take place before the player has a chance to release the START pushbutton.

The circuit in Fig. 6-2b should be used where there is a possibility that automatic reset will occur before the START button is released. It does not have any provision for automatic lockout.

The circuit in Fig. 6-2c has none of the disadvantages of the two other circuits, plus it features an optional lockout input. (If the lockout feature isn't to be used, that terminal should be connected to +5V.)

When selecting the circuit most appropriate for your game design, consider the complexity, polarity of the outputs, and requirements of the game.

These three circuits, incidentally, can be used in the same ways for manually resetting game operations. Simply switch the $\overline{\text{START}}$ and START labels to $\overline{\text{STOP}}$ and STOP respectively.

Automatic Stop Circuits

Games, or cycles within games, can be stopped or reset automatically by means of the circuits in Fig. 6-3. These are both pulse-generating circuits: The active-high STOP pulse from the circuit in Fig. 6-3b is directly compatible with the flip-flop circuit in Fig. 6-1b, while the active-low output from the circuit in Fig. 6-3a is compatible with that in Fig. 6-1a.

In both cases, the stopping action is initiated whenever a given set of *stop* conditions are met within the system. IC1-A in Fig. 6-3a, for example, normally shows a logic-1 output. Whenever all the inputs to this NAND gate find their way to a logic-1 state (presumably at the time all the conditions for automatic resetting are met), the

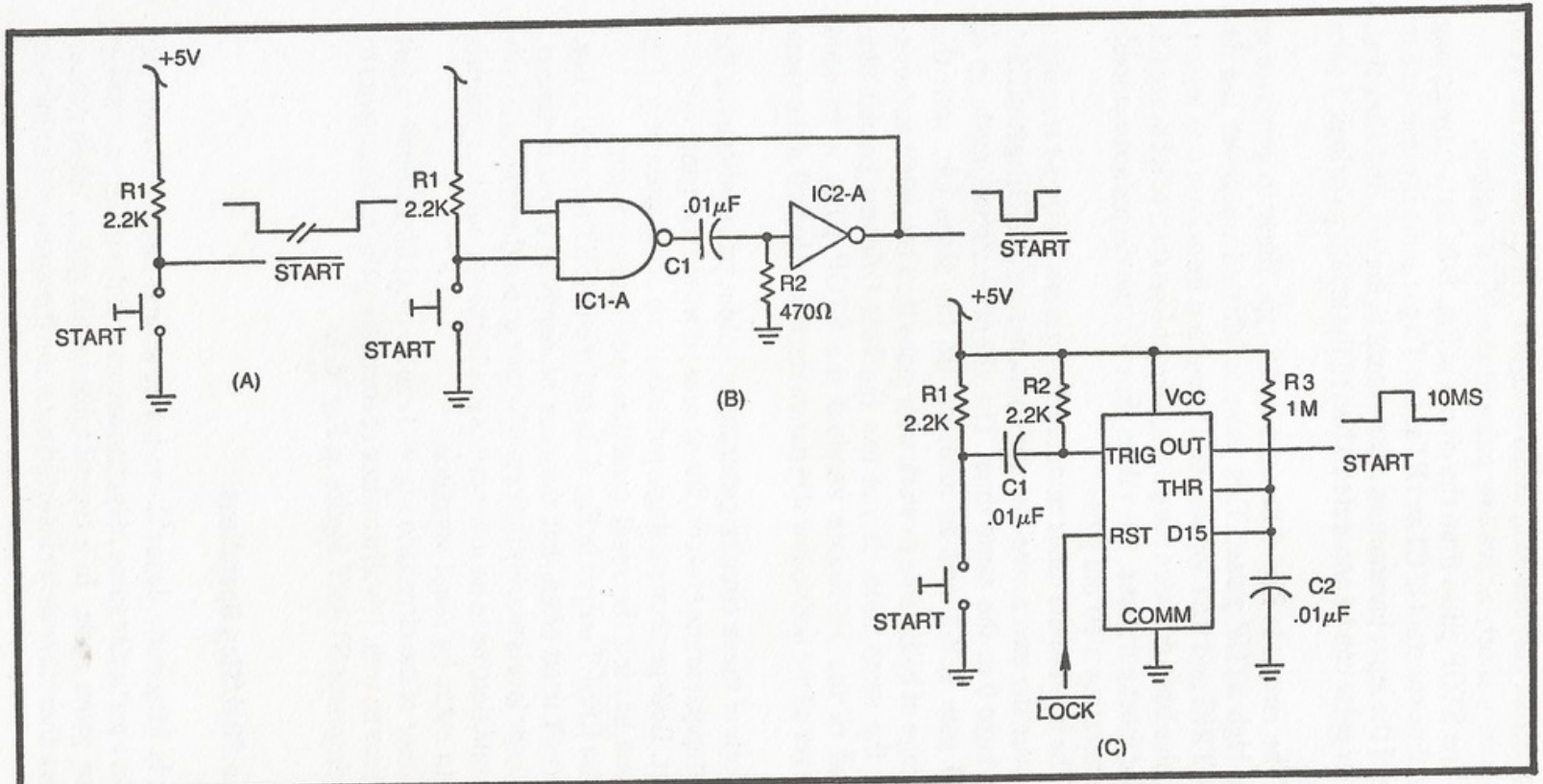


Fig. 6-2. Manual start switch circuits. (a) A simple switch arrangement. (b) A start pulse generator. (c) A positive pulse start generator with the lockout feature.

output of that gate suddenly drops to logic 0. This transition from 1 to 0 initiates a short, active-low pulse at the $\overline{\text{STOP}}$ output.

The STOP pulse from the circuit in Fig. 6-3a lasts only as long as the time constant of C1 and R1 allow. To get another pulse, one or more of the input parameters must return to logic 0. And after that, another pulse can be generated when all the inputs go to logic 1 once again.

The circuit in Fig. 6-3b uses a 555-type timer to generate an active-high STOP pulse. This pulse is initiated whenever the device's TRIG and RST inputs see a transition from logic 0 to logic 1. The pulse has a duration roughly equal to the product of R1 times C1. And for most purposes, the values shown in the diagram are suitable for generating a 10 ms pulse.

The input to the timer circuit makes the critical 0-to-1 transition whenever the two active-low stop parameters at NOR gate IC2-A are at logic 0 at the same time. The alternate circuit, made up of NAND gate IC2-A and an inverter, do the same job, with the advantage of being able to work with more than two input parameters at the same time. If you use the NAND/invert combination (instead of the NOR-gate version) the STOP pulse is initiated whenever all the active-low *stop* parameters go to logic 0 at the same time.

Both of these circuits generate an output pulse whenever the input stop parameters—conditions sensed within the game cycle—are met. Both generate a single pulse that can occur again only after the game has left the reset condition and enters it again.

The $\overline{\text{RESET}}$ output in Fig. 6-3a and the RESET terminal in Fig. 6-3b aren't used often, but they can be handy in a few instances. These logic levels merely indicate whether or not the system is in its reset condition, or to be more precise, whether or not the system is ready to enter its reset condition.

Either of the circuits in Fig. 6-3 can be used for automatic start operations as well. It's all a matter of connecting the pulse outputs to the appropriate START inputs in Fig. 6-1.

Delayed Start/Stop Operations

It is frequently desirable to insert a time delay between the moment a particular game response occurs and the initiation of a new game or game cycle. In a typical table-tennis game, for instance, there is a short time delay inserted between the time a player misses

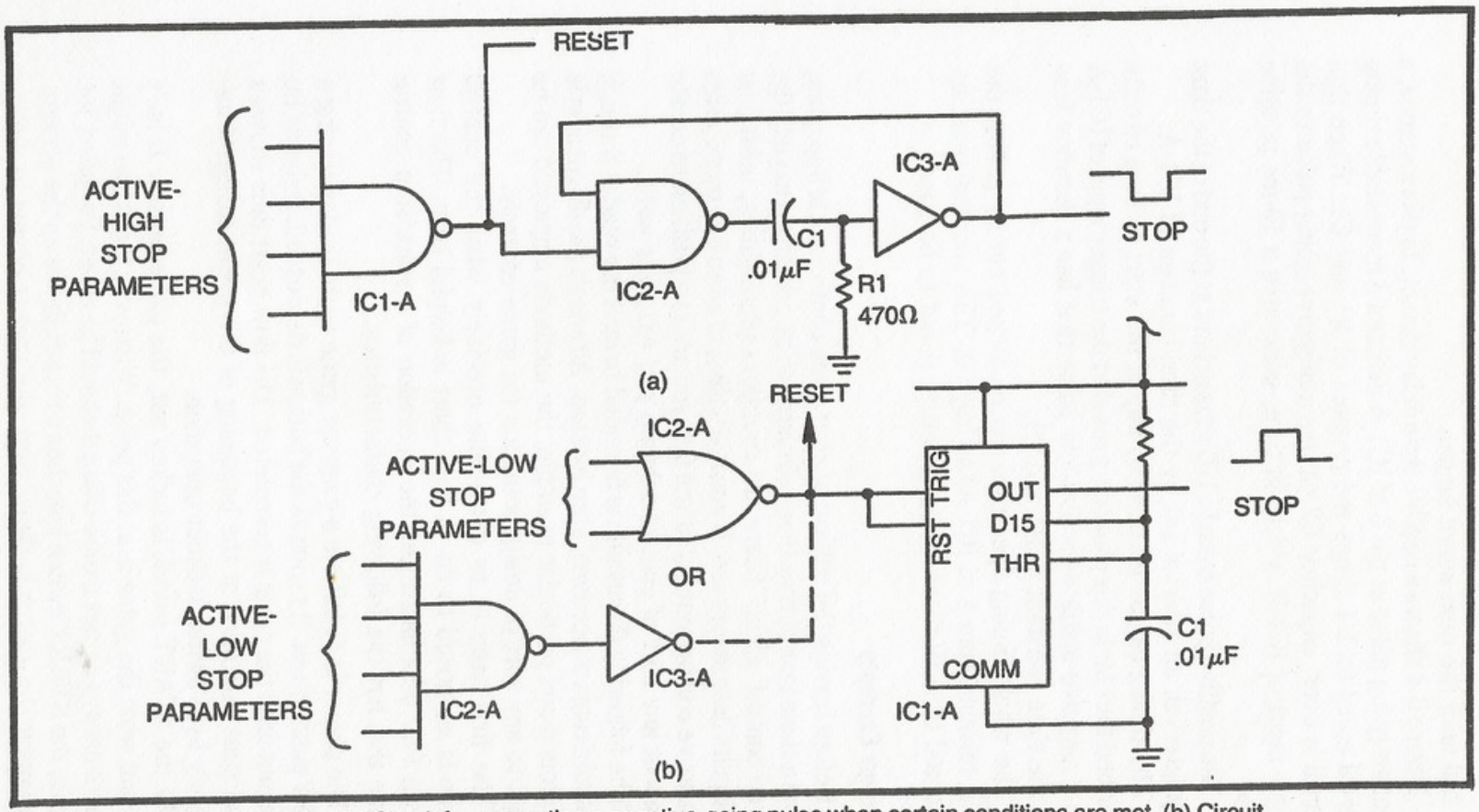


Fig. 6-3. Automatic stop circuits. (a) Circuit for generating a negative-going pulse when certain conditions are met. (b) Circuit for generating a positive-going pulse when the stop parameters are met.

the ball until the next serve begins.

Figure 6-4 shows a relaible time delay circuit. In this instance, a negative-going pulse at pin 6 of IC1-A initiates a monostable timing interval equal to 1.1 times the product of R1 and C1. When that interval is over, capacitor C2 passes a negative-going pulse to the trigger input of IC1-B which, in turn, generates a 10-ms positive pulse.

The positive pulse from IC1-B thus occurs at the end of the time delay interval, an interval set by the time constant of IC1-A.

The timing can be initiated by any of the start or stop circuits described thus far in this chapter, provided the trigger input of IC1-A sees a negative-going or active-low pulse that has a duration less than the time constant of R1 and C1.

The whole timing operation can be locked out by pulling the $\overline{\text{LOCK}}$ terminal (pin 4 of IC1-A) to logic 0. This terminal must be connected to +5V if the lockout feature is not to be used.

A Design Example

Before leaving the subject of start/stop controls, it is important to take a close look at how these circuits can be combined to yield the desired control effect. There are countless possibilities, using just the circuits described here. If you are willing to add a few more gates and passive components, you will find you are in a position to create just about any sort of control scheme you will ever want.

The following discussion is intended to lead the reader through a control design procedure, step by step. Although the discussion is built around one particular example, the methods employed can be applied to any sort of design situation for game controls.

The first step is to determine precisely what the control scheme is supposed to do. A flow chart is helpful here. The flow chart in Fig. 6-5a illustrates the operation of a start/stop control scheme that has the following characteristics.

The player is to initiate a game or game cycle by depressing a START pushbutton. As soon as the button is depressed, it should be locked out until the cycle is completed. The flow chart thus shows a manual start operation at the beginning of the cycle, followed immediately by a switch lockout operation.

As the START switch is locked out, the game begins. It isn't important what the game is at this point. Whatever the game might do, there always comes a time when it should be reset. Suppose, for example, the START switch launches a rocket figure on the screen. The moment the switch is depressed, the rocket begins to move,

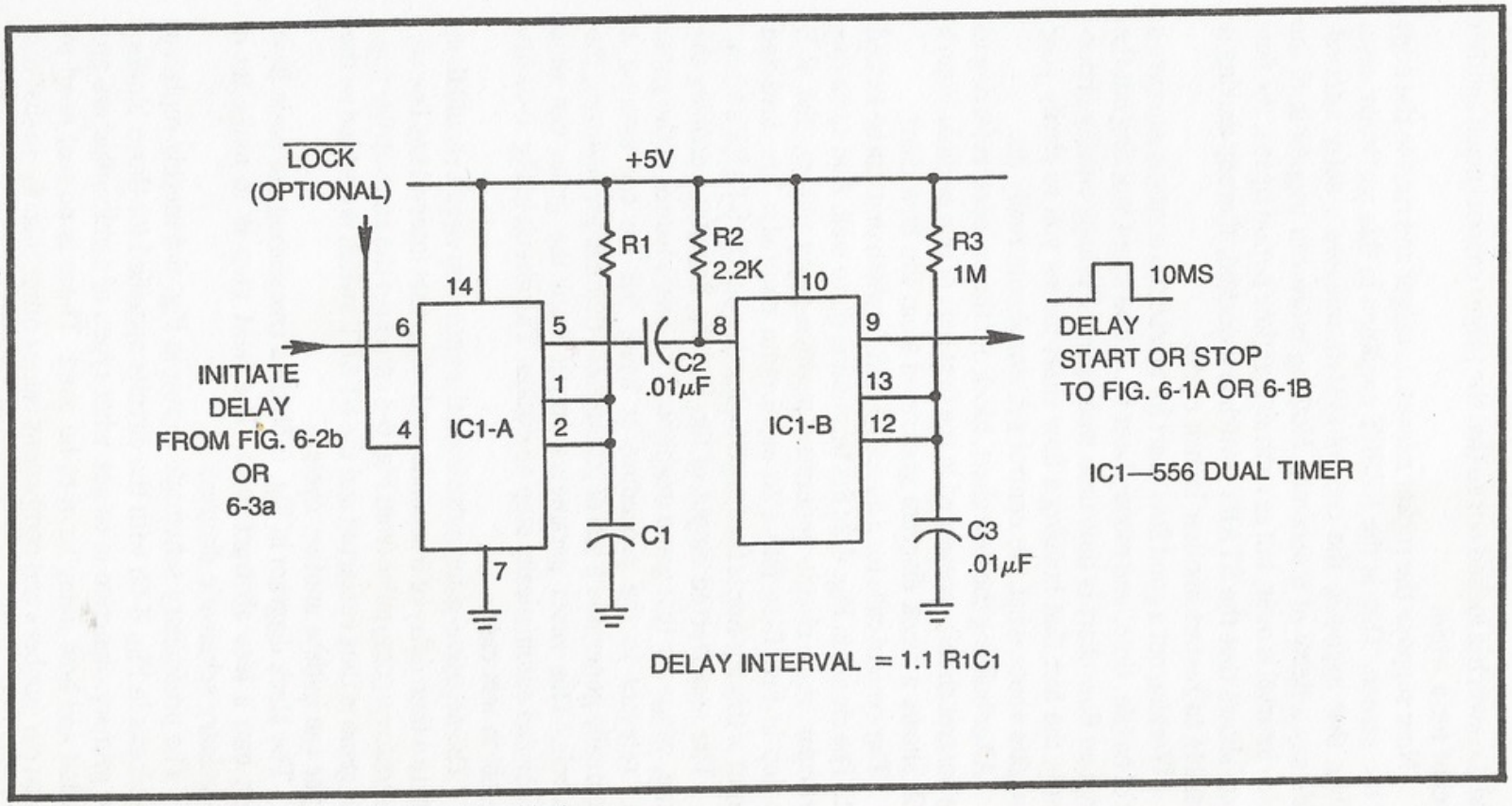


Fig. 6-4. A delayed start/stop circuit with optional lockout.

and the switch is locked out so that the player cannot launch another rocket for a while.

Now suppose the rocket misses its target and runs to the edge of the screen. That is the RESET condition in this particular case. When that happens, the control system initiates a delay interval, let's say a delay of 5 seconds. Nothing more can happen until the delay interval is over, but as soon as that time period lapses, the flow chart shows that the START switch is unlocked, thereby making it possible to restart another launch cycle.

Working out a good flow chart for start/stop controls can save a lot of hassle, time, and money later on. Time spent working out the simplest flow chart is thus time well spent. Perhaps equally important is the fact that building a flow chart forces you to clarify your thoughts about what the control system should really do.

After building the flow chart, block out the process, referring to the control circuits presented in the first part of this section. Figure 6-5b shows a block diagram generated from the flow chart.

The system calls for a manual START switch that can be locked out. The circuit in Fig. 6-2c fits this notion quite well. And since this particular start circuit generates positive-going pulses, the R-S flip-flop in Fig. 6-1b falls into place quite naturally. The inverted output of this flip-flop is then used for locking out the START switch.

The noninverted output of the R-S flip-flop stage enables the game, whatever that game might be. The exact nature of the game isn't relevant to the procedure at hand, but we can assume it eventually generates a set of parameters that call for resetting the controls. This reset parameter output from the game can thus initiate and automatically stop operation. The circuit in Fig. 6-3a fits the bill in this case.

The negative-going pulse from the auto stop circuit should then initiate a time delay of 5 seconds, and when that interval has lapsed, the positive-going pulse from Fig. 6-4 can reset the start-up flip-flop. The game is thus restarted and the START switch is enabled so the player can initiate another cycle.

The block diagram in Fig. 6-5b thus transforms the basic flow chart into a less abstract form. The next step is to make up a preliminary schematic diagram.

The preliminary schematic shown in Fig. 6-6 merely replaces the blocks in Fig. 6-5b with the circuits specified for those blocks. The primary objective is to see what types of semiconductors are needed and how many have to be used. There is no real need to assign pin numbers and component values other than those that are

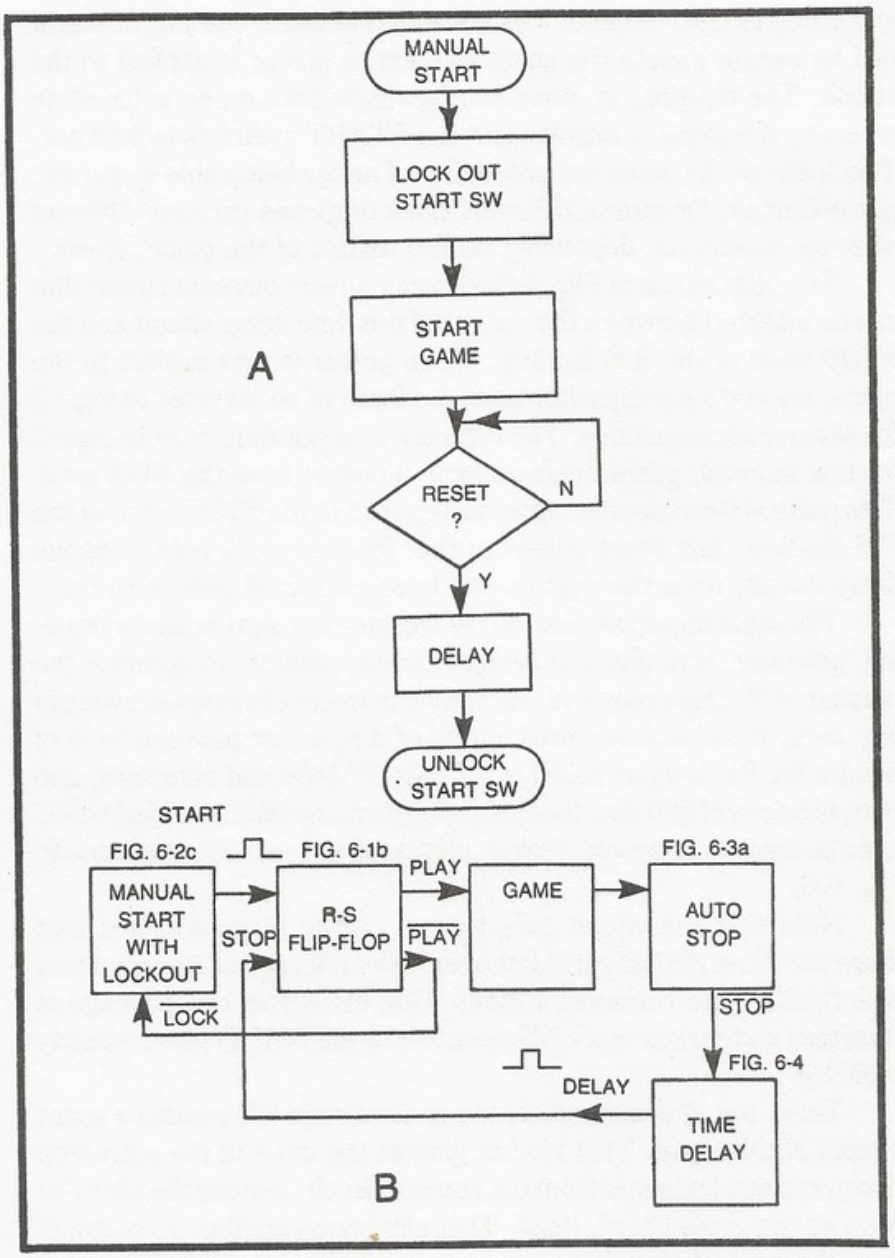


Fig. 6-5. Basic flowchart for control circuit designs. (a) A typical scheme using manual start and automatic reset with time delay. (b) Corresponding block diagram.

peculiar to the system being designed. In this case, the only unique control parameter is the length of the reset time delay. We have specified 5 seconds, so the preliminary schematic shows the values of R and C that are appropriate for this delay interval, 470 kΩ and 10 μF.

Another problem became apparent while setting up the preliminary drawing. This particular scheme ought to have a power-on

initializing circuit. Without this power-on feature, the player might find he cannot initiate the game as soon as power is applied to the circuit. The flip-flop, in other words, might take on an initial state whereby the game is enabled and the START switch is locked out. This leads to the awkward possibility of never being able to get the game started. Of course different kinds of games present different start-up conditions, depending on the nature of the game, itself.

The little circuit in Fig. 6-6b shows a power-on reset circuit that can be inserted between the output of the time delay circuit and the STOP input of the R-S flip-flop. When power is first applied to the game, the 0.01-μF capacitor pulls the input of an inverter to logic 0 for several microseconds. The inverter's output thus rises to logic 1 for that interval, guaranteeing a logic-0 output from the NOR gate. This pulse is then inverted again and applied to the STOP input of the R-S flip-flop. Any reset pulses to that flip-flop come only from the delay circuit, once the system has been initialized after turn-on.

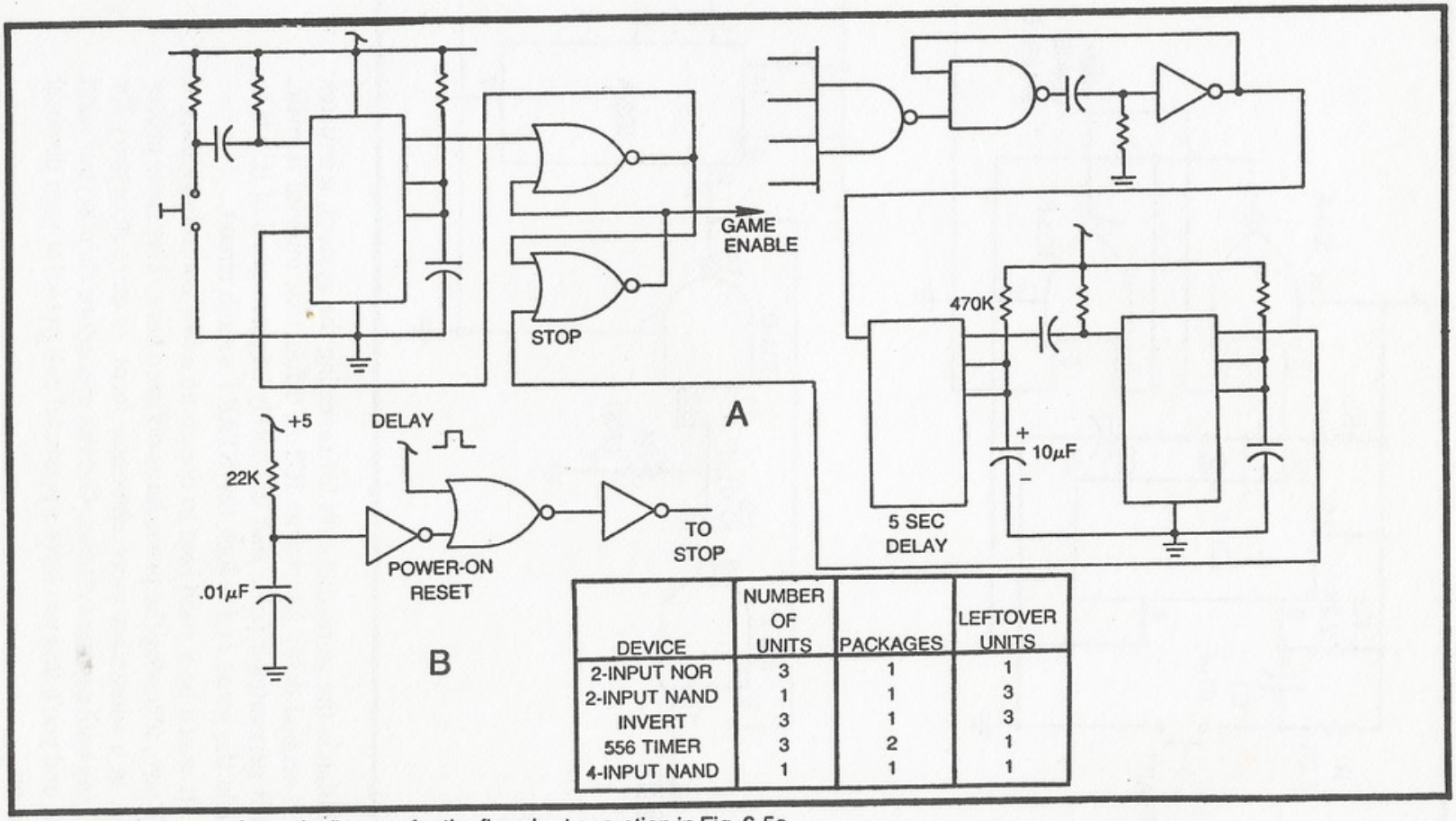
The most important reason for working out a preliminary drawing, however, is to give the designer an opportunity to optimize the number of IC chips required. As shown in the preliminary drawing in Fig. 6-6a, the circuit requires a total of 6 ICs: one package each of 2-input NORs, 2-input NANDs, 4-input NANDs and inverters, and two packages of 556 dual timers. Using so many different kinds of IC logic packages, however, leaves plenty of spares. See the table in Fig. 6-6.

Note that the circuit calls for only three inverters, and that there are three NAND gates left over. Why not replace the inverters with NAND-gate versions of them. That eliminates one package of inverters and makes more efficient use of the NAND gates already available.

Then, too, it is quite likely the game circuit will contain a spare 4-input NAND gate. That kind of gate at the input of the auto-stop circuit can be eliminated from the control circuit, leaving the job to be done on the game board, itself. That eliminates another IC package from the control circuit.

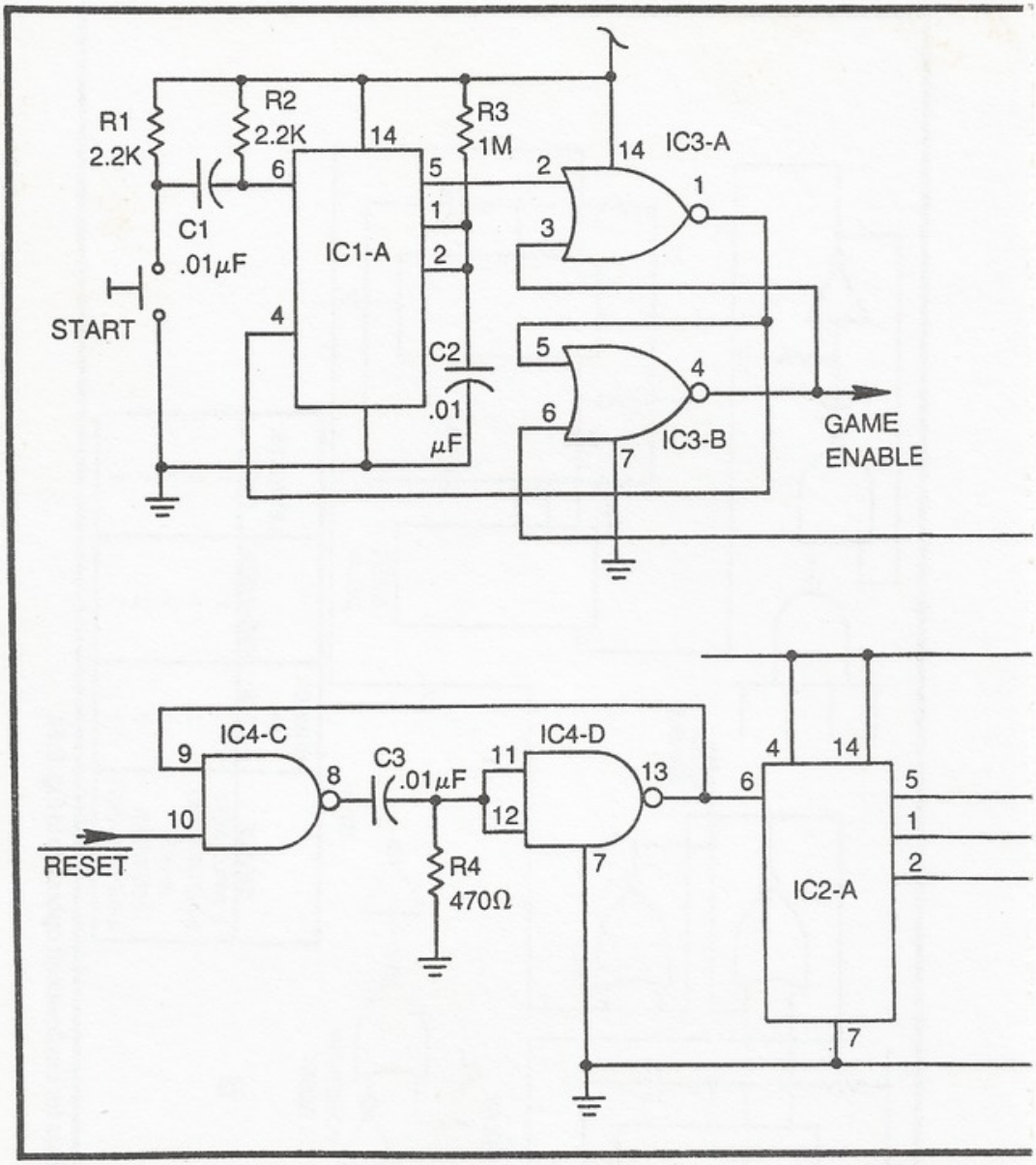
Now the circuit requires only four IC packages. The final step in the design procedure is to redraw the schematic, incorporating the modifications just described. The final circuit appears in Fig. 6-7.

After studying the flow diagram, block diagram, and preliminary schematic, you should have no trouble understanding the operation of the control circuit in Fig. 6-7. R8 and C6 make up the power-on initializing circuit, ensuring the START switch is enabled the moment power is first applied. Depressing the START switch then enables the game. And whenever the $\overline{\text{RESET}}$ input indicates a



DEVICE	NUMBER OF UNITS	PACKAGES	LEFTOVER UNITS
2-INPUT NOR	3	1	1
2-INPUT NAND	1	1	3
INVERT	3	1	3
556 TIMER	3	2	1
4-INPUT NAND	1	1	1

Fig. 6-6. Preliminary schematic diagram for the flowchart operation in Fig. 6-5a.



condition in the game that calls for resetting the system, it initiates the 5-second delay interval at IC2-A. After that interval is over, IC2-B generates a pulse that ultimately triggers pin 6 of IC3-B to disable the game and unlock the START switch circuit.

It would be a good idea to dream up a few control systems of your own, following the procedures outlined here. This same procedure, in a somewhat more elaborate form, is quite necessary for planning entire game systems. Get the procedure clear in your mind now, and you'll find you have a powerful design tool at your disposal later on.

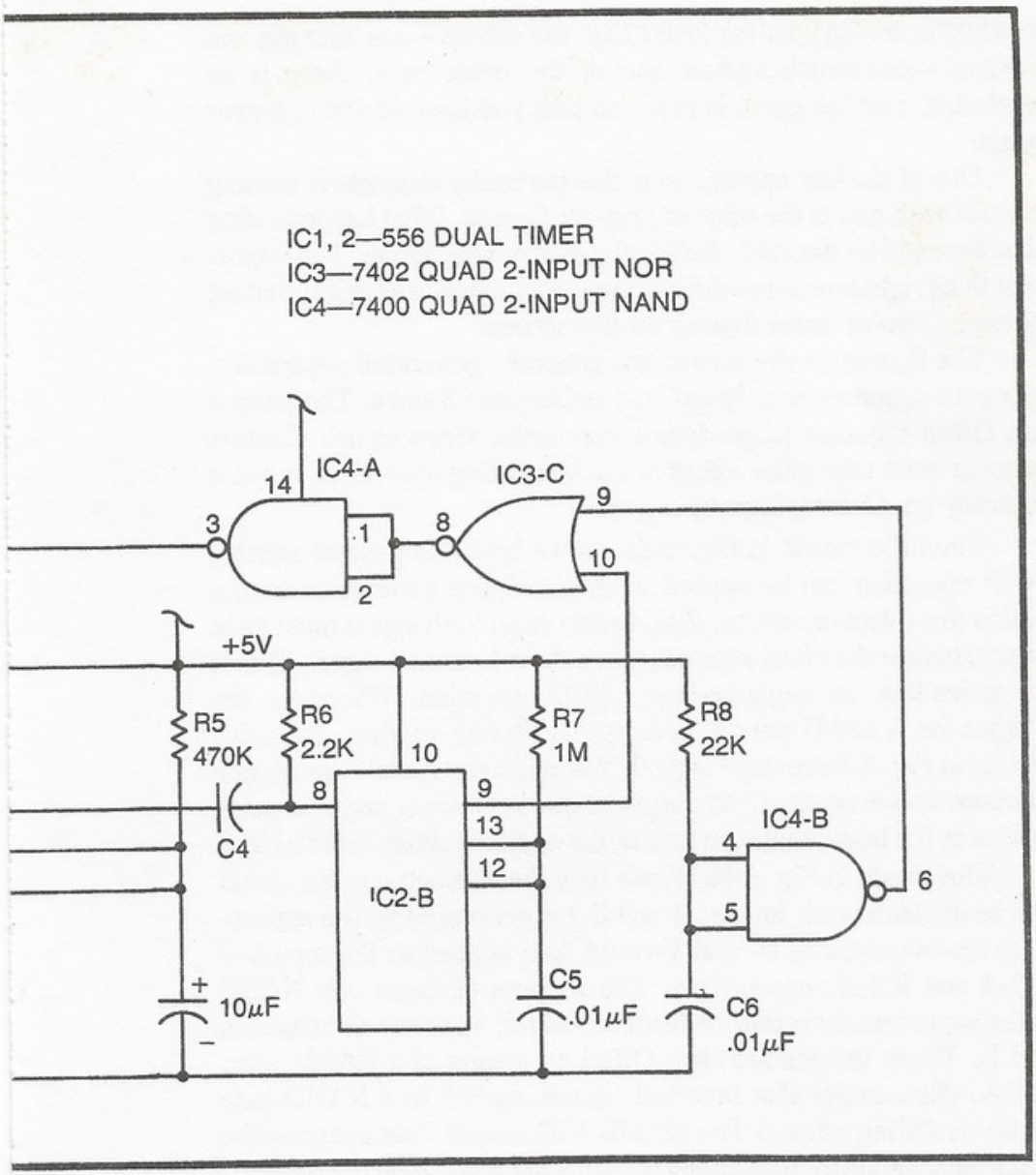


Fig. 6-7. Finalized schematic for the control operation described in connection with the flowchart in Fig. 6-5a.

FIGURE-CONTACT-SENSING CIRCUITS

Most TV games are designed so that they carry out an automatic operation whenever two or more figures come into contact with one another on the screen. Suppose you are working up an auto racing game. The cycle begins with your opponents' cars moving rapidly across the screen. And your job is to accelerate your car, moving into the traffic without touching any of the other cars. If you are setting up this game right, things will start getting a little hairy as

you begin moving your car faster than the others—one little slip and whamo!—you brush against one of the other cars, there is an explosion, and the game is reset so that you have to start all over again.

One of the key operations in this particular example is sensing contact with one of the other racing-car figures. What happens after that depends on the kind of control circuitry you devise. The important thing right now is providing a means for sensing a critical contact between two or more figures on the screen.

The figures on the screen are generally generated separately, using the schemes already outlined in Chapters 3 and 4. The images are ORed together to produce a composite video signal. Contact sensing must take place ahead of the last ORing operation, and it is basically an ANDing logic operation.

The little circuit in Fig. 6-8a shows how this contact-sensing AND operation can be applied when the figure generators output active-low (black-on-white) data. In this case, both inputs must be at logic 0 before the NOR gate outputs a logic-1 contact signal. This is an active-low, or negative-logic, AND operation. Whenever the images for A and B are not in contact with one another, the CON output in Fig. 6-8a rests at logic 0. When images A and B touch one another, however, the CON output begins generating positive-going pulses at the horizontal scan rate of the system (about 15,750 Hz).

The circuit in Fig. 6-8b shows how the contact-sensing circuit can be implemented. Images A and B are generated by the appropriate combinations of H- and V-count data applied to the inputs of IC1-A and IC1-B respectively. The outputs of these two NAND gates represent inverted, or black-on-white, versions of images A and B. These images are then ORed by means of a NAND gate, IC2-A. (Remember that inverted signals applied to a NAND gate yields an ORing effect.) The GAME VID output thus contains the video information for presenting both images A and B on the screen.

The inputs to IC3-A, however, are taken directly from the two image-generating NAND gates. IC3-A is the contact sensing circuit already described in connection with Fig. 6-8a. Whenever images A and B touch one another on the screen, their video data is lining up such that the CON output is showing positive active-high pulses. These pulses cannot occur at any other time.

In short, the circuit in Fig. 6-8b generates video information for two different images, combines them into a single composite game video signal and senses any contact between the two images.

The circuit in Fig. 6-8c performs the same function as that in Fig. 6-8a. In this latter example, however, the video data for the two

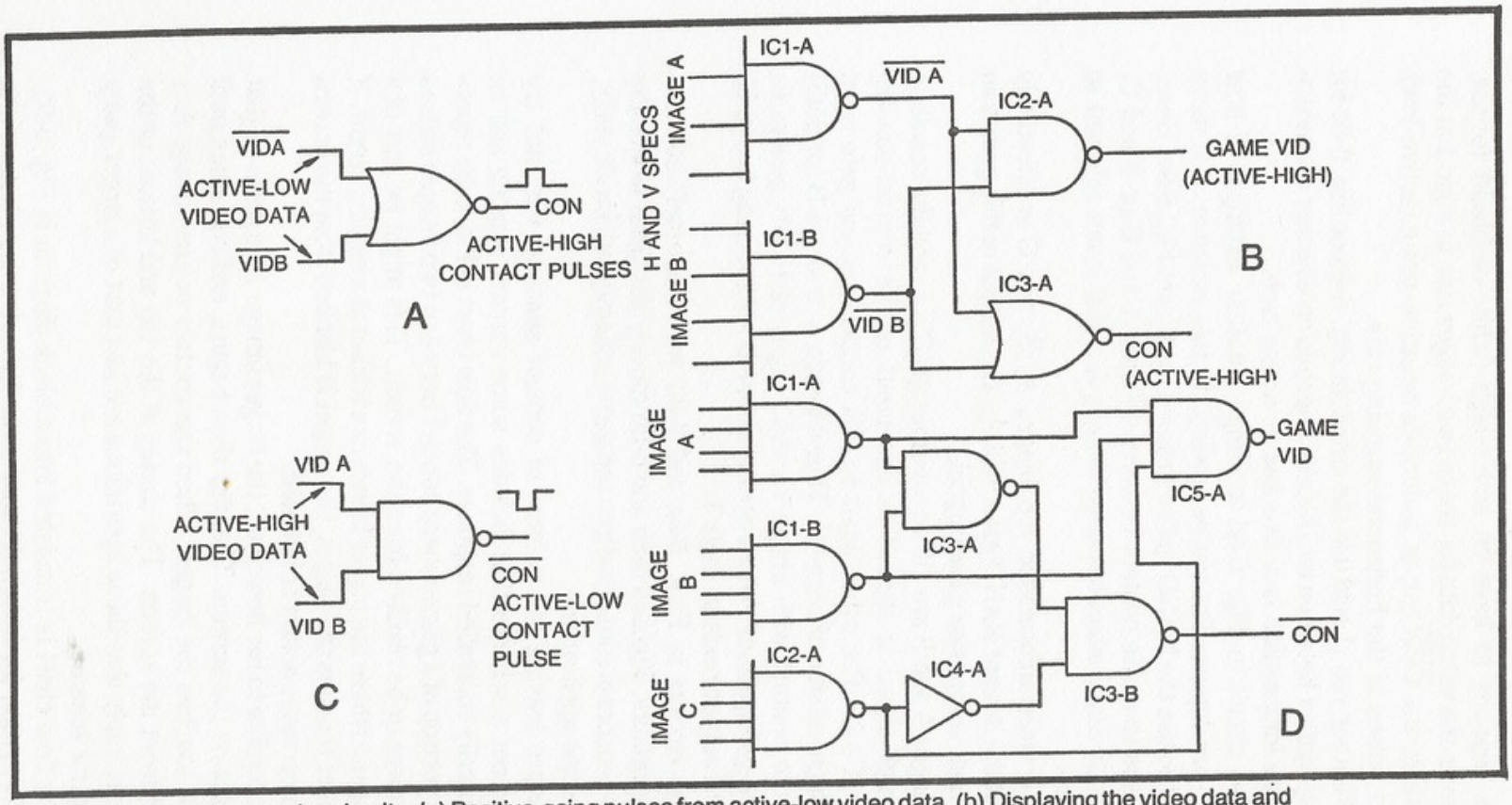


Fig. 6-8. Some contact sensing circuits. (a) Positive-going pulses from active-low video data. (b) Displaying the video data and sensing contact between two figures on the screen. (c) Negative-going contact pulses from active-high video data. (d) A circuit for displaying three figures and sensing contact between A and C or B and C.

images happens to have an active-high (white-on-black) format. Whenever the video data for these two images rise to logic 1 at the same time, the $\overline{\text{CON}}$ output generates negative-going (active-low) contact pulses at the horizontal-scanning rate.

Whether you should use the circuit in Fig. 6-8a or Fig. 6-8c for sensing contact between two figures depends on whether the available image information is active low or active high.

The circuit in Fig. 6-8d is simply another example of how contact-sensing circuits can be applied. In this instance, it is more efficient to use the NAND-gate sensing circuit from Fig. 6-8c. Here, the idea is to sense contact between images A and C or B and C. Apparently any contact between images A and B is not relevant to the game.

The video information for images A, B, and C is effectively ORed at the 3-input NAND gate, IC5-A. The output of this gate is an active-high composite game signal.

Images A and B are ORed together at IC3-A, and the resulting active-high signal is applied to one input of the contact-sensing circuit, IC3-B. The other input to this contact-sensing gate is an active-high version of image C. Thus if image A or image B (or both) come into contact with image C, the output of IC3-B generates negative-going pulses. When neither A nor B are touching image C, the $\overline{\text{CON}}$ output rests at logic 1.

The circuits in Figs. 6-8a and 6-8c are "universal" contact-sensing circuits. Figures 6-8b and 6-8d merely illustrate where the contact sensors are inserted into the game scheme and, incidentally, two specific applications.

Before leaving the subject of contact sensing, why not try working out a scheme that includes some contact sensing and an automatically controlled response. The flow chart in Fig. 6-9a represents a portion of a game where contact between two images causes one of them to be blanked from the screen. This might be part of a target game where the player launches a rocket at a moving target. If the rocket touches the target, the target is blanked from the screen until some reset action is initiated.

According to the flowchart, the target (image A) is reset so that it appears on the screen. The play then begins, and continues until image B touches the target. When the contact occurs, image A is blanked from the screen. The circles at the top and bottom of the flow chart imply that these operations are just part of a larger game and control scheme.

The flow chart is translated into a block diagram in Fig. 6-9b. The information for image A must pass through a gate before it is

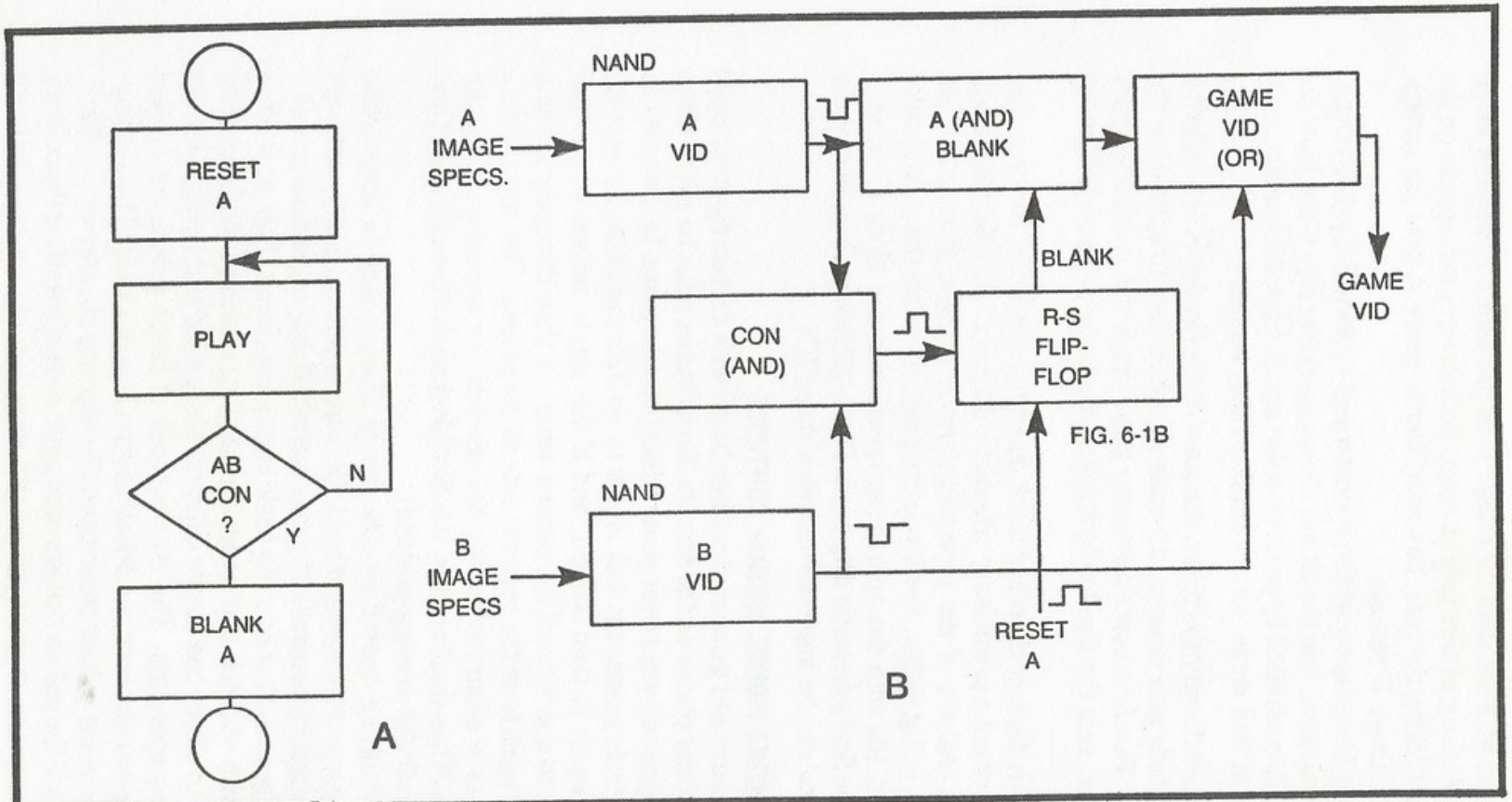


Fig. 6-9. A design example for contact sensing. (a) Flowchart for sensing contact between images A and B, then blanking image A. (b) Block diagram.

combined with the data for image B to produce a composite video signal. As long as this gate is open, both images can appear on the screen. Closing the gate, however, blanks image A from the screen, but lets image B remain.

The blanking operation is controlled by an R-S flip-flop circuit. In this instance, it is the circuit shown earlier in Fig. 6-1b. Now the flip-flop is controlled by positive-pulse inputs from the contact sensing circuit and some sort of outside reset circuit.

When the game is reset, the state of the flip-flop is set such that the A blank gate is open, allowing image A data to appear on the screen. The first contact-sensing pulse from the contact sensor, however, sets the flip-flop to a state where the A image is blanked off.

With the flow chart and block diagram completed, the next step is to work out a preliminary schematic diagram. This diagram, along with an analysis of the type and number of logic gates required, appears in Fig. 6-10a. See if you can properly relate the gate circuits in Fig. 6-10a with the operations specified in the block diagram.

The final schematic diagram in Fig. 6-10b shows how this whole operation can be implemented with three ICs.

INITIALIZING FIGURE MOTION CONTROLS

Games and game cycles often begin with certain figures placed at particular places on the screen. Such figures must be set to those initial positions whenever some critical event occurs. In the case of a table-tennis game, the ball is set to an initial position for serving purposes. It is then served, and if the player misses it, the ball travels to one side of the screen where it then disappears, being served again from the opposite side of the screen. The critical event in this case occurs when the ball reaches the opposite side of the screen. When that happens, it is first blanked and then initialized (set to a particular serving position).

Initializing operations are nearly always used in conjunction with figures that move around the screen automatically, although some manually controlled figures use initializing operations as well.

Figure 6-11 shows the basic initializing circuit. It is most often used with vertical- and horizontal-slipping counters, and the basic idea is to select one of two different sources of reset pulses for the counting operation. The two sources of reset pulses are labeled *slipping-counter reset* and *initializing reset* in this case. The slipping-counter reset pulses come from the slipping-counter circuit itself. These are the pulses that are normally used for resetting the counter to achieve the desired speed and direction of motion. They are

directed to the loading inputs of the slipping counter whenever the *initializing control* input is at logic 1.

When the initializing control input is at logic 1, IC1-A is effectively opened so that inverted versions of the slipping-counter reset pulses emerge from its output. While the initializing control is at logic 1, however, inverter IC2-A switches that logic level to 0, as far as the pin-9 input of IC1-C is concerned. IC1-C is thus effectively gated off, thereby preventing any initializing reset pulses from reaching IC1-B. Setting the initializing control input to logic 1 blocks any incoming initializing reset pulses, but allows negative-going slipping-counter reset pulses to appear at the output of IC1-B. These pulses are applied to the loading line of the slipping-counter circuit, letting it operate in its normal, motion-generating mode.

Setting the initializing control input to logic 0 completely changes the situation. In this case, IC1-A is gated off by the logic-0 level appearing at its pin-2 input, but IC1-C is gated on by virtue of the logic-1 level now appearing at its pin-9 input. Inverted versions of the initializing reset pulses thus appear at the output of IC1-B.

In short, the circuit in Fig. 6-11 is simply a digital selector circuit. It selects one of two sources of reset pulses, depending on the logic level present at the initializing control input.

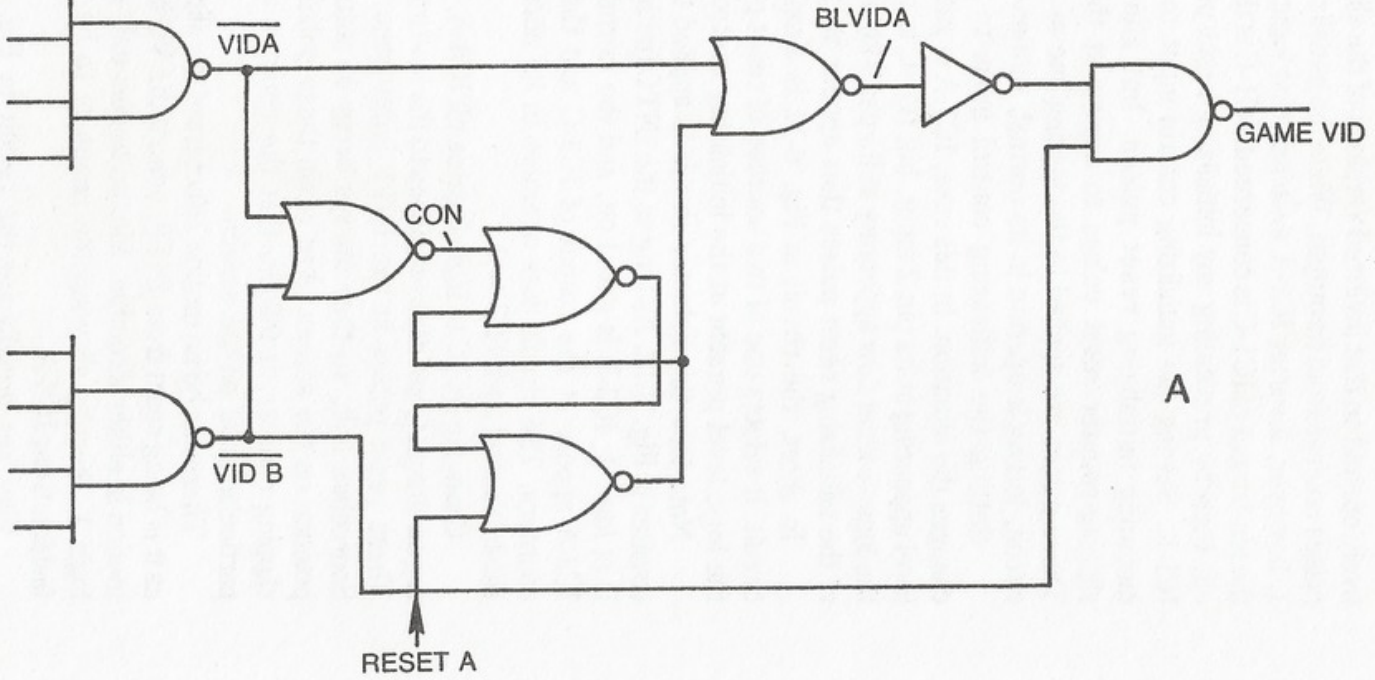
Note how this selector circuit is applied to a vertical-slipping counter in Fig. 6-12. As long as the INTC (initializing-control) input is at logic 1, IC3-B is gated on, and the normal reset pulses from IC3-A appear at the output of IC3-C and the load inputs of the counters. The circuit thus operates in its normal slipping-counter mode as long as INTC = 1.

Changing INTC to logic 0 gates off IC3-A, however, and delivers an alternate set of reset pulses to the load inputs of the counters. These reset pulses at the INTP input generally come from the Sourcebox unit, so they always occur at some particular vertical position on the screen. And when these pulses are applied to the slipping counter, it follows that the counter will be reset at that particular point on the screen.

There can be no motion effect from the slipping counter as long as it is being reset from INTP, even if the VC inputs are set for fast motion in either direction. Motion begins only when INTC is set to logic 1, thereby allowing the counters to be reset in the normal fashion from IC3-A.

Why not breadboard the circuit in Fig. 6-12 and try it for yourself. For the purposes of this experiment, connect VRST from the Sourcebox through an inverter to the INTP connection. Whenever the figure is to be initialized (INTC set to logic 0), the

	NUMBER OF UNITS	PACKAGES	SPARE UNITS
4-INPUT NAND	2	1	0
2-INPUT NAND	1	1	3
2-INPUT NOR	4	1	0
INVERT	1	1	5



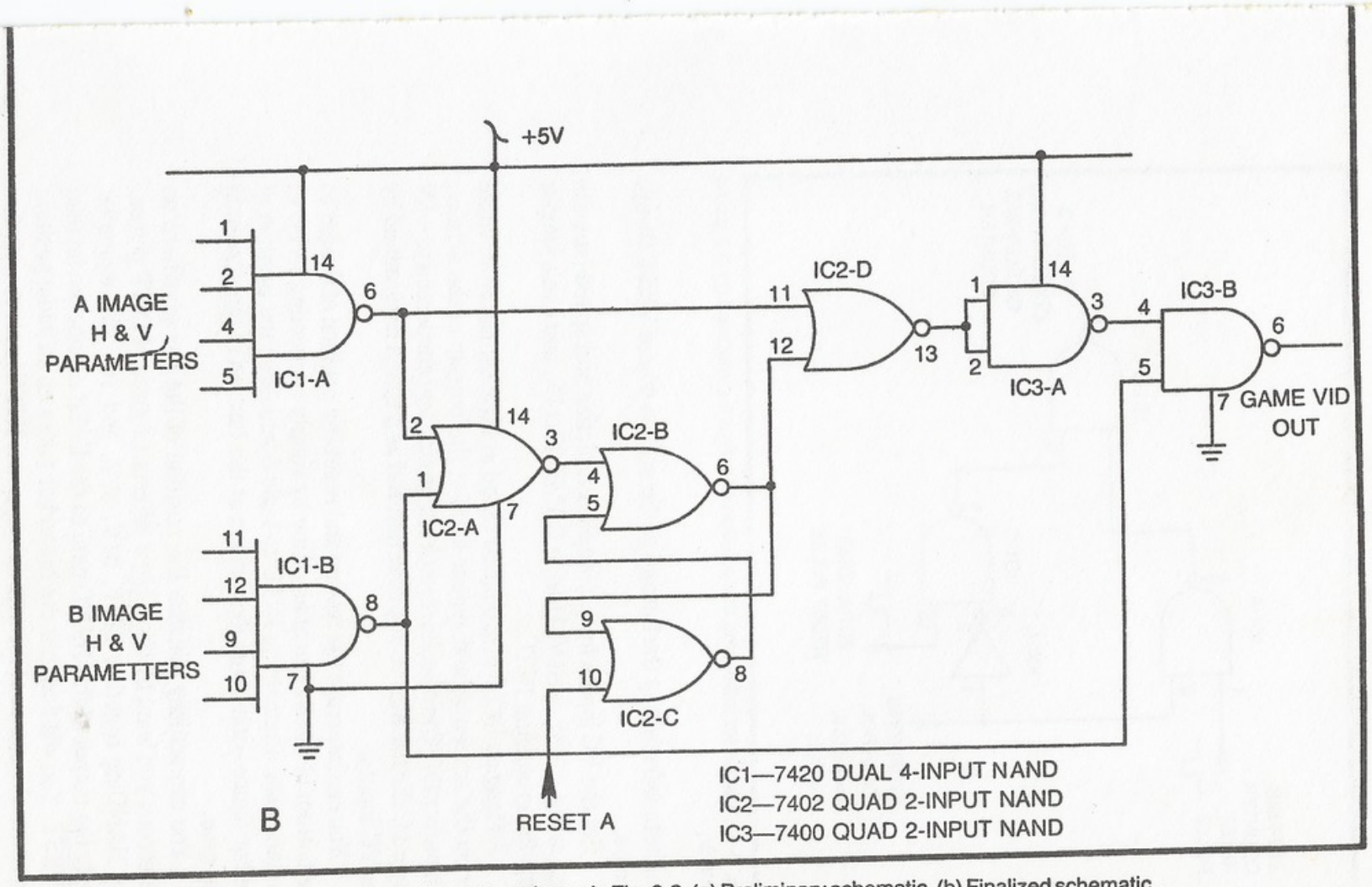


Fig. 6-10. Schematic diagrams for the contact sensing scheme in Fig. 6-9. (a) Preliminary schematic. (b) Finalized schematic.

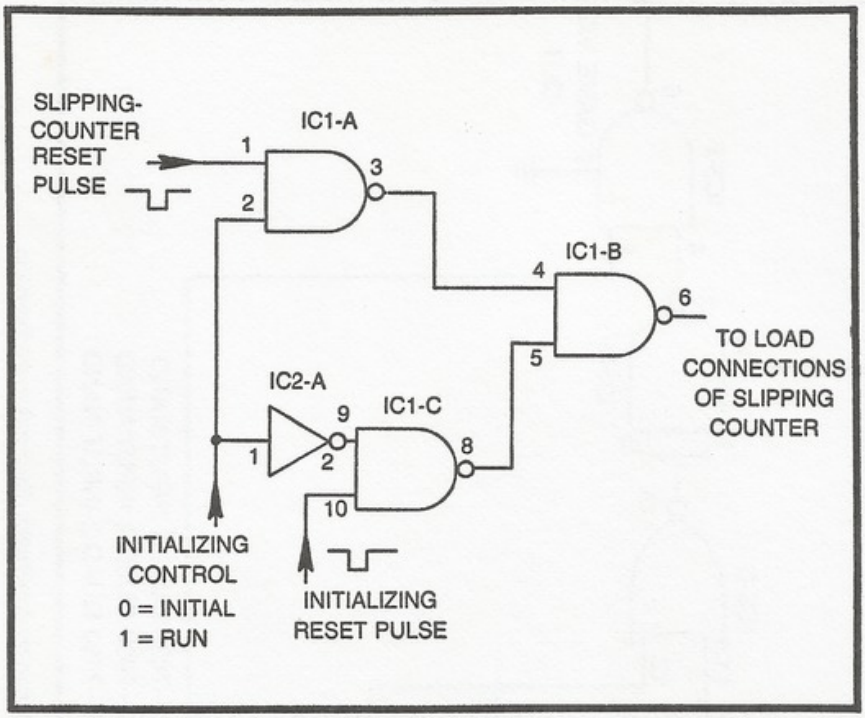


Fig. 6-11. Circuit for initializing the position of a figure generated by a slipping counter.

counters will reset at the bottom of the screen where VRST always occurs.

Set the VC inputs for any desired direction and speed (see the data in Fig. 5-13), feed VM128 to GAME VID IN, and use a jumper wire for grounding INTC.

As long as INTC is grounded (fixed at logic 0), the wide white bar on the screen should appear fixed at the bottom of the screen. Whenever INTC is disconnected from ground and connected to +5V (logic 1), the bar moves in a direction and at a speed determined by the VC inputs.

You can interrupt the motion and reset the position of the bar to the bottom of the screen at any time by simply connecting INTC to ground again. Connecting INTC to logic 0 initializes the position of the bar figure—initializes its position at the bottom of the screen in this case.

You can actually initialize the position of the bar anywhere on the screen you want by choosing a different source of INTP pulses. Try NANDing together $\overline{128V}$, 64V, 32V, and 16V, for example. Apply the output of this NAND gate to the INTP connection instead of $\overline{VRST}$. You will find that the broad bar takes on an initial position just above the center of the screen whenever INTC is set to logic 0.

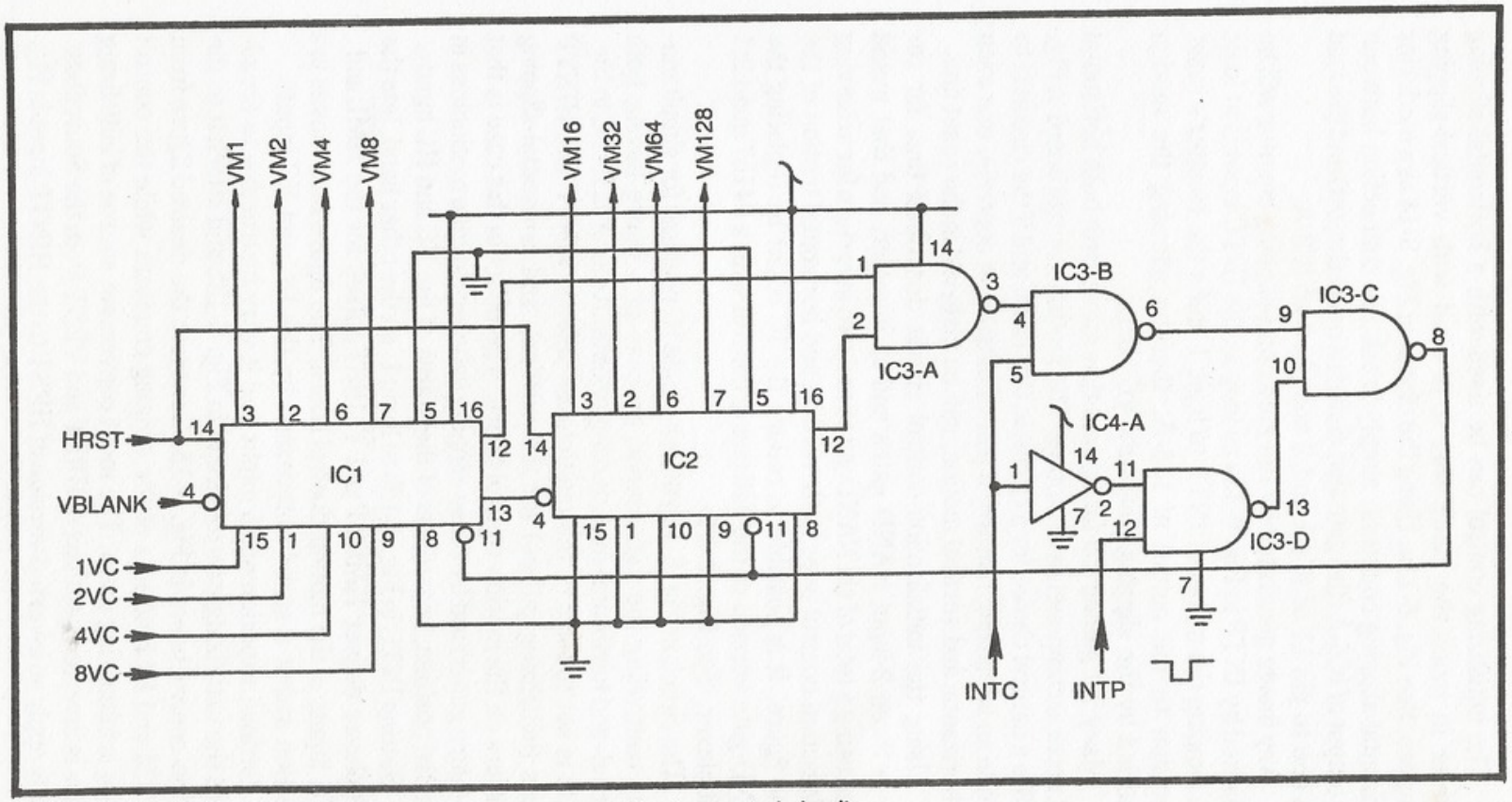


Fig. 6-12. Interfacing a vertical slipping counter to its initialization control circuit.

The initializing control can be used with a horizontal-slipping counter in much the same way it is used with vertical-slipping counters. See Fig. 6-13a. Using the circuit in Fig. 5-14 as a model for horizontal-slipping counters, simply break the connection between the output of IC5-A (the gate that signals a normal reset) and the load bus line to pin 11 of ICs 1 and 2 and pin 3 of IC3-A.

Any image generated by the horizontal-slipping counter will be initialized by INTP in Fig. 6-13a as long as the INTC input is at logic 0. Changing the status of INTP to logic 1 shifts the counter's reset operation to the output of IC5-A, thereby allowing the motion specified by the slipping counter's VC inputs.

Many TV games call for figures that move with both horizontal and vertical components of motion. The motion-control board in Fig. 5-15 is a natural choice for a circuit in this case; and if the circuit is to include an initializing feature, a pair of initializing controls, one each for horizontal and vertical motion, can be inserted in the reset line.

Using the initialization-control circuit described thus far requires three 2-input NAND gates and an inverter, and that would mean using a total of six NAND gates and two inverters for achieving initialization control over both vertical and horizontal motion of the same figure. It is possible to reduce the IC count by replacing the NAND-gate version of the initializing control with a 74157 quad 2:1 multiplexer. See Fig. 6-13b.

The circuit in Fig. 6-13b uses a single IC package for simultaneously controlling the initialization process of a figure having both vertical- and horizontal-motion components. As long as INTC in Fig. 6-13b is set to logic 0, the multiplexer selects VINTP and HINTP inputs (initializing pulses) for the vertical- and horizontal-slipping counters on the motion-control board. The effect in this case is that the figure generated by these slipping counters stands motionless in its initial position, regardless of the status of the VC and HC inputs.

Setting INTC in Fig. 6-13b to logic 1, on the other hand, lets the multiplexer deliver $\overline{\text{HMRST}}$ and $\overline{\text{VMRST}}$ pulses for the $\overline{\text{HML}}$ and $\overline{\text{VML}}$ inputs of the motion-control board. The figure thus moves in a direction and at a speed determined by the HC and VC inputs.

Serious experimenters might find it very instructive to breadboard the initializing-control circuit in Fig. 6-13b and attach it to the motion-control board in Fig. 5-15. Generate the desired figure from the VM and HM outputs of the slipping counters while the control circuit is initializing them. The most convenient source of initializing pulses is inverted versions of HRST and VRST from the Sourcebox. Simply apply an inverted version of HRST to the HINTP input in Fig.

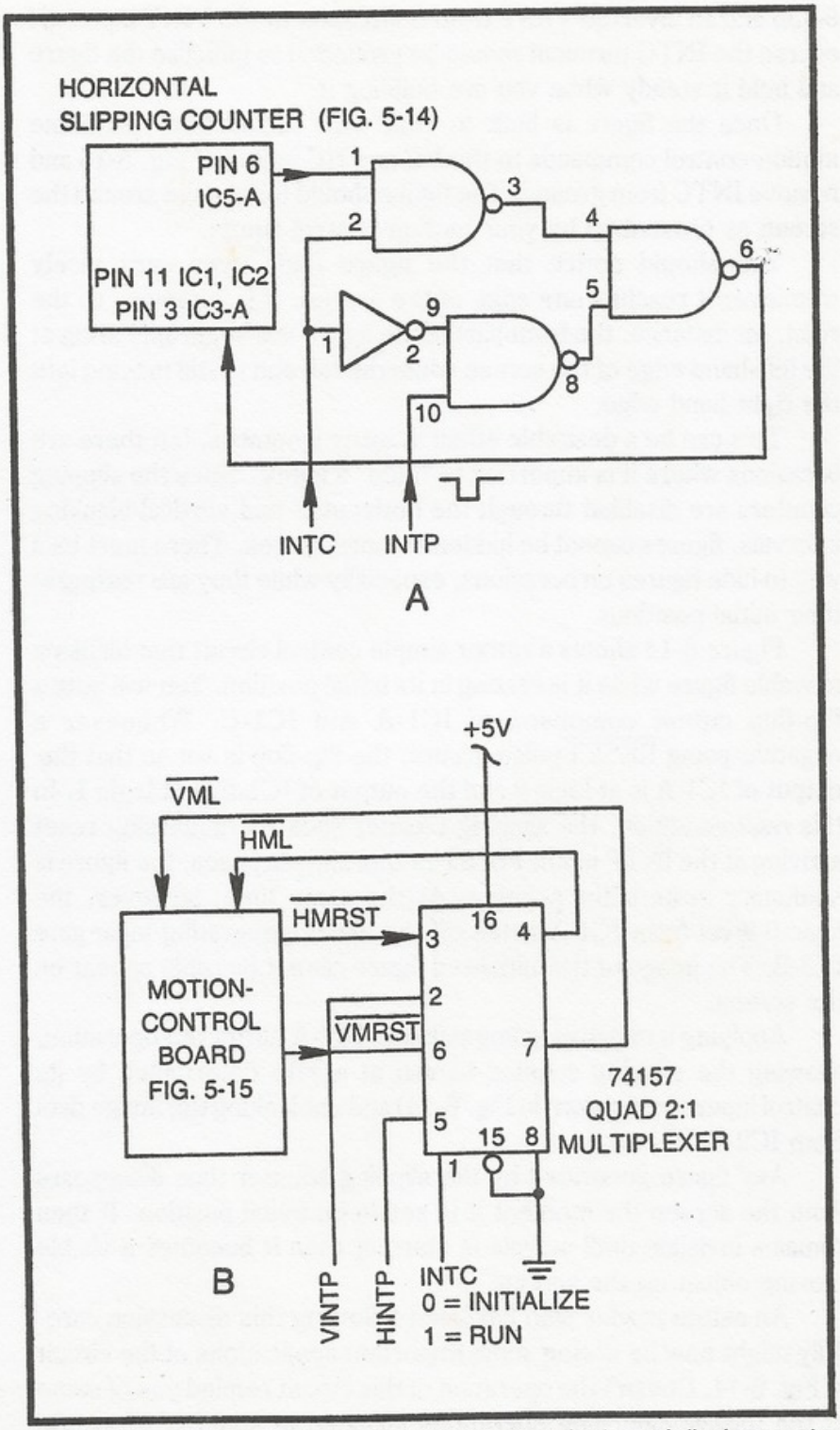


Fig. 6-13. More initialization controls. (a) Interfacing a horizontal slipping counter to an initialization control circuit. (b) Interfacing a complete motion control circuit to a horizontal and vertical initialization control that uses a quad 2:1 multiplexer.

6-13b and an inverted VRST from Sourcebox to the VINT input. Of course the INTC terminal should be grounded to initialize the figure and hold it steady while you are building it.

Once the figure is built to your own satisfaction, set some motion-control commands to the VC and HC inputs of Fig. 5-15 and remove INTC from ground. The figure should then move around the screen as prescribed by your motion-control inputs.

You should notice that the figure folds over very nicely whenever it reaches one edge of the screen. If it is moving to the right, for instance, the front part of the figure will begin appearing at the left-hand edge of the screen while the tail end is still moving into the right-hand edge.

This can be a desirable effect in many instances, but there are occasions where it is important to "hide" a figure. Since the slipping counters are disabled through the horizontal- and vertical-blanking intervals, figures cannot be hidden in those spaces. There must be a way to hide figures on occasions, especially while they are resting in their initial positions.

Figure 6-14 shows a rather simple control circuit that blanks a movable figure while it is resting in its initial position. You will note a flip-flop circuit composed of IC1-A and IC1-C. Whenever a negative-going $\overline{\text{RESET}}$ pulse occurs, the flip-flop is set so that the output of IC1-A is at logic 0 and the output of IC1-C is at logic 1. In this *reset* condition, the slipping counter sees the initializing reset arriving at the INTP input. For all intents and purposes, the figure is stationary in its initial position. At the same time, however, the logic-0 level from IC1-A gates off the figure-generating logic gate IC2-B. The image of the initialized figure cannot possibly appear on the screen.

Applying a negative-going pulse to IC1-A alters the operation, allowing the slipping counter to run at a rate determined by its control inputs (not shown in Fig. 6-14) and unblanking the image data from IC2-B.

Any figure generated by the slipping counter thus disappears from the screen the moment it is set to its initial position. It then remains invisible until a cycle is started; then it becomes a visible moving object on the screen.

An astute reader who has been following this discussion carefully might now be seeing some important applications of the circuit in Fig. 6-14. Doesn't the operation of this circuit remind you of some of the missile-launching operations included in some of the more popular commercial TV games?

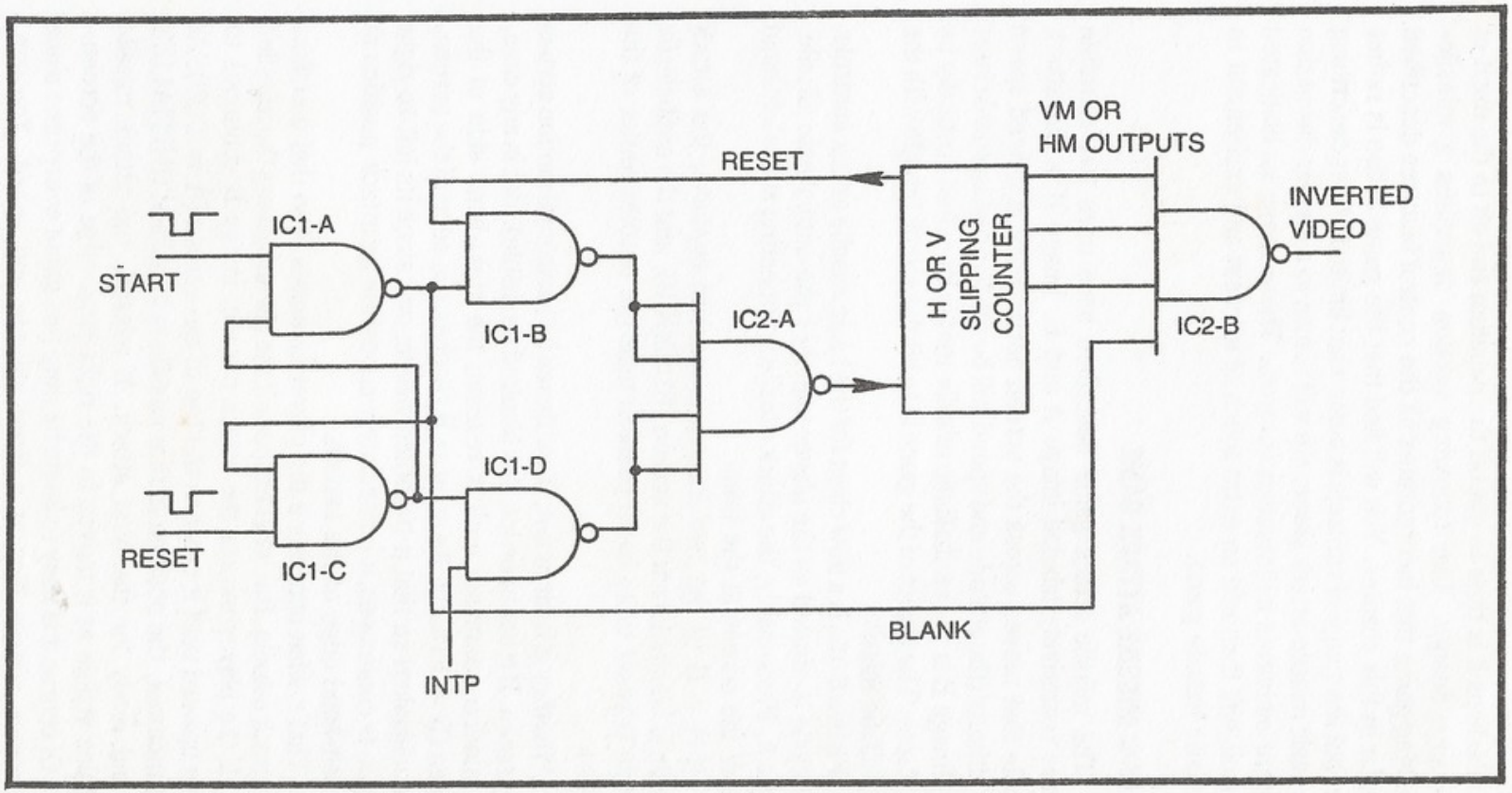


Fig. 6-14. A complete motion control circuit having start/stop, figure initialization, and figure blanking features.

Perhaps it is time to get off the pabulum and on to the meat of TV-game design. The following section describes a missile-launching game that incorporates all the control features described thus far in this chapter. You will find that the game action is rather nice, but the range of controls is somewhat limited. After describing this basic missile-attack game, we will return to a further discussion of more-elaborate initialization controls. These more sophisticated controls will then add an extra touch of interest and excitement to the basic missile game.

A BASIC MISSILE ATTACK GAME

The missile attack game described here uses two movable figures tentatively labeled image A and B. Image B is the attack missile that moves across the screen horizontally at a fixed speed and altitude (the altitude and speed will be made programmable later on.) Image B is the antiballistic missile that is launched vertically by the player. The object of the game is to hit the attack missile with the antiballistic missile.

Figure 6-15a is a flow chart for the basic missile attack controls. The cycle is started as the player launches the antiballistic missile, image A. Presumably, the attack missile is appearing at the left-hand side of the screen at the time.

If A = B (if the two images come into contact), the attack missile is blanked from the screen (BLANK B), and the antiballistic missile is reset to its initial position near the bottom center of the screen.

Whether B is hit or not, it continues its horizontal motion across the screen. If it has been hit, it is blanked (invisible). But in any case, the motion continues until it reaches the right-hand side of the screen (B = EDGE). As soon as B reaches the edge of the screen, it is unblanked so that it becomes visible; and since its left-to-right motion is continuous, it immediately appears in its attack position at the left-hand edge of the screen.

That's what happens if the player launches B so that it strikes the attack missile. But what happens if the player misses the missile?

If the player misses the attack missile, image B continues to move upward until it reaches the top of the screen (A = TOP). At that moment, the position of that missile is initialized (INITIALIZE A) and ready for the next attack. If missed, the attack missile remains visible as it moves to the right-hand edge of the screen.

Of course it is easy to begin beating this game every time since the attack missile flies at a fixed altitude and speed. You can,

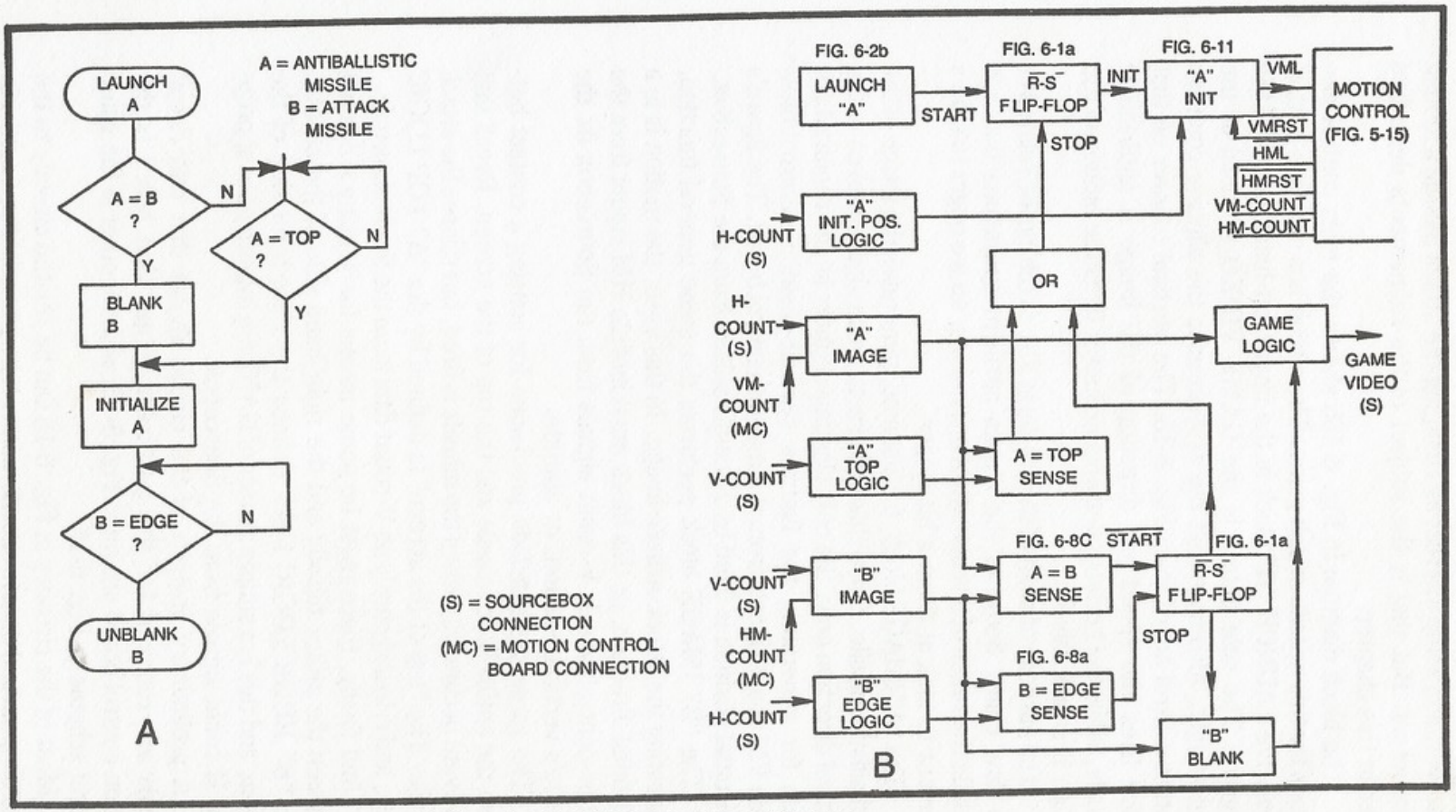


Fig. 6-15. A basic missile attack game. (a) Flowchart. (b) Functional block diagram.

however, add some interest by changing the attack missile's motion parameters. But that is the subject of the refinements described later in this chapter.

The block diagram in Fig. 6-15b shows the main control elements for this missile attack game. The circuit has only one control input, the LAUNCH "A" block in the upper left-hand corner of the diagram. The sole output is the GAME VIDEO terminal at the center right. Also note that the system uses the slipping-counter-motion-control board from Fig. 5-15. The vertical counter on that board fixes the speed and direction of the image A antiballistic missile, while the horizontal-slipping counter fixes the motion of the image B attack missile.

The purpose of most of the blocks in Fig. 6-15b can be related to the flow chart. Some of the blocks perform operations that are outside the realm of direct control operations, so we ought to take a moment to look at them a bit closer.

The "A" IMAGE block, for instance, generates the image of the antiballistic missile. Since this little rectangular figure moves in the vertical direction only, the only slipping-counter inputs required are those for generating the figure's vertical-position information—some VM count signals from the motion-control board. The figure's horizontal position is fixed by H-count pulses from the Sourcebox.

The "B" IMAGE block performs the same general function, generating the attack-missile image. In this case, the motion is in a horizontal direction, so this block must include HM inputs from the motion-control board. V-count signals from the Sourcebox fix the missile's vertical position, or altitude.

The game must include provisions for sensing a contact between the antiballistic missile and the top of the screen. Recall that this event occurs only when the missile is fired, but misses the attack missile. The "top of the screen" is defined by the "A" TOP LOGIC block, and is based purely on V-count data from the Sourcebox unit.

And finally, there must be some means for sensing a contact between the attack missile and the right-hand side of the screen. The "B" EDGE LOGIC block defines the right-hand side of the screen, and this is a simple matter of NANDing together the appropriate H-count signals from the Sourcebox.

A preliminary analysis of the circuit shows that two circuit boards are required for this particular game. One board is the motion-control board shown in Fig. 5-15, and the other is the game control scheme in Fig. 6-16.

Most of the circuitry in Fig. 6-16 can be related directly to the block diagram and then to specific control circuits described earlier in

this chapter. The missile launching and initializing circuits, for example, appear at the top of the schematic diagram. The six NAND gates and two inverters are simply one-for-one combinations of the simpler circuits specified in the block diagram for LAUNCH "A", $\overline{\text{R-S}}$ FLIP-FLOP, and "A" INIT.

IC8-A defines the vertical information for the antiballistic missile. And using the three VM inputs specified at the input of IC8-A, it generates a movable horizontal bar that is 32VM pulses tall. The horizontal position of the antiballistic missile is taken from the point on the screen where 256H makes the transition from black to white, near the middle of the screen. This 256H signal from the Sourcebox is inverted by IC6-C to create a negative-going edge that triggers the pulse-shortening circuit composed of IC2-C and IC7-E. The value of C2 determines the width of the image. A value of 0.002 μF is specified here, but you might want to change that value to generate a figure width more suitable to your own ideas about how the figure should look.

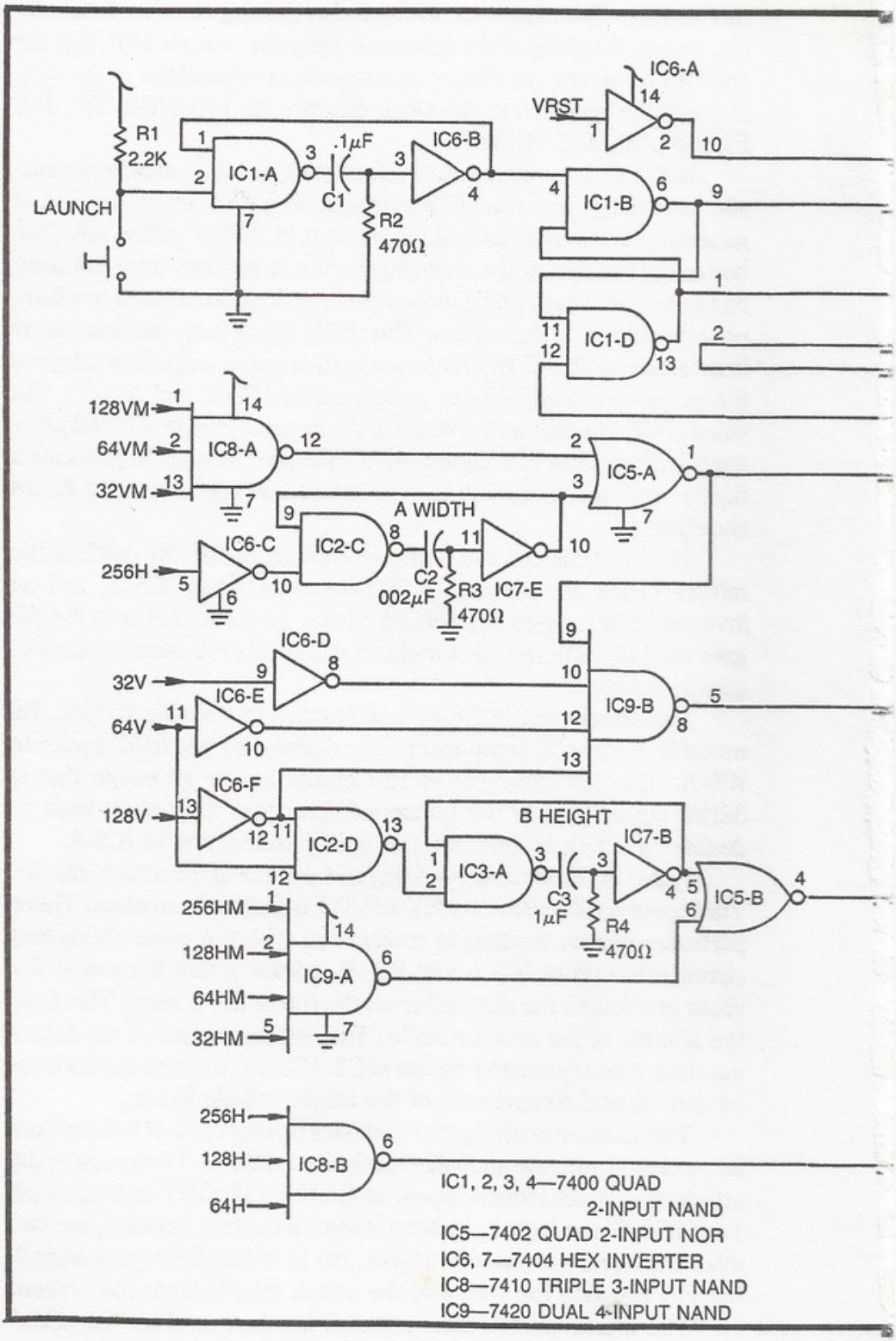
The vertical and horizontal components for the antiballistic missile figure are effectively ANDed together in IC5-A, and an inverted (black-on-white) version of the figure is sent to a NAND gate (IC3-C), where it is uprighted and combined with the attack-missile figure.

The attack-missile figure is generated in a similar fashion. Its movable horizontal components are defined by the HM inputs to IC9-A. This particular set of HM inputs create an image that is 32HM pulses long in the horizontal direction. You might want to double its length by omitting the 32HM connection to IC9-A.

IC2-D is responsible for fixing the altitude of the attack missile. The inputs in this case are $\overline{128\text{V}}$ and 64V from the Sourcebox. These particular inputs, working in conjunction with the pulse-shortening circuit made up of IC3-A and IC7-B, create a thin horizontal line about one-fourth the distance from the top of the screen. This fixes the altitude of the attack missile. The vertical height of the attack missile can be adjusted by means of C3. IC5-B combines the horizontal and vertical components of the attack-missile figure.

The attack-missile figure must pass through IC4-D before it can be combined with the antiballistic missile at IC3-C. The image of the attack missile will indeed appear at GAME VID OUT as long as pin 14 of IC4-D is at logic 1. If there is ever a contact between the two missile images, as sensed by IC4-A, pin 14 of IC4-D drops to logic 0, thereby blanking the image of the attack missile from the screen.

The attack missile then remains blanked until its "invisible" image reaches the right-hand side of the screen, as defined by



IC1, 2, 3, 4—7400 QUAD
2-INPUT NAND
IC5—7402 QUAD 2-INPUT NOR
IC6, 7—7404 HEX INVERTER
IC8—7410 TRIPLE 3-INPUT NAND
IC9—7420 DUAL 4-INPUT NAND

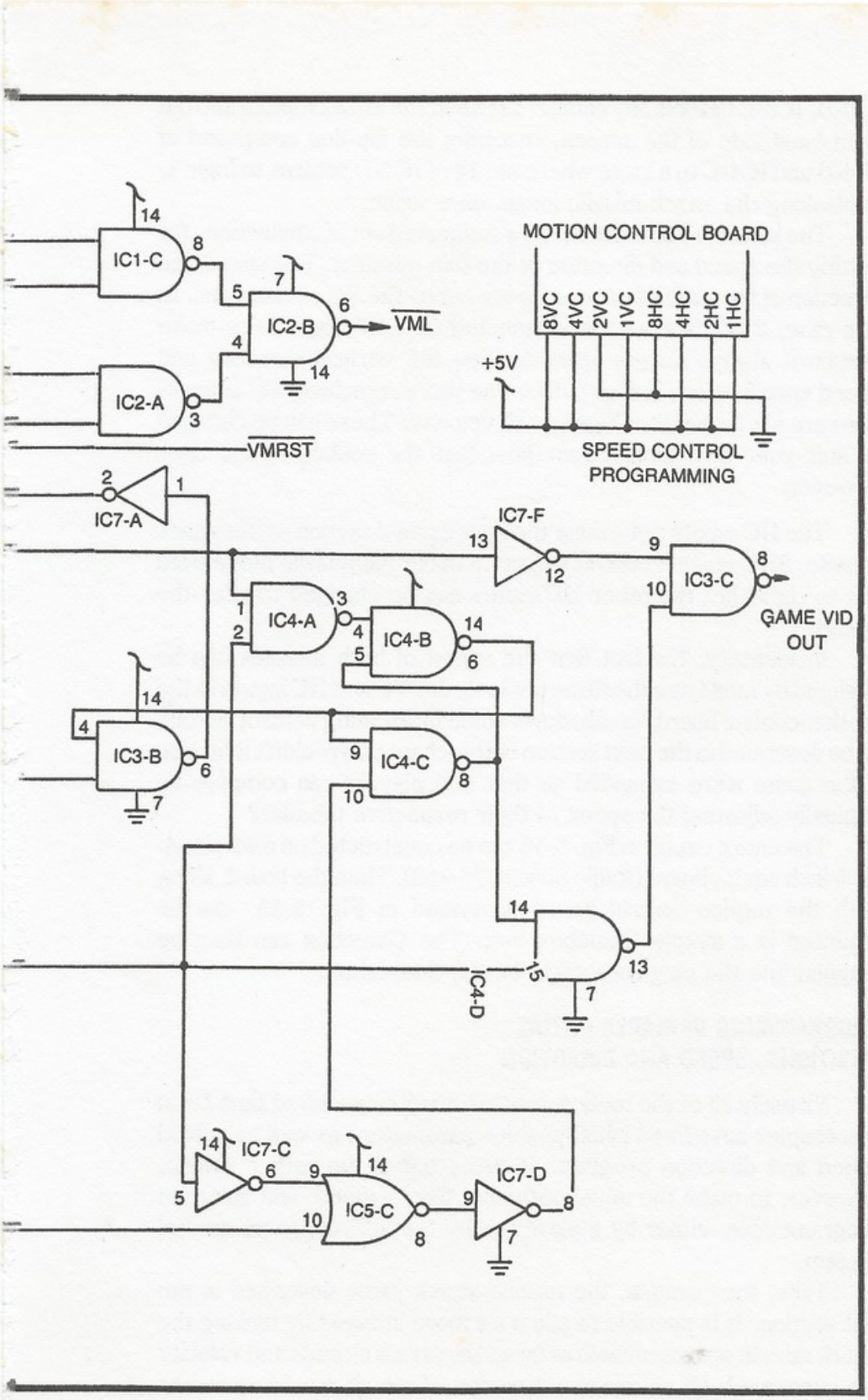


Fig. 6-16. Schematic diagram for the basic missile attack game.

IC8-B. IC5-C detects any contact between the attack missile and the right-hand side of the screen, resetting the flip-flop composed of IC4-B and IC4-C to a state where pin 14 of IC4-D returns to logic 1, unblanking the attack-missile image once again.

The insert in Fig. 6-16 shows a suggested set of connections for setting the speed and direction of the two missiles. The speed and direction of the antiballistic missile are set by the VC connections. In this case, 8VC must remain connected to +5V (logic 1) to make certain it always moves upward. (See the vertical direction and speed specifications in Fig. 5-13.) The three remaining VC connections are set for a rather high launch velocity. These can be changed to suit your own ideas about how fast the rocket should rise, however.

The HC inputs determine the speed and direction of the attack missile. 8HC must remain at logic 0 to make the missile move from left to right, but the other HC inputs can be changed to alter the speed.

Incidentally, the fact that the speed of both missiles can be changed by modifying the three lower-order VC and HC inputs to the motion-control board foreshadows some interesting control circuits to be described in the next section of this chapter. Wouldn't it be nice if the game were expanded so that two players can compete by manually adjusting the speed of their respective missiles?

The entire circuit in Fig. 6-16 can be constructed on a 44-pin, 4- by 4-inch circuit board (Radio Shack 276-153). Then the board, along with the motion-control board described in Fig. 5-15, can be mounted in a simple Gamebox unit. The Gamebox can then be plugged into the plug receptacle on the Sourcebox.

PROGRAMMING VARIABLE INITIAL POSITIONS, SPEED AND DIRECTION

Virtually all of the motion-control circuits described thus far in this chapter have fixed initial-position parameters as well as a fixed speed and direction program. It turns out to be rather simple, however, to make the initial positions, figure speed, and direction programmable, either by a player or by the internal game-control system.

Take, for example, the missile-attack game described in the last section. It is possible to add a lot more interest by making the attack missile programmable as far as the attack altitude and velocity are concerned. Of course the direction of attack can be varied as well, but that wouldn't be altogether appropriate in this case.

Then consider the typical table-tennis game. Whenever the ball is served, the machine ought to be able to set the ball's vertical position and direction just before the serving operation takes place.

The circuits described in this chapter show how to program a wide variety of initial positions, speeds, and directions. The examples are oriented toward manual, or player, controls, but it turns out that the schemes for automatically setting these parameters aren't much different.

Programming the Initial Position

The circuit in Fig. 6-17 shows four SPDT switches, labeled A through D, each connected to one input of an EXCLUSIVE OR gate. The second input to each of the gates goes to a source of count pulses from the Sourcebox, 32H, 64H, 128H, and 256H in this particular example. The outputs of the four EXCLUSIVE OR gates are NANDed together at a 4-input NAND gate, IC2-A.

One of the most useful properties of an EXCLUSIVE OR gate is that it can operate as an inverter or noninverter, depending on the logic level presented to the second input. Consider IC1-D in Fig. 6-17. Whenever the A switch is set to its "1" position, the 32H signal applied to the other input emerges from the gate with the same phase as the input. Setting switch A to its "0" position, however, causes IC1-D to invert its 32H input.

An EXCLUSIVE OR gate thus passes a noninverted version of its signal input whenever the control input is at logic 1, but it inverts the signal whenever the control input is at logic 0.

The circuit in Fig. 6-17 is capable of generating vertical bars on the screen that are 32H wide and positioned at any 1 of 16 different locations. The table accompanying the schematic shows all possible combinations of switch settings and the output specifications that result. If, for example, D = 0 while C = B = A = 1, the effect would be the same as applying $\overline{256H}$, 128H, 64H, and 32H directly to the inputs of the NAND gate.

If you want to check out this circuit in a rather simple fashion, run the output of the NAND gate through an inverter and to the GAME VID IN connection on the Sourcebox. You will indeed find that you can create a white vertical bar that has a position on the screen determined by the settings of switches A through D. In a sense, this circuit is a switch-programmable line generator. Of course you can apply any combination of H- and V-count inputs to the four signal inputs to create horizontal and vertical lines. You can also expand the circuit to accommodate eight signal inputs by using eight

switches, four more EXCLUSIVE OR gates, and a 7430 8-input NAND gate.

The circuits in Fig. 6-18a and 6-18b show how this circuit can be applied to the missile-attack game. The circuit in Fig. 6-18a lets the player adjust the horizontal position of the antiballistic missile. The 256H and 128H inputs to IC10-A fix the positions to a region spanning the middle half of the screen. The player has control over the position of the antiballistic missile within that range by manipulating switches SA, SB, and SC. Using three switches in this manner offers eight different launch positions.

To use this circuit in conjunction with the missile game drawn up in Fig. 6-16, disconnect IC6-C in that circuit, and connect the output of IC11-A in its place. Now you will be able to program the horizontal position of the antiballistic missile, giving you a better chance of shooting down the attack missile. If you manipulate the switches quickly enough, you can actually steer the antiballistic missile while it is in flight.

The circuit in Fig. 6-18b can also be added to the basic missile-attack game. This circuit, however, controls the altitude of the attack missile. The altitude is fixed within the upper third of the screen by the 128V signal that is always inverted by IC12-A before it is applied to NAND gate IC11-B. A player can vary the altitude of the attack missile to any one of eight different positions within that range by means of switches SA through SC.

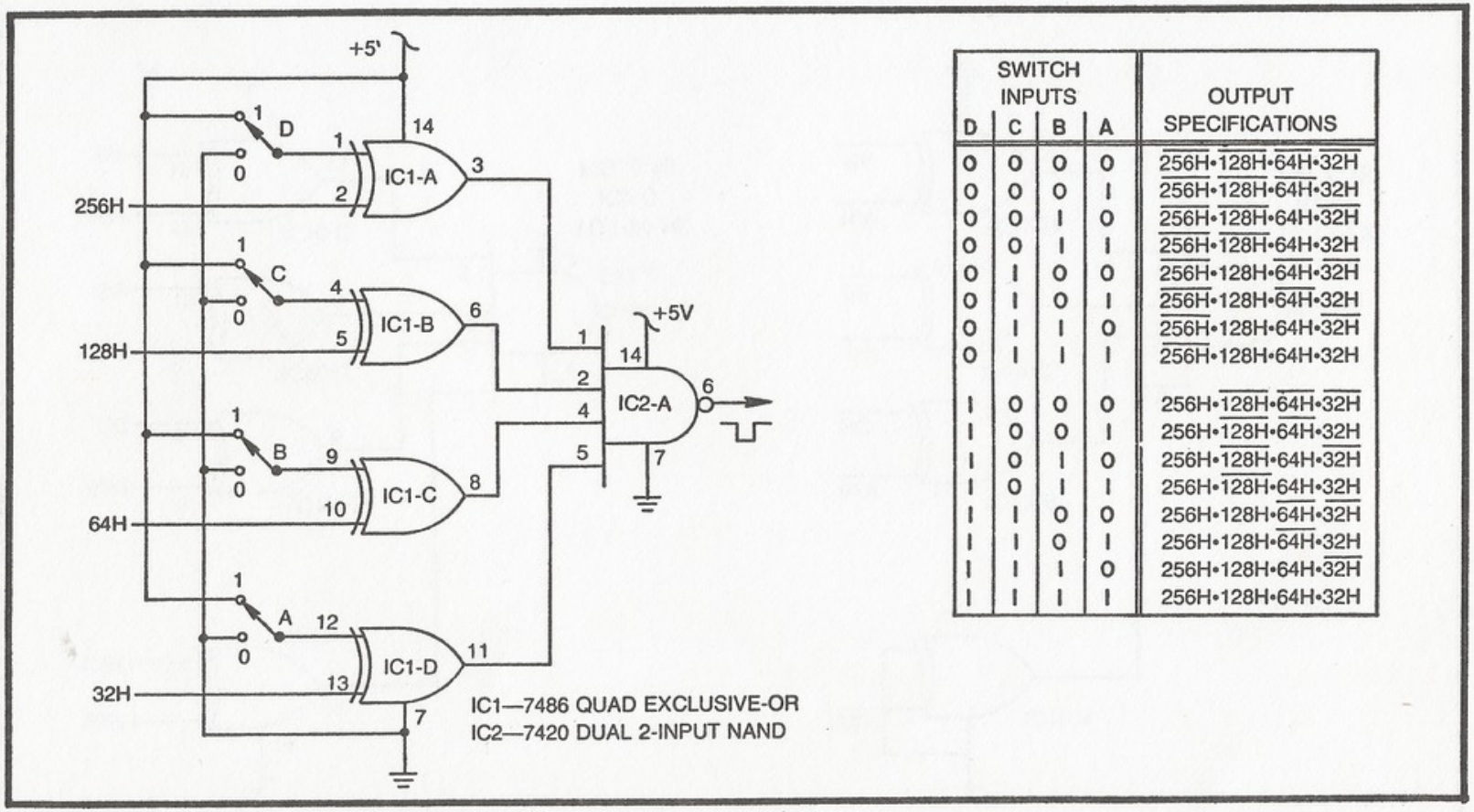
Incorporating this altitude-programming circuit into the missile-attack game is a matter of disconnecting IC2-D in Fig. 6-16 and connecting the output of IC11-B to pin 2 of IC3-A.

Of course these are merely two specific examples of how the position-selection scheme in Fig. 6-17 can be applied. This circuit is quite useful in any case where you would like to have program control over the horizontal or vertical position of an object on the screen.

The same circuit can also be used for defining the initial position of a movable figure. In this case, the output of the NAND gate is used as the source of initializing pulses. See the block diagram in Fig. 6-18c.

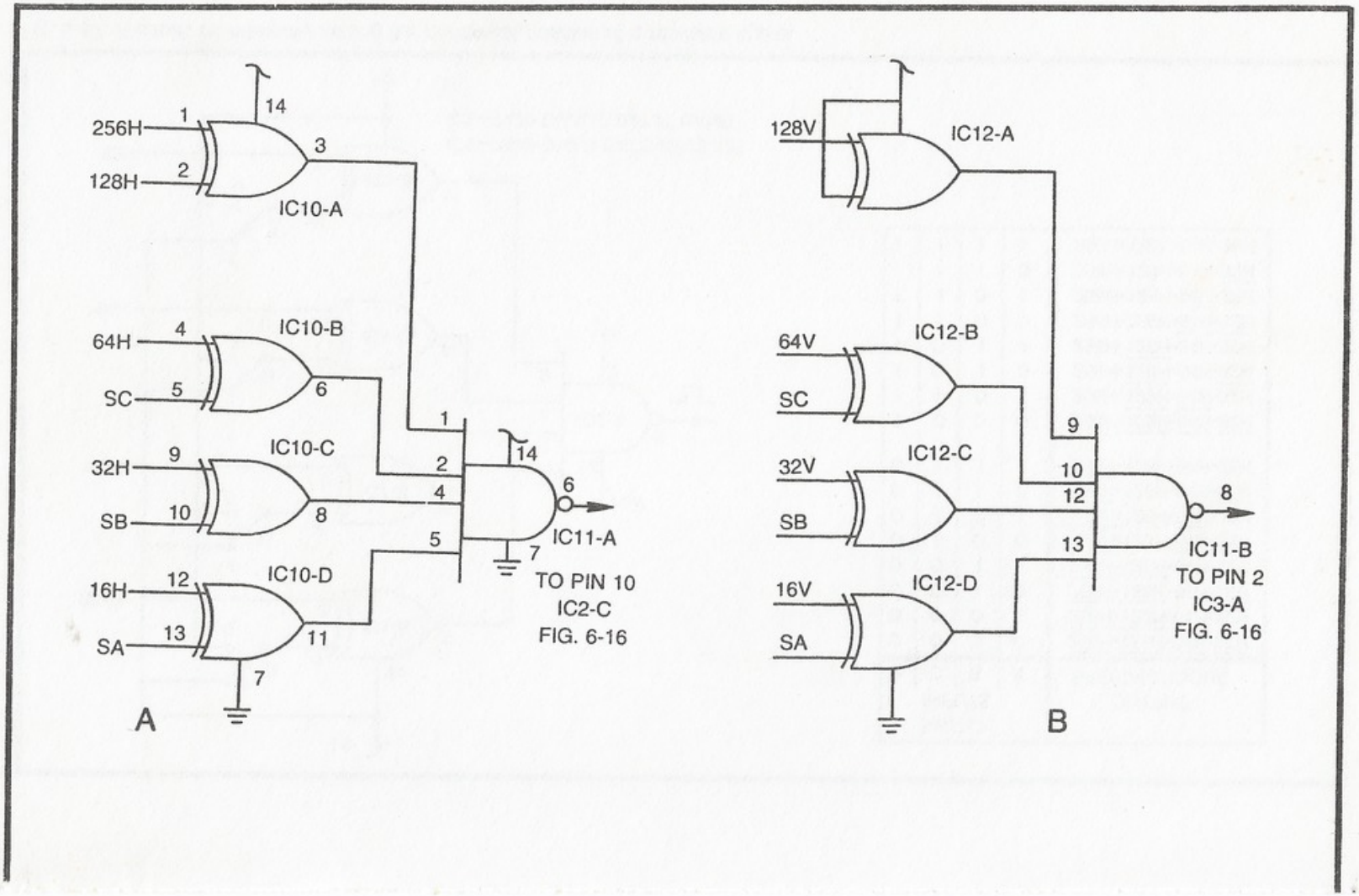
If you would like to check out the operation of this circuit, construct the circuit shown back in Fig. 6-13a, then couple the output of IC2-A in Fig. 6-17 to the INTP input of Fig. 6-13a.

Setting INTC to logic 0, you will find you can program the initial position of any figure generated by the slipping counter. The figure can then be "launched" from that initial position by setting INTC to logic 1.



SWITCH INPUTS D	C	B	A	OUTPUT SPECIFICATIONS
0	0	0	0	$\overline{256H}\cdot\overline{128H}\cdot\overline{64H}\cdot\overline{32H}$
0	0	0	1	$\overline{256H}\cdot\overline{128H}\cdot\overline{64H}\cdot 32H$
0	0	1	0	$\overline{256H}\cdot\overline{128H}\cdot 64H\cdot\overline{32H}$
0	0	1	1	$\overline{256H}\cdot\overline{128H}\cdot 64H\cdot 32H$
0	1	0	0	$\overline{256H}\cdot 128H\cdot\overline{64H}\cdot\overline{32H}$
0	1	0	1	$\overline{256H}\cdot 128H\cdot\overline{64H}\cdot 32H$
0	1	1	0	$\overline{256H}\cdot 128H\cdot 64H\cdot\overline{32H}$
0	1	1	1	$\overline{256H}\cdot 128H\cdot 64H\cdot 32H$
1	0	0	0	$256H\cdot\overline{128H}\cdot\overline{64H}\cdot\overline{32H}$
1	0	0	1	$256H\cdot\overline{128H}\cdot\overline{64H}\cdot 32H$
1	0	1	0	$256H\cdot\overline{128H}\cdot 64H\cdot\overline{32H}$
1	0	1	1	$256H\cdot\overline{128H}\cdot 64H\cdot 32H$
1	1	0	0	$256H\cdot 128H\cdot\overline{64H}\cdot\overline{32H}$
1	1	0	1	$256H\cdot 128H\cdot\overline{64H}\cdot 32H$
1	1	1	0	$256H\cdot 128H\cdot 64H\cdot\overline{32H}$
1	1	1	1	$256H\cdot 128H\cdot 64H\cdot 32H$

Fig. 6-17. A circuit for manually setting the horizontal position of a movable figure.


256H
128H
IC10-A
64H
SC
IC10-B
32H
SB
IC10-C
16H
SA
IC10-D
IC11-A
TO PIN 10
IC2-C
FIG. 6-16
A
128V
IC12-A
64V
SC
IC12-B
32V
SB
IC12-C
16V
SA
IC12-D
IC11-B
TO PIN 2
IC3-A
FIG. 6-16
B

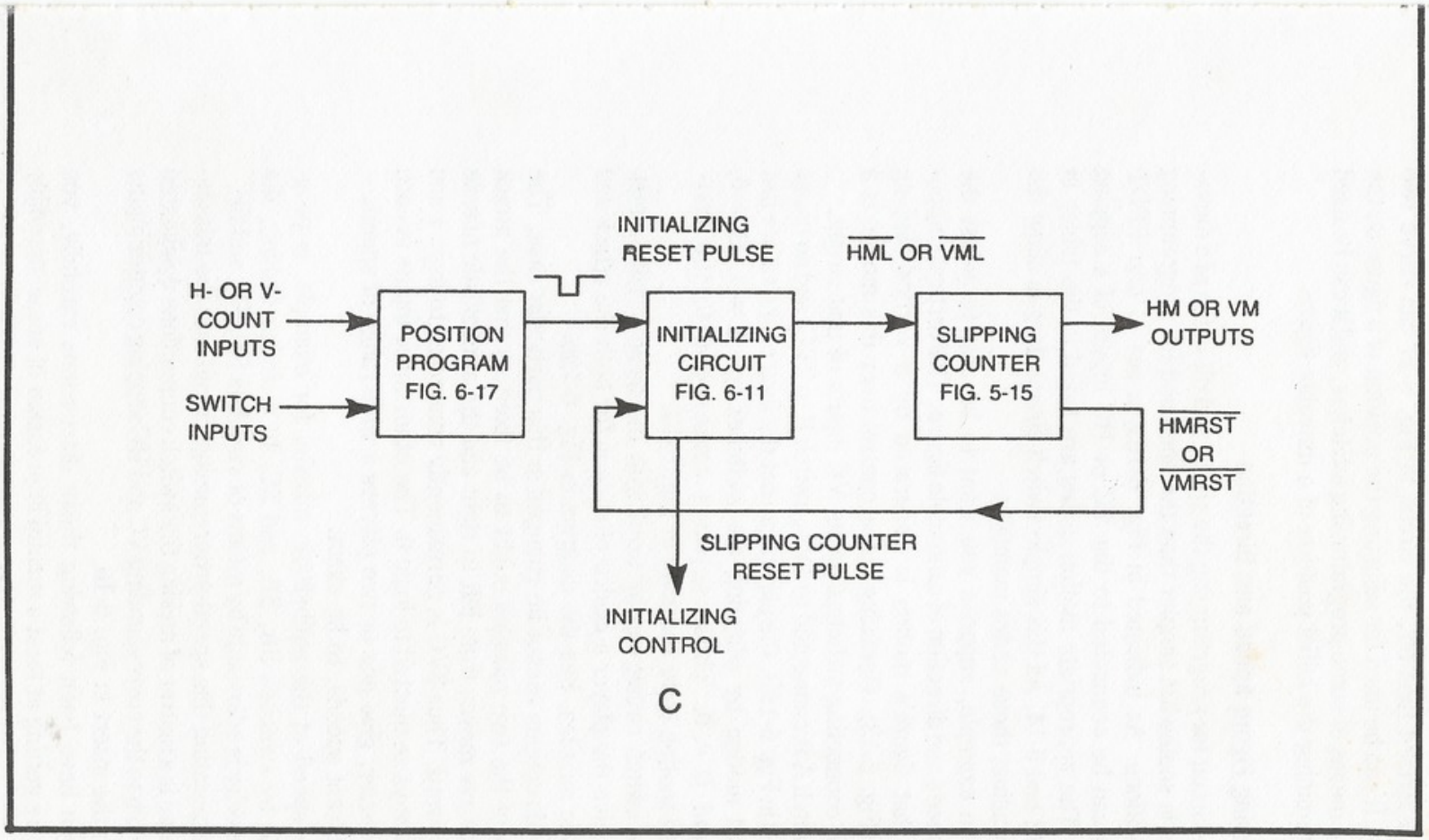


Fig. 6-18. Programming the initial positions of the figures for the basic missile attack game. (a) Horizontal position programming for the antiballistic missile. (b) Vertical initial position programming for the attack missile. (c) Interfacing block diagram.

As described thus far, the circuit in Fig. 6-17 can serve two purposes: It can be used for adjusting the position of a figure on the screen by means of some programming switches, and it can be used for programming the initial position of a movable figure.

Programming Figure Speed and Direction

The circuit for programming the speed and direction of a movable figure is somewhat simpler than the one used for programming initial positions. As indicated in Fig. 6-19a, a set of four SPDT switches can be connected to the VC or HC inputs of a slipping counter. The appropriate motion codes are listed in the tables in Figs. 5-13 and 5-14. All this simple switch circuit does is allow the player to adjust those codes manually.

As an example, suppose you want to be able to adjust the vertical speed and direction of a movable figure. The vertical component of that figure's motion is generated by a vertical-slipping counter (Fig. 5-13). Obtaining some control over this motion is a matter of connecting switches to the VC inputs of that circuit.

If switch A is connected to 1VC, switch B to 2VC, and so on, as illustrated in Fig. 6-19B, the player can set the figure for a rather fast downward motion by adjusting the switches for A = 1, B = 0, C = 1 and D = 0. These settings correspond to the fast-downward-motion code specified in Fig. 5-13.

This switch circuit can be used with the basic missile attack game to give the player a choice of speeds for both the attack and antiballistic missiles. See the diagram in Fig. 6-19b.

The directions cannot be changed in this particular case. The direction of the two missiles ought to be fixed so that the attack missile always moves from left to right and the antiballistic missile moves upward. Thus 8VC is permanently connected to logic 1 and 8HC is always connected to logic 0. The other three inputs in each case, however, give one or two players a wide range of speeds—eight different speeds, to be exact.

The speed of the antiballistic missile, for example, is programmable by switches SA, SB, and SC. In a similar fashion, the attack velocity is adjustable by means of switches SD, SE, and SF.

Incorporating this speed-programming circuit into the missile-attack game is a matter of making the switch connections designated in Fig. 6-19b to the corresponding VC and HC slipping counter inputs shown in the insert in Fig. 6-16.

If you have been following these discussions carefully, you ought to be getting at least a mental impression of some incredibly

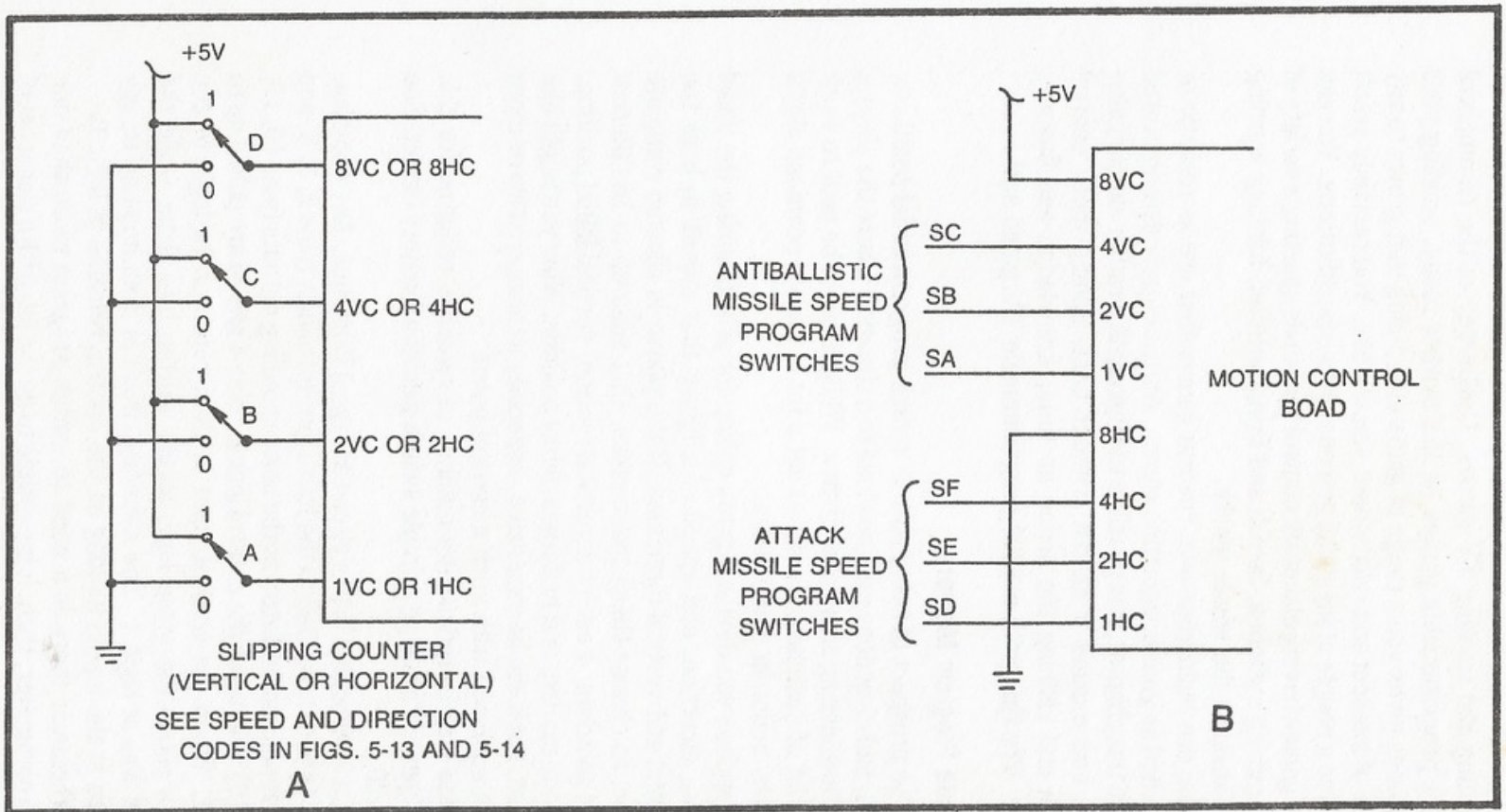


Fig. 6-19. Switch programming motion direction and speed. (a) Manual switch circuitry. (b) Application example for the basic Attack Missile game.

interesting and exciting TV games. Unlike any of the commercial fixed or programmable games on the market today, building your own games gives you a range of game interfacing that grows to any degree of interest and excitement you choose. You certainly aren't limited to a couple of joysticks and one or two pushbuttons. You can devise games having elaborate control terminals sporting a variety of programming switches, launch and firing buttons, blinking warning lights, sirens—the whole works.

You can build your own custom games that are as realistic or far-fetched as your imagination allows. And as clearly demonstrated so far in this chapter, it is possible to begin with a rather simple game format and expand it almost without limit, adding more control circuits and refining the action as time, knowledge and finances permit. Try that with a $200 programmable TV game set!

A Simple Program Memory

The prospect of being able to switch-program initial positions, speeds, and directions can soon lead to a situation where the players face a bewildering array of switches. While it might be neat to work with a lot of switches, having to set a lot of switch positions slows down the action in some cases.

Suppose you devise a game calling for programming the initial position, direction, and speed of a figure that moves in both the horizontal and vertical directions. If the player is allowed complete freedom to choose these parameters, that adds up to 16 different control switches, 4 each for vertical motion, vertical initial position, horizontal motion, and horizontal initial position. Now you might like that idea, but it can be expensive, especially if there are two or more players equipped with such a control panel.

One way to reduce the number of switches required for this sort of game is by using a simple switch-position memory circuit. See Fig. 6-20.

ICs 2 through 5 in this circuit are quad D latches. Any combination of 1s and 0s applied to the four input terminals (pins 2, 3, 6, and 7) will appear immediately at the corresponding outputs (pins 16, 15, 10 and 9) whenever the control inputs (pins 4 and 3) are pulled up to logic 1. When the control input is then returned to logic 0, the outputs retain the same logic levels written into them while the control was at logic 1. The memory circuit is then immune to any changes in the inputs as long as the control remains at logic 0.

Whenever there is a need to switch-program more than one motion parameter, then, it is possible to do the job with a single set of

four switches. As shown in the example in Fig. 6-20, a player can enter vertical-motion data by first setting the positions of the data select switches (SA through SD) to the desired combinations of 1s and 0s and *then* depressing the LOAD V MOTION pushbutton for a moment. Depressing that particular button ultimately applies a logic-1 level to the control inputs of IC2, thereby writing that combination of 1s and 0s into the memory. The VC outputs of IC2 then retain that data until the LOAD V MOTION button is depressed again.

Vertical-position data can be entered in the same fashion, depressing LOAD V POSITION in this case.

The data from the select switches can be loaded into the output latches in any desired sequence and at any time. Some specific examples cited later in this book include some special automatic controls for locking out the programming operations through certain critical phases of the game.

THE TAGALONG FEATURE

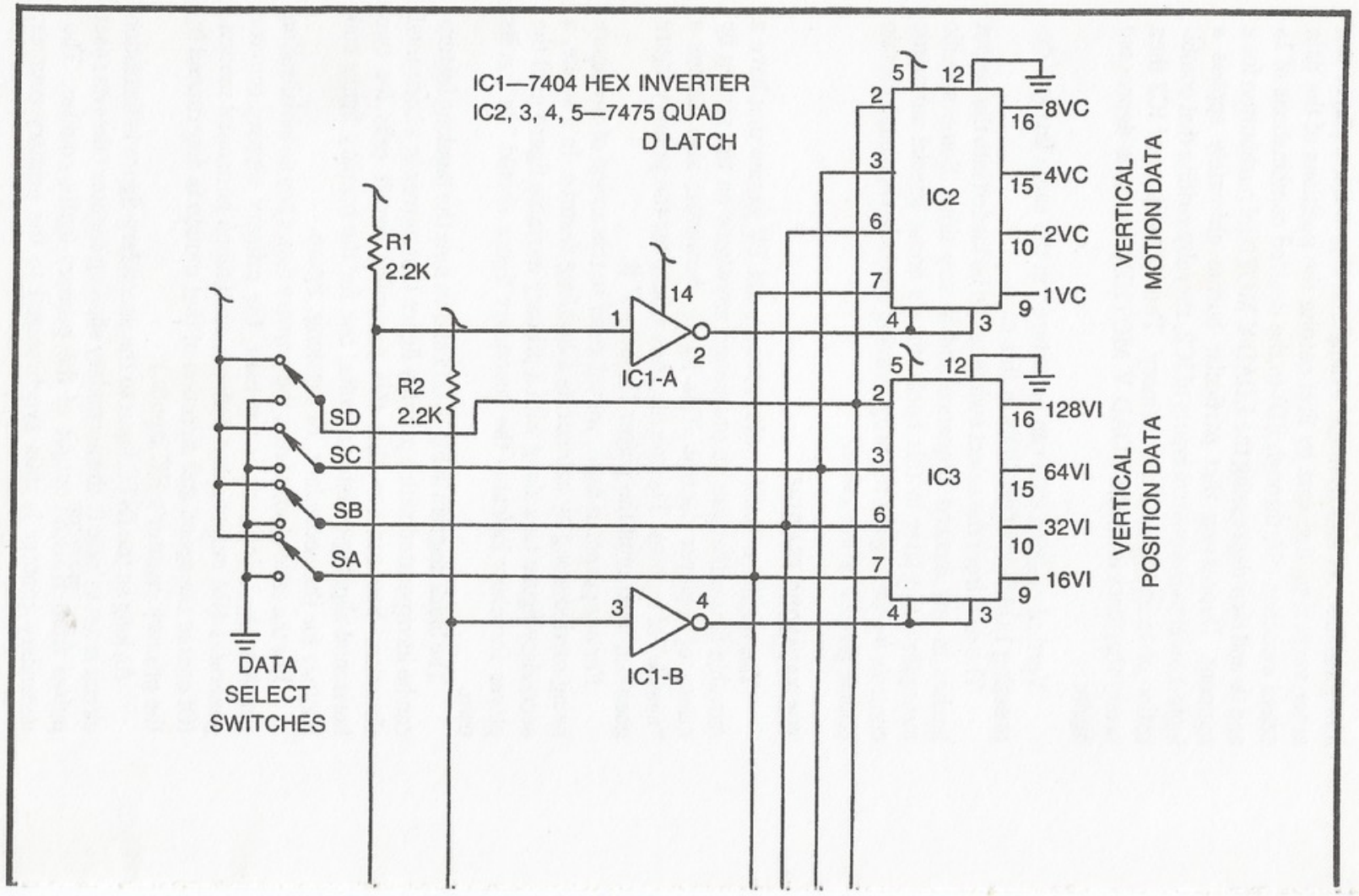
You might be familiar with commercial TV games that have a movable figure that can be positioned anywhere on the screen by means of a player control. This figure, however, also carries a "missile" of its own. The missile goes wherever the primary figure goes—at least until the player "launches" it.

For our purposes here, we will refer to this notion of one figure being carried along by another as a *tagalong* feature. In a sense, a secondary figure tags along with a primary movable figure until the player somehow indicates the secondary figure should "fly" on its own.

The block diagram in Fig. 6-21a shows how the tagalong feature can be incorporated with a primary figure that moves in a horizontal direction. You can see that this particular circuit calls for two horizontal-slipping-counter circuits, one for the primary figure and another for the secondary, or tagalong, figure.

For the sake of simplicity, the primary figure has no initialization circuitry. And since this is the case, the primary slipping counter generates HM outputs that define a continuous horizontal motion. (Of course the speed and direction of that motion is determined by the primary counter's HC inputs.)

As long as the INTC input to the secondary-figure initialization circuit is set at logic 0, the secondary slipping counter takes its reset pulses from $\overline{\text{HMRST}}$ output of the primary-figure counter. The secondary counter is thus synchronized to the primary counter,


IC1—7404 HEX INVERTER
IC2, 3, 4, 5—7475 QUAD
D LATCH
R1
2.2K
R2
2.2K
SD
SC
SB
SA
DATA
SELECT
SWITCHES
IC1-A
IC1-B
IC2
IC3
8VC
4VC
2VC
1VC
128VI
64VI
32VI
16VI
VERTICAL
MOTION DATA
VERTICAL
POSITION DATA

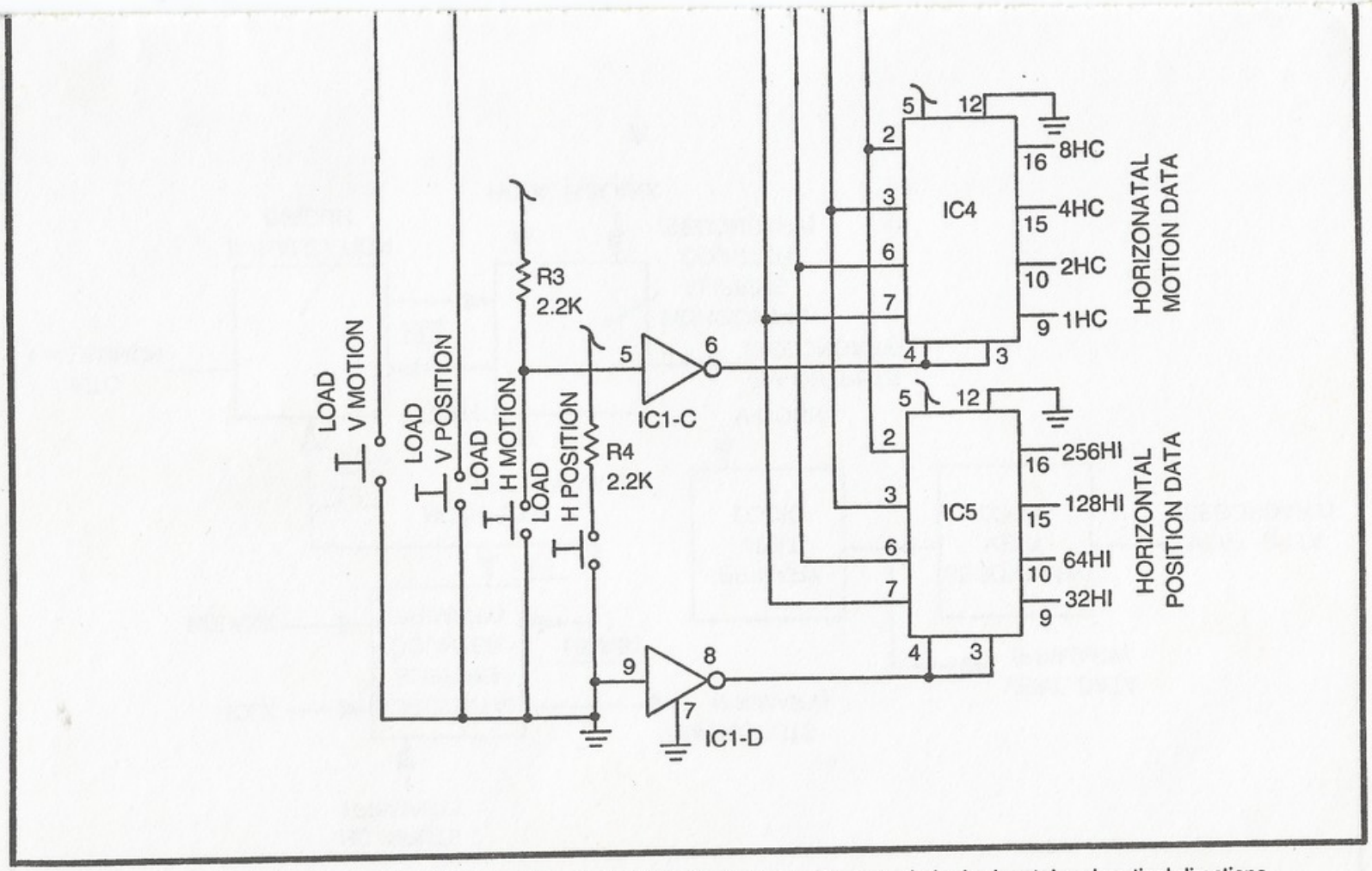
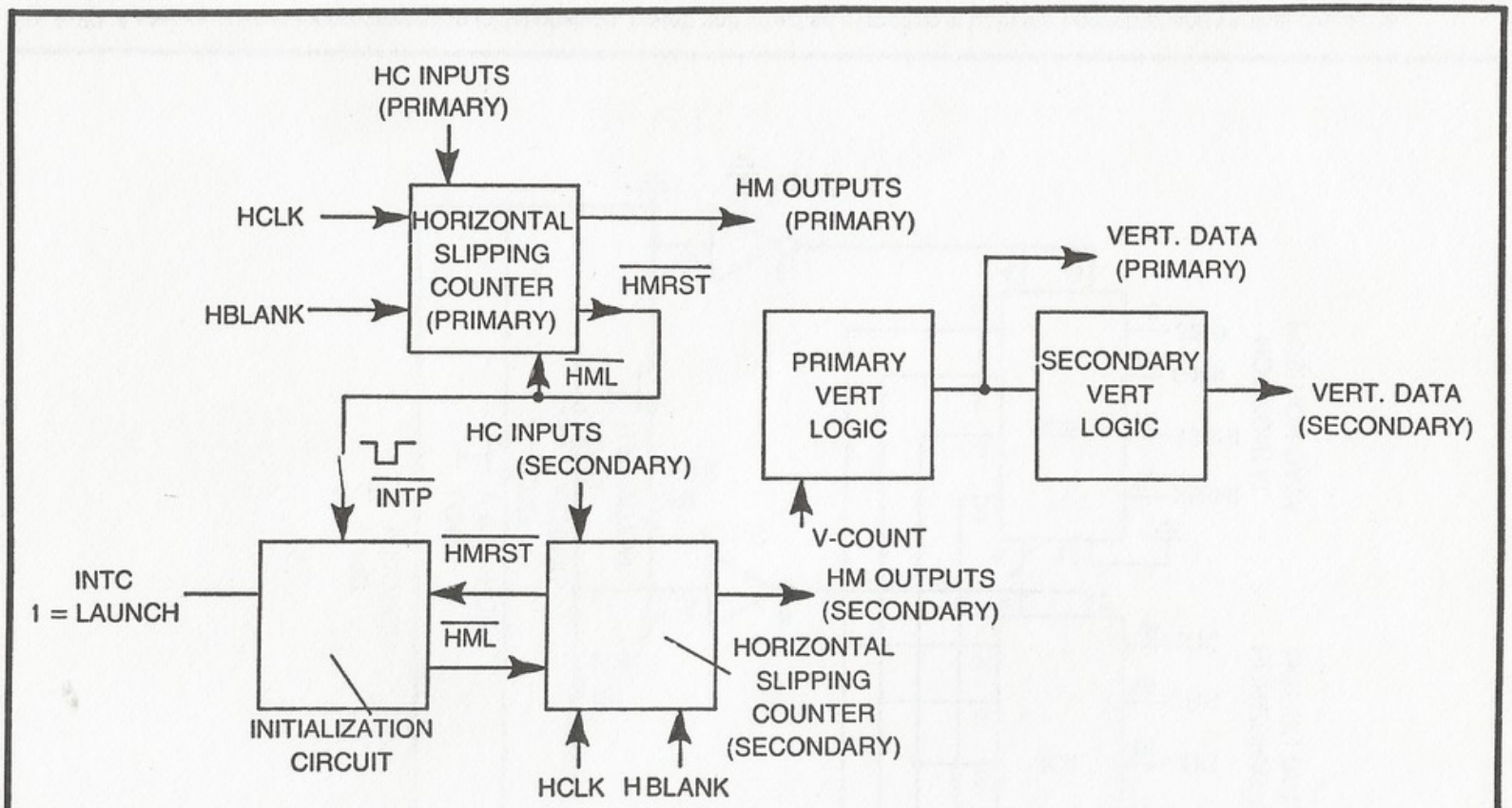


Fig. 6-20. A memory circuit for programming initial position, speed and direction of motion in both the horizontal and vertical directions.


HC INPUTS
(PRIMARY)
HCLK
HBLANK
HORIZONTAL
SLIPPING
COUNTER
(PRIMARY)
HM OUTPUTS
(PRIMARY)
HMRST
HML
VERT. DATA
(PRIMARY)
PRIMARY
VERT
LOGIC
SECONDARY
VERT
LOGIC
VERT. DATA
(SECONDARY)
HC INPUTS
(SECONDARY)
INTP
V-COUNT
HMRST
INTC
1 = LAUNCH
HM OUTPUTS
(SECONDARY)
HML
HORIZONTAL
SLIPPING
COUNTER
(SECONDARY)
INITIALIZATION
CIRCUIT
HCLK
H BLANK

A

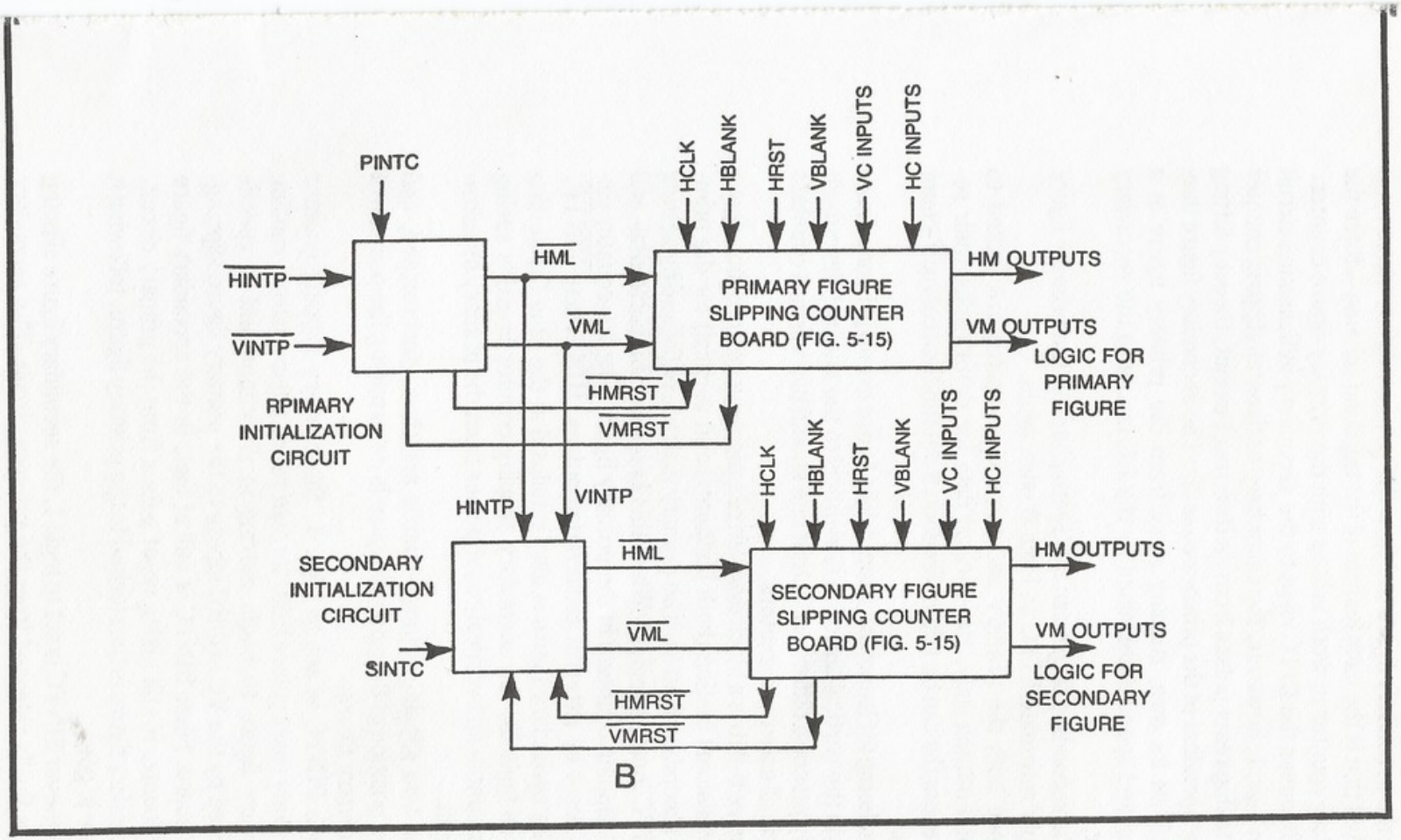


Fig. 6-21. Tagalong block diagrams. (a) The secondary figure follows its primary figure in a horizontal direction until launched by INTC = 1 (b) A complete tagalong control scheme, including initialization circuitry for the primary figure

forcing the secondary figure to follow the primary figure wherever it goes. And that is the main feature of the tagalong scheme—force the secondary counter to work in step with the primary-figure counter.

Whenever the INTC input to the secondary initialization circuit is set to logic 1, however, the secondary horizontal-slipping counter begins taking reset pulses from its own reset circuit, thereby letting it run independent of the primary counter. The secondary figure thus takes off on its own, running away from the primary figure at a direction and speed determined by the HC inputs to the secondary counter.

The secondary figure can be "reattached" to the primary figure by simply returning INTC to logic 0 once again.

Since both the primary and secondary figures are locked to horizontal motion only, the vertical data for determining their respective lengths can be taken direction from the Sourcebox V-count outputs.

A scheme for launching a secondary figure from a primary figure moving in the vertical direction is identical to the one just described. The only practical difference is that one should use vertical counters and vertical-count parameters.

Figure 6-21b is a block diagram of a tagalong system that allows two-dimensional motion (both horizontal and vertical) for the primary and secondary figures. The primary figure is initialized by setting the PINTC input to logic 0. When this happens, the horizontal- and vertical-slipping counters in the primary figure slipping-counter circuit are reset by external initialization pulses, $\overline{\text{HINTP}}$ and $\overline{\text{VINTP}}$. And if the secondary figure is also initialized at the time (by setting SINTC to logic 0), the secondary slipping-counter board is seeing both horizontal- and vertical-reset pulses from the primary initialization circuit.

The two slipping-counter boards are thus synchronized, and the initial position of the primary figure is determining the position of the secondary figure.

Once PINTC is set to logic 1, the primary slipping counter begins taking reset pulses from its own reset-pulse system, causing the primary figure to begin moving in directions and at speeds determined by the VC and HC inputs to the primary figure slipping-counter board. Since SINTC is still at logic 0, the secondary figure slipping counter is still taking reset pulses from the primary circuit. The secondary figure is thus locked to the primary figure, following it wherever it goes.

Whenever SINTC is set to logic 1, the secondary figure slipping counter is finally released from the primary circuit. The secondary

figure thus departs from its home, moving away in a direction and at a speed determined by the VC and HC inputs to the secondary figure slipping counter.

Returning SINTC to logic 0 returns the secondary figure to its "home" at the primary figure, whether the primary figure has been initialized or not.

The speed and direction of the secondary figure can be totally independent of the speed and direction of the primary figure. The relationships between the speed and direction of the two figures are strictly determined by the HC and VC inputs of the two pairs of slipping counters. In some instances, however, you might want the secondary figure to depart at a faster speed, but in the same direction as the primary figure. In this case, connect together the 8VC and 8HC inputs to the two slipping counters. This will lock together their directions of travel. The settings of their three lower-order VC and HC inputs then determine the relative speeds after the secondary figure is launched.

The Torpedo Attack and Dogfight games in Chapter 7 illustrate the application of tagalong circuits.