How To Use EPROMs for Non-Memory Applications

Creating complex digital logic circuits with EPROMs simplifies electronic designing

By John D. Anderson

EPROMs are usually considered to be just computer memory devices. However, they can be programmed to suit other digital functions. All you need is an EPROM programmer. Add a knowledge of how programmable read-only memory devices work, which we'll explore here, and you'll be able to create your own custom-designed digital ICs to plug into an electronic circuit . . . right in your own workshop.

To prove how easy it is to use EPROMs that aren't in the traditional computer memory bailiwick, we'll describe three such applications that can be implemented with judicious programming, the keyboard translator, keyboard controller and stepper-motor controller circuits shown here, all actual working circuits, illustrate only a few of the possible ways an EPROM or PROM can be programmed to suit a specialized application. Applications beyond these are limited only by your imagination and technical ability.

You will note that the three circuits presented here use the 2716 EPROM as the custom-design ele-

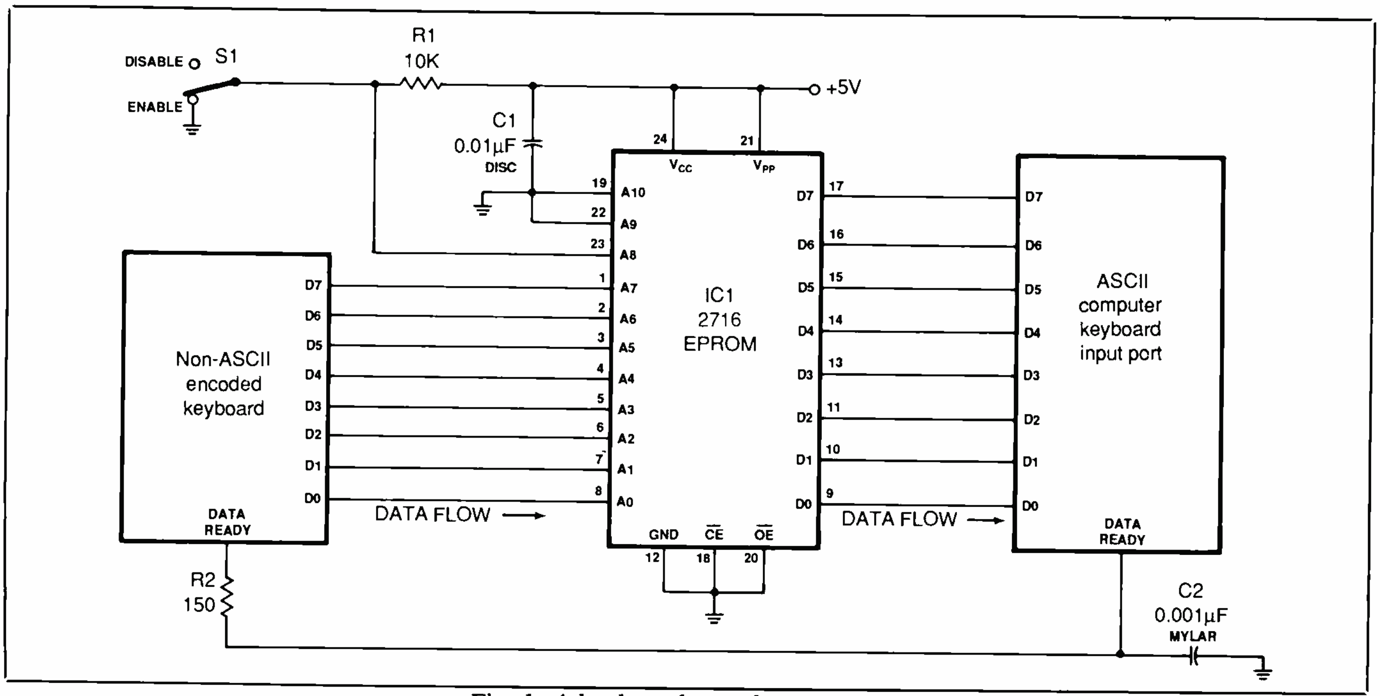


Fig. 1. A keyboard-translator circuit.

Table 1. Programming Data for Keyboard Interface*

Key Pressed	Keyboard Output (EPROM Address*)	EPROM Output	Computer "sees" ASCII Character
A	00000000	01000001	A
B	00000001	01000010	B
C	00000010	01000011	C
D	00000011	01000100	D
etc.	etc.	etc.	etc.

Address lines A8, A9 and A10 are tied low. *Partial table listing; see text for details.

ment. This EPROM was selected only because of its low cost and ready availability. In actuality, just about any other EPROM or PROM could be used if you happen to have them on hand.

In all three circuits, the EPROM is forced into the "read" (output-only) mode by hard-wiring pins 18 and 20 low and pin 21 high. This arrangement causes the EPROM to output a programmed data word on data-output lines D0 through D7 for every address word that appears on address-input lines A0 through A10. Technically speaking, we're turning a computer memory chip into a programmable logic array, or PLA.

Keyboard Translator

Figure 1 shows how a 2716 EPROM can be used to as an interface between a nonstandard encoded typewriter keyboard and a computer that requires ASCII-type keyboard coding. (Computer builders take note—maybe you *can* use those bargain-basement keyboards after all.) This circuit is easy to build. You simply connect the output lines of the keyboard to the address inputs of the 2716 EPROM and the EPROM's data output lines to the computer's keyboard input port. A simple resistor/capacitor network is included in the circuit to delay the KEY PRESSED strobe pulse to compensate for the 450-nanosecond data delay through the EPROM.

Table 1 shows a partial listing of the data to be programmed into the EPROM to turn it into a keyboard translator. Switch *S1* on address line A8 permits the translation function to be enabled and disabled. In Table 1, you can see that when address line A8 is high, the 8-bit data word coming out of the EPROM on data lines D0 through D7 is the same as the 8-bit address word being supplied by the keyboard. This makes the translator "transparent" to the keyboard and computer. (Because only the lower 512 bytes of the EPROM need be programmed, address lines A9 and A10 are tied low.) You should draw up a complete ASCII table and fill in the required output data. This table will prove to be very helpful as you program the EPROM and later

Table 2. Programming Data for Keyboard Encoder/Latch

EPROM Address Input											EPROM Output							
A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
S8	S7	S6	S5	S4	S3	S2	S1	Feedback			Encoder Output			Not used		Feedback		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	X	X	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	1	X	X	0	0	1
0	0	0	0	0	0	0	0	0	1	0	0	1	0	X	X	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1	1	X	X	0	1	1
0	0	0	0	0	0	0	0	1	0	0	1	0	0	X	X	1	0	0
0	0	0	0	0	0	0	0	1	0	1	1	0	1	X	X	1	0	1
0	0	0	0	0	0	0	0	1	1	0	1	1	0	X	X	1	1	0
0	0	0	0	0	0	0	0	1	1	1	1	1	1	X	X	1	1	1
0	0	0	0	0	0	0	1	0	0	0	0	0	0	X	X	0	0	0
0	0	0	0	0	0	1	X	0	0	1	0	0	1	X	X	0	0	1
0	0	0	0	0	1	X	X	0	1	0	0	1	0	X	X	0	1	0
0	0	0	0	1	X	X	X	0	1	1	0	1	1	X	X	0	1	1
0	0	0	1	X	X	X	X	1	0	0	1	0	0	X	X	1	0	0
0	0	1	X	X	X	X	X	1	0	1	1	0	1	X	X	1	0	1
0	1	X	X	X	X	X	X	1	1	0	1	1	0	X	X	1	1	0
1	X	X	X	X	X	X	X	1	1	1	1	1	1	X	X	1	1	1

X = Don't Care. Since D3 and D4 are not used in basic circuit, data here can be either a 0 or a 1 with no change in circuit operation.

on for debugging the circuit should this become necessary.

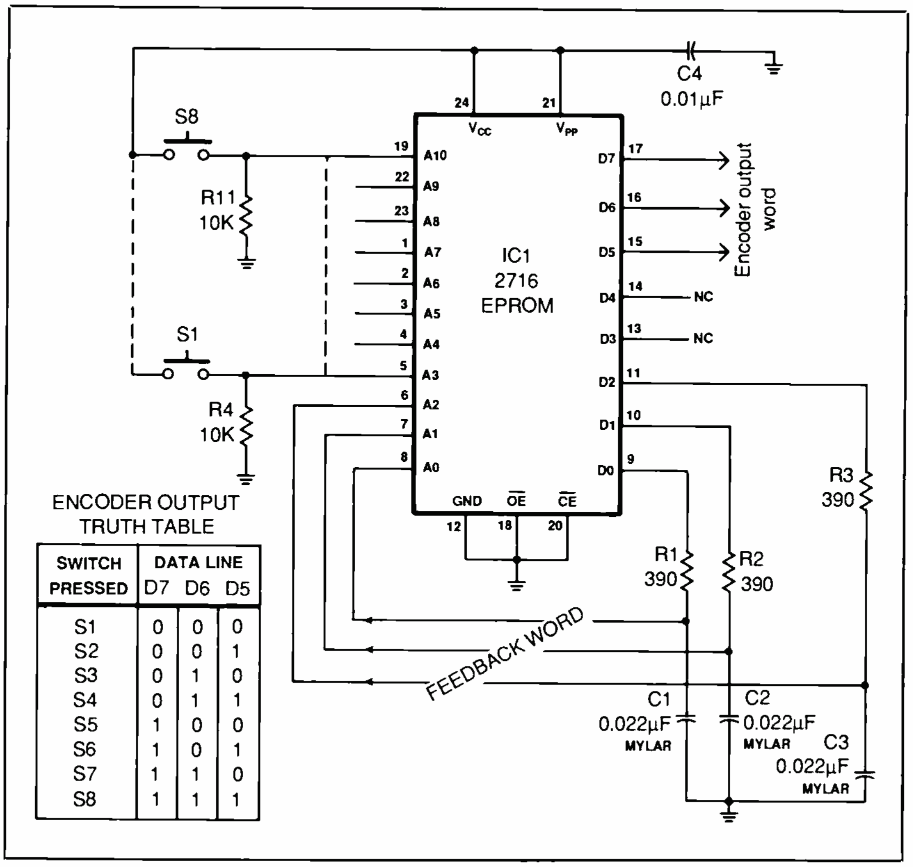
Keyboard-Controller System

An EPROM can also be used as a priority encoder/latch for an eight-switch keyboard, as shown in Fig. 2. (This type of circuit is commonly used in video and audio selector systems.) The circuit outputs and holds a 3-bit binary word that corresponds to the last keyswitch pressed. If two or more keyswitches are held down simultaneously, the circuit outputs the code for the highest-priority (highest-numbered) one.

Figure 2 shows how to wire the EPROM into a digital feedback configuration, with data output lines D0, D1 and D2 driving address input lines A0, A1 and A2, respectively. These feedback lines provide the encoder's latch (memory) function. Resistor/capacitor networks *R1/C1*, *R2/C2* and *R3/C3* on feedback lines A0/D0, A1/D1 and A2/D2, respectively, prevent circuit "racing" (unstable output caused by unstable, fast-changing feedback lines). The 3-bit encoder output is provided on data output lines D5, D6 and D7. Data lines D3 and D4 aren't used here but could be programmed for other outputs, such as a KEY PRESSED signal or an expanded 5-bit encoder output.

Data words to be programmed into the EPROM are detailed in Table 2. As long as no key is pressed, the data output doesn't change because the feedback input word to the EPROM generates the same feedback output word from the EPROM. Hence, the data output is "locked" into a stable state (for simplicity, each 3-bit feedback word corresponds to the same 3-bit data output word).

A key must be pressed to change the output data word. Pressing any keyswitch pulls a corresponding address line high and forces the EPROM to output a different data word. Releasing the key causes the feedback lines to lock the new data word on the EPROM output.



ENCODER OUTPUT TRUTH TABLE

SWITCH PRESSED	DATA LINE D7	D6	D5
S1	0	0	0
S2	0	0	1
S3	0	1	0
S4	0	1	1
S5	1	0	0
S6	1	0	1
S7	1	1	0
S8	1	1	1

Fig. 2. A keyswitch encoder with feedback.

Although the Fig. 2 circuit provides a basic switch-debounce function, short noise spikes may appear on the output data word as switch contacts bounce as they're open and closed. The noise generated by contact bouncing can appear because the EPROM output data word is undefined while the address input lines are changing. If you note a noise problem as you operate this circuit, you can eliminate it by bypassing the affected data lines to ground with 0.001-microfarad disc capacitors. Since they already pass through the resistor/capacitor network, the feedback lines don't require bypassing.

Stepper-Motor Controller

Figure 3 shows how an EPROM can be used to generate the timing signals required for controlling a four-phase stepper motor. Table 3 details how each winding of the motor must be turned on and off to cause the stepper shaft to rotate one step in a given direction. This table is typical for most 4-winding stepper motors.

In Fig. 2, EPROM data lines D4 through D7 switch on and off Darlington transistor amplifiers *Q4* through *Q1* through current-limiting

Table 3. Stepper-Motor

Winding Energizing Sequence

Winding Number

1	2	3	4
OFF	ON	ON	OFF
OFF	ON	OFF	ON
ON	OFF	OFF	ON
ON	OFF	ON	OFF

cycle repeats

For clockwise rotation, read down; for counterclockwise rotation, read up.

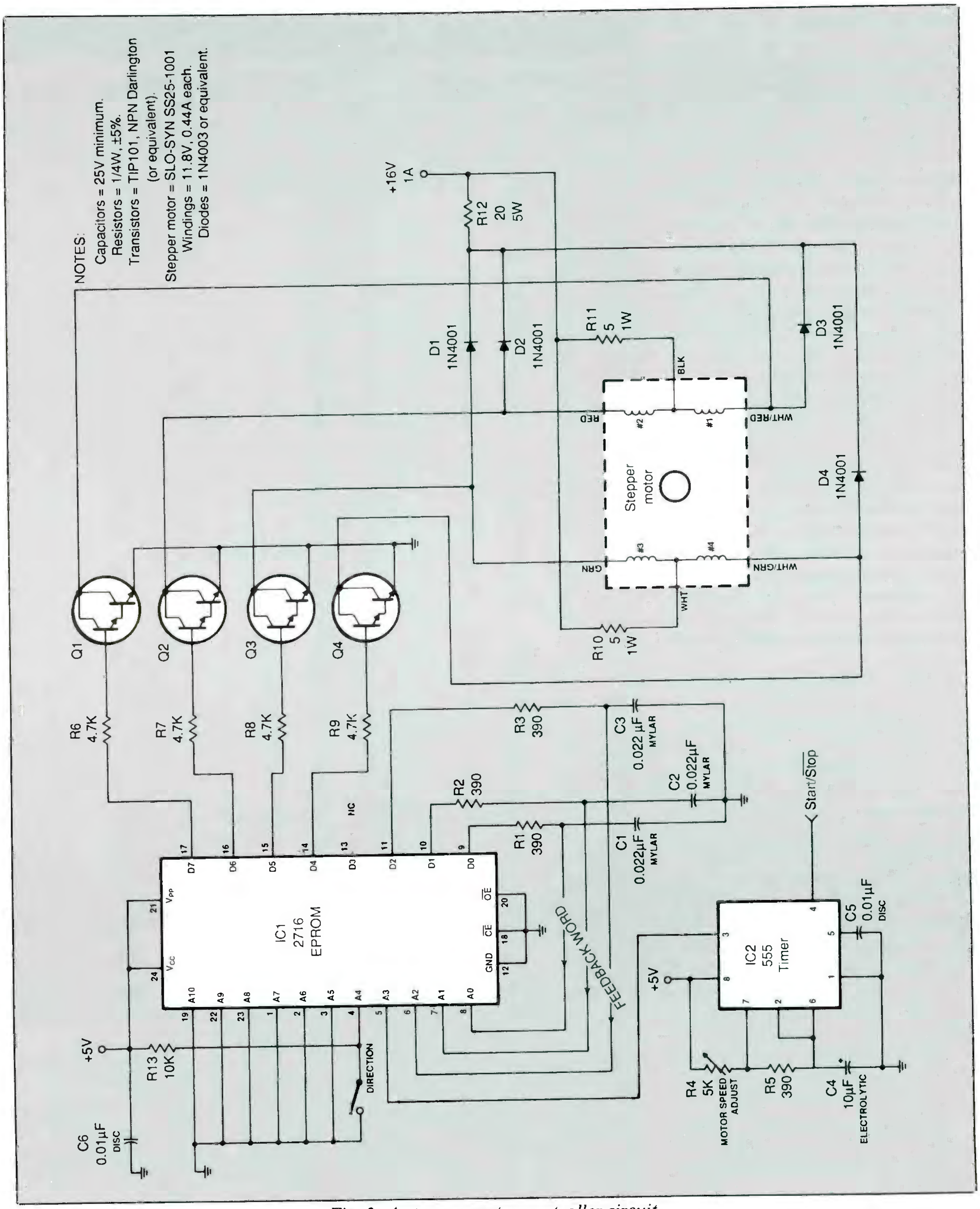

NOTES:
Capacitors = 25V minimum.
Resistors = 1/4W, ±5%.
Transistors = TIP101, NPN Darlington (or equivalent).
Stepper motor = SLO-SYN SS25-1001
Windings = 11.8V, 0.44A each.
Diodes = 1N4003 or equivalent.

Fig. 3. A stepper-motor controller circuit.

resistors *R9* through *R6*. In turn, these transistors control the current flow to the stepper motor individually for each of its windings. Resistors *R10, R11* and *R12* and diodes *D1* through *D4* suppress transient switching spikes generated by the motor's highly inductive windings. Digital feedback is provided by EPROM data lines D0, D1 and D2. Data line D3 isn't used in this circuit, but it could be programmed to provide another output signal. A clock generator built around a 555 IC provides pulses to the EPROM. The EPROM moves the motor shaft one step for each clock pulse.

When using an EPROM as a sequence generator driven from an external clock, it's important to arrange the output data so that only *one* feedback line changes state every time the clock input changes states (either a low-to-high or a high-to-low transition) to prevent two feedback lines from changing state at exactly the same time. Also, make sure the output data word is stable during both high and low periods of the clock pulse.

Table 4 shows the data to be programmed into the EPROM for the Fig. 3 circuit. Assume that the feedback lines A0/D0, A1/D1 and A2/D2 are at position 001, motor winding output signals D4 through D7 are at 0110, and the STEP and DIRECTION inputs A3 and A4 are both 0 (second line in Table 4). The address applied to the EPROM is 00000000001. (Address lines A5 through A10 aren't used in this application and are tied low to ground). The data word output is 01100001, which is the data word programmed into the EPROM for that address. At this point, the EPROM output is stable and unchanging because the feedback word (001) is causing the EPROM to output the same feedback word.

Now assume that STEP input A3 goes high. At the instant it does, the address applied to the EPROM input is 00000001001. (Keep in mind, though, that the capacitors on the feedback lines don't allow the feedback word to change instantly.) For that address input, the EPROM outputs the data word 01010011, which means that the motor winding signal is now 0101, the motor has advanced one step, and the feedback word is 011. As feedback capacitor *C2* charges up, the address inputs change to 00000001011. For this address input, the EPROM outputs the same data word, 01010011, and the output is stable as long as the STEP input is high.

When the STEP input changes from high to low, the EPROM address input changes to 00000000011. Now the output data word is 01010010, which means the motor winding signal is the same (0101) but

(Continued on page 91)

Table 4. Programming Data for Stepper-Motor Controller

EPROM Address*					EPROM Output							
A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Dir.	Step	Feedback			#1	#2	#3	#4	**	Feedback		
					Clockwise Rotation							
0	0	0	0	0	0	1	1	0		0	0	1
X0	0	0	0	1	0	1	1	0		0	0	1
0	1	0	0	1	0	1	0	1		0	1	1
X0	1	0	1	1	0	1	0	1		0	1	1
0	0	0	1	1	0	1	0	1		0	1	0
X0	0	0	1	0	0	1	0	1		0	1	0
0	1	0	1	0	1	0	0	1		1	1	0
X0	1	1	1	0	1	0	0	1		1	1	0
0	0	1	1	0	1	0	0	1		1	1	1
X0	0	1	1	1	1	0	0	1		1	1	1
0	1	1	1	1	1	0	1	0		1	0	1
X0	1	1	0	1	1	0	1	0		1	0	1
0	0	1	0	1	1	0	1	0		1	0	0
X0	0	1	0	0	1	0	1	0		1	0	0
0	1	1	0	0	0	1	1	0		0	0	0
X0	1	0	0	0	0	1	1	0		0	0	0
					Counterclockwise Rotation							
1	0	0	0	0	1	0	1	0		1	0	0
X1	0	1	0	0	1	0	1	0		1	0	0
1	1	1	0	0	1	0	0	1		1	0	1
X1	1	1	0	1	1	0	0	1		1	0	1
1	0	1	0	1	1	0	0	1		1	1	1
X1	0	1	1	1	1	0	0	1		1	1	1
1	1	1	1	1	0	1	0	1		1	1	0
X1	1	1	1	0	0	1	0	1		1	1	0
1	0	1	1	0	0	1	0	1		0	1	0
X1	0	0	1	0	0	1	0	1		0	1	0
1	1	0	1	0	0	1	1	0		0	1	1
X1	1	0	1	1	0	1	1	0		0	1	1
1	0	0	1	1	0	1	1	0		0	0	1
X1	0	0	0	1	0	1	1	0		0	0	1
1	1	0	0	1	1	0	1	0		0	0	0
X1	1	0	0	0	1	0	1	0		0	0	0

X = Stable state. *A5 thru A10 tied low to ground. **Not used.

the feedback word has changed again—to 010. When feedback capacitor *C1* discharges, making the address input 0000000010, the output data word remains 01010010 and the output will be stable until the STEP input goes high again.

The above sequence of events continues with the stepper motor winding signal changing at every low-to-high transition of the STEP input. At the same time, the feedback word changes at every STEP transition. If DIRECTION input A4 is high, the process is the same, except that the feedback word changes in reverse order at the low-to-high STEP transition. The Fig. 3 stepper-motor controller circuit has room for expansion at the address inputs to the EPROM. These uncommitted inputs could be used to connect mechanical limit switches to the EPROM. Then a program could be worked up to inhibit the stepper motor in one direction if the corresponding limit switch is activated by the rotation of the motor shaft past a certain point.

Another use of the uncommitted address inputs is shown in Fig. 4. Here an R/C network and Schmitt-trigger inverter (the latter to clean up the slowly rising voltage across the capacitor) generate a power-on reset signal that forces the EPROM to output a specific data word whenever power is applied. When programming the EPROM for the rest function, simply "burn in" the reset data word at all locations above address 10000000000.

Construction Tips

When you build the circuits presented here—and any others of your own design—be sure to connect a 0.01-microfarad disc capacitor between pins 12 and 24 of the EPROM to reduce switching noise on the power-supply lines. These and most other circuits can be wired in any manner that suits you, including on printed-circuit boards of your own design and fabrication, on solderless bread-

om page 32)

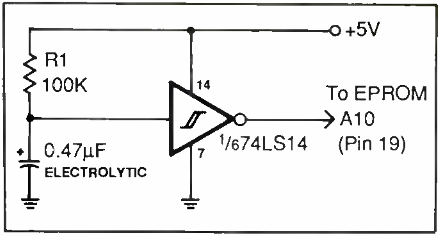


Fig. 4. A power-on reset for the stepper-motor controller.

boarding sockets and on perforated board with appropriate Wire Wrap or solder hardware. Though component layout is rarely critical, you should keep all component leads and interconnecting wire runs as short and direct as possible, especially when you point-to-point wire or Wire Wrap the circuits.

After any circuit in which an EPROM is used is wired and ready to put into operation, always cover the window on the EPROM with a self-stick opaque label or black electrical tape. If you don't cover the window (or at least house the circuit in a light-tight enclosure), you run the risk of accidentally erasing the data programmed into the EPROM. Bright direct sunlight can erase the programmed-in data in short order, and even fluorescent light can erase the data in less than a week. ME