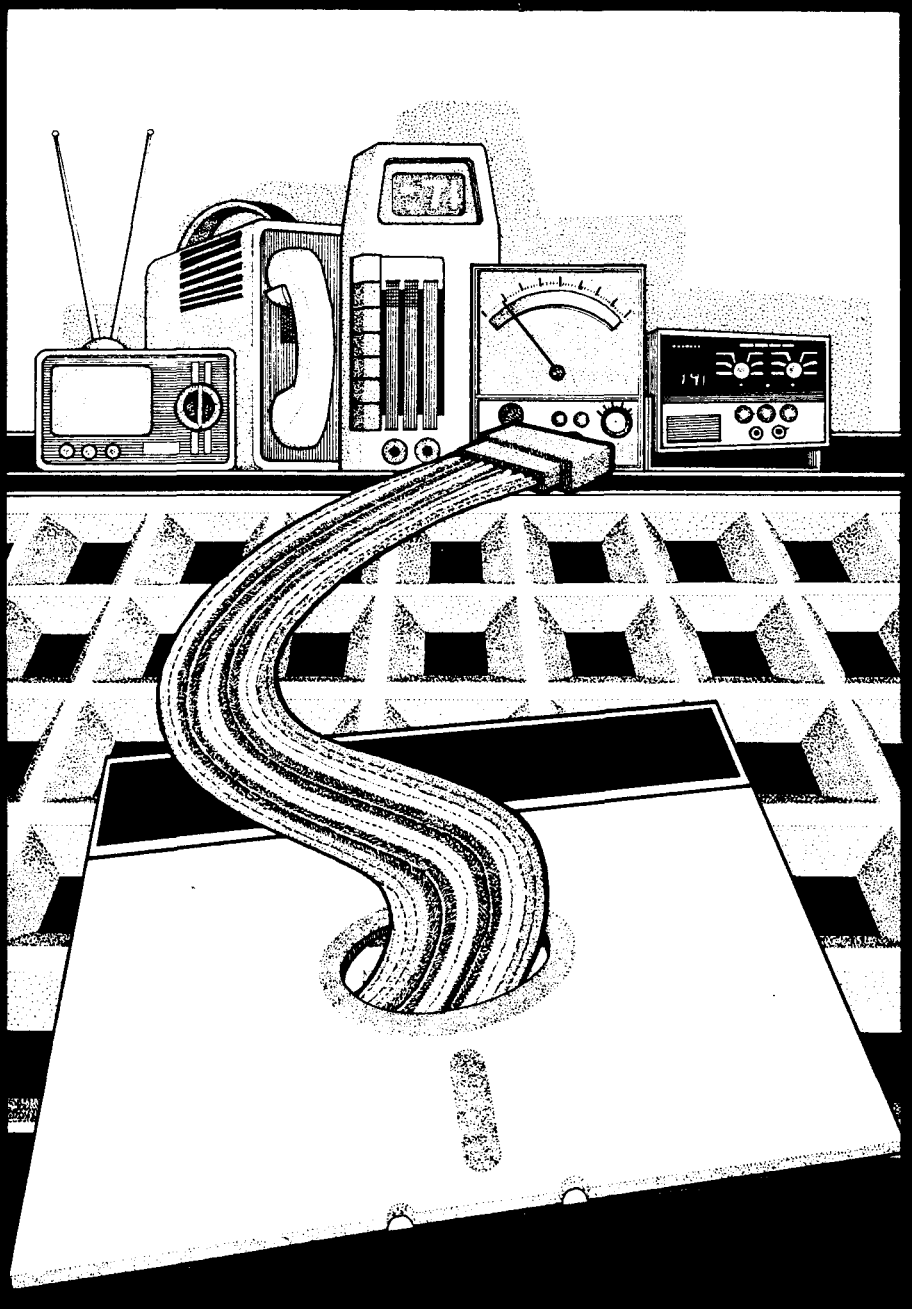
22009

TRS-80® Models I, III, & Color Computer Interfacing Projects

By William Barden, Jr.



12.20

TRS-80® Models I, III, and Color Computer Interfacing Projects

William Barden, Jr., is a computer consultant specializing in small-configuration computer systems. He has nearly 20 years' experience in programming, systems analysis, and design on a variety of computer systems. Mr. Barden is a member of the Association for Computing Machinery and the IEEE. His major interest is home computer systems. Amateur radio, mathematical games, and sailing are among his other interests. Other Sams books by Mr. Barden are *How to Buy and Use Minicomputers & Microcomputers, How to Program Microcomputers, Microcomputers for Business Applications, The Z-80 Microcomputer Handbook, Z-80 Microcomputer Design Projects, Guidebook to Small Computers,* and *Microcomputer Math.*

TRS-80® Models I, III, and Color Computer Interfacing Projects

by

William Barden, Jr.

Howard W. Sams & Co., Inc.
4300 WEST 62ND ST. INDIANAPOLIS, INDIANA 46268 USA

FIRST EDITION
FIRST PRINTING—1983

International Standard Book Number: 0-672-22009-1
Library of Congress Catalog Card Number: 82-60876

Edited by: *Charlie Buffington*
Illustrated by: *Jill Martin*

Printed in the United States of America.

Preface

The Radio Shack TRS-80 Models I and III computers and the Color Computer can be interfaced easily and inexpensively to "real-world" devices such as telephones, audio inputs, temperature- and pressure-sensors, clock timers, and windspeed instruments. In this book we show you how with a selection of projects that include:

- Voice input and synthesis
- Light detectors (two types)
- Thermometers (two types)
- Pressure sensor
- Musical note generator
- General-purpose input/output board with 24 discrete lines
- Anemometer to measure windspeed
- Tachometer "wand"
- Tachometer-input sensing by reed switches
- Serial-out driver for the cassette port
- Data communications plugboard
- Half-year clock
- Joysticks for Models I and III
- Many others

In some cases the projects require implementation of some special purpose hardware that connects to the computer input/output ports; in other cases, no special hardware is needed, as the computer systems themselves provide everything that is necessary.

We've tried to make the projects as foolproof as possible. Many can be assembled with two or three integrated circuits, mounted on a simple project board. Detailed construction information is provided for each design.

In general, we've attempted to take a "systems" approach to the problem of interfacing. Too often there is a dichotomy between the hardware and software. We've all seen computer systems where an applications problem is solved by interfacing a custom-designed device that uses 315 (more or less!) integrated circuits; in this case, one suspects the designer has a strong hardware background. Similarly, there's the implementation

where everything is software-driven in a 2000-instruction (or close to it) hand-coded machine-language program using a single computer input/output line; the designer here is obviously from the software clan. We've tried to take the lazy way out by combining the best of hardware and software worlds. After all, the important thing is that the computer system can be used to accomplish some pretty spectacular real-world things. We've tried to do this in the most efficient possible fashion, using both hardware and software techniques.

Along with the projects, we've included quite a bit of descriptive material on the internal design of the Models I and III and Color Computer electronics. In part, at least, this book is a tutorial on the workings of the RS-232-C ports, the PIAs of the Color Computer, the cassette logic of the three systems, and the general input/output bus of each computer, as much as being a collection of interfacing projects.

Section I covers analog-to-digital conversion for the three computers, a way of reading in and processing electrical analog inputs representing real-world variables such as temperature, light intensity, and pressure. Chapter 1 describes the Color Computer analog-to-digital circuitry. Chapter 2 shows how the Color Computer can measure temperature and light intensity. Chapter 3 describes circuitry and special software that will let you record your voice or other audio and play it back through the Color Computer audio output, providing a means of low-cost speech synthesis. Chapter 4 describes an analog-to-digital converter for the Models I and III computers, which is very similar to the Color Computer circuitry. In this case, the unit will handle joystick inputs, and complete information is included for the Models I and III joysticks. Chapter 5 describes a second type of analog-to-digital converter for the Model III, one that is not as fast as the first, but simple and extremely accurate.

Section II describes the RS-232-C ports on the three systems. Chapter 6 defines RS-232-C standards in general and the circuitry in the Models I and III computers in particular. Chapter 7 presents a half-year clock project for the Color Computer that will run for hundreds of hours using battery power. Chapter 8 discusses the RS-232-C lines of the Models I and III, and offers a plugboard project that can be used to break out the lines and make RS-232-C interfacing much easier.

Section III is on the cassette output port of the Models I and III computers. Chapter 9 discusses general interface circuitry. Three projects in Chapters 10–12 include a musical tone generator, a telephone dialer, and a serial-out driver.

Section IV presents material on the cassette input of the Model III and the Color Computer. Chapter 13 shows how discrete inputs can easily be implemented on the two systems. Various switch-type inputs are discussed. Chapter 14 describes a low-frequency event counter for the Model III and the Color Computer. Chapter 15 features an anemometer or windspeed sensor that can be built and the software to drive it.

Section V deals with the system input/output bus of the Color Computer and the Models I and III computers. Chapter 16 discusses internal input/output bus logic of the Color Computer, followed by a complete design for a general purpose input/output board in Chapter 17. Chapter 18 describes the Models I and III internal I/O bus, followed by a general-purpose I/O board in Chapter 19.

The concluding section, Section VI, goes into switches and transducers for all three systems and ways to interface them. Chapter 20 provides instructions on general interfacing methods. Chapter 21 describes various types of switches and sensing devices. Chapter 22 offers simple operational amplifiers for analog-to-digital circuitry. Chapter 23 contains a number of analog-to-digital projects, including a solar cell light detector, thermistor, LM334 temperature sensor, dc motor generator, tachometer wand, and pressure sensor.

It's easy to interface to the real world. This book will show you how!

WILLIAM BARDEN, JR.

To Forrest Mims III, Dan Likins, and Dennis Kitsz and their excellent hardware concepts

Note:
Printed circuit boards and parts kits for many of the circuits shown in this book are available from:

William Barden, Jr., Inc.
P.O. Box 3568
Mission Viejo, CA 92692

Contents

Section II

Section III

Section VI

Chapter 20

Chapter 21

Chapter 22

Chapter 23

Section I

Analog-to-Digital Conversion for the Models I and III and the Color Computer

chapter 1

Color Computer Analog-to-Digital Channels

Many physical quantities reflecting a real-world state or condition may be represented by an electrical *analog* of voltage, resistance, or current. For example, a *thermistor* changes resistance in accordance with temperature. Certain types of crystals generate a voltage when stressed, hence we have crystal microphones that produce a voltage output in step with sound input. Photoresistors change resistance when they are exposed to varying light intensities.

One of the problems with many types of *transducers* such as these is that they do not operate in a *linear* manner. Changes in the physical quantity to be measured or monitored do not produce corresponding equivalent changes in the electrical property over a wide range. Manufacturers strive to maintain linearity in operation of such devices. As a result, transducers become expensive. Fortunately, we can completely bypass linearity problems in the analog-to-digital (a/d or adc) inputs, because we can easily convert input values to the corresponding physical values by a table of values. As a result, we can use many "garden variety" devices as transducers.

Another powerful aspect of computers is that they enable us to do more than read instantaneous input values. We can use computers as *data acquisition* devices. Inputs can be *sampled* many times a second and then the information can be stored in memory, on cassette, or on disk. We can retrieve the input data as often as required and process it in any way we wish.

In the remainder of this chapter and the next two chapters, we discuss analog-to-digital conversion on the Color Computer. Much of the following material also is applicable to the Models I and III, which we cover in detail in Chapters 4 and 5.

ANALOG-TO-DIGITAL INPUTS

The Color Computer has an amazing amount of circuitry built into it for the price. One of its most interesting features is the joystick interface, which allows the user to control screen cursor position by the use of two *joystick* controls. Actually what we're seeing in this use of the joysticks is one of the most mundane aspects of the circuitry. The computer joystick inputs can be used instead for reading in temperature, amount of light in a room, or other real-world physical quantities, and all that is needed are a few inexpensive components. The four channels of data coming into the Color Computer make the Color Computer a data acquisition system for storage and processing of all types of real-world data.

In this chapter we investigate the Color Computer hardware that handles the joystick inputs and the software that drives the input electronics. In the following chapter you'll learn how to implement some simple analog-to-digital projects.

JOYSTICK CIRCUITRY

Let's first take a look at the hardware. Fig. 1–1 shows a block diagram of the Color Computer joystick circuitry. There are two joysticks, each one having an X and Y channel. These connect to a *data selector*, which selects one of the four channels. The output of the data selector goes to a comparator. The second input to the comparator is from the output of an analog-to-digital converter driven by six lines from a PIA (peripheral interface adapter). The output from the comparator goes to one input line of a second PIA.

A more detailed diagram of the electronics is shown in Fig. 1–2. Parts placement on this diagram will correspond to the functional blocks of Fig. 1–1. We'll use Fig. 1–2 in the discussion following and explain some of the parts for those of you who are software types.

Joysticks

The joysticks are simply variable resistors or potentiometers as shown in Fig. 1–3. Move the joystick control in the up/down direction only and

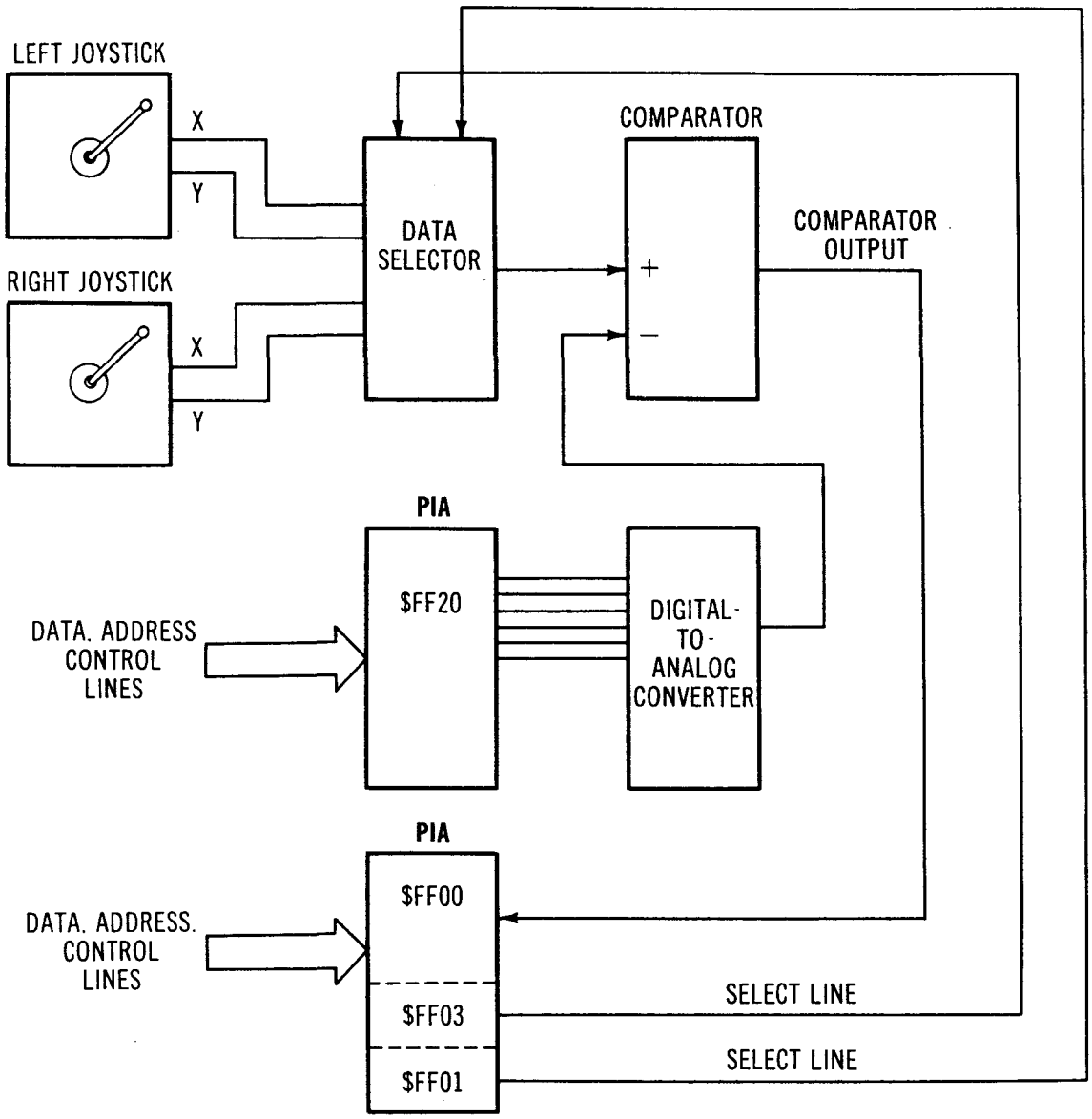


Fig. 1–1. Color Computer joystick circuitry block diagram.

the Y potentiometer wiper moves across the potentiometer, varying the resistance from 0 to 100 kΩ. Move the joystick control in the right/left direction only and the X potentiometer wiper varies the resistance from 0 to 100 kΩ. Every position of the joystick can be translated into X and Y components, with resulting X and Y positions and resulting resistance values. As both potentiometers are connected between +5 volts (from the Color Computer) and ground, the voltage output to the X and Y channels varies between approximately 0 volts (up or left position) and +5 volts (down or right position). A switch on each joystick connects another input pin (pin 4) to ground when it is pressed.

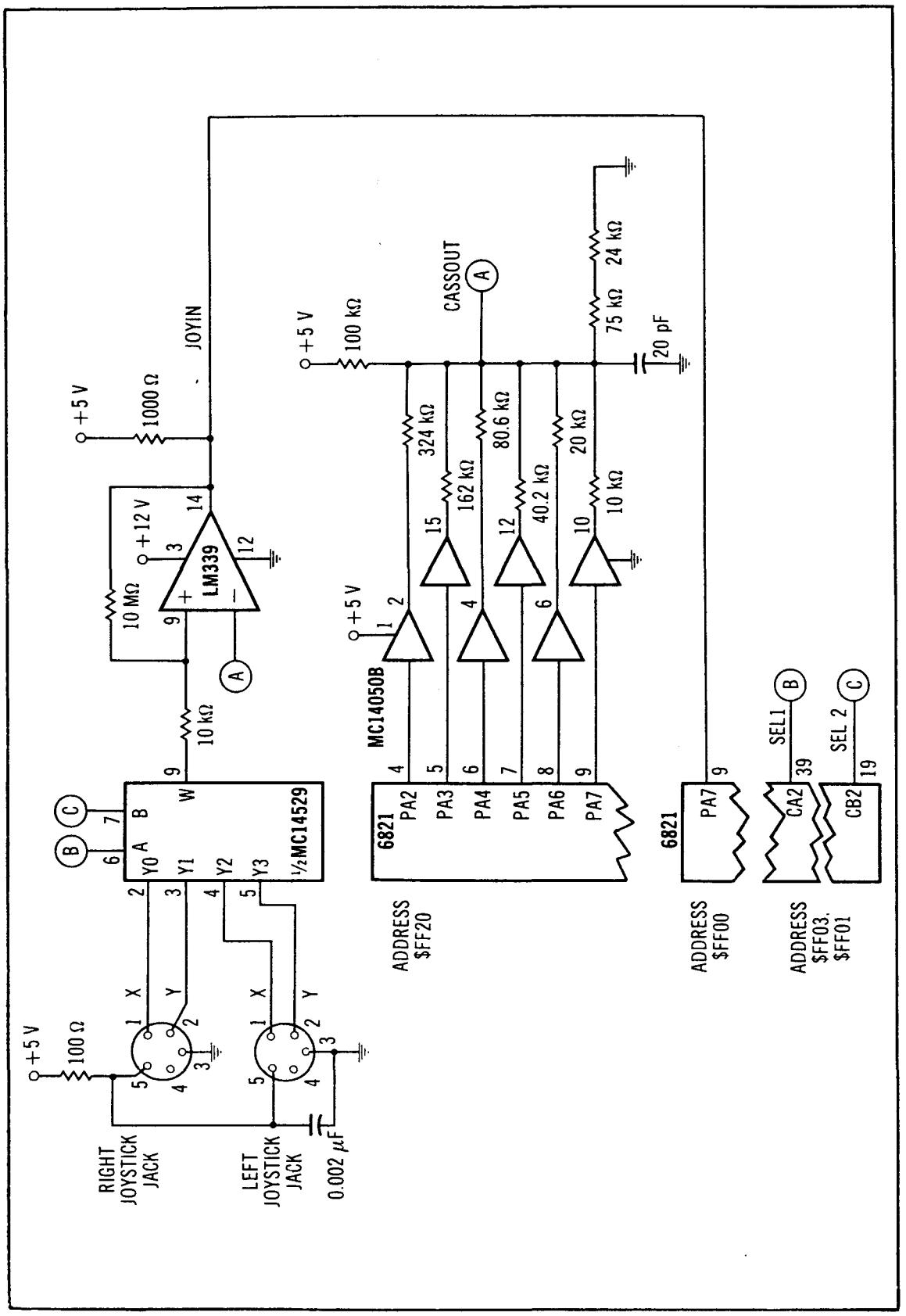


Fig. 1–2. Detailed joystick circuitry.

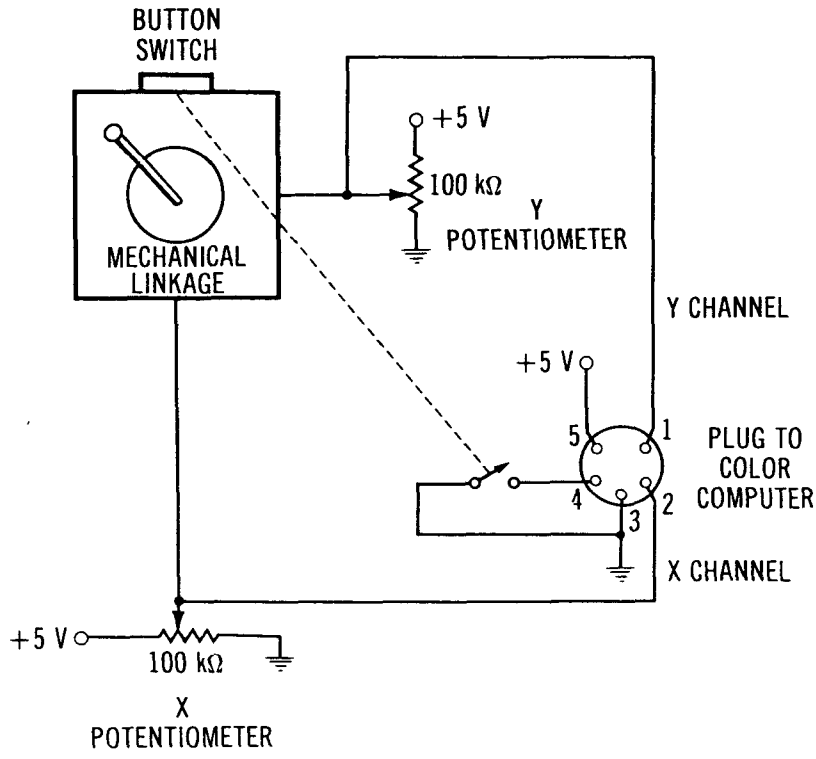


Fig. 1–3. Joystick electronics.

Data Selector

The MC14529 is an analog switch. This device selects one of four input channels and routes it to output W (Fig. 1–2). The signal is not otherwise processed as it passes to the LM339 comparator. So the voltage input from one of the channels is fed unchanged to the LM339 positive (+) input.

The selection of the channel is determined by two *select* lines, SEL1 and SEL2. These lines are outputs from the second 6821 PIA. More about the PIAs later. For now, simply note that we can switch the channels easily by changing SEL1/SEL2 to 00, 01, 10, or 11.

The Comparator

The LM339 is a common voltage-comparator device that compares two inputs. The inputs are two dc levels that can vary from 0 to some positive voltage. The output is either on or off. In this case, the two inputs will vary from 0 to +5 volts (approximately), and the output will be either 0 (+ input greater than − input) or +5 volts (+ input less

than − input). The output, then, represents a binary 0 or 1 and reflects the comparison of a joystick voltage and a second input called CASSOUT.

Digital-to-Analog Converter

The six buffers of the MC14050B and the resistor network make up a digital-to-analog converter. The d/a (or dac) takes the six lines labeled PA7 through PA2 and converts the binary value of 000000 through 111111 into a corresponding voltage of 0 volts through +5 volts dc. Since there are 64 separate values represented over this range (111111 is 63), the voltage represented will be in steps of 5/64 volts (approximately) from 0 volts, i.e., 5/64, 10/64, 15/64, . . . up to 320/64, or 5 volts.

The method used for this conversion is a resistor-network, voltage-divider-type conversion, where each resistor produces a weighted voltage. The output of each of the MC14050B buffers is either 0 volts or +5 volts (approximately). If the output of a buffer is 0 volts, the resistor associated with the buffer can be considered to be tied to ground; if the output is +5 volts, the resistor can be considered to be tied to +5 volts. The resulting resistor network for a typical configuration is shown in Fig. 1–4. The output is the total voltage from ground to the output point. Table 1–1 shows the approximate output voltages for the range of input values.

The PIAs

The PIA is Motorola's *peripheral interface adapter*, basically a 20-line device in which most lines can be programmed to be an input or output. In the standard Color Computer configuration, the PIA lines feeding the digital-to-analog converter are assigned address $FF20, the PIA lines selecting the channel by the data selector are assigned address $FF01, and the PIA line for JOYIN is assigned address $FF00. There are other lines involved with the PIAs: lines to read the keyboard, lines to handle RS-232 communication, and so forth, but the lines pertaining to the joystick inputs are shown in Fig. 1–5.

Each set of lines is *memory-mapped* in the Color Computer. A PEEK (65280) can be used to read the JOYIN bit, while a POKE 65312 will output a value to the digital-to-analog converter.

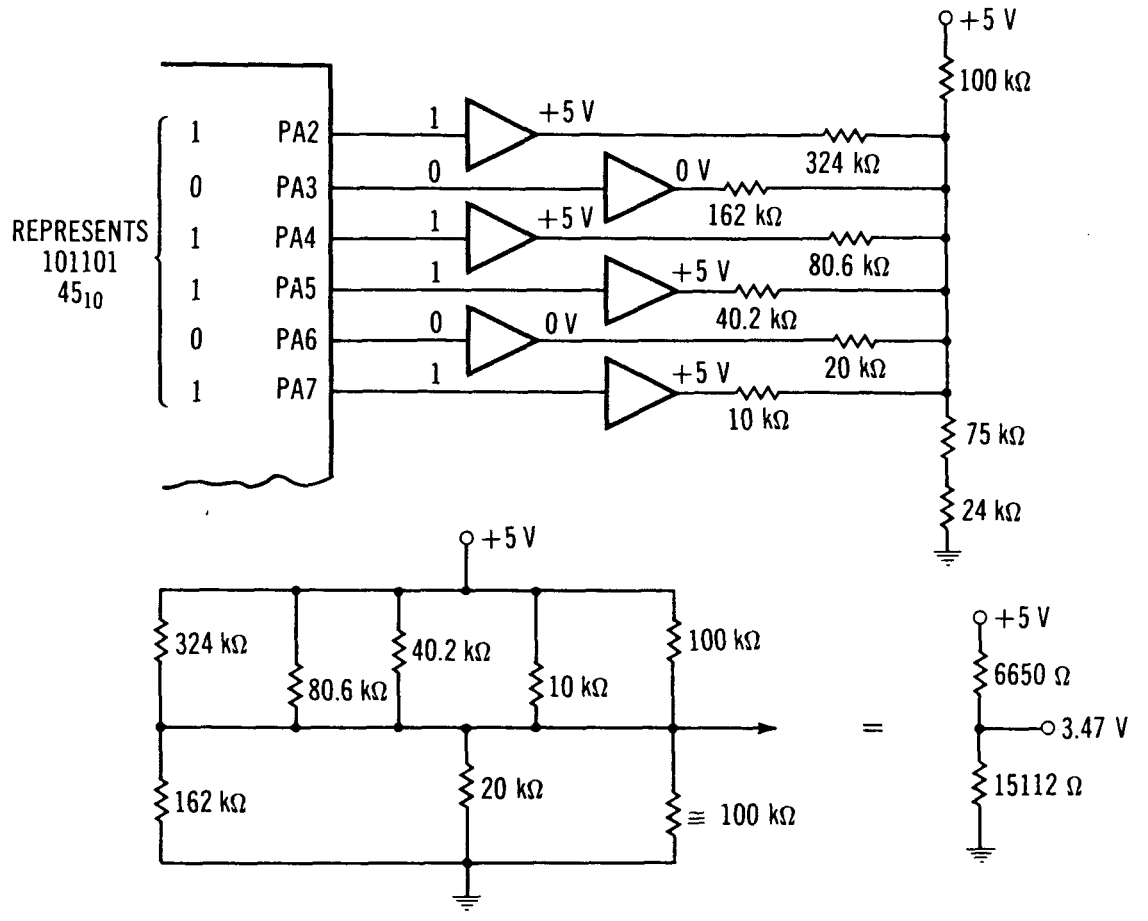


Fig. 1–4. Analog-to-digital converter operation.

JOYSTICK SOFTWARE

From here on in, the problem is simply a matter of programming.

Successive Approximation Methods of Analog-to-Digital Conversion

The algorithm for finding the position of either joystick goes something like this:

1. Select the joystick and X/Y channel by outputting to the SEL1/SEL2 lines. To select the right joystick and X, for example, a 0 must be output to bit 3 of address 65283 and a 1 output to bit 3 of address 65281.
2. The input from the joystick is now at the positive (+) pin of the comparator. Assuming that we aren't playing a hot game of Space

Table 1–1. Analog-to-Digital Converter Voltages

Input Value	Output	Input Value	Output
0	0.230	32	2.53
1	0.302	33	2.61
2	0.373	34	2.68
3	0.444	35	2.75
4	0.517	36	2.82
5	0.588	37	2.89
6	0.659	38	2.96
7	0.731	39	3.04
8	0.805	40	3.11
9	0.876	41	3.18
10	0.947	42	3.25
11	1.01	43	3.32
12	1.09	44	3.40
13	1.16	45	3.47
14	1.23	46	3.54
15	1.30	47	3.61
16	1.38	48	3.69
17	1.45	49	3.76
18	1.52	50	3.83
19	1.59	51	3.90
20	1.67	52	3.98
21	1.74	53	4.05
22	1.81	54	4.12
23	1.88	55	4.19
24	1.95	56	4.26
25	2.03	57	4.34
26	2.10	58	4.41
27	2.17	59	4.48
28	2.24	60	4.55
29	2.31	61	4.62
30	2.38	62	4.69
31	2.46	63	4.76

Invaders, that input should remain relatively constant for some period of time, although in normal use it could be fluctuating from 0 to +5 volts in a quarter of a second or less.

3. Output a value of 100000 (32, or about 2.5 volts) to the analog-to-digital converter by doing a POKE 65312,128.
4. Look at the output of the comparator by doing a PEEK (65280) and testing bit 7 by an AND 128. If the output is a 0, the channel value is less than the a/d value. In this case, take half of the

remaining range (010000, 16, or about 1.38 volts) and try again. If the output is a 1, the channel value is greater than the a/d value. In this case, take half of the remaining range (110000, 48, or 3.69 volts) and try again.

5. Repeat this process for six tries. For each try, take one-half of the remaining range and try again. At the end of the six tries, take the value most recently output; it will be within 5/64 of the actual voltage.

Savvy readers will recognize this algorithm as our old friend, the binary search. In this case a binary search was used to converge on the X or Y input voltage by a *successive approximation.*

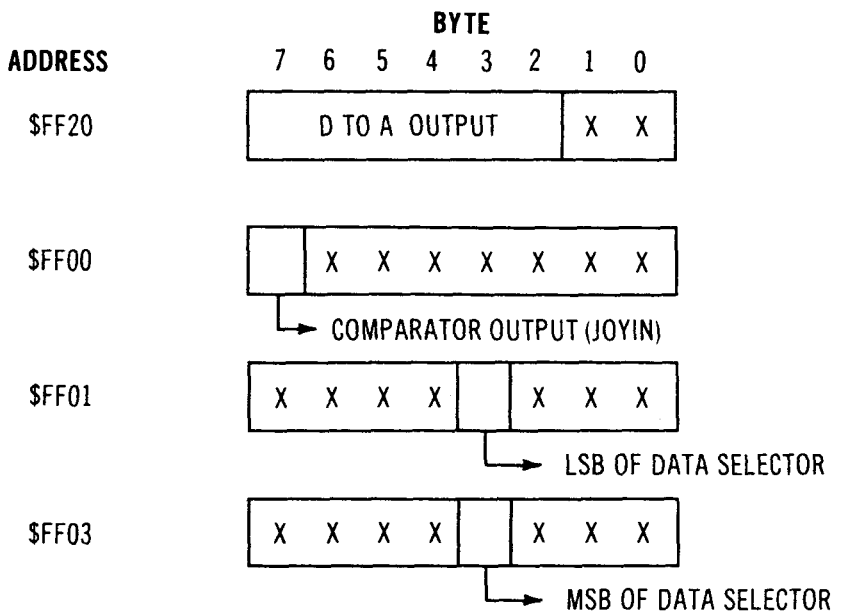


Fig. 1–5. PIA addresses and bit configuration for joysticks.

A Slow BASIC Example

To show you that this method does work, run the BASIC program shown in Fig. 1–6. This program zeros in on the X channel of the right joystick. Move the joystick and the program will report back on the new X position for each iteration.

BASIC Joystick Commands

The JOYSTK command in Color BASIC accomplishes the same function as the program above. The format of JOYSTK is

JOYSTK (j)

where j is 0 for right joystick, X; 1 for right joystick, Y; 2 for left joystick, X; and 3 for left joystick, Y. JOYSTK(0) must be executed before JOYSTK(1), (2), or (3) can be returned.

As with other BASIC operations, there is a limit to how fast JOYSTK can be performed. Assuming that we want to read the X/Y coordinates of one joystick (see Fig. 1–7), the speed of operation is about 23 reads per second. This is not too bad, but does prevent such things as smooth plotting of points on the screen for rapid cursor movement, as in Fig. 1–8.

Fig. 1–6. BASIC program for a/d conversion of right joystick X.

```
90 REM SELECT RIGHT,X
100 A=PEEK(65283)
110 A=A AND 247
120 POKE 65283,A
130 A=PEEK(65281)
140 A=A AND 247 OR 4
150 POKE 65281,A
160 REM SETUP VALUE, DELTA
170 V=128: D=64
175 BINARY SEARCH HERE
180 POKE 65312,V
190 A=PEEK(65280)
200 A=A AND 128
210 IF A=0 THEN V=V-D ELSE V=V+D
220 D=D/2
230 IF D<>1 THEN GOTO 180
235 REM NOW GET 6 LS BITS
240 V=V AND 252
250 V=V/4
260 PRINT V
270 GOTO 100
```

Machine Language

The answer to a faster read of the joysticks, as the reader might suspect, is in 6809 machine language. There are two driver subroutines in Color BASIC associated with the joysticks, one to select the joystick channel and one to read in all four channels into four page 0 locations.

The Select Joystick subroutine in Color BASIC is located at location $A9A2. It is entered with the B register containing the joystick channel number 0 through 3. Fig. 1–9 shows the disassembled code. User stack

Fig. 1–7. BASIC JOYSTK test case 1.

```
100 REM TYPICAL JOYSTICK PROGRAM
110 A=JOYSTK(0)
120 PRINT JOYSTK(0),JOYSTK(1)
130 I=I+1
140 GOTO 120
```

Fig. 1–8. BASIC JOYSTK test case 2.

```
100 REM PROGRAM TO PLOT POINTS FROM JOYSTICK
110 PMODE 4,1: PCLS: SCREEN 1,0
120 PSET (JOYSTK(0)*4,JOYSTK(1)*3)
130 GOTO 120
```

```
>>DIS $A9A2,$A9B2
A9A2   CE FF01     LDU  #FF01      $FF01 TO U
A9A5   8D 00       BSR  A9A7       DO $A9A7 TWICE
A9A7   A6 C4       LDA  0,U        READ $FF01 PIA
A9A9   84 F7       ANDA #F7        RESET SELECT BIT
A9AB   57          ASRB            SHIFT OUT BIT TO C
A9AC   24 02       BCC  A9B0       GO IF SELECT BIT = 0
A9AE   8A 08       ORA  #08        SELECT BIT = 1
A9B0   A7 C1       STA  U++        STORE IN $FF01 PIA. BUMP TO $FF0 <
A9B2   39          RTS             RETURN

>>
```

Fig. 1–9. Select joystick subroutine.

pointer register U is first loaded with $FF01. A following BSR to $A9A7 performs the $A9A7 code twice. A is first loaded with the current configuration of the $FF01 PIA bits. An AND of $F7 resets the select bits. The ASRB shifts the least-significant bit of the B register into the carry flag. If this is a 1-bit, an OR of 8 sets the select bit. The STA ,U + + stores SEL2, and increments the user stack pointer by two so that it holds $FF03. The RTS returns to A9A7, where the same operation is repeated for the second select bit in PIA $FF03.

The main code for the joysticks is at $A9E0 (Fig. 1–10). This code is entered without parameters and stores the values of channels 0 through 3 into page 0 locations $15A, $15B, $15C, and $15D. The X index register is first loaded with one more than the address of joystick variable storage. B is loaded with a loop count of 3. The code from $A9E5 through $AA17

```
DIS $A9E0,$AA19
A9E0   8E 015E     LDX  #015E      POINT TO STORAGE +1
A9E3   C6 03       LDB  #03        LOOP COUNT -3 TO 0
A9E5   86 0A       LDA  #0A        # OF RETRIES
A9E7   ED E3       STD  --S        SAVE # TIMES. RETRIES
A9E9   8D B7       BSR  A9A2       SELECT JOYSTICK 3-0
A9EB   CC 4080     LDD  #4080      $40 - A. $80 - B = DELTA. START
A9EE   A7 E2       STA  -S         SAVE
A9F0   CA 02       ORB  #02        ?? RS-232 OUT
A9F2   F7 FF20     STB  FF20       CURRENT TRY TO D/A
A9F5   C8 02       EORB #02        FLIP BIT?
A9F7   B6 FF00     LDA  FF00       GET JOYIN
A9FA   2B 03       BMI  A9FF       GO IF 1
A9FC   E0 E4       SUBB 0,S        SUBTRACT DELTA
A9FE   8C EBE4     CMPX #EBE4      BYTES 2. 3 = ADD B 0.5
AA01   A6 E0       LDA  S+         GET DELTA
AA03   44          LSRA            FIND NEXT DELTA
AA04   81 01       CMPA #01        TEST FOR END
AA06   26 E6       BNE  A9EE       GO IF NOT DELTA OF 1
AA08   54          LSRB            ALIGN FINAL VALUE TO 00XXXXXX
AA09   54          LSRB
AA0A   E1 1F       CMPB -01,X      GET LAST VALUE
AA0C   27 04       BEQ  AA12       GO IF EQUAL
AA0E   6A E4       DEC  0,S        NOT EQUAL-RETRY
AA10   26 D9       BNE  A9EB       TRY 10 TIMES
AA12   E7 82       STB  -X         STORE VALUE IN STORAGE
AA14   EC E1       LDD  S++        GET COUNT
AA16   5A          DECB            DECREMENT COUNT (#)
AA17   2A CC       BPL  A9E5       GO IF NOT 4 JOYSTICKS
AA19   39          RTS             RETURN

>>
```

Fig. 1–10. Read joysticks subroutine.

is the outer loop. For each of four passes, a channel value is found and put into a joystick variable.

Outer loop—A is loaded with 10. This is the number of retries for the joystick value. If the same value is not found a second time, up to 10 tries are made to find a matching value. The number of times in B and the number of retries are stored in the stack by the STD. A call is then made to $A9A2 to select the current joystick channel. This corresponds to the loop count of 3 to 0 in B. The code from $A9EB through $AA10 is inner loop 1. It finds the value of the channel. At the end of this loop ($AA12) the value is stored in the variable storage area by STB ,X−. This auto-decrement causes X to point to the next lower value before the store occurs. Next, the count in B and the number of retries in A are retrieved by the LDD, the count is decremented, and a BPL causes a loop back to $A9E5 if the count is not −1.

Inner loop 1—The code from $A9EB through $AA10 is the inner loop that finds the value for the current channel. Within this code is inner loop 2, from $A9EE through $AA06, which actually does the binary search. $40 is loaded into A and $80 into B to start the search. $80 is 100000XX for the initial value of 32, while $40 is 010000XX for the *delta*, the size of the remaining range.

At the end of the binary search at $AA08, the final PIA-format value is in B. This value is aligned to the right by the two LSRBs to represent a binary value of 0 through 63. It is then compared with the previous value. If these are the same, a branch is made to $AA12 to store the value in the outer loop. If the value is different, the number of retries is decremented and, if not 0, another binary search is done by a branch to $A9EB.

Inner loop 2—The code from $A9EE through $AA06 is the binary search to find the channel value. A (the delta) is saved in the stack. The current value in B is then output to the analog-to-digital converter by the STB $FF20. The output of the comparator is read by the LDA $FF00. If this value is a 1, the delta is added to the current value; if it is a zero, the delta is subtracted from the current value. The next value is then found by retrieving the delta from the stack and shifting it right one bit position. If the result is 1 the smallest delta has been processed and B holds the final value. If the next delta is not one, a branch back to $A9EE goes to the next iteration in the search.

This subroutine can be used for high-speed processing of the joystick position from other assembly language code. Best case is when the joystick position is fixed and only one retry is necessary (for comparison). A test program from BASIC indicates that it takes about 1½ ms for each set of four values. To find only the X channel of joystick 0, call location $A9E5 with B=0 and X pointing to $15A. In this case, the time should be about 400 μs, although we haven't verified this.

USING THE JOYSTICK INPUTS FOR OTHER USES

As we've seen from the above discussion, we have a built-in set of four analog-to-digital channels in the Color Computer, channels in which the input voltage may be from 0 to +5 volts dc and in which data may be sampled at rates of up to 2500 samples per second for a single channel. There are many other uses to which such channels can be put.

In the next chapter you'll find two types of real-world inputs that utilize the capabilities of the Color Computer: a light detector and a thermometer to show you how simple this can be.

chapter **2**

Color Computer Analog-to-Digital Conversion Projects

In this chapter we utilize one of the four Color Computer joystick channels to perform analog-to-digital conversion of real-world quantities. This is an "introduction" to a/d techniques. Later in the book we look at some advanced applications.

STANDARD PLUG

As a first step, let's make a standard plug for the a/d inputs. The standard joystick plug is a 5-pin DIN male plug. Radio Shack carries it in most stores. Be certain to get a thin-walled type; the thicker plastic type will not fit into the jack. Use any 4-conductor wire or simply four single wires to connect to the DIN pins as shown in Fig. 2–1. If you'd like, you can add a fifth wire for the button switch, although we won't be talking about its use now.

Is it possible to zap the power supply by inadvertently connecting +5 volts to ground? Not too likely, as the +5 volt lead is connected internally via a 100-ohm resistor, with the resultant current only 50 mA, so use the cable leads with impunity.

A LIGHT DETECTOR

Our first application uses just two components attached to the right joystick X channel as shown in Fig. 2–2. The primary component is a cadmium sulfide (CdS) photocell. This device currently costs less than $1.50 at your local Radio Shack store. Its resistance is dependent upon

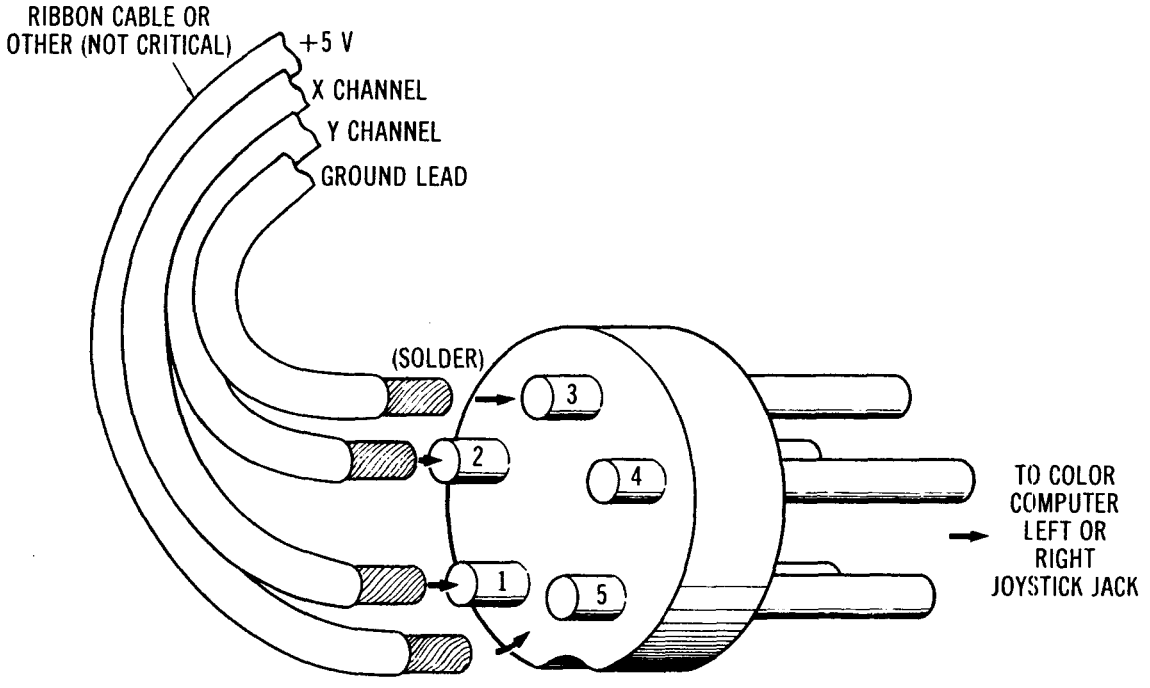


Fig. 2–1. Five-pin DIN standard plug.

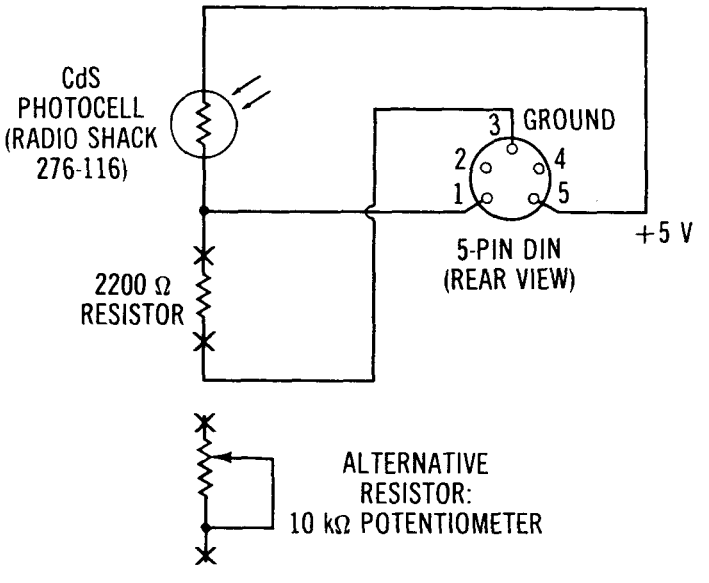


Fig. 2–2. Light detector components.

the amount of light striking it, and varies from about 5 megohms (5,000,000 ohms) in complete darkness (it was hard to read the ohmmeter here) to about 20 ohms in direct sunlight. Some other readings are shown in Table 2–1.

Needless to say, this is quite a wide range. For this example, we choose the normal house interior settings out of direct sunlight for a program that would determine when the room was adequately lighted, a range of about 500 to 5000 ohms. The input voltage to the 0 channel is given by:

$$V = R1/(R1 + R(cs)) \times 5 \text{ volts}$$

where R(cs) is the resistance of the photocell and R1 is the resistance of the second component (¼- or ⅛-watt carbon resistor, about 25 cents or less). For a midpoint R(cs) of 2750, R1 should be 2750. We choose the closest standard resistance value of 2200 ohms. Vary the resistance as required for the light conditions you are testing.

Table 2–1. Cadmium Sulfide Photocell Readings

Condition	Reading (ohms)
Facing sun	20
Sunlit outdoors	30
Overcast outdoors	50
Shaded outdoor screen	100
Inside house facing window	180
Inside house facing interior	830
Artificially lighted (well-lighted) room	2200
Interior of closet, swathed in old raccoon coat	5 M

The resistance could also be a potentiometer with the center and one outer pin tied together (actually a rheostat) to allow the use of this circuit with a variety of conditions. Both the fixed resistor or potentiometer are available from Radio Shack or other electronics parts stores.

Read channel 0 using the BASIC JOYSTK(0) command or by calling the joystick assembly language subroutine.

The light detector could be used for a number of things: electronic exposure meter for the darkroom, light level detector for artificial lighting, solar panel aiming (together with an output to control panel positioning), or burglar alarm (detectable drop in output as person walks by sensor).

In testing, the CdS photocell was sensitive enough to detect differences in clothing color and the whiteness of various types of paper. Many of the differences were not discernible by the human eye.

A THERMOMETER

The second application also uses two components (shown in Fig. 2–3). One of these is a thermistor. A thermistor's resistance varies in accord-

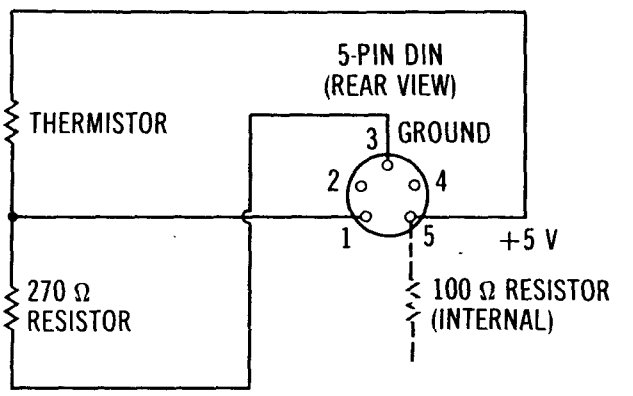


Fig. 2–3. Thermometer components.

ance with ambient temperature. We used a rather gross type of thermistor for this application, a replacement television thermistor. It has a resistance of about 120 ohms at 25 °C (77°F) and about 1.8 ohms at 65 °C (77°F). A thermistor of this type has a slow response to temperature changes, but it is inexpensive (less than $2.50). Better quality thermistors operable over a wide range of resistance values are available from a manufacturer's representative in larger cities and are on the order of $6.00 to $10.00. Choose one with a resistance of tens of thousands of ohms to reduce the effect of the 100-ohm resistor in series with the +5-volt pin.

A plot of the values obtained by reading JOYSTK(0) is shown in Fig. 2–4. Even with this unsophisticated thermistor, the temperature resolu-

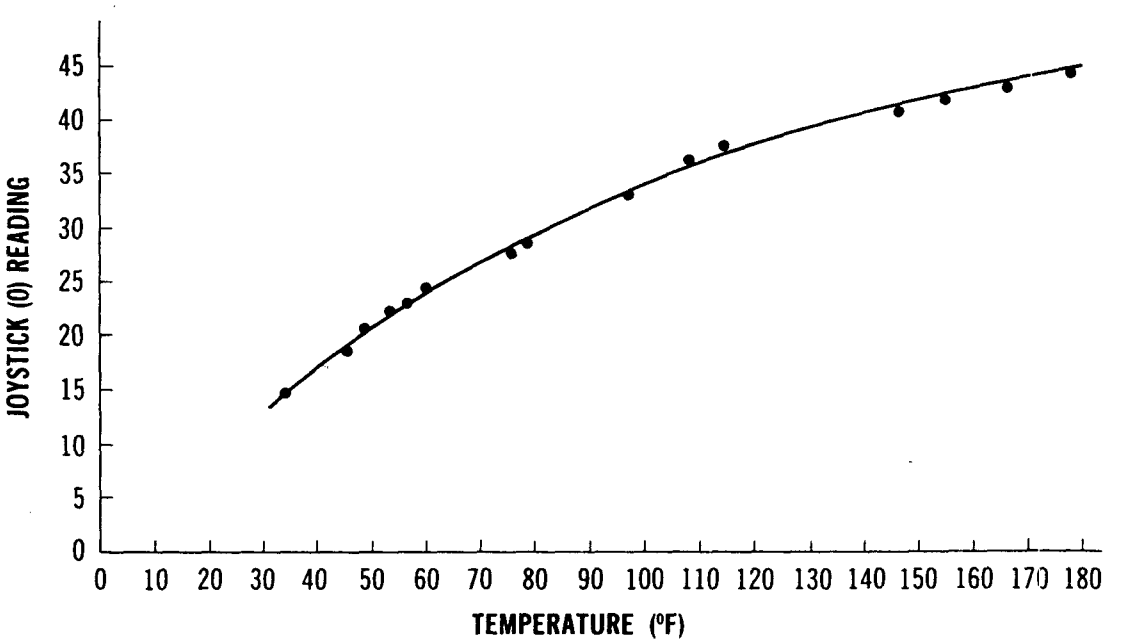


Fig. 2–4. Thermometer readings.

tion is 3 to 4 degrees at lower temperatures (effect of the 100-ohm resistance is less pronounced). This particular thermistor took several seconds to respond to changes in temperature.

It's quite easy to see how many interesting temperature applications could be implemented with this simple circuit: measurements of liquid temperature, fire detection, flow gauges (moving fluid cools a thermistor), a weather station, and the like.

OTHER APPLICATIONS

Don't hesitate to try other transducers with the joystick inputs. Anything that can resolve physical quantities into resistance or voltage can be measured by the Color Computer joystick inputs.

A small dc motor, for example, might be used in reverse, as a generator, when driven by anemometer-type wind cups; the motor would generate a voltage which could be applied directly across pins 3 (ground) and 1 (X input). Some amplification by a single transistor might be necessary.

A solar cell can be used in similar fashion. Tie its output directly to pins 1 and 3 to read voltage generated by sunlight.

Used with a microphone and small amplifier, the Color Computer could also be used as a sound detector for security systems.

A spring-loaded sliding potentiometer (only several dollars) could be used with a second resistor to provide an output for a scale to weigh elephants or letters. The same device can be used to convert linear movement into a form readable by the Color Computer. With two multiturn potentiometers (under $10 each), a little bit of cord, and a few pulleys, it's not too hard to see how an X/Y plotter can be constructed to enable manual digitization of two-dimensional drawings or patterns.

With a photocell, simple lens (for example, a partial microscope assembly), and some transistor amplification, it's possible to construct an automatic digitizer that would convert shades of grey into digital form for screen display.

Remove the stops from a linear taper potentiometer (not hard to do) and you have a resistor whose resistance value is an analog of compass heading or rotational position. Use a second resistor in the voltage-divider circuit we've been discussing.

The analog-to-digital channels are there, and the possibilities are endless! In Chapters 22 and 23 we show you some advanced techniques for measuring real-world physical quantities. Meanwhile, in the next chapter, there's a high-speed a/d technique for recording and playing back voice or other audio inputs.

chapter **3**

Voice Synthesis for the Color Computer

In this chapter we take three resistors, one inexpensive integrated circuit, two capacitors, one plug, one less than $2.00 microphone, and some software and create a way to record and play back your voice on the Color Computer! The quality will be better than Speak 'N Spell from Texas Instruments. The circuit here can take any sound input, digitize it, store it in memory, and play it back on request, all with the few components mentioned above! The catch is that we have only enough memory to record about 1⅓ seconds. However, by sacrificing some quality, you may be able to process the digitized data and expand the record time up to a factor of ten (13 seconds) or more. This project is primarily meant to show you how to easily capture the sounds, record them, and play them back. The improvements are up to you.

VOICE FREQUENCY PARAMETERS

Textbooks tell us that the range of hearing for humans extends from 20 to 20,000 Hz. In fact, most people are capable of hearing much less than the 20,000 Hz. How much can the upper frequency limit be reduced without losing significant voice quality? The average telephone circuit has an upper frequency limit of 3500 Hz, and, although it's not pleasant to listen to music while waiting for the airline reservations clerk to come on the line, voice suffers surprisingly little. Amateur radio operators, to reduce operating bandwidth, also restrict audio frequencies to 3000 Hz or so without experiencing too much voice degradation.

If we are to store and play back acceptable voice sound, therefore, we had best design circuits capable of playing back up to 3500 Hz. But to play back 3500 Hz, we first have to capture the voice data. A fundamen-

tal rule of recording data digitally is that the sampling frequency must be twice the maximum frequency required under optimum conditions. Here, then, we must be capable of recording at rates of 7000 Hz or better. In other words, we must take the voice input and convert it to digital form at 7000 samples per second or better.

SAMPLING WITH AN ANALOG-TO-DIGITAL CONVERTER

To convert the voice signal to digital form, we need an analog-to-digital converter, or an adc. The adc will take the analog voice input and convert it to a digital value, as shown in Fig. 3–1. The larger the number of bits in

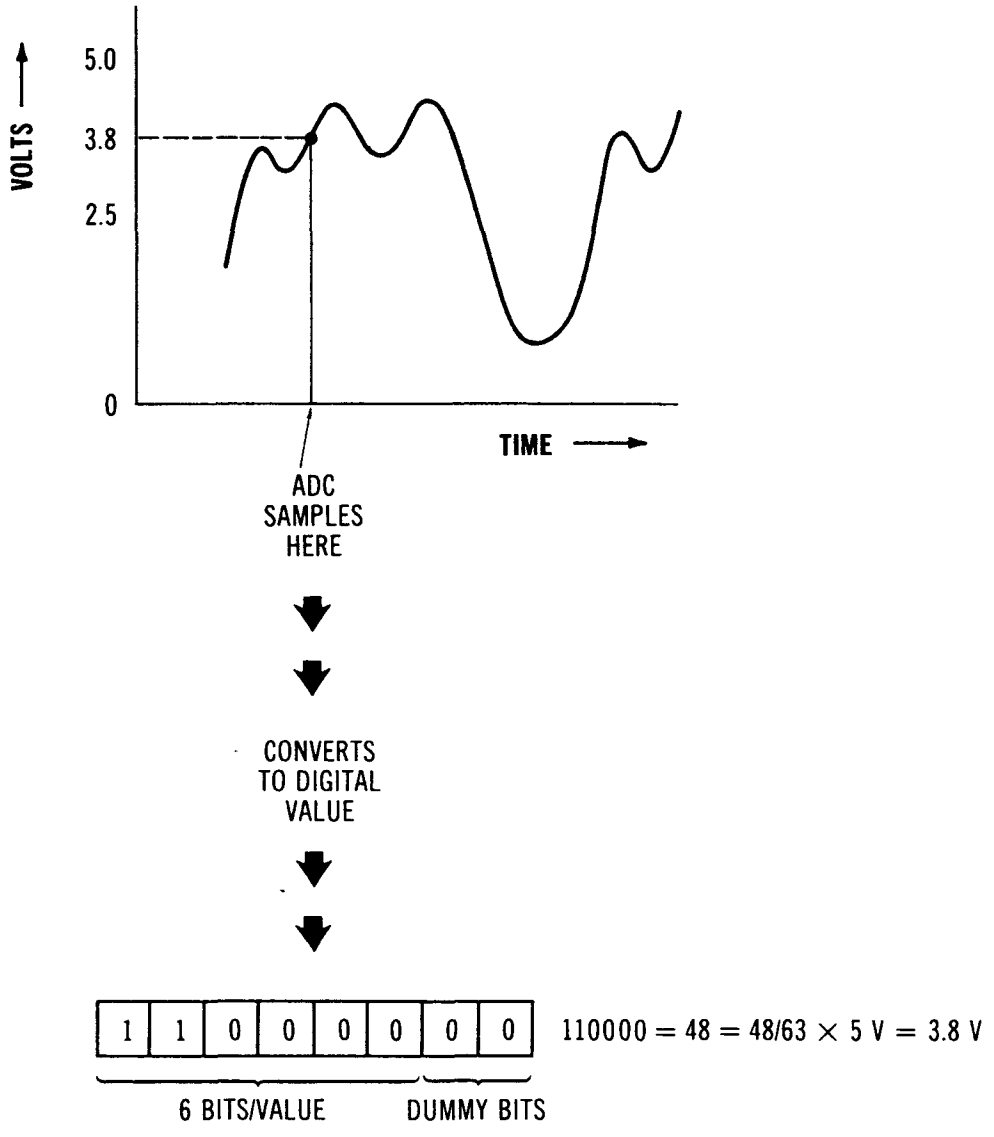


Fig. 3–1. Analog-to-digital converter action.

the sample, the finer the resolution in the digital representation of the analog value. If the adc is a 6-bit adc, for example, each digital value will be within 1/64th of the analog input value. A 5-bit adc will produce values within 1/32nd of the analog input value, and so on. When the digitized form of the input is replayed, the output waveform will approximate the original by a series of square waves. The greater the sampling rate and the resolution of the adc, the more the output will resemble the original, as shown in Fig. 3–2.

Let's assume that we're using a 6-bit adc. Although we could pack the data four values to three 8-bit bytes, it's less trouble and faster to simply put a 6-bit adc value in each byte, as shown in Fig. 3–3. A sampling rate of 7000 Hz, therefore, will eat up 7000 bytes of memory for each second of audio recorded.

A variety of ways to reduce the amount of storage required for audio data exist. Many are tied to the special electronics implemented in commercial voice synthesis integrated circuits. Texas Instruments, National,

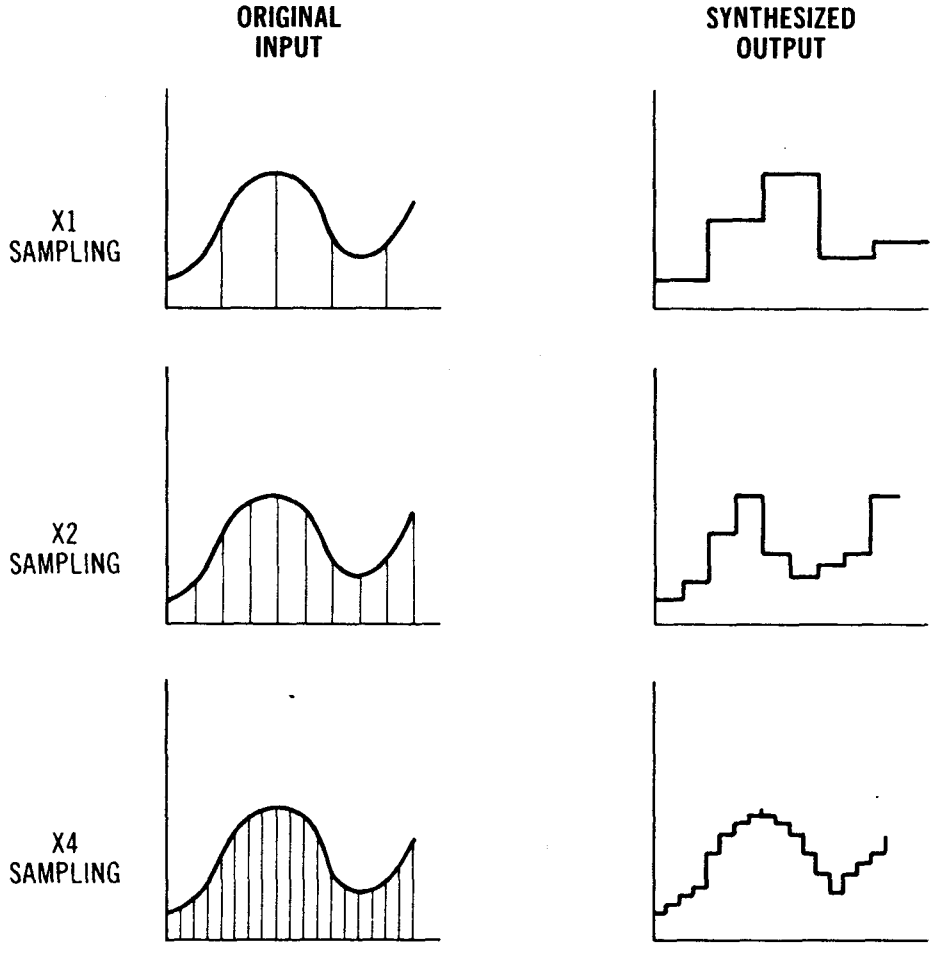


Fig. 3–2. A/d sampling rates.

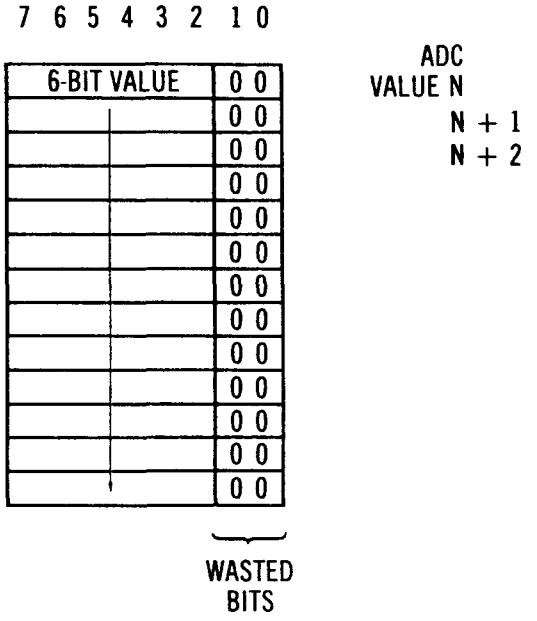


Fig. 3–3. Storage of adc samples.

and other companies are producing hardware that can synthesize voices that occupy only hundreds of bytes per second of speech. In these implementations, special processing software looks for silent periods, symmetry of waveforms, replication of patterns, does Fourier analysis of the waveforms, and uses other techniques. The result of the processing is a compact, specially encoded form of the voice data for the special hardware involved. We'll stick with our "brute force" approach for the time being, however, and discuss ways to cut down on the storage requirements later.

To play back digitized sounds, we need the complement of an adc, or a digital-to-analog converter (a dac). The dac will take the number of bits in each digitized value and produce a weighted voltage level for each one bit. If the data were originally captured by a 6-bit adc, then a 6-bit dac is required to reproduce each analog sample.

In theory, then, this brute force voice capture and synthesis is simple enough. Take an analog voltage input from the audio source, sample it 7000 times per second with an adc, store the adc output values in the memory of a digital computer, and then play back the values from memory with a dac. The process is shown in Fig. 3–4.

COLOR COMPUTER DAC AND ADC HARDWARE

If you have read Chapter 1, you know that the Color Computer has a built-in 6-bit dac and adc circuit. The dac is used to synthesize sine waves

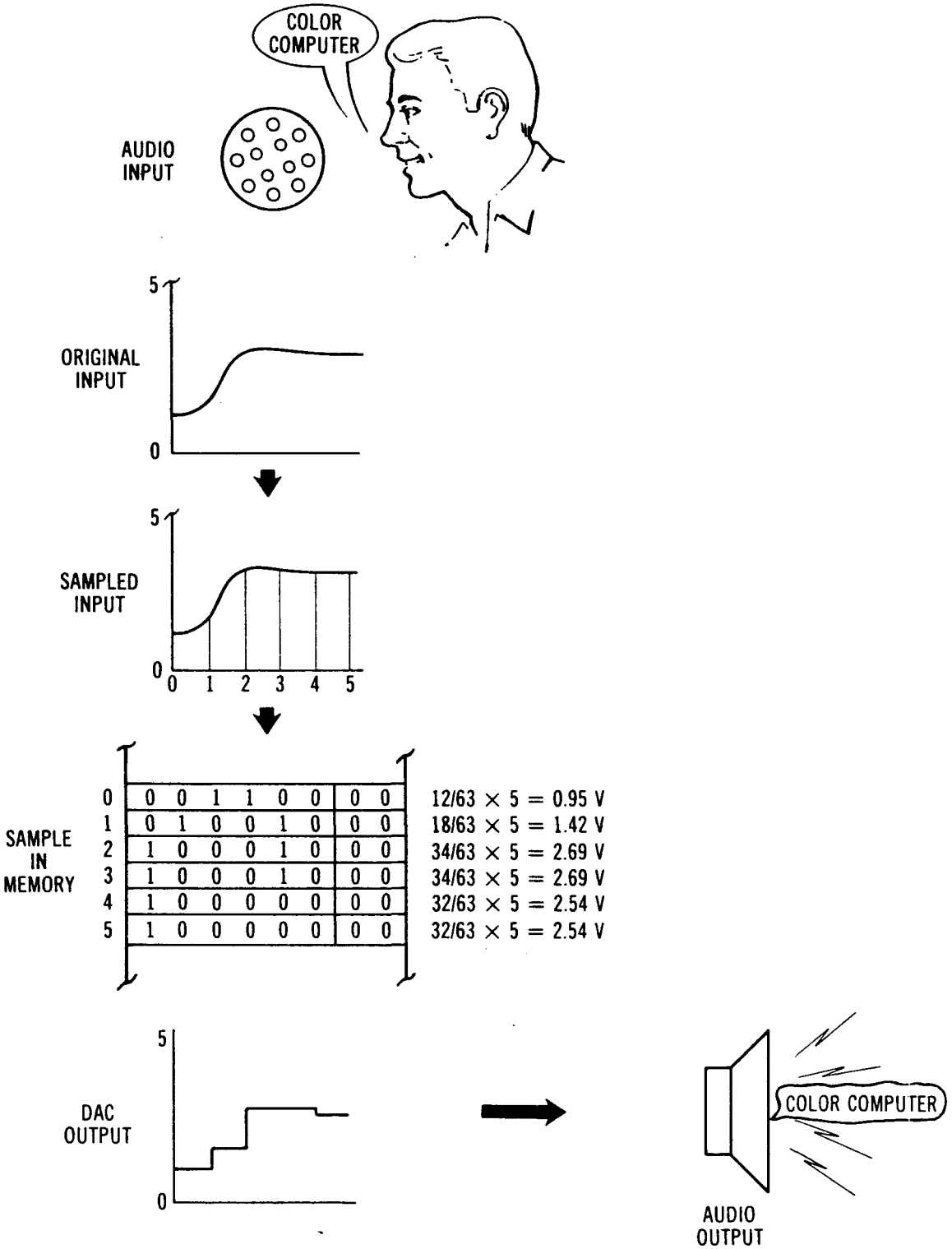


Fig. 3–4. Brute force voice synthesis.

for recording of cassette data and for generating musical tones. The adc exists partially in hardware and partially in software, and is used to perform analog-to-digital conversion on the joystick positions. Can we utilize the existing hardware of the Color Computer to implement sound

recording and playback? Let's review the Color Computer circuitry to see what's required.

Color Computer DAC

The dac (Fig. 3–5) is a 6-bit dac that will operate just as fast as data can be output to it. We have to use assembly language coding, however, to get decent output rates of thousands of times per second. BASIC would only allow several hundred operations per second, far too few for our purposes here.

Each 6-bit digitized value can be output to address $FF20, the PIA (peripheral interface adapter) for the dac. The value will be held in the PIA until overwritten by the next value. The output of the dac is very rapid (less than a microsecond), and it appears that the dac is no problem in our scheme. The output of the dac goes to an rf/audio modulator that converts the video to a television signal with audio. Audio from the dac will be heard through the audio circuits of the television used with the Color Computer.

Color Computer ADC

The adc is shown in Fig. 3–6. It uses a comparator IC that compares two inputs. The output of the comparator is either a one or zero depending upon whether the positive (+) input is lower or higher than the

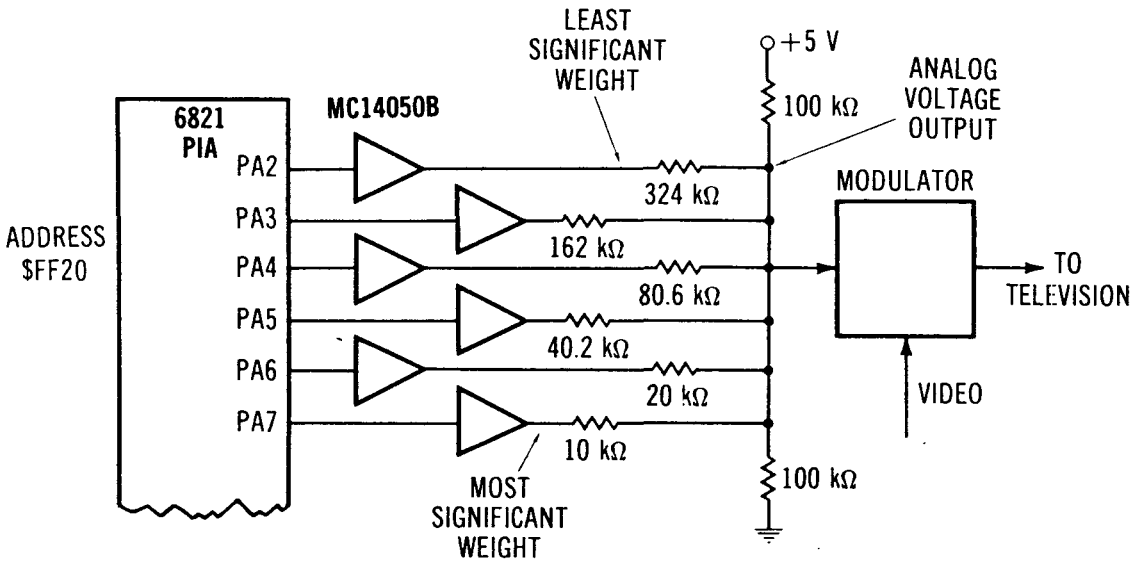


Fig. 3–5. Color Computer digital-to-analog circuitry.

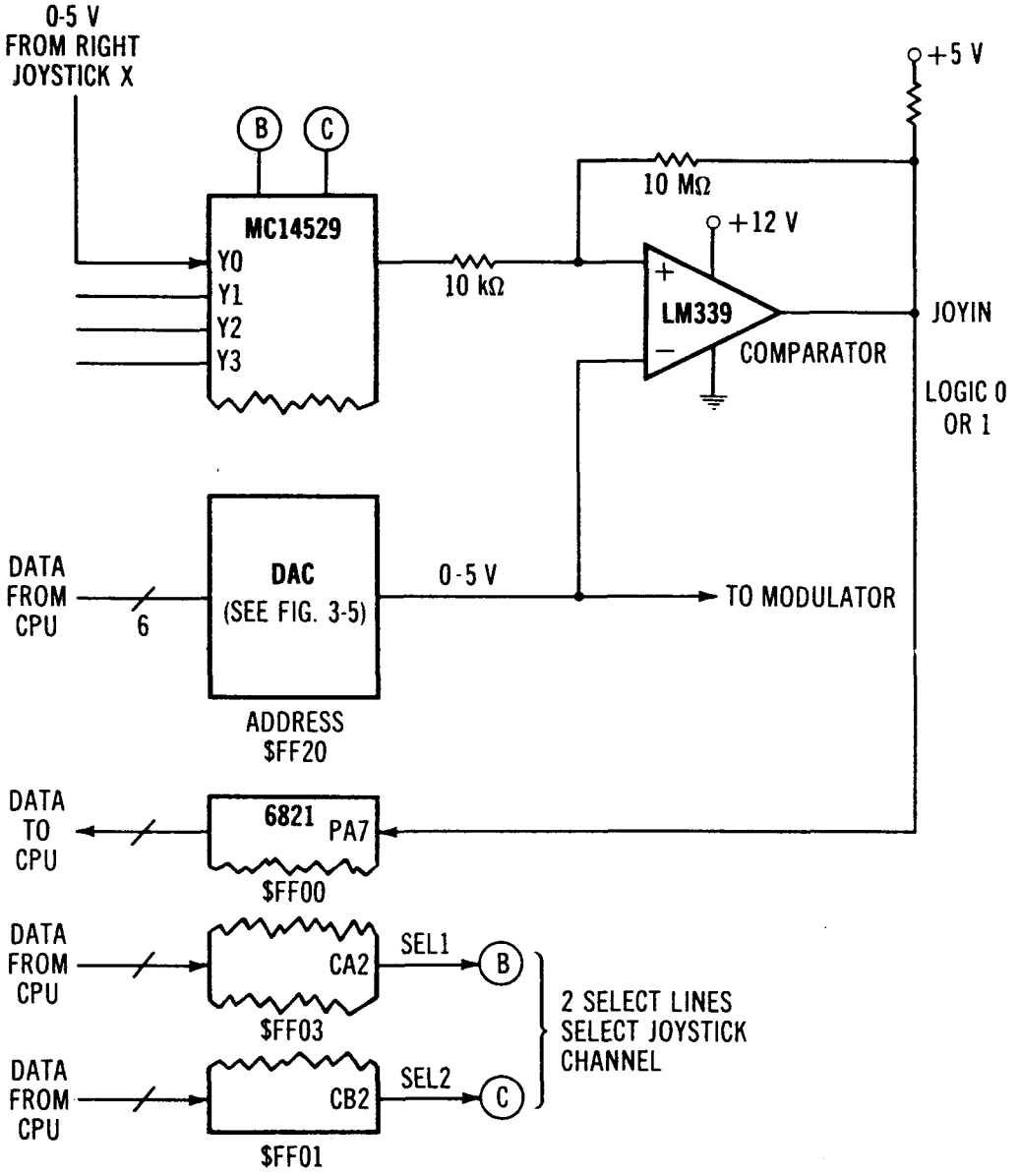


Fig. 3–6. Color Computer adc circuitry.

negative (−) input. The output of the comparator is extremely fast. The output can be tested by looking at bit 7 of the byte read from address **$FF00**.

One of the inputs to the comparator is from the external joystick input. This is a voltage level of 0 to +5 volts. The joystick input can be a voltage from the joystick potentiometer, or it can be any voltage in that range from any external device, including an audio amplifier. The second input to the comparator is from the dac and is also 0 to +5 volts. The analog-to-digital conversion is accomplished by rapidly changing the adc output and bracketing the joystick voltage, based on the comparator output.

Ah, there's the rub. "Rapidly changing" is fast in human terms, but may not be fast enough for our purposes. The analog-to-digital conversion is done in Color BASIC by a "successive approximation" software routine. This machine-language routine uses a type of binary search to converge on the joystick input value (see Chapter 1). It takes six steps to perform the conversion. However, conversions are done for all four joystick values, right joystick X and Y and left joystick X and Y. In addition, the routine compares the current value of each channel with the previous value until they match. All of this overhead allows sampling rates of only 600–700 samples per second, too slow for our needs. We need a high-speed adc!

VOICE SYNTHESIS SOFTWARE

INPUT Routine

The software for such an adc is shown in Fig. 3–7. It may not be the absolute fastest adc routine, but it does allow conversions of about 7733 samples per second. One technique used in the routine is linear coding without loops, eliminating the loop overhead.

If you're not acquainted with assembly language, the routine is not as imposing as it looks. The data on the extreme left of the listing is the hexadecimal location in memory where the instruction is found. The two columns following are the machine code bytes of the instruction in hexadecimal. The next column is simply a line number. The remaining four columns are the assembly language program with optional label, op code mnemonic, operands, and comments. The $ signs are used to signify a hexadecimal value. The # sign indicates that the value is an immediate value in the instruction, rather than a variable in memory.

There are six sections of the code that are virtually identical. Each one starts with STB $0FF20 and ends with BRA INPXXX. In each section a dac value is output by STB $0FF20. The dac immediately changes the value output to the PIA from the B register to a voltage level. The output of the comparator is then read into the A register by LDA ,Y. The Y register was previously loaded by the PIA address for the comparator input, $0FF00. If the value in A has bit 7 set, a Branch on Minus (BMI) is taken, and an add of a "delta value" adds one-half of the existing range to

```
172B                00100          ORG     $172B
                    00110 ******************************************
                    00120 * SPEECH SYNTHESIS PROGRAM                 *
                    00130 * ACCUMULATES 1 1/3 SECONDS WORTH OF INPUT *
                    00140 * PLAYS BACK ON REQUEST                    *
                    00150 * ENTER AT INPUT TO RECORD                 *
                    00160 * ENTER AT OUTPUT TO PLAY BACK            *
                    00170 ******************************************
                    00180 *
          17C4      00190 BUFFER   EQU     $4000-10300
          3FFF      00200 BUFEND   EQU     $3FFF    END OF BUFFER
172B 17   0085      00210 INPUT    LBSR    SELECT   SELECT RIGHT,X
172E 108E FF00      00220          LDY     #$0FF00 LOAD INPUT PIA ADDRESS
1732 8E   17C4      00230          LDX     #BUFFER LOAD BUFFER PNTR ADDRESS
1735 C6   80        00240 INP005   LDB     #$80       LOAD START VALUE
1737 F7   FF20      00250          STB     $0FF20       OUTPUT FIRST VALUE
173A A6   A4        00260          LDA     ,Y           INPUT COMPARATOR
173C 2B   04        00270          BMI     INP015       GO IF TOO LOW
173E C0   40        00280          SUBB    #$40         SUBTRACT DELTA
1740 20   04        00290          BRA     INP020       GO TO SECOND ITERATION
1742 CB   40        00300 INP015   ADDB    #$40         ADD DELTA
1744 20   00        00310          BRA     INP020       GO TO SECOND ITERATION
1746 F7   FF20      00320 INP020   STB     $0FF20       OUTPUT SECOND VALUE
1749 A6   A4        00330          LDA     ,Y           INPUT COMPARATOR
174B 2B   04        00340          BMI     INP025       GO IF TOO LOW
174D C0   20        00350          SUBB    #$20         SUBTRACT DELTA
174F 20   04        00360          BRA     INP030       GO TO THIRD ITERATION
1751 CB   20        00370 INP025   ADDB    #$20         ADD DELTA
1753 20   00        00380          BRA     INP030       GO TO THIRD ITERATION
1755 F7   FF20      00390 INP030   STB     $0FF20       OUTPUT THIRD VALUE
1758 A6   A4        00400          LDA     ,Y           INPUT COMPARATOR
175A 2B   04        00410          BMI     INP035       GO IF TOO LOW
175C C0   10        00420          SUBB    #$10         SUBTRACT DELTA
175E 20   04        00430          BRA     INP040       GO TO FOURTH ITERATION
1760 CB   10        00440 INP035   ADDB    #$10         ADD DELTA
1762 20   00        00450          BRA     INP040       GO TO FOURTH ITERATION
1764 F7   FF20      00460 INP040   STB     $0FF20       OUTPUT FOURTH VALUE
1767 A6   A4        00470          LDA     ,Y           LOAD COMPARATOR
1769 2B   04        00480          BMI     INP045       GO IF TOO LOW
176B C0   08        00490          SUBB    #8           SUBTRACT DELTA
176D 20   04        00500          BRA     INP050       GO TO FIFTH ITERATION
176F CB   08        00510 INP045   ADDB    #8           ADD DELTA
1771 20   00        00520          BRA     INP050       GO TO FIFTH ITERATION
1773 F7   FF20      00530 INP050   STB     $0FF20       OUTPUT FIFTH VALUE
1776 A6   A4        00540          LDA     ,Y           INPUT COMPARATOR
1778 2B   04        00550          BMI     INP055       GO IF TOO LOW
177A C0   04        00560          SUBB    #4           SUBTRACT DELTA
177C 20   04        00570          BRA     INP060       GO TO SIXTH ITERATION
177E CB   04        00580 INP055   ADDB    #4           ADD DELTA
1780 20   00        00590          BRA     INP060       GO TO SIXTH ITERATION
1782 F7   FF20      00600 INP060   STB     $0FF20       OUTPUT SIXTH VALUE
1785 A6   A4        00610          LDA     ,Y           INPUT COMPARATOR
1787 2B   04        00620          BMI     INP065       GO IF TOO LOW
1789 C0   02        00630          SUBB    #2           SUBTRACT DELTA
178B 20   04        00640          BRA     INP070       GO FOR NEXT VALUE
178D CB   02        00650 INP065   ADDB    #2           ADD DELTA
178F 20   00        00660          BRA     INP070       GO FOR NEXT VALUE
1791 E7   80        00670 INP070   STB     ,X+        STORE VALUE
1793 8C   3FFF      00680          CMPX    #BUFEND    TEST FOR END OF BUFFER
1796 26   9D        00690          BNE     INP005     GO IF NOT END
1798 39             00700          RTS              END-RETURN
```

Fig. 3–7. INPUT subroutine.

the value in the B register. If the value in A has bit 7 reset, the SUBB #$XX is done to subtract one-half of the existing range.

The six sections taken together constitute a binary search to find the input value. At INP070, the B register holds the final value. It is stored in the next memory location pointed to by the X register. The ,X+ form of

the instruction automatically increments the X register by one to point to the next location after the current store. The X register is then compared to BUFEND, the last location for storing digitized values. If there is space left, the routine branches back to INP005 to sample the next value.

The INPUT routine takes 6 × 19.1 + 14.6 microseconds for each adc conversion, allowing 7733 samples per second. Note that during each 129.2-μs conversion, the input voltage may change and that the final value may be off by 25% or more, as shown in Fig. 3–8. In the majority of cases, however, the result is fairly close for these high sampling rates and audio frequencies. The buffer is 10,300 bytes long, making the total time of input about 1⅓ seconds.

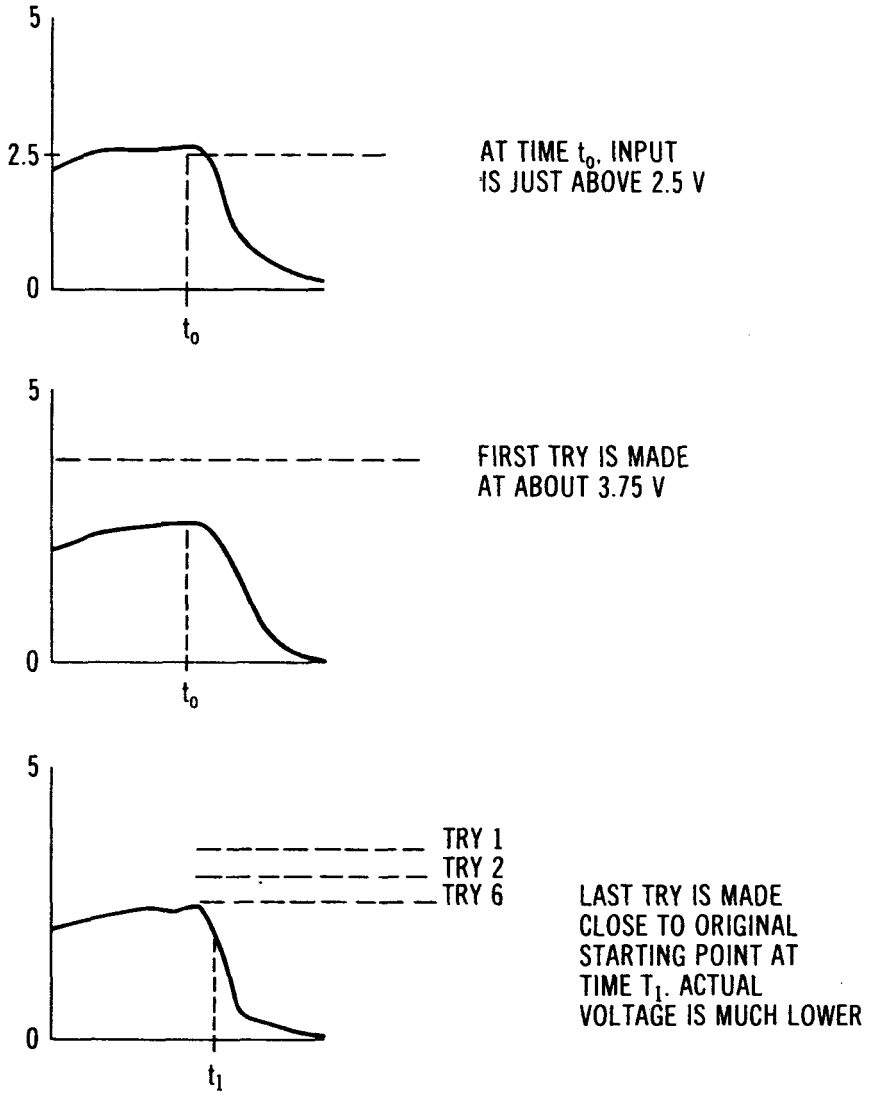


Fig. 3–8. Error on adc conversion.

OUTPUT Routine

The OUTPUT subroutine is considerably simpler. It is shown in Fig. 3–9. The routine starts at the beginning of the BUFFER, delays about 1/7700 second, fetches a value from memory (LDA ,X+), outputs the value to the dac (STA $0FF20), tests for the end of the buffer (BUFEND), and then returns for the next value if there is more data remaining in the buffer. The delay routine simply loops back to OUT020 if a count in the A register has not been decremented down from 19 to 0.

SELECT Routine

The SELECT subroutine alters the routing in the Color Computer so that the right joystick X channel is chosen for the adc and so that the dac output goes to the television sound. It is entered at the beginning of both INPUT and OUTPUT.

BASIC Driver

The 6809 assembly language programs in Figs. 3–7 and 3–9 are *relocatable;* that is, they can be moved anywhere in memory and still operate properly without reassembly. Fig. 3–10 shows the programs converted to DATA values in an Extended Color BASIC program.

Most of the program relocates the DATA values to memory locations $172B through $17C3. The loop from 180 through 260 replicates the six sections of the INPUT routine six times to condense the number of

```
1799 8D   18      00710 OUTPUT  BSR     SELECT   SELECT DAC OUTPUT
179B 86   3C      00720         LDA     #$3C     LOAD INITIALIZATION VALUE
179D B7   FF23    00730         STA     $0FF23   INITIALIZE PIA FOR OUTPUT
17A0 8E   17C4    00740         LDX     #BUFFER  POINT TO BUFFER
17A3 86   13      00750 OUT010  LDA     #19        DELAY COUNT
17A5 4A           00760 OUT020  DECA                 DELAY LOOP
17A6 26   FD      00770         BNE     OUT020     DELAY
17A8 A6   80      00780         LDA     ,X+      GET VALUE
17AA B7   FF20    00790         STA     $0FF20   OUTPUT TO DAC
17AD 8C   3FFF    00800         CMPX    #BUFEND  TEST FOR END OF DATA
17B0 26   F1      00810         BNE     OUT010   GO IF NOT END
17B2 39           00820         RTS              END-RETURN
17B3 B6   FF01    00830 SELECT  LDA     $0FF01   GET PIA CONFIGURATION
17B6 84   F7      00840         ANDA    #$0F7    RESET LSB OF MUX SELECT
17B8 B7   FF01    00850         STA     $0FF01   STORE
17BB B6   FF03    00860         LDA     $0FF03   GET PIA CONFIGURATION
17BE 84   F7      00870         ANDA    #$0F7    RESET MSB OF MUX SELECT
17C0 B7   FF03    00880         STA     $0FF03   STORE
17C3 39           00890         RTS              RETURN
          0000    00900         END
00000 TOTAL ERRORS
```

Fig. 3–9. OUTPUT subroutine.

```
100 PCLEAR 1:CLEAR 10,&H1720
110 REM VOICE SYNTHESIS PROGRAM IN BASIC FORM
120 DATA 247,255,32,166,164,43,4,192,0,32,4,203,0,32,0
130 DATA 23,0,133,16,142,255,0,142,23,196,198,128
140 DATA 231,128,140,63,255,38,157,57,141,24,134,60,183,255,35
150 DATA 142,23,196,134,19,74,38,253,166,128,183,255,32
160 DATA 140,63,255,38,241,57,182,255,1,132,247,183,255,1,182,255,3
170 DATA 132,247,183,255,3,57
180 FOR J=0 TO 5
190 RESTORE
200 FOR I=&H1737+J*15 TO &H1745+J*15
210 READ A
220 POKE I,A
230 NEXT I
240 POKE &H173F+J*15,2^(6-J)
250 POKE &H1743+J*15,2^(6-J)
260 NEXT J
270 FOR I=&H172B TO &H1736
280 READ A
290 POKE I,A
300 NEXT I
310 FOR I=&H1791 TO &H17C3
320 READ A
330 POKE I,A
340 NEXT I
350 DEFUSR0=&H172B:DEFUSR1=&H1799
360 INPUT "RECORD (R) OR PLAY (P)?";A$
370 IF A$="R" THEN A=USR0(0) ELSE IF A$="P" THEN A=USR1(0) ELSE GOTO 360
380 GOTO 360
```

Fig. 3–10. BASIC input/output driver.

DATA values. Values of 64, 32, 16, 8, 4, and 2 are POKEd for the "delta values" in two places. The following loops move the remainder of the values.

There are two entry points to the code, one at INPUT and one at OUTPUT. In this fixed location for the program, INPUT is at location $172B and OUTPUT is at location $1799. USR0 defines the INPUT start and USR1 defines the OUTPUT start. A discussion of program operation follows a description of the special hardware for this project.

VOICE SYNTHESIS SPECIAL HARDWARE

The normal joystick inputs are shown in Fig. 3–11. Each joystick plug is a 5-pin DIN plug described in Chapter 1. One pin is connected to the X channel (right/left), one to the Y channel (up/down), one to ground, one to +5 volts dc, and one to a switch on the joystick. The output for X and Y is taken from a voltage divider made up of the potentiometer "legs." The output will vary from 0 volts to about +5 volts.

In this application we're using only the X channel of the right joystick. We'd like to convert an audio signal, which is essentially an ac voltage, to a level of 0 to 5 volts dc. The 0 to 5-volt level can then be sampled, digitized, and stored in memory by the adc hardware and software.

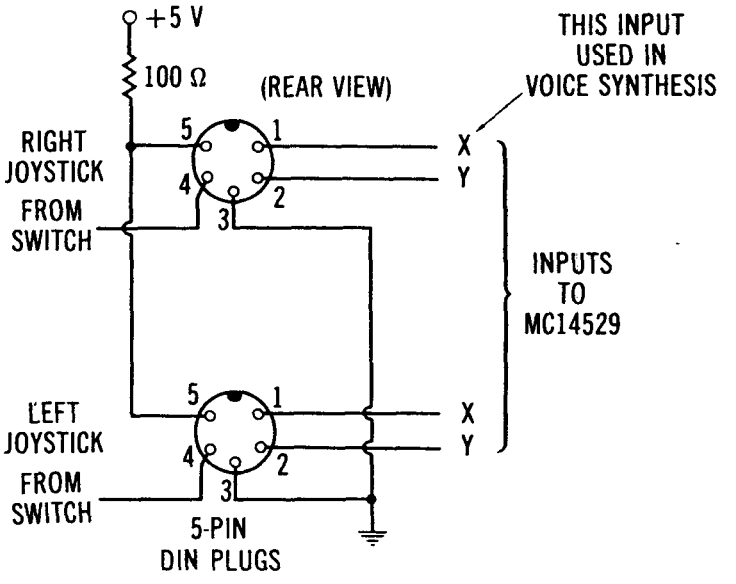


Fig. 3–11. Color Computer joystick channels.

The circuit in Fig. 3–12 is a simple operational amplifier that will take a voltage level and amplify it by a factor of 10. A crystal microphone has a relatively high output level (tenths of a volt) and is a good match for a X10 amplifier. The normal output voltage level of the amplifier with no input sound is about midrange at 2.3 volts. The output will vary about this bias. The power for the amplifier is supplied by the +5 volts from pin 1 of the DIN plug. As the amplifier requires less than 4 mA, there is no problem in using the +5-volt supply of the Color Computer, other than a voltage drop of about 0.4 volt across the 100-ohm resistor on the +5-volt lead.

The easiest way to construct the amplifier is to mount the parts on a prototype board, as shown in Fig. 3–13. Radio Shack has one for around $6.50 (RS 276-175). This board consists of 23 rows of 12 holes each, as shown in the figure. The outer vertical columns on the left and right can be used for ground and power buses as shown. Fig. 3–13 shows the arrangement of the components on the prototype board. The resistor and capacitor leads can be cut to length and then pushed into the proper holes without soldering or wire-wrapping. The LM3900N op amp can also be pushed into the board (the holes are properly spaced).

The microphone used in this project is really a crystal microphone cartridge (Radio Shack 270-095, for a little over $1.50). You will have to solder two wires to the cartridge. Tin the other ends of the wire and plug into the board as shown.

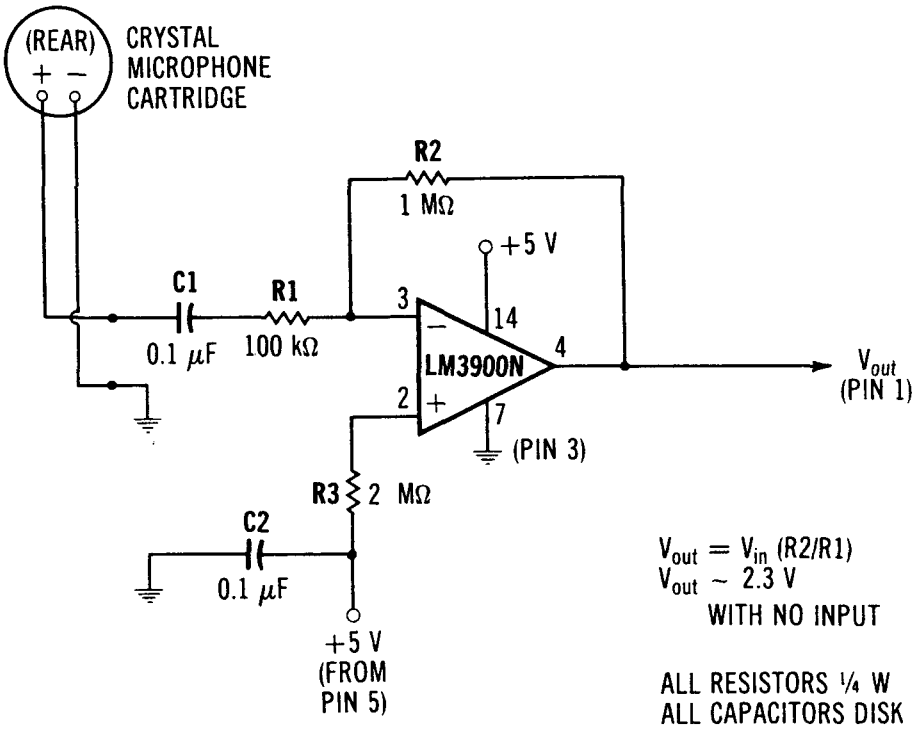


Fig. 3–12. Voice synthesis op amp.

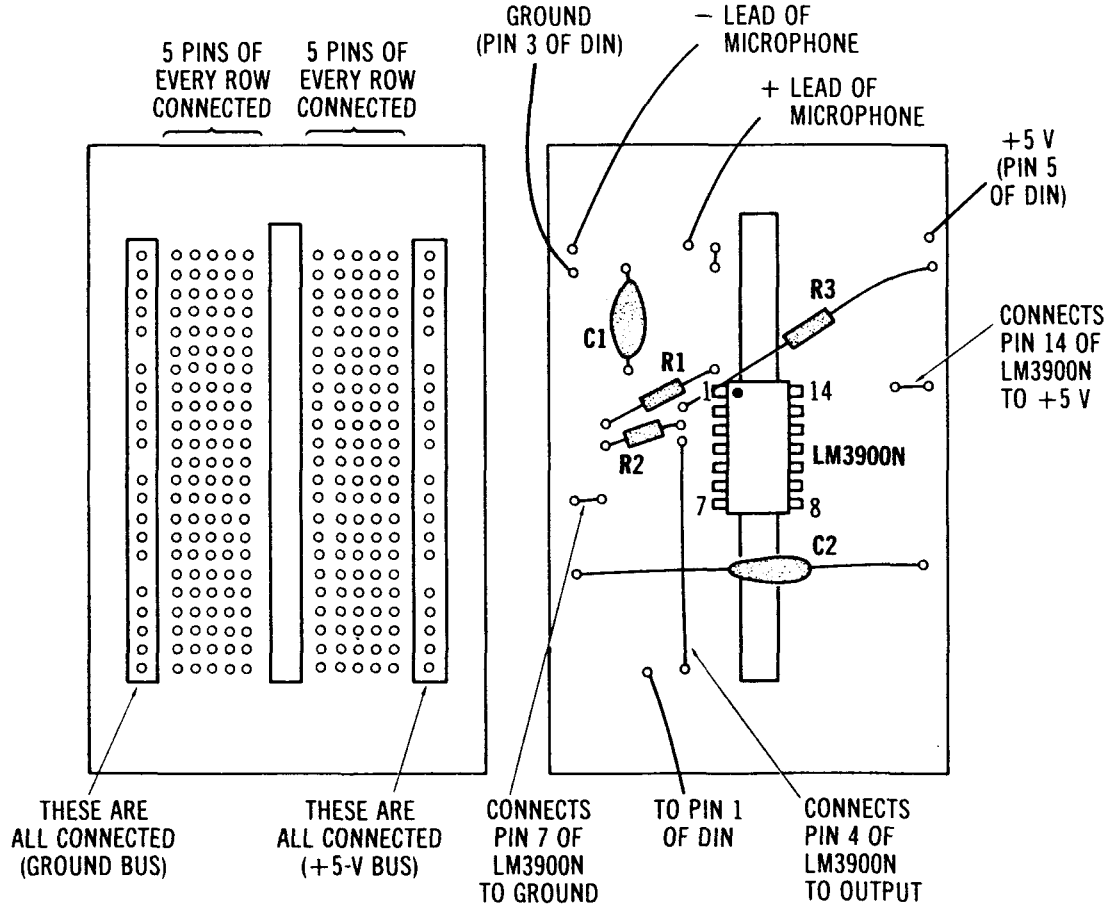


Fig. 3–13. Voice synthesis breadboard.

Three wires go from the board to the Color Computer right joystick plug, as shown in the figure. One wire attaches to ground (pin 3), one attaches to +5 volts (pin 5), and one attaches to the X channel (pin 1). All parts are available from Radio Shack or your friendly neighborhood electronics store and should cost under $10.00.

OPERATION OF THE VOICE SYNTHESIZER

To operate the unit, plug the completed circuit into the right joystick plug. Turn on the Color Computer and execute the following program:

```
100 PRINT JOYSTK(0)
110 GOTO 100
```

You should see a printout of about 30, representing 30/64ths of 4.6 volts, or about 2.3 volts. If the displayed values are less than 30, decrease the value of R3; if the displayed values are greater than 32, increase the value of R3. The optimum value is 30, although values from 26 through 34 are acceptable. If the bias is offset too far from 30, however, audio signals will clip on either the top or bottom, as shown in Fig. 3–14, resulting in distorted sound. Talk into the microphone while running the program. You should see the values change, although the pattern isn't predictable. Look for low (close to 0) and high (close to 63) values.

If all looks proper, load the program shown in Fig. 3–7 and execute it. When the message RECORD (R) OR PLAY (P)? is displayed, type R. At the same time, speak loudly into the microphone element while holding it close. Speaking off to the side eliminates voice "pops." You have about 1⅓ seconds to record the message. The time allows such messages as "HELP! COMPUTER FAILURE!", "TWAS BRILLIG AND THE SLITHY . . . ", and "INPUT ERROR, DUMMY!".

The program will record the audio and then return to the prompt message again. Enter P to play back the message through the television audio. You may want to repetitively play back the message once recorded by looping back to the P USR call. The quality of the sound output is quite good, even though the duration of the speech is short.

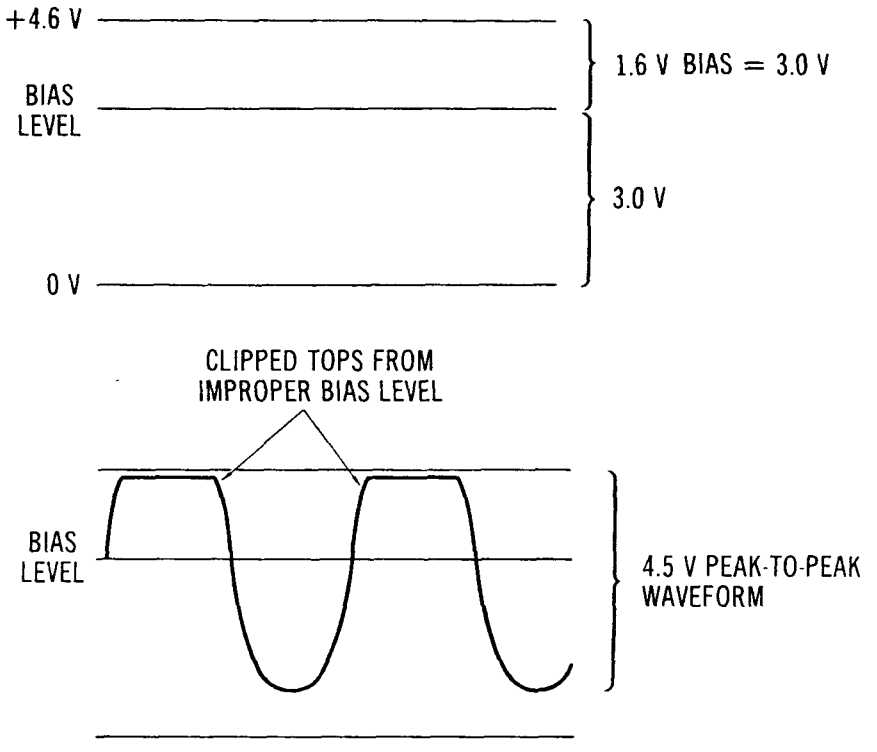


Fig. 3–14. Clipping on audio input.

CONDENSING THE DATA

That's the basic hardware and software for acquiring and playing back the data. Now we come to the question of how the data can be condensed. There are three approaches here: (1) altering the sampling parameters during acquisition of the data, (2) processing the data after acquisition, and (3) a combination of the two.

Altering the Sampling Parameters

The program above recorded the data at a sampling rate of 7700 samples per second. The rate can be reduced by putting in a time delay after the STB ,X+ in the INPUT routine. A simple

```
        LDA    #X     CONSTANT
LOOP    DECA          DECREMENT
        BNE    LOOP   LOOP IF NOT ZERO
```

would do the trick. This would delay the acquisition of data by about 5.62 × "X" microseconds. Sampling rates (samples/second) for various values of X are shown in Table 3-1. The program must be reassembled if this change is made, as the displacement values for the branches in some cases are no longer valid. From the quality of the speech at the 7700 samples per second rate, degradation at sampling rates of 6000 per second or so is probably acceptable.

Another parameter that can be varied in acquisition is the resolution of the adc. We used a 6-bit adc, allowing for 64 different levels. Certainly one or two bits could be deleted from this resolution without too much degradation. If two bits could be deleted, twice as much data could be stored in memory by packing two nibbles per byte in memory. This would call for a little more overhead in the INP070 area as the values were stored, but the net effect would probably maintain the same sampling rate (or better) as the instructions from INP050 through INP070 could be deleted.

Data Processing After Acquisition

There are a number of possibilities in processing after acquisition. In this method of compression, the adc values are post-processed by an analysis program.

First of all, the waveforms may be symmetrical about the horizontal axis. Therefore, keep one half and throw the other away, as shown in Fig. 3–15. The trick here is recognizing repetitions of the cycle.

Another possibility is to delete the dead time between words. In a string of words, there are large areas where there is no sound, and these

Table 3–1. Sampling Rate Vs. Delay Time in INPUT

X	Sampling Rate
1	7410
2	7114
3	6841
4	6587
5	6414
10	5390
20	4137
30	3357

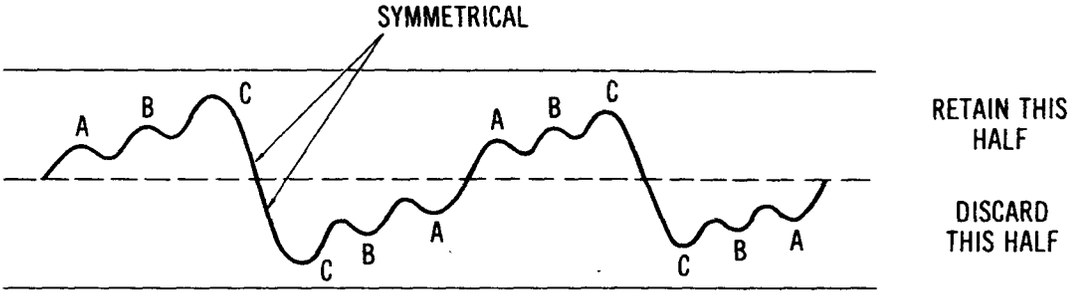
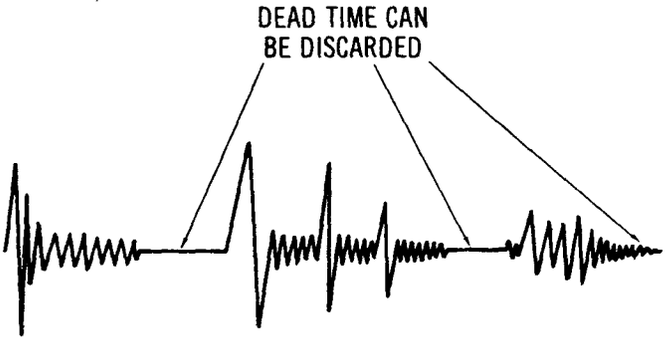


Fig. 3–15. Data compression of symmetrical waveforms.



"COLOR COMPUTER IS ..."

VALUE	00	LEGITIMATE ADC VALUE
VALUE	00	LEGITIMATE ADC VALUE
111111	11	FLAG WORD (WASTED BITS NOT 0)
DELAY IN MS		DELAY COUNT
VALUE	00	LEGITIMATE ADC VALUE
VALUE	00	LEGITIMATE ADC VALUE

Fig. 3–16. Data compression by elimination of dead time.

are a waste of storage. A special flag value could be detected on output and a delay of n number of milliseconds could be performed based on the value following the flag value, as shown in Fig. 3–16.

Another compression technique would be to look for portions of the data that change slowly. Certain sounds, such as vowels, have a much lower level than consonants like P that almost explode over a wide dynamic range. If the change is small enough, it can be held in 4 bits instead of 8, reducing memory requirements. Again, a flag value can be used on output to get into this slow change mode, as shown in Fig. 3–17.

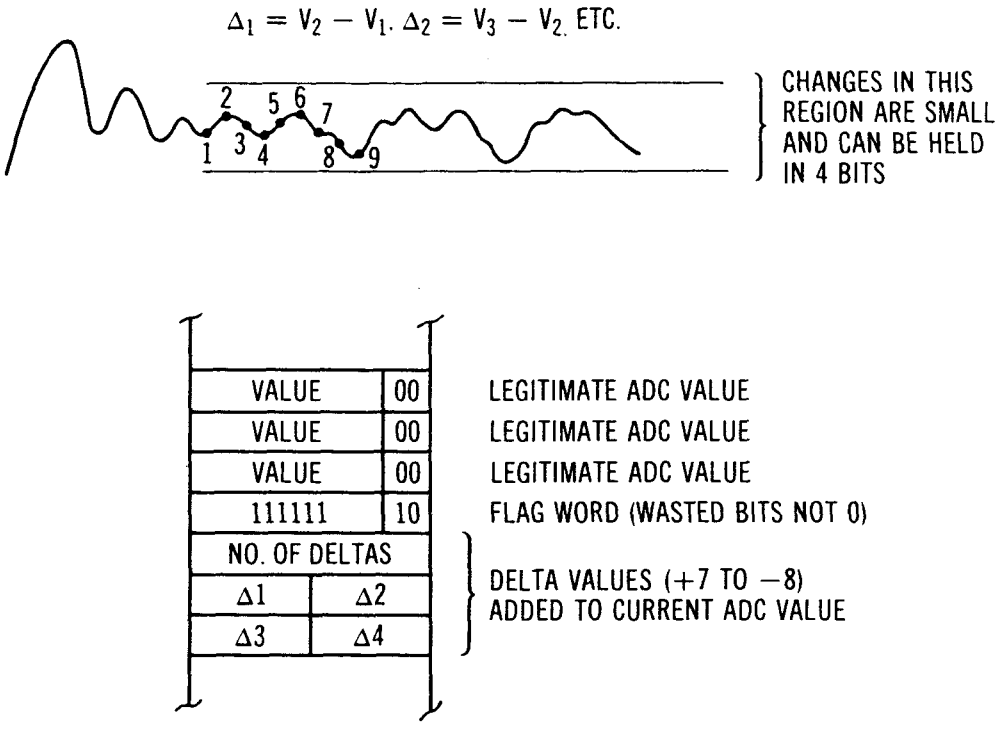


Fig. 3–17. Data compression by nibble encoding.

We hope that we've stimulated your imagination with this chapter. Half the battle is getting the data digitized. The rest is mere programming!

chapter 4

A/D and Joysticks for the Models I and III

So far we've examined how the Color Computer uses analog-to-digital hardware to implement joysticks, how the Color Computer firmware reads in joystick positions, and how real-world analog inputs can be used in place of the joysticks to provide an easy means for processing of such inputs as temperature and light intensity changes. In this chapter we give "equal time" to Models I and III users.

The Models I and III do not have the built-in hardware for digital-to-analog and analog-to-digital conversion that the Color Computer has, but it can be added very easily—two common integrated circuits and a small number of resistors. For those who think they're not electronics-oriented, step-by-step instructions are provided on how to fabricate and test the circuit. After you're done you'll have a working joystick, a two-channel digital-to-analog converter, and a two-channel analog-to-digital converter—all for less than $20.

JOYSTICK CIRCUIT

The block diagram of the joystick circuit is shown in Fig. 4–1. An expansion interface is necessary on the Model I, as we're driving the circuit from the line printer port. The joystick circuit uses the address decoding, latches, and input gates of the line printer port on the assumption that the joystick would normally be used in operations not involving the line printer.

The bulk of the circuit duplicates the circuitry of the Color Computer (see Chapter 1), with some minor variations. There is a resistor-network digital-to-analog converter (dac) and two comparators, one for the X

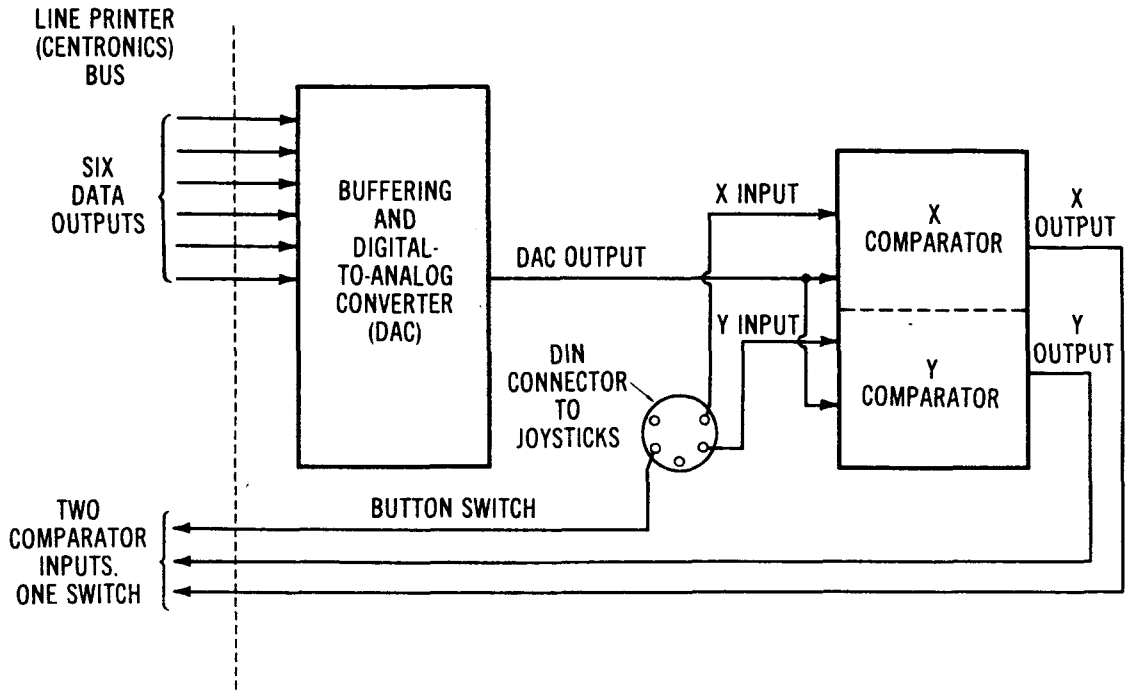


Fig. 4–1. Block diagram of the Models I and III joystick circuitry.

channel and one for the Y channel. The comparators use the output of the dac to do a successive approximation of the analog input voltages of the X and Y joystick channels.

There are six outputs to the dac; and six outputs make up a 6-bit digital value which is converted to an analog value by the dac. There are three inputs to the computer line printer port, one for the X channel, one for the Y channel, and a spare for a joystick switch.

Line Printer Port

Before we discuss the operation and construction of the joystick circuit, let's look at the operation of the line printer port on the Models I and III, as the joystick circuitry to some extent *emulates* a line printer.

A simplified version of the line printer port circuitry for the Model I is shown in Fig. 4–2. It consists of two 74LS175 chips, each containing four flip-flops, four buffers of a 74LS367 chip, and a one-shot implemented by one half of a 74LS123. When a character is output to line printer address 37E8H in the Model I, the clock signal (CK) strobes the 8 bits of data into the two 74LS175s. That data remains in the 74LS175s until a new character is output or until a System Clear (CLR) is done. When status is input for the line printer, the read from address 37E8H in the Model I gates the four lines for BUSY, OUTPAPER, UNIT SELECT, and

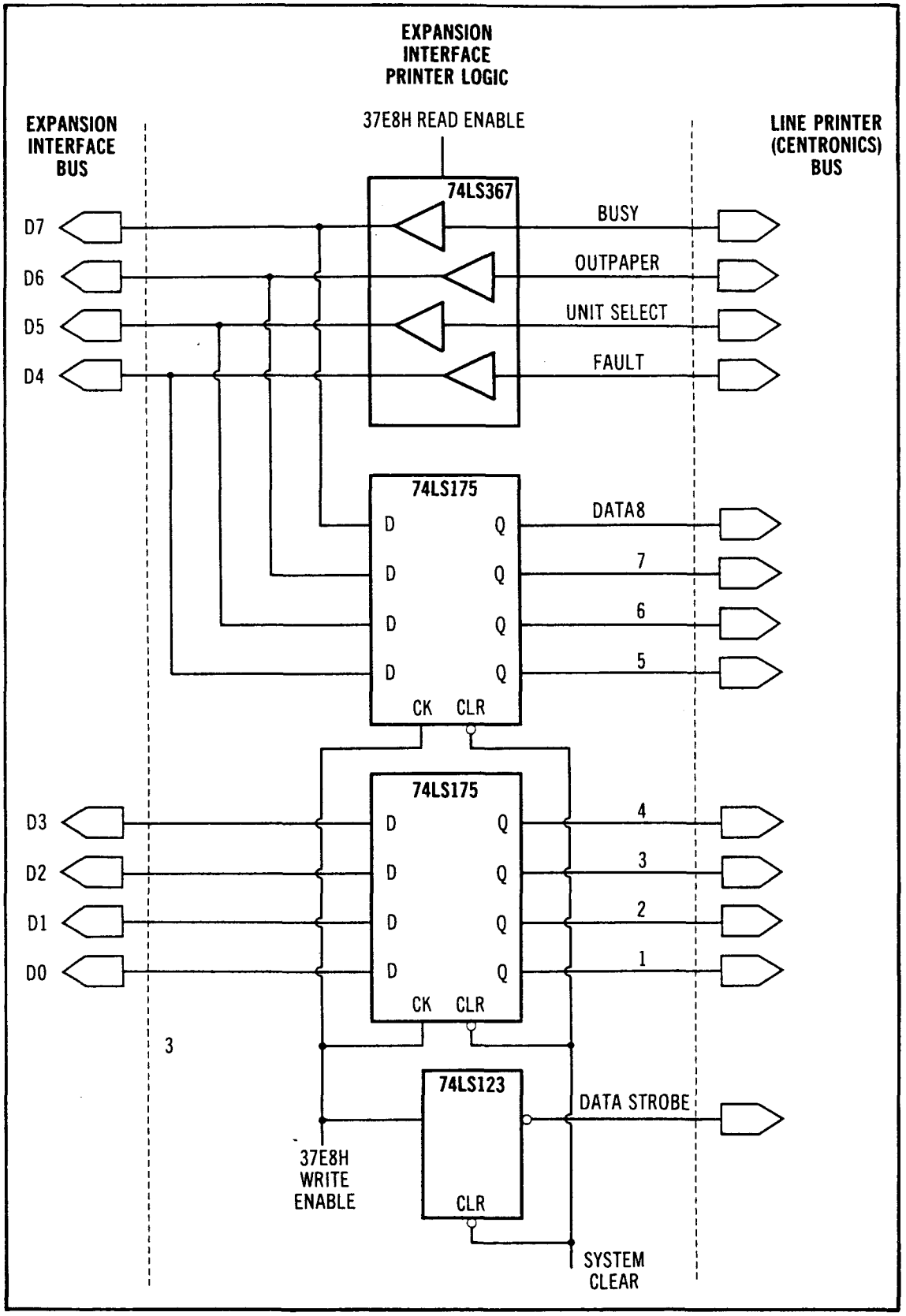


Fig. 4–2. Model I line printer circuitry.

FAULT onto data lines D7, D6, D5, and D4, respectively. The one-shot output is the signal DATA STROBE, which becomes active when a write is done; this strobe tells the line printer that data is available on line printer lines DATA8 through DATA0.

The Model I goes through the following sequence to output data to the line printer:

1. It executes an LD A,(37E8H) to read the line printer status.
2. It tests bits 7 (BUSY) and 6 (OUTPAPER) of the status. If they are both zero, indicating that the line printer has finished with the last character and that there are no abnormal conditions, the line printer is ready to accept the next character. If the line printer is not ready, a loop back to the LD A,(37E8H) is done to read in status again.
3. If the line printer is ready, an LD (37E8H),A is done to output the next character (in the A register) to the line printer. This activates the one-shot and strobes the 8 bits of the character into the 74LS175s.
4. The one-shot resets after a short delay, strobing the 8 bits of the character into the line printer electronics, starting the printing cycle and setting BUSY.

Memory Mapping vs. I/O Mapping

The Model I is *memory-mapped* for the line printer port, with address 37E8H representing the line printer address. The Model III is *I/O mapped* for the line printer, with port address 0F8H representing the line printer address. Aside from using different chips, the Model III has the same logical implementation as the Model I. Output to the Model I is done via LD (37E8H),A; output to the Model III is done via OUT (0F8H),A. Input of status to the Model I is done via LD A,(37E8H); input of status to the Model III is done via IN A,(0F8H). (The Model III also may use an LD A,(37E8H) to input status, but we'll do all of our inputs using LDs.)

The above instructions are, of course, Z-80 machine-language instructions. Model I BASIC inputs from address 37E8H may be done by means of a PEEK (14312); Model I BASIC outputs to address 37E8H may be done by POKE 14312,X, where X is the byte to be output. Model III BASIC inputs are done by means of INP (248) and outputs are done by OUT 248,X, where X is the byte to be output.

Using the Line Printer Port for the Joystick Circuitry

We can easily make the joystick circuitry emulate a line printer. First of all, we can forget about the DATA STROBE output. It's only there for the line printer electronics. Since data simply stays in the 74LS175s (or their Model III equivalents), we can simply do a write to 37E8H (or 0F8H) to output 8 bits to DATA8 through DATA1. Whenever we want to read in data, all we have to do is read 37E8H (or 0F8H) to input four bits.

We've chosen to dedicate DATA6 through DATA1 as outputs from the program to the dac, the BUSY input as the X-channel comparator input, and the OUTPAPER input as the Y-channel comparator input. These eight lines, plus GROUND, are all we need to perform the joystick operation, to do digital-to-analog conversions and to do analog-to-digital conversions. They're shown in Fig. 4–3. A ninth line is optionally usable as a joystick "button" input.

How the Joystick Circuit Works

The detailed joystick circuit is shown in Fig. 4–4. The physical layout of the circuit corresponds to the physical layout of the block diagram in Fig. 4–1.

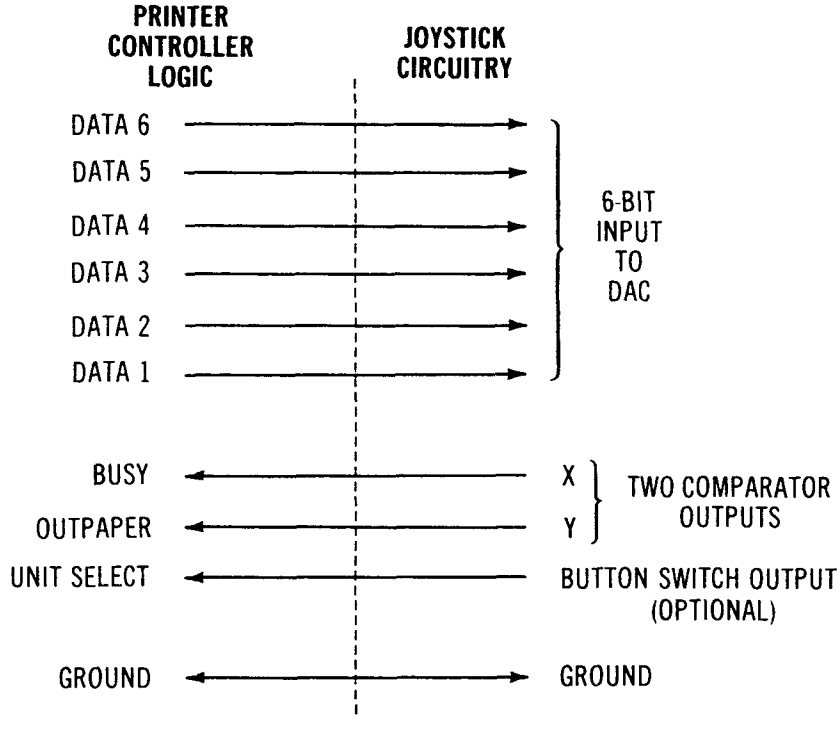


Fig. 4–3. Line printer I/O lines.

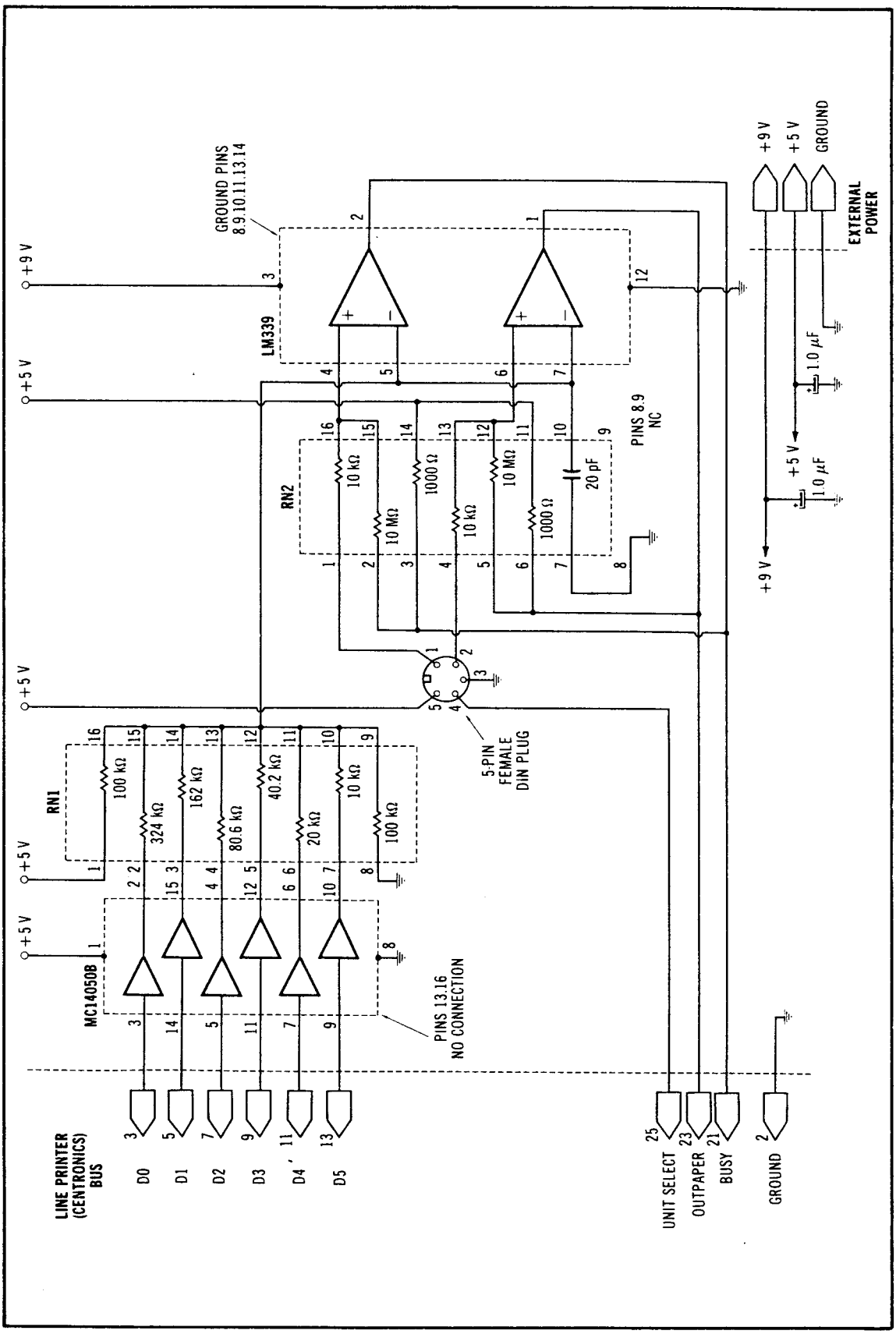


Fig. 4–4. Detailed joystick circuitry for the Models I and III.

The joystick schematic is shown in Fig. 4–5. It is essentially two potentiometers with the two ends of each connected between +5 volts and ground. The wiper of each pot varies with the position of the joystick. Output on the wiper varies between 0 volts and +5 volts. The X-channel 0-volt position is to the left and the Y-channel 0-volt position is toward the top.

Two joysticks are available for the Color Computer (Radio Shack 26-3008, $24.95). One of these can be used with this joystick circuitry. A second alternative is to get the 100K joystick pot for $4.95 (RS 271-1705). This pot is shown in Fig. 4–6 with the required connections.

Each of the joystick voltage outputs goes into one of the comparator positive (+) inputs. The negative (−) input for both comparators comes from the output of the digital-to-analog converter. Each comparator compares the present joystick voltage with the dac output. If the joystick voltage is lower than the dac output, a logic 0 is generated by the comparator. If the joystick voltage is higher than the dac output, a logic 1 is generated by the comparator. The outputs of both comparators feed input lines BUSY (X) and OUTPAPER (Y). To find where any given joystick voltage is, all we must do is vary the dac output from 0 to +5 volts until we get a comparator output of 1 for the channel. And that's precisely what we do with the dac.

The dac is a 6-bit dac. Each resistor is approximately double the resistance of the next lower resistance. Each resistor is connected to the

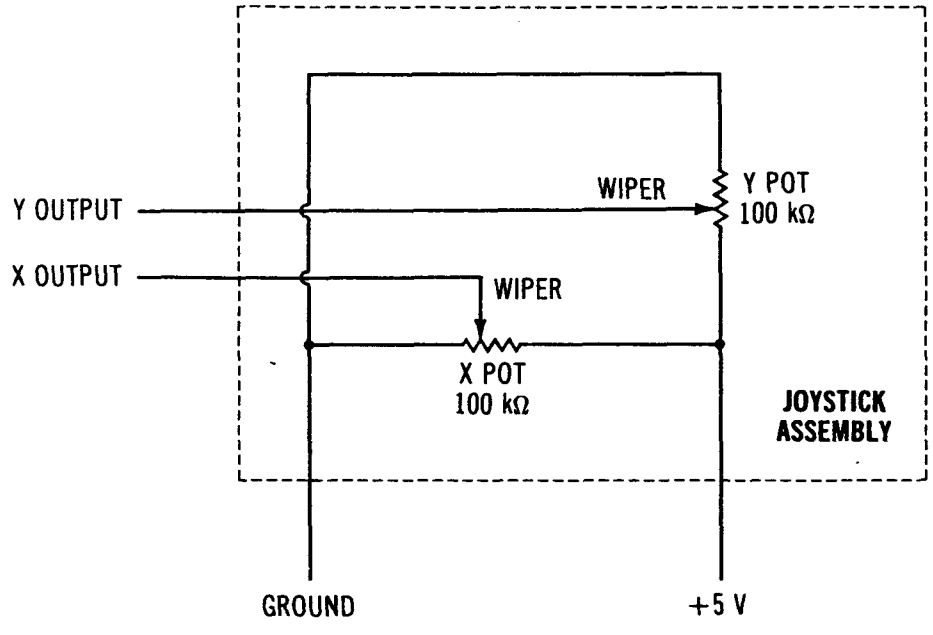


Fig. 4–5. Joystick schematic.

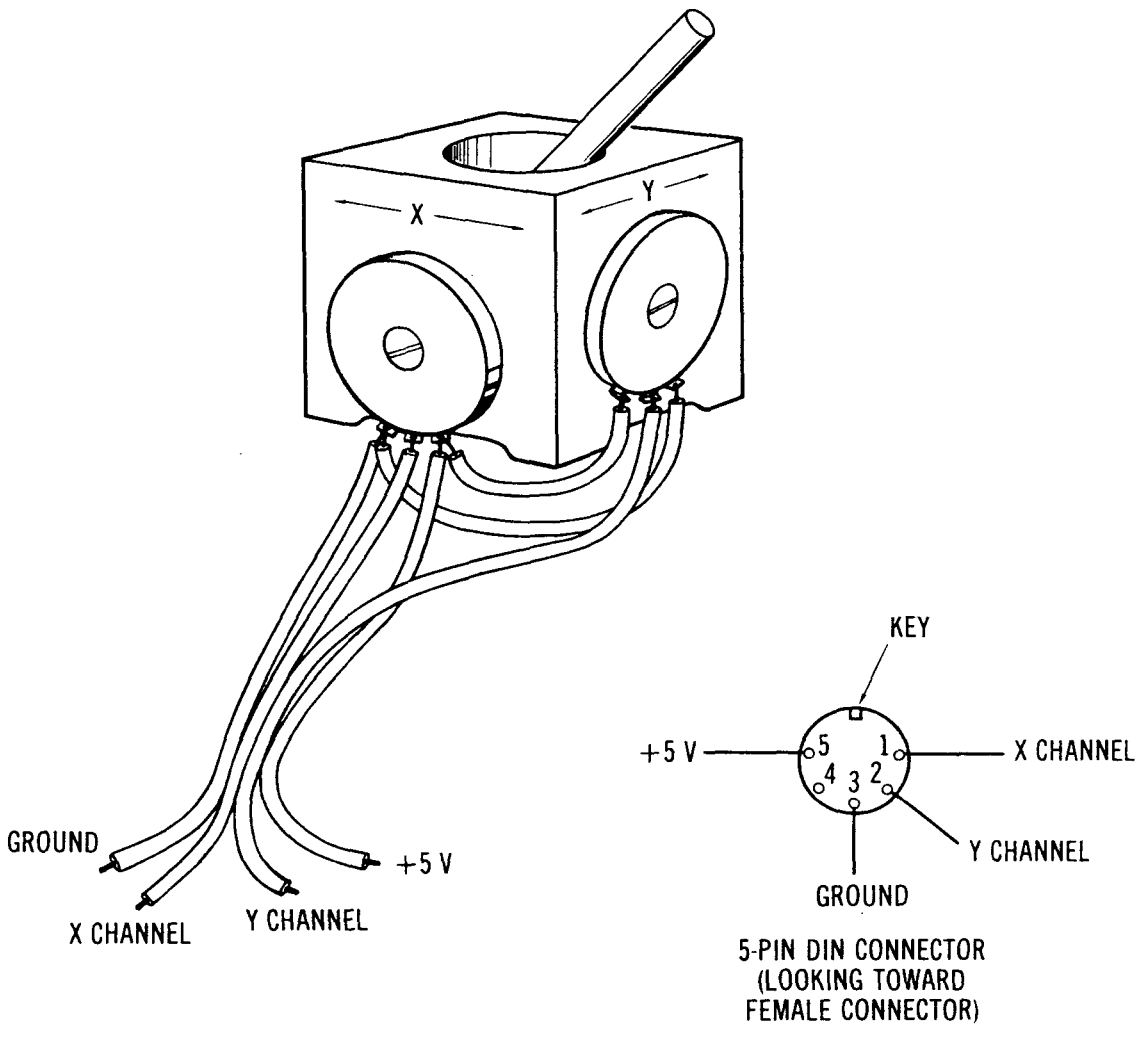


Fig. 4–6. Discrete joystick potentiometer.

output of one bit of the MC14050B. This is a CMOS buffer with an output of close to 0 volts for an input of 0 or about +4.95 volts for an input of 1. By varying the 6-bit input from 000000 through 111111, we will get a total voltage output of about 0.25 volt through 4.75 volts in steps of about 70 mV (see Fig. 4–7). The resistances produce weighted voltage outputs dependent upon their bit position.

If we want an analog-to-digital conversion, we can simply forget about the comparator output and take the output from pin 12 of the MC14050B. The voltage output will be the analog equivalent of the 6-bit input value. If we want to find the joystick voltage, which is really position, then we simply vary the dac input from 000000 through 111111 until we read a one on the proper comparator output. That's all there is to it! Almost . . .

CONSTRUCTING THE JOYSTICK CIRCUIT

Parts List

A parts list for the joystick circuit is shown in Table 4–1. All of the parts can be obtained at your local Radio Shack store or at a similar type of electronics supplier. The resistor values are somewhat critical. If you cannot get 1% resistors of the values indicated, you can use "hand-selected" 5% resistors. Measure the resistance of each with a multimeter and choose values that are within 5% of the proper values. There is enough variation in most resistors that you should be able to come fairly close to the proper values. Two resistors may be used in series to get a total resistance that is correct. The prototype circuit was made by hand selection of resistors and works well.

Soldering and Wire Wrapping

You will need a small (30-watt) soldering iron and rosin-core solder for the circuit. You'll also need a wire-wrap tool or gun. If you've never wire-

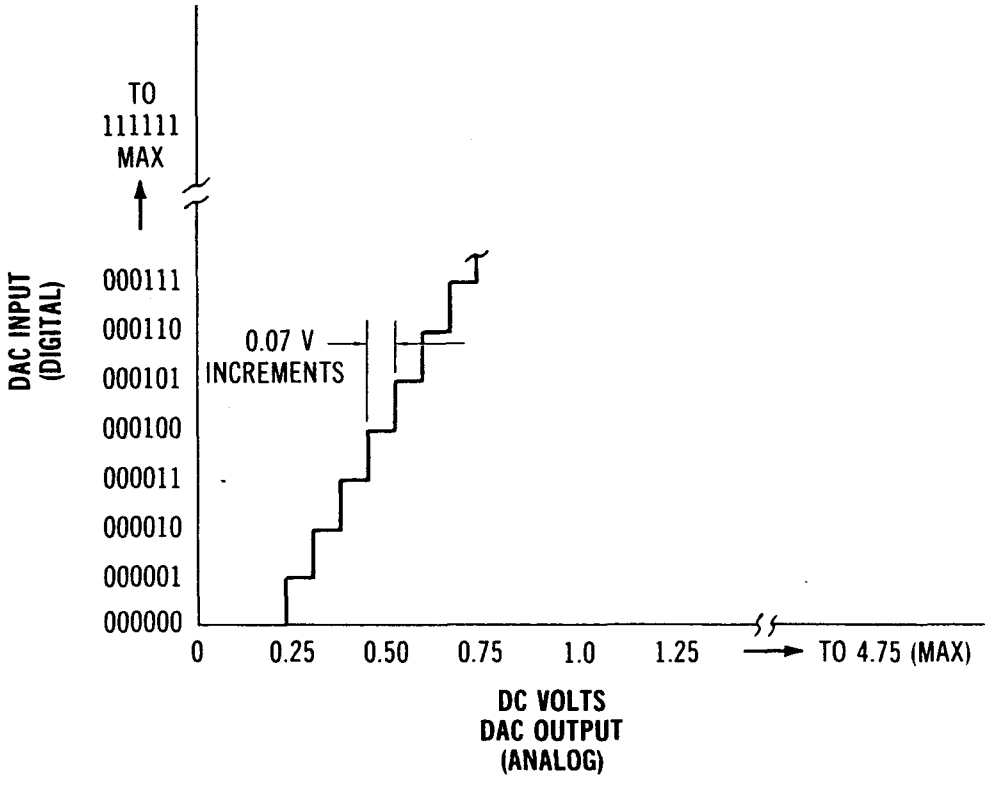


Fig. 4–7. Dac output.

Table 4–1. Parts Required for Joystick Circuitry

Amt.	Description
1	Experimenter's pc board or prototype board
1	5-pin female DIN socket, chassis mounting
3	16-pin wire-wrap socket
1	14-pin wire-wrap socket
1	34-pin edge connector for pc board
2	16-pin DIP header
2	100 KΩ 1/8 W or greater 5% resistor
2	10 MΩ 1/8 W or greater 5% resistor
2	10 KΩ 1/8 W or greater 5% resistor
2	1000 Ω 1/8 W or greater 5% resistor
1	324 KΩ 1/8 W or greater 1% resistor
1	162 KΩ 1/8 W or greater 1% resistor
1	80.6 KΩ 1/8 W or greater 1% resistor
1	40.2 KΩ 1/8 W or greater 1% resistor
1	20 KΩ 1/8 W or greater 1% resistor
1	10 KΩ 1/8 W or greater 1% resistor
1	20- or 47-pF disc capacitor
2	1-μF electrolytic capacitor
1	MC14050B (4050B) IC
1	LM339 IC
Misc.	Wire-wrap, hookup wire

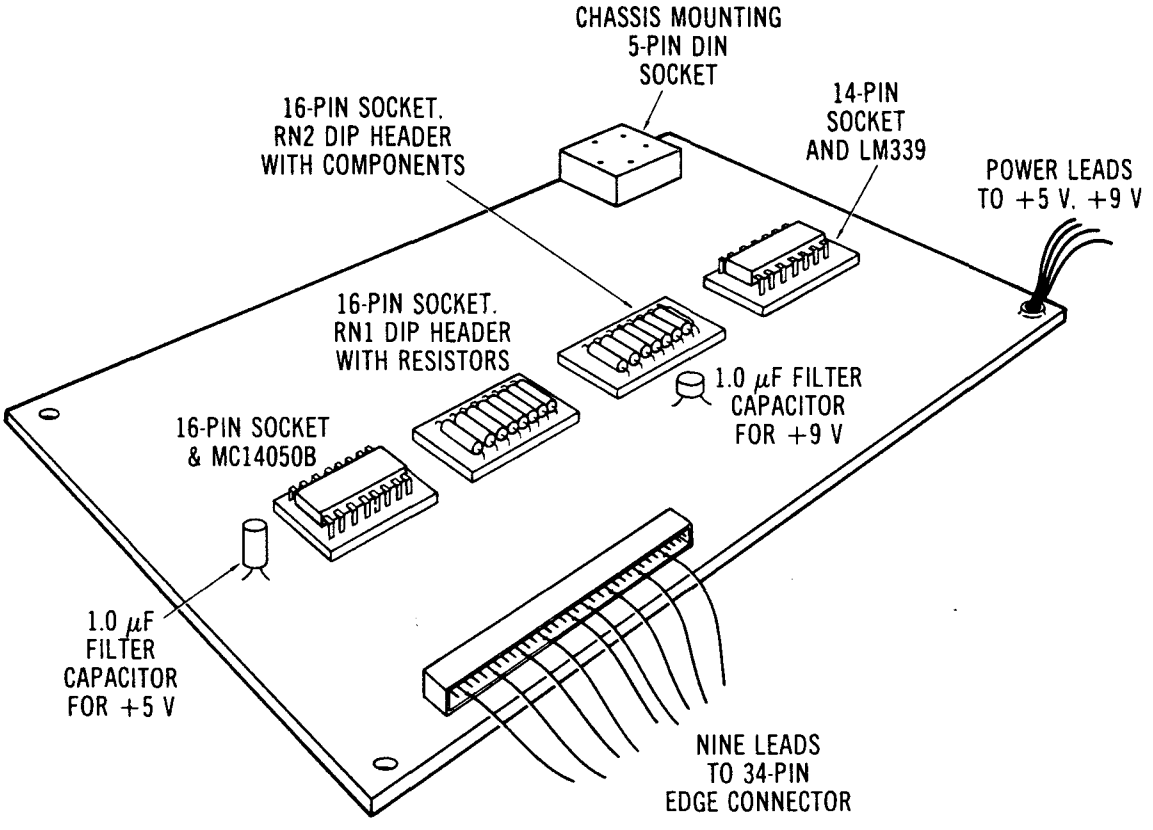


Fig. 4–8. Physical layout of joystick circuitry.

wrapped, you'll find that it's very easy to do and you'll be able to make about one connection per minute. Assuming that you have all the parts, it'll probably take about an hour and a half for the entire job.

Mounting the Parts

The circuit is mounted on a small prototype board (RS 276-170). The general layout is shown in Fig. 4–8. This board is bare on one side and has

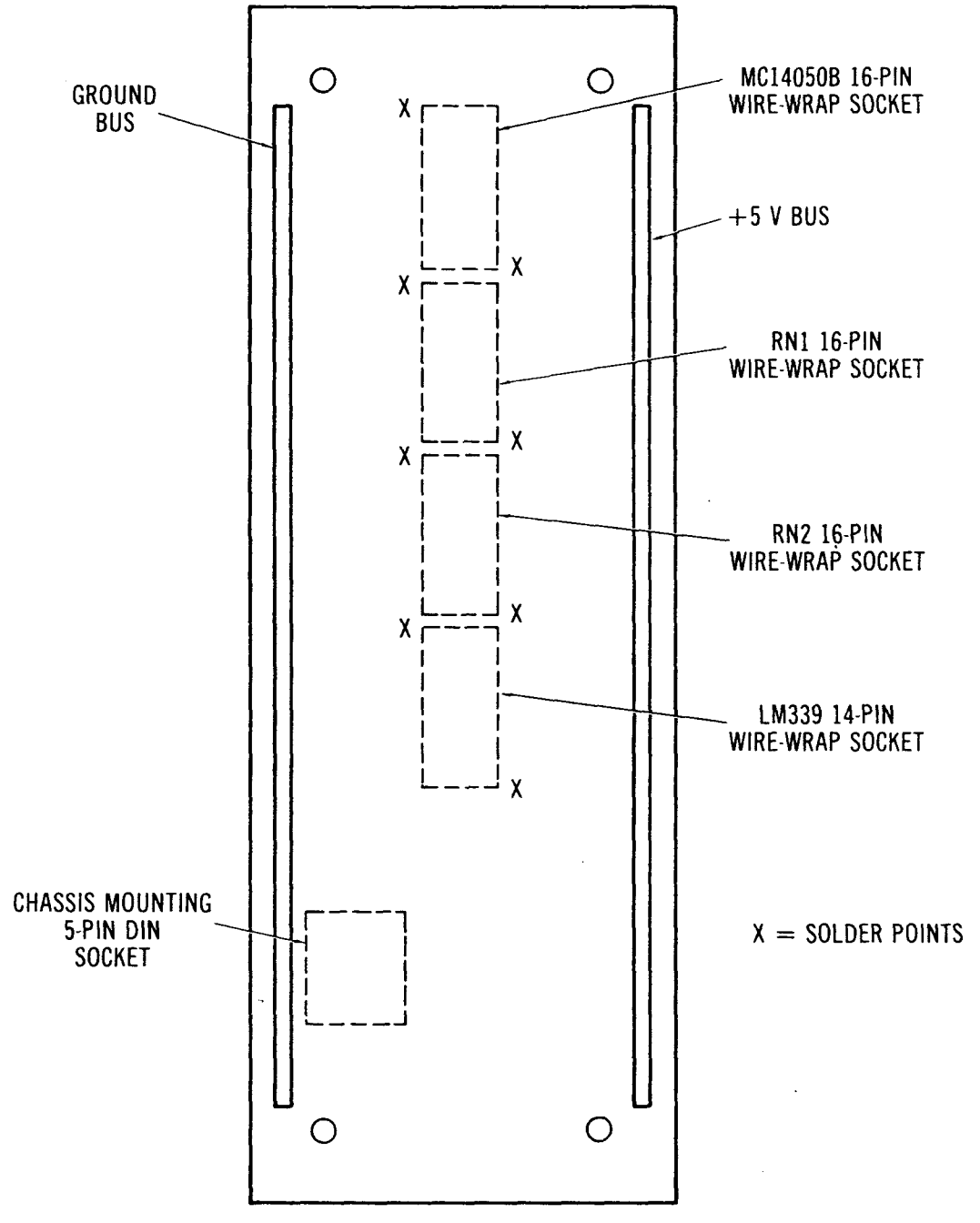


Fig. 4–9. Joystick circuitry parts placement.

55 rows with solder pads on the other. The spacing of the holes is compatible with the spacing on the pins of the four wire-wrap IC sockets. Mount the 4 IC sockets by soldering alternate corners of the socket, as shown in Fig. 4–9. Use the left-hand strip for the ground bus and the right-hand strip for the +5-volt bus. The four sockets are designated MC14050B, RN1, RN2, and LM339.

The 34-pin edge connector may be hard to find. Radio Shack is now carrying both it and a 40-pin edge connector. In a pinch, you can use the 40-pin edge connector for the Model I by inserting a cardboard "filler" in one side so that the edge connector is properly keyed. You'll have to use a 34-pin connector for the Model III, as the cutout in the cover will only pass a 34-pin width. I soldered the wires to the pins of the edge connector, even though the edge connector was really meant as an insulation-displacement type that pokes metal contacts through a ribbon cable. The pin layout for the edge connectors is shown in Fig. 4–10. The edge connector is designated EC.

The 5-pin DIN connector is another problem. If you plan on using the Color Computer joysticks, the matching 5-pin plug will probably have incompatible spacing. Consider cutting the joystick cable and attaching the plug to an audio-type DIN plug or attaching the wires directly. If you are using the joystick pot, you should be able to get a DIN chassis mounting plug and matching connector. The DIN plug is designated DIN.

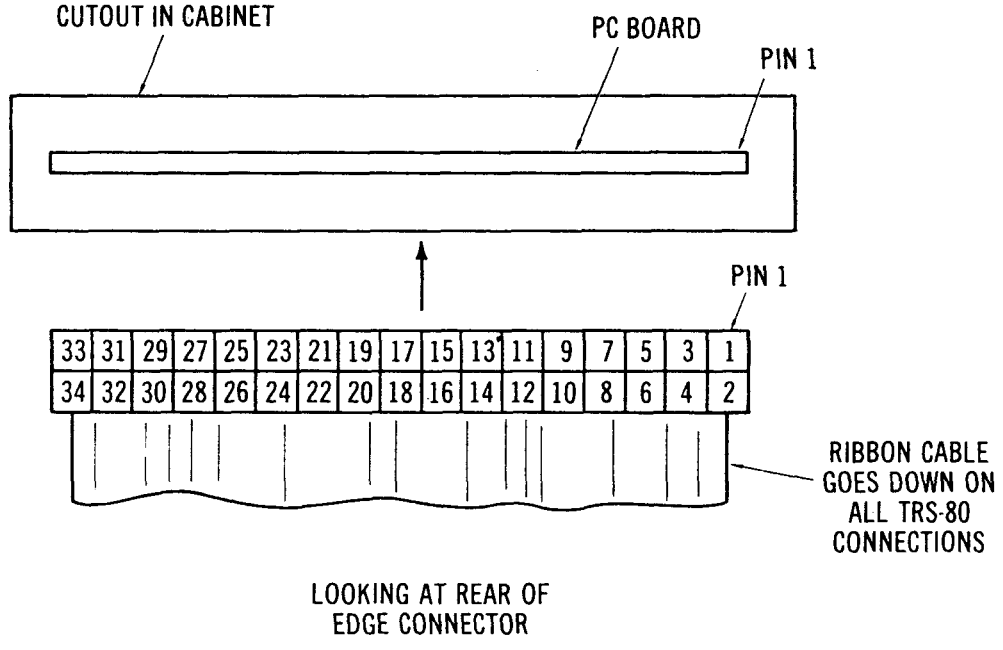


Fig. 4–10. Edge connector pins.

Table 4–2. Wire-Wrap List for Joystick Circuitry

34-Pin Edge Connector to Other Pins	EC–3 to MC14050B–3 EC–5 to MC14050B–14 EC–7 to MC14050B–5 EC–9 to MC14050B–11 EC–11 to MC14050B–7 EC–13 to MC14050B–9 EC–21 to LM339–2 EC–23 to LM339–1 EC–25 to DIN–4
MC14050B to RN1	MC14050B–2 to RN1–2 MC14050B–15 to RN1–3 MC14050B–4 to RN1–4 MC14050B–12 to RN1–5 MC14050B–6 to RN1–6 MC14050B–10 to RN1–7
RN1	RN1–16 to RN1–15 RN1–15 to RN1–14 RN1–14 to RN1–13 RN1–13 to RN1–12 RN1–12 to RN1–11 RN1–11 to RN1–10 RN1–10 to RN1–9 RN1–12 to LM339–5
RN2	RN2–1 to DIN–1 RN2–2 to LM339–2 RN2–2 to RN2–3 RN2–4 to DIN–2 RN2–5 to LM339–1 RN2–5 to RN2–6 RN2–10 to LM339–7 RN2–12 to LM339–6 RN2–12 to RN2–13 RN2–15 to LM339–4 RN2–15 to RN2–16
LM339	LM339–7 to LM339–5

Wire-Wrap Connections

Make the wire-wrap connections shown in Table 4–2. Most of these are wire-wrap to wire-wrap, although some will be wire-wrap to solder. All of these connections can be made with 30-gauge wire-wrap wire, although you might consider thin stranded wire for the edge connector leads. Route the edge connector leads through board holes for strain relief. Now connect the grounds shown in Table 4–3. You may wire-wrap ground pins on the same socket and then route one wire to the ground bus. Connect the +5-volt connections in Table 4–4 in similar fashion.

Power Connections

Now run four wires as shown in Fig. 4–11. Two hookup wire (22-gauge stranded) leads run from the ground bus. One +5-volt lead runs from the +5-volt bus. One +9-volt lead runs from pin 3 of the LM339. These leads may be in a 4-wire ribbon cable and routed through one hole for convenience. Two of the leads, one ground lead and the +9-volt lead, attach to a 9-volt transistor battery. The other two leads connect to a +5-volt supply. Leave the power leads unconnected for the time being.

Without plugging in any chips, test the connections by using a multimeter or continuity checker. A common pin works fine for getting into the IC socket pins. Cross off the circuit on the schematic as each path checks out. (The author had two miswires, probably a typical number.)

Table 4–3. Ground Connections for Joystick Circuitry

LM339-8, LM339-9, LM339-10, LM339-11, LM339-12, LM339-13, and LM339-14
MC14050B-8
RN1-8
DIN-3
RN2-7
EC-2

Table 4–4. The +5-Volt Connections for Joystick Circuitry

MC14050B-1	RN2-11 and RN2-14
RN1-1	DIN-5

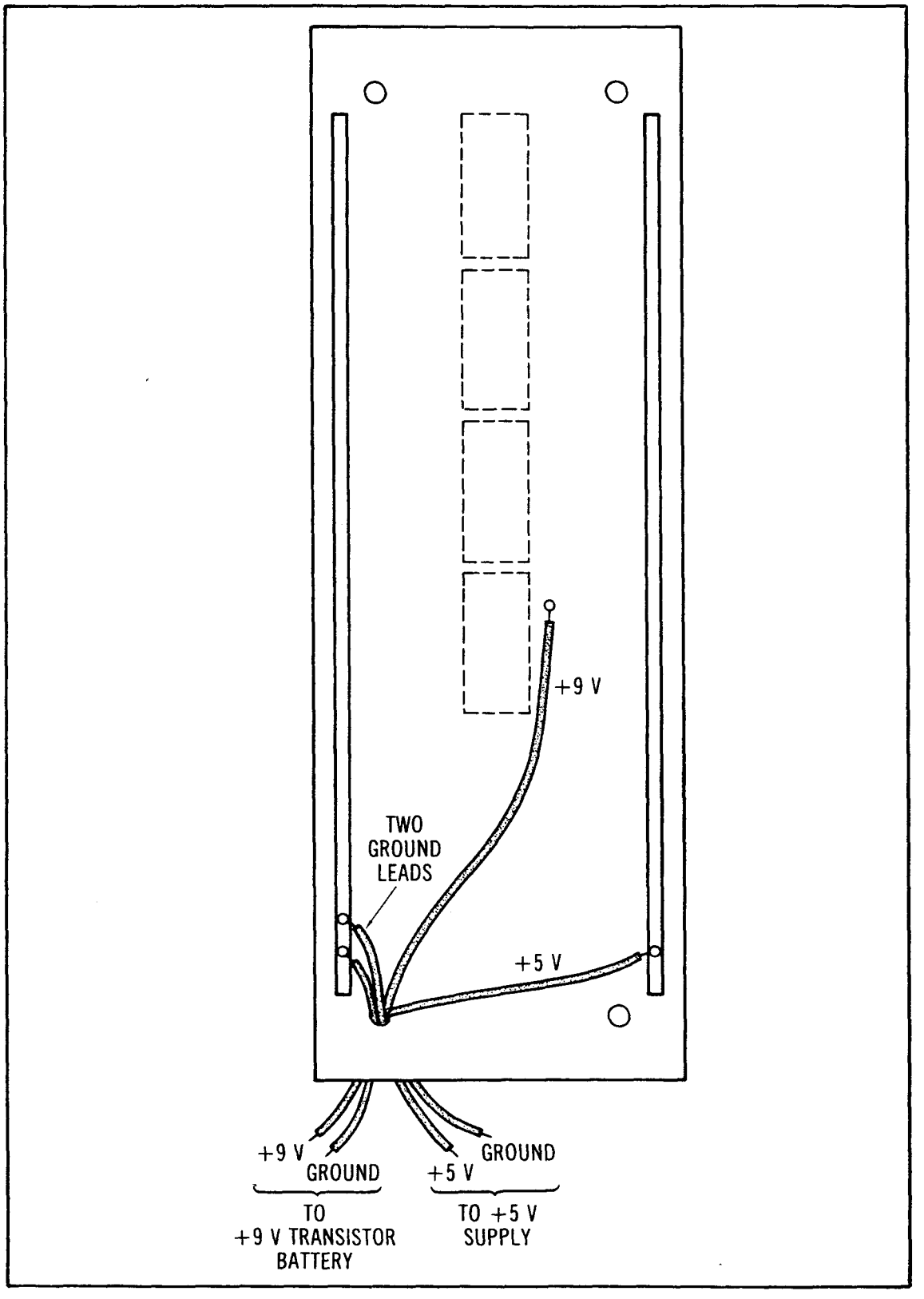


Fig. 4–11. Power connections.

This check takes very little time and is well worth it considering the grief that can be caused by connection errors.

Solder two 1.0-μF filter capacitors between +9 volts and ground and +5 volts and ground as shown in Fig. 4–8. Make certain that the polarity of the capacitors (note the + sign or − sign) is oriented in the right direction.

Construct two DIP headers as shown in Fig. 4–12. One of these will have the dac resistors, while the other has the resistors for the LM339. If you apply a lot of heat during the soldering you might want to remeasure the resistance values for the six dac resistors; they may have changed due to the heat.

Now, plug in the dip headers, the MC15050B, and the LM339. Connect the line printer connector (pin 1 is on the top right), turn on the Model I or III, and connect the +5 volts and +9 volts. Make the following test: Watch for smoke! Try a fingertip test of the board components. They should be warm but not hot. If everything seems all right, plug in the joystick connector and repeat the test. You're now read for program debugging.

PROGRAM TESTING

These preliminary tests are included as a means to "bring up" the circuit one step at a time. If you feel like going directly to the final

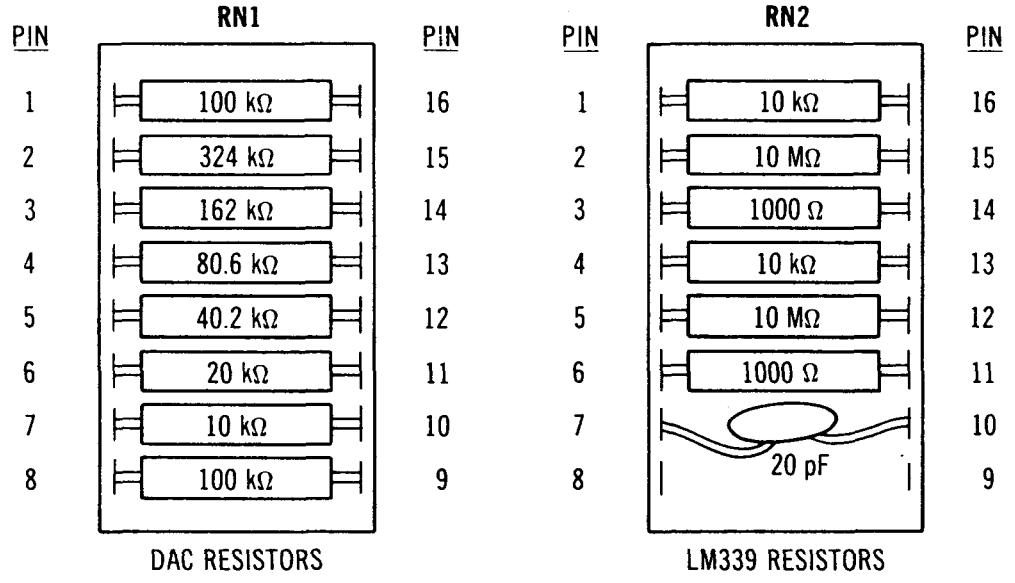


Fig. 4–12. DIP header layout.

program instead of following this procedure, by all means do so. If you have problems, fall back to these preliminary tests.

The first program tests the output of the dac. You'll need a voltmeter to run it. If you don't have a voltmeter, go on to the next test. Hook the voltmeter between the ground and the output of the dac, pin 12 of the MC15050B. Fig. 4–13 shows the program. Substitute 120 OUT 248,V for statement 120 if you are using a Model III. The program steps the dac over the range of voltages by outputting values of 000000 through 111111. Each increment should be about the same amount — 70 mV. Table 4–5 shows the values obtained with the prototype.

If you do not get what my calculus instructor called a "monotonically increasing" set of voltages (see Fig. 4–14), you have a problem. If any successive output is lower than the previous, you must recheck the resistance values. If any resistor is off by a considerable percentage, it is possible to get, say, a voltage output of +2.50 for an output of 31 and a +2.48 for an output of 32. This will lead to problems in finding the proper value if not corrected.

Figs. 4–15 and 14–16 show the comparator tests for the Models I and III, respectively. This test steps the dac from 0 through 63 (+0.25 through +4.75 volts) and displays the step number, X input, and Y input. The X and Y inputs will be either 0 or 1. If the input is a 0, the X or Y voltage is less than the current dac voltage. Move the joystick and observe that the comparator inputs change. Moving the joystick to the upper left corner should reset both comparator inputs to 0 after several steps, for example. Also observe that when the input changes from 0 to 1 that successive inputs remain at 1. If there is a 1 followed by several zeros, you have the "not monotonically increasing" problem.

If all seems well with this test, you're ready for a machine-language driver for the joysticks. Figs. 4–17 and 4–18 show Z-80 drivers for the Models I and III, respectively. The only difference is that one uses a memory-mapped LD, while the other uses I/O-mapped INs and OUTs. Both programs are completely relocatable, even though assembled at 8000H. You may reassemble using your own editor/assembler, or simply key in the program using DEBUG. Another alternative is to convert the hex code to decimal and incorporate the 62 bytes in a DATA statement which is then used to fill a block of memory.

The calling sequence in disk BASIC is the same for both the Models I and III. It's shown in Fig. 4–19. This program clears the screen with 128 bytes (graphics zero) and defines the USR0 routine at 8000H. Next, a call

```
100 REM DAC TEST. OUTPUT VOLTAGES FROM 0 TO 63
110 FOR V=0 TO 63
120 POKE 14312,V
130 CLS: PRINT @ 534,"DAC VALUE=";V
140 IF INKEY$="" GOTO 140
150 NEXT V
```

Fig. 4–13. BASIC program for dac testing.

Table 4–5. Dac Values Obtained for Prototype

Binary	Dac Output	Binary	Dac Output
0	0.240	32	2.48
1	0.312	33	2.55
2	0.387	34	2.63
3	0.460	35	2.70
4	0.530	36	2.77
5	0.602	37	2.84
6	0.677	38	2.92
7	0.749	39	2.99
8	0.785	40	3.03
9	0.857	41	3.10
10	0.932	42	3.18
11	1.005	43	3.25
12	1.075	44	3.32
13	1.147	45	3.39
14	1.222	46	3.47
15	1.294	47	3.54
16	1.419	48	3.67
17	1.492	49	3.74
18	1.568	50	3.82
19	1.640	51	3.89
20	1.710	52	3.96
21	1.782	53	4.04
22	1.858	54	4.12
23	1.930	55	4.19
24	1.966	56	4.22
25	2.03	57	4.30
26	2.11	58	4.37
27	2.18	59	4.44
28	2.25	60	4.52
29	2.32	61	4.59
30	2.40	62	4.67
31	2.47	63	4.74

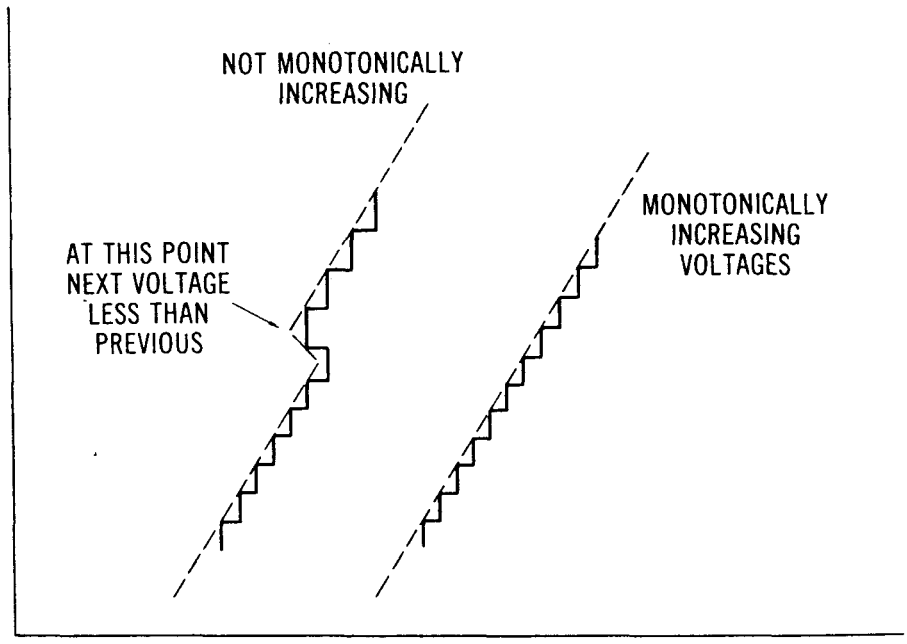


Fig. 4–14. Dac output problem is indicated by the output on the left.

```
100 REM COMPARATOR TEST
110 FOR V=0 TO 63
120 POKE 14312,V: CLS
130 PRINT @ 520,"VALUE=";V;
140 PRINT @ 540,"X=";(PEEK(14312) AND 128)/128;
150 PRINT @ 560,"Y=";(PEEK(14312) AND 64)/64;
160 FOR I=0 TO 1000:NEXT I
170 NEXT V
```

Fig. 4–15. BASIC program for the Model I comparator test.

```
100 REM COMPARATOR TEST
110 FOR V=0 TO 63
120 OUT 248,V: CLS
130 PRINT @ 520,"VALUE=";V;
140 PRINT @ 540,"X=";(INP(248) AND 128)/128;
150 PRINT @ 560,"Y=";(INP(248) AND 64)/64;
160 FOR I=0 TO 1000:NEXT I
170 NEXT V
```

Fig. 4–16. BASIC program for the Model III comparator test.

is made of USR0. The X,Y position of the joystick is returned as variable A. The X position is in the most significant byte, and the Y position is in the least significant byte.

Both X and Y are returned as values of 0 through 63. The X value (B) is multiplied by 2 and used in a SET command. The Y value is converted from 0 through 63 to 0 through 48 and used in the same SET command. As long as the cursor position remains fixed, one pixel of the SET appears on the screen. If the joystick is moved, the last pixel is RESET and the new one SET. The display will be a one-pixel display of the joystick position on a clear screen.

```
8000            00100          ORG     8000H
                00110 ;*****************************************************
                00120 ;* SUBROUTINE TO READ JOYSTICK                       *
                00130 ;*      ENTRY: NO PARAMETERS                         *
                00140 ;*      EXIT:  (H,L)=X VALUE 0-63, Y VALUE 0-63      *
                00150 ;* SUBROUTINE IS RELOCATABLE ANYWHERE IN RAM. SUBROU- *
                00160 ;* TINE IS SETUP FOR STANDARD MODEL I/III BASIC USR  *
                00170 ;* CALL.                                             *
                00180 ;*****************************************************
                00190 ;
8000 0E80       00200 READJY   LD      C,128          ;MASK FOR X
8002 180A       00210          JR      SRCHJY         ;READ X VALUE
8004 F5         00220 REA010   PUSH    AF             ;SAVE X VALUE
8005 0E40       00230          LD      C,64           ;MASK FOR Y VALUE
8007 1805       00240          JR      SRCHJY         ;READ Y VALUE
8009 E1         00250 REA020   POP     HL             ;X TO H
800A 6F         00260          LD      L,A            ;Y TO L
800B C39A0A     00270          JP      0A9AH          ;***BASIC RTN***
                00280 ;
                00290 ;*****************************************************
                00300 ;* SUBROUTINE TO SEARCH FOR X OR Y VALUE             *
                00310 ;*      ENTRY: (C)=128 FOR X, 64 FOR Y               *
                00320 ;*      EXIT:  (A)=ANALOG VALUE 0-63                 *
                00330 ;* SUBROUTINE FINDS ANALOG VALUE WITH 8 RETRIES.     *
                00340 ;*****************************************************
                00350 ;
800E 21FFFF     00360 SRCHJY   LD      HL,-1          ;DUMMY VALUE FOR COMPARE
8011 E5         00370          PUSH    HL             ;INITIALIZE LAST VALUE
8012 21E837     00380          LD      HL,37E8H       ;PRINTER ADDRESS
8015 0608       00390          LD      B,8            ;8 TRIES
8017 1640       00400 SRC005   LD      D,40H            ;START VALUE
8019 1E20       00410          LD      E,20H            ;START DELTA
801B CB1A       00420 SRC010   RR      D                  ;ALIGN TO H'WARE FORM
801D 72         00430          LD      (HL),D             ;OUTPUT VALUE TO DAC
801E CB12       00440          RL      D                  ;BACK TO SCALED DELTA
8020 7E         00450          LD      A,(HL)             ;GET COMPARATOR INP
8021 A1         00460          AND     C                  ;TEST CHANNEL
8022 7A         00470          LD      A,D                ;CURRENT VALUE TO A
8023 2003       00480          JR      NZ,SRC020          ;GO IF COMP=1
8025 83         00490          ADD     A,E                ;TOO LOW-ADD 1/2
8026 1801       00500          JR      SRC030             ;CONTINUE
8028 93         00510 SRC020   SUB     E                  ;TOO HIGH-SUB 1/2
8029 57         00520 SRC030   LD      D,A                ;SAVE ADJUSTED VALUE
802A CB3B       00530          SRL     E                  ;DELTA/2
802C 20ED       00540          JR      NZ,SRC010          ;GO IF DELTA NOT 0
802E CB3A       00550          SRL     D                ;CONVERT TO 0-63 FORM
8030 F1         00560          POP     AF               ;GET LAST VALUE
8031 BA         00570          CP      D                ;TEST WITH CURRENT
8032 D5         00580          PUSH    DE               ;SAVE CURRENT
8033 2802       00590          JR      Z,SRC040         ;GO IF EQUAL
8035 10E0       00600          DJNZ    SRC005           ;NOT EQUAL-8 RETRIES
8037 F1         00610 SRC040   POP     AF             ;RESTORE LAST
8038 CB79       00620          BIT     7,C            ;TEST FOR RETURN POINT
803A 20C8       00630          JR      NZ,REA010      ;X CASE
803C 18CB       00640          JR      REA020         ;Y CASE
8000            00650          END     READJY
00000 Total errors
```

Fig. 4–17. Assembly language driver for the Model I.

The pixel may have some jitter. It may have a tendency to jump from one spot to the next. This is normal and occurs when the voltage increment is close to the input voltage value. For most positions, however, the pixel will be fixed in the one position. Although a resolution of 64 X and 48 Y does not seem like a precise enough resolution, it is more than adequate for positioning the joystick. The mechanical limitations of the joystick make it very difficult to remain on the same horizontal row when moving across, and greater resolution, as with 7 bits instead of 6, would be overkill.

```
8000            00100           ORG     8000H
                00110 ;*******************************************************
                00120 ;* SUBROUTINE TO READ JOYSTICK                         *
                00130 ;*      ENTRY: NO PARAMETERS                           *
                00140 ;*      EXIT:  (H,L)=X VALUE 0-63, Y VALUE 0-63        *
                00150 ;* SUBROUTINE IS RELOCATABLE ANYWHERE IN RAM. SUBROU-  *
                00160 ;* TINE IS SETUP FOR STANDARD MODEL I/III BASIC USR    *
                00170 ;* CALL.                                               *
                00180 ;*******************************************************
                00190 ;
8000 0E80       00200 READJY  LD      C,128           ;MASK FOR X
8002 180A       00210         JR      SRCHJY          ;READ X VALUE
8004 F5         00220 REA010  PUSH    AF              ;SAVE X VALUE
8005 0E40       00230         LD      C,64            ;MASK FOR Y VALUE
8007 1805       00240         JR      SRCHJY          ;READ Y VALUE
8009 E1         00250 REA020  POP     HL              ;X TO H
800A 6F         00260         LD      L,A             ;Y TO L
800B C39A0A     00270         JP      0A9AH           ;***BASIC RTN***
                00280 ;
                00290 ;*******************************************************
                00300 ;* SUBROUTINE TO SEARCH FOR X OR Y VALUE               *
                00310 ;*      ENTRY: (C)=128 FOR X, 64 FOR Y                 *
                00320 ;*      EXIT:  (A)=ANALOG VALUE 0-63                   *
                00330 ;* SUBROUTINE FINDS ANALOG VALUE WITH 8 RETRIES.       *
                00340 ;*******************************************************
                00350 ;
800E 21FFFF     00360 SRCHJY  LD      HL,-1           ;DUMMY VALUE FOR COMPARE
8011 E5         00370         PUSH    HL              ;INITIALIZE LAST VALUE
8012 0608       00380         LD      B,8             ;8 TRIES
8014 1640       00390 SRC005  LD      D,40H             ;START VALUE
8016 1E20       00400         LD      E,20H             ;START DELTA
8018 CB1A       00410 SRC010  RR      D                   ;ALIGN TO H'WARE FORM
801A 7A         00420         LD      A,D                 ;PUT IN A FOR OUTPUT
801B D3F8       00430         OUT     (0F8H),A            ;OUTPUT VALUE TO DAC
801D CB12       00440         RL      D                   ;BACK TO SCALED DELTA
801F DBF8       00450         IN      A,(0F8H)            ;GET COMPARATOR INP
8021 A1         00460         AND     C                   ;TEST CHANNEL
8022 7A         00470         LD      A,D                 ;CURRENT VALUE TO A
8023 2003       00480         JR      NZ,SRC020           ;GO IF COMP=1
8025 83         00490         ADD     A,E                 ;TOO LOW-ADD 1/2
8026 1801       00500         JR      SRC030              ;CONTINUE
8028 93         00510 SRC020  SUB     E                   ;TOO HIGH-SUB 1/2
8029 57         00520 SRC030  LD      D,A                 ;SAVE ADJUSTED VALUE
802A CB3B       00530         SRL     E                   ;DELTA/2
802C 20EA       00540         JR      NZ,SRC010           ;GO IF DELTA NOT 0
802E CB3A       00550         SRL     D                 ;CONVERT TO 0-63 FORM
8030 F1         00560         POP     AF                ;GET LAST VALUE
8031 BA         00570         CP      D                 ;TEST WITH CURRENT
8032 D5         00580         PUSH    DE                ;SAVE CURRENT
8033 2802       00590         JR      Z,SRC040          ;GO IF EQUAL
8035 10DD       00600         DJNZ    SRC005            ;NOT EQUAL-8 RETRIES
8037 F1         00610 SRC040  POP     AF              ;RESTORE LAST
8038 CB79       00620         BIT     7,C             ;TEST FOR RETURN POINT
803A 20CB       00630         JR      NZ,REA010       ;X CASE
803C 18CB       00640         JR      REA020          ;Y CASE
8000            00650         END     READJY
00000 Total errors
```

Fig. 4–18. Assembly language driver for the Model III.

HOW THE PROGRAM WORKS

The program in Figs. 4–17 and 4–18 consists of two parts. SRCHJY is the actual search program that finds the comparator value for the current joystick channel. This program is called twice by driver routine READJY. The CALL is made by loading the C register with 128 or 64 and doing a JR to SRCHJY. The JR keeps the code relocatable and allows the program to be located anywhere in memory.

```
100 REM JOYSTICK-CONTROLLED CURSOR
110 FOR I=15360 TO 16383
120 POKE I,128
130 NEXT I
140 D=0: E=0
150 DEFUSR0=&H8000
160 A=USR0(0)
170 B=INT(A/256)
180 C=(A-B*256)*47/63
190 IF (D<>B OR E<>C) THEN RESET (D*2,E)
200 SET (B*2,C)
210 D=B: E=C
220 GOTO 160
```

Fig. 4–19. Disk BASIC calling sequence.

The value in C serves two purposes: it acts as a flag for the return point and serves as a mask value for the X/Y comparator bit. The X-channel comparator bit is found by ANDing the result of the read from the line printer port with 128, and the Y-channel comparator bit is found by ANDing the result of the read with 64.

READJY calls SRCHJY twice and merges the result into the HL register for return. H will contain an X value of 0 through 63 and L a Y value of 0 through 63. The JP 0A9AH is the standard BASIC method of returning an argument to BASIC from a machine-language subroutine. Convert this to a normal RET if the program will be stand-alone (non-BASIC).

The SRCHJY subroutine operates similarly to the Color Computer joystick subroutine. A successive approximation analog-to-digital conversion is performed. A start value of 32, or half the voltage range, is first output to the dac. A delta value of 16 is initialized. The comparator output is then read in. Depending upon the comparator output, the next value tried is 32 plus or minus the delta. The delta is then halved. This successive approximation continues until the delta has been reduced to ½ (the value is scaled up by two to permit the last delta of ½).

As the input may change rapidly, eight tries are made to obtain the same X or Y input value. The minimum number of times through SRCHJY will be two, the maximum eight. If the value does not match the previous value after eight tries, the last value is used.

USING THE JOYSTICK INPUT FOR ANALOG-TO-DIGITAL CONVERSION

In Chapter 2, we described some real-world analog inputs that can be used in place of the joystick of a Color Computer. You may want to refer to that chapter; the techniques are identical for this a/d converter. Basi-

cally, anything that can be converted into a voltage can be used as an input to the DIN connector and converted to an increment of 0 through 63.

The examples we used in Chapter 2 were a cadmium sulfide photocell which has a variable resistance dependent upon the amount of light striking it. When used with a resistor in a divider network, a varying input voltage is generated. The second example used was a thermistor, a resistor whose resistance varied inversely with temperature. Many other devices may be connected to the joystick input in this fashion. Of course, two devices may be used simultaneously, one for the X channel and one for the Y channel.

chapter **5**

Cheap and Easy Model III Analog-to-Digital Conversion

The joystick controller described in Chapter 4 is also an analog-to-digital converter for the Model III. The controller applies the two voltage analogs of the X and Y joystick positions to an analog-to-digital converter (adc). The dac output zeros in on the analog input voltage by the use of a comparator. In this chapter, we will show you a much neater implementation of an adc, one that uses only three chips and can be hooked directly to the cassette port of the Model III, thus eliminating a great deal of wire-wrapping and connector preparation. In fact, you can utilize the existing cassette cable, the assembly that plugs into the 5-pin DIN jack on the back of the Model III.

This adc is also a little bit better than the earlier version in that it will convert an analog voltage to 7-bit resolution instead of 6 bits. In addition, it's extremely accurate, down to about 20 mV for a 2.5-volt range. One disadvantage, however, is that this adc allows only a half dozen or so samples per second. This is no detriment, though, if you are monitoring slowly changing real-world events such as temperature, pressure, or ambient light.

BASICS OF THE MODEL III CASSETTE PORT

The Model III is capable of writing data on cassette tape in either 500- or 1500-baud format. The circuitry used to perform the output is identical; the 500- and 1500-baud data formats and software, however, are different. The 500-baud tape format is shown in Fig. 5–1. Each bit time is made up of a leading clock pulse followed by a second pulse for a 1 or no pulse for a 0. The duration of each pulse is about 250 microseconds (top and bottom). The 1500-baud tape format is shown in Fig. 5–2. Here,

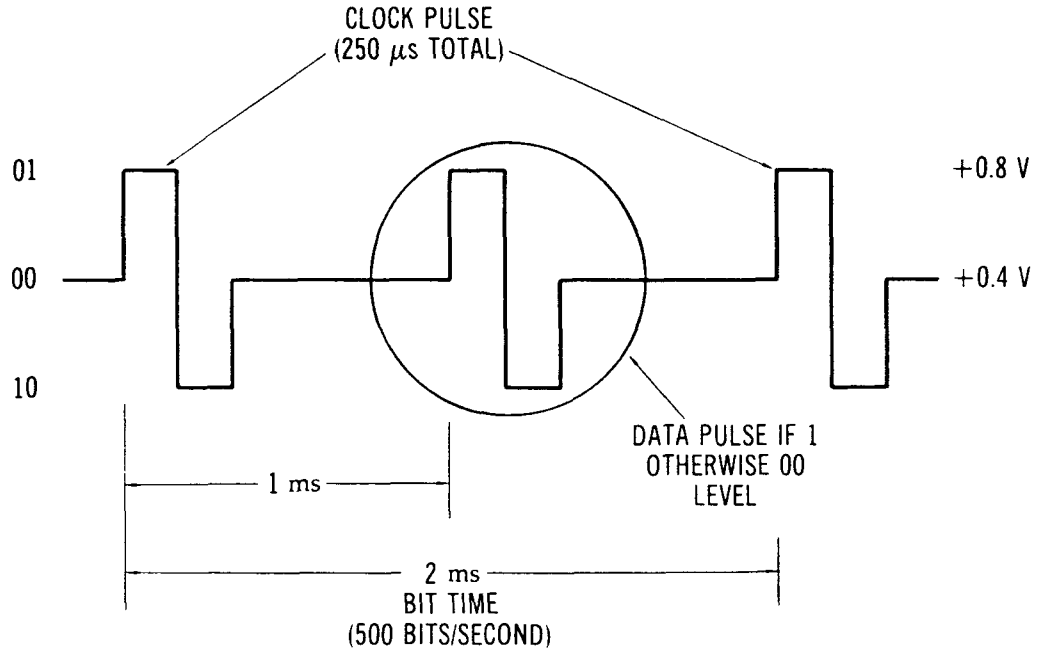


Fig. 5–1. 500-baud tape format.

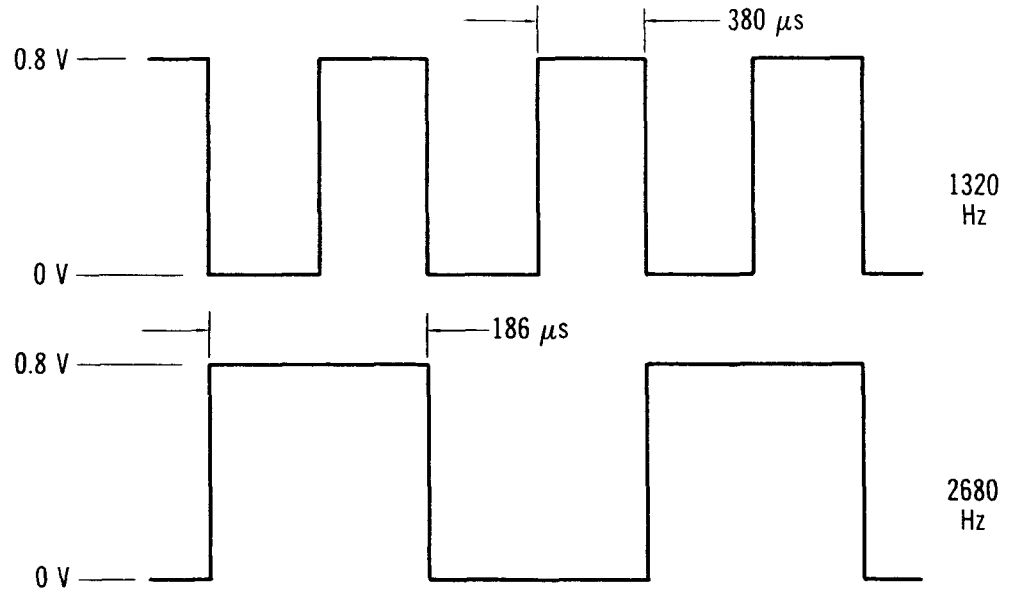


Fig. 5–2. 1500-baud tape format.

the format is that of frequency-shift keying, one frequency for a 0, and a second frequency for a 1 bit.

Both formats use software that drives a simple circuit that produces three voltage levels (only the two extremes are used for 1500 baud), 0 volts, 0.46 volt, and 0.85 volt (see Fig. 5–3). The levels are produced by outputs to a 2-bit latch with input/output port address 0FFH. The latch

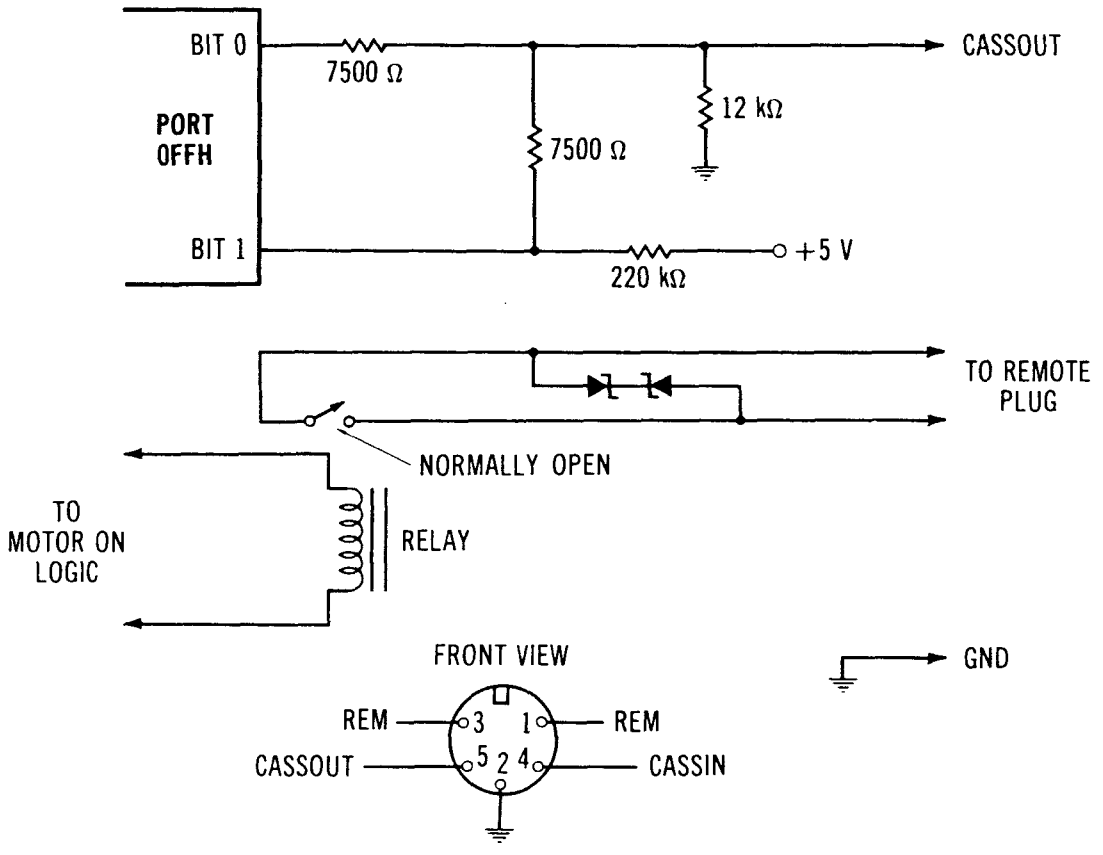


Fig. 5–3. Cassette tape output signals.

simply records the two least-significant bits output to I/O address 0FFH. If a XXXXXX00 is output, a 0.46-volt level is produced. XXXXXX01 produces a 0.85-volt level, and XXXXXX10 produces a 0-volt level.

Toggling the latch bits in an assembly language program can produce any frequency square-wave output, up to the limits of the electronics in the cassette circuitry. Unfortunately, this action must be in assembly language, by an OUT (0FFH instruction), as BASIC is much too slow to achieve the relatively short pulses required. A BASIC OUT (255) performs the same function at much slower speeds.

The Model III has two separate circuits for reading cassette data, one for 500 baud and one for 1500 baud. The 500-baud circuit works by essentially rectifying the incoming pulse and then looking for the dc level from the rectification. (It's very similar to the Model I circuitry.) The 1500-baud circuit uses a much different approach, as shown in Fig. 5–4.

The circuit in Fig. 5–4 is a zero-crossing detector that looks for negative pulses. The output of the LM339 comparator is a zero with no input or a positive pulse, but it switches to a one with the negative-going edge

of the pulse. It remains a one until the pulse switches back to a positive level, as shown in Fig. 5–5.

The CASSDIN (cassette data in) bit is read by performing an IN (0FFH) and checking bit 0, the 1500-baud cassette bit (bit 7 is the 500-baud bit). Again, assembly language allows us to sample the CASSDIN bit hundreds of tens of times per second. A BASIC INP (255) will also perform the same function, but at much lower speeds. It appears, then, that we can easily output square waves and read in a voltage level. How is this going to help us implement an analog-to-digital converter?

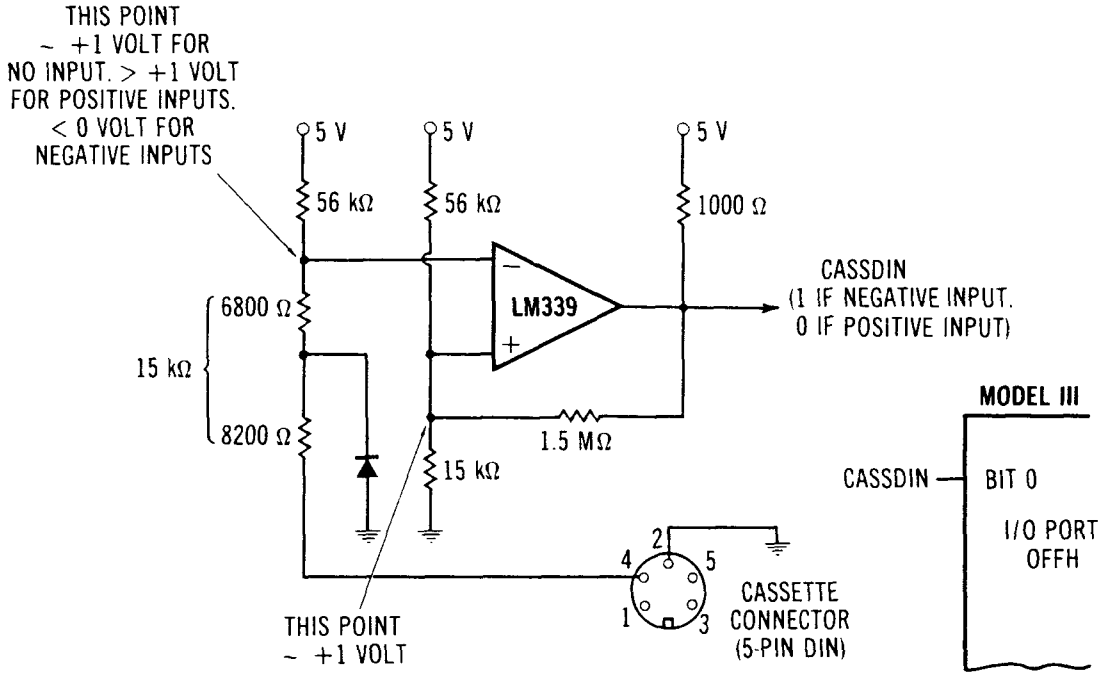


Fig. 5–4. Cassette tape input logic.

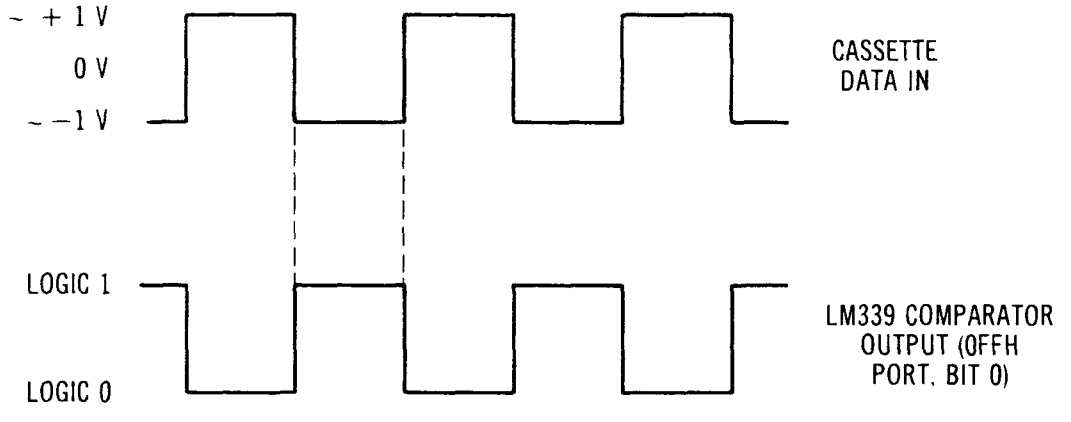


Fig. 5–5. Zero-crossing detector signals.

A "DIAMOND" IN THE "SHACK"

While paging through the Radio Shack catalog, I happened to see an adc chip called the TL507C. At first glance, it didn't appear very imposing (it's an 8-pin device and rather ugly, as semiconductors go), but it turned out to be a diamond in the Shack.

The internal workings of the TL507C are shown in Fig. 5–6. It uses a ramp method of adc conversion, and is made up of a 7-bit counter, a few gates, and two comparators. The counter is the heart of the chip. It simply counts from 127 through 0 and repeats this cycle continuously. The counter may be reset at any time by bringing up the RESET input to a high (V_{cc}) level. If the RESET is held at ground, the counter is powered up to some indeterminate state, but settles into the 128-bit count cycle within 128 counts.

The output of the 7 counter bits is connected to binary-weighted resistors. The resistors are actually a digital-to-analog converter resistor ladder network. Each output of the counter produces a voltage that is twice as great as the preceding stage. The analog voltages are added together in the operational amplifier to produce the analog voltage corresponding to the value in the 7 bits of the counter. As the counter counts from 127 through 0, a ramp of voltages is generated, with each step of the ramp changing the output voltage by 1/128th of full scale (see Fig. 5–7).

The TL507C is designed so that one of two voltages may be used for power. The V_{cc_1} input is the normal power supply input and may range from +5 to +6 volts dc. The V_{cc_2} input may be used instead; in this case

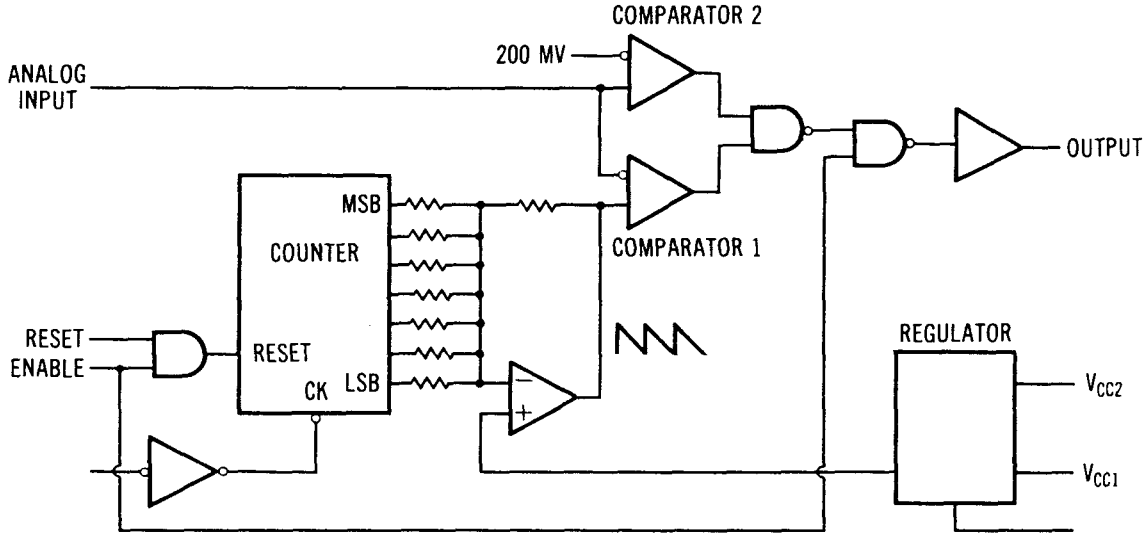


Fig. 5–6. TL507C internal logic.

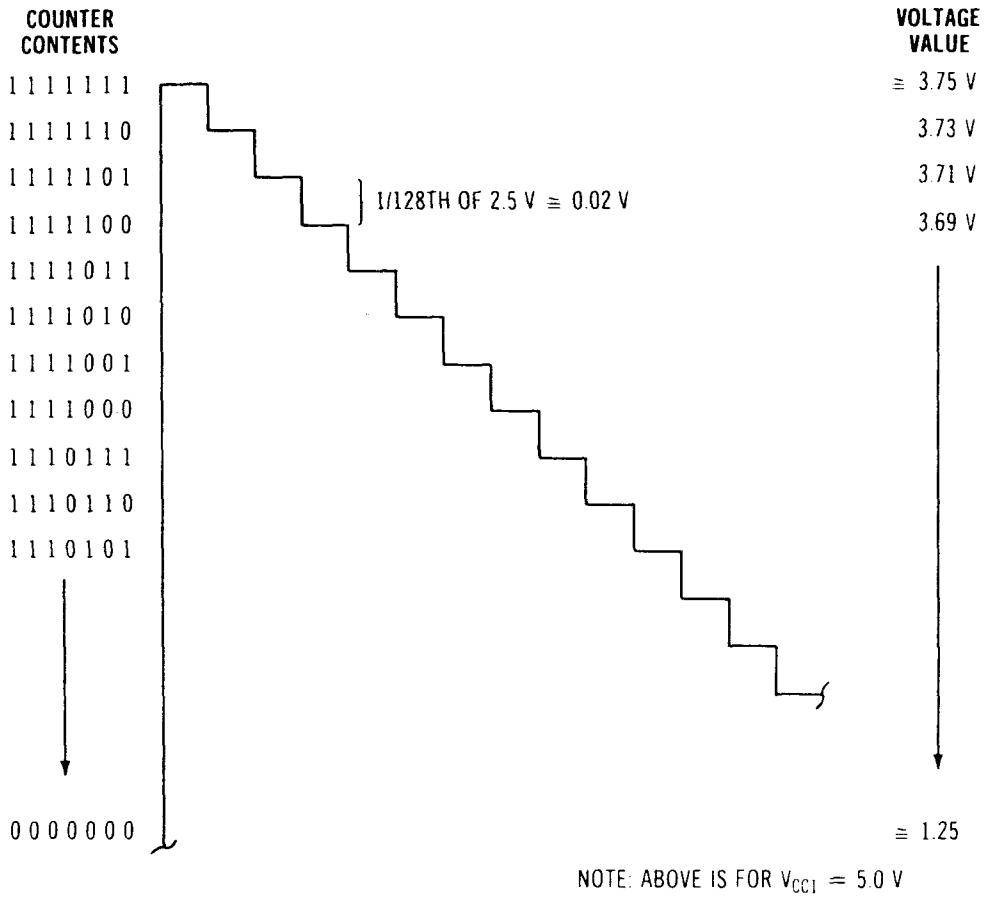


Fig. 5–7. Ramp voltages from the TL507C.

the power supply may be unregulated and may range from 8 to 16 volts. In the design that we've used here, we've chosen to use V_{CC1} and not V_{CC2}. In this case, the V_{CC2} input is simply not connected.

The ramp voltage generated during a count cycle ranges from 0.75 of V_{CC1} through 0.25 of V_{CC1}. This range of voltages is accurately controlled by the TL507C. With a V_{CC1} of +5 volts, therefore, the ramp will range from +3.75 through +1.25, as shown in Fig. 5–8.

The ramp output goes to comparator 1. This comparator compares the analog input voltage and the ramp voltage. The output of the comparator is a 0 or 1 depending upon whether the analog input voltage is greater (0) or less than (1) the ramp voltage. For any given analog input voltage, the comparator output will generate a square wave as shown in the square wave labeled TL507C output in Fig. 5–8. As the ramp repeats continuously (with a constant stream of clock pulses), the duty cycle (relationship of on time to total cycle time) will vary with the analog input voltage. With a large analog input voltage, the comparator output will quickly go

to 0, while with a small analog input voltage, the comparator output will reach 0 near the end of the ramp. The width of an on pulse, therefore, is directly proportional to the analog input voltage. If this pulse width can be measured, the analog input voltage can easily be determined. This

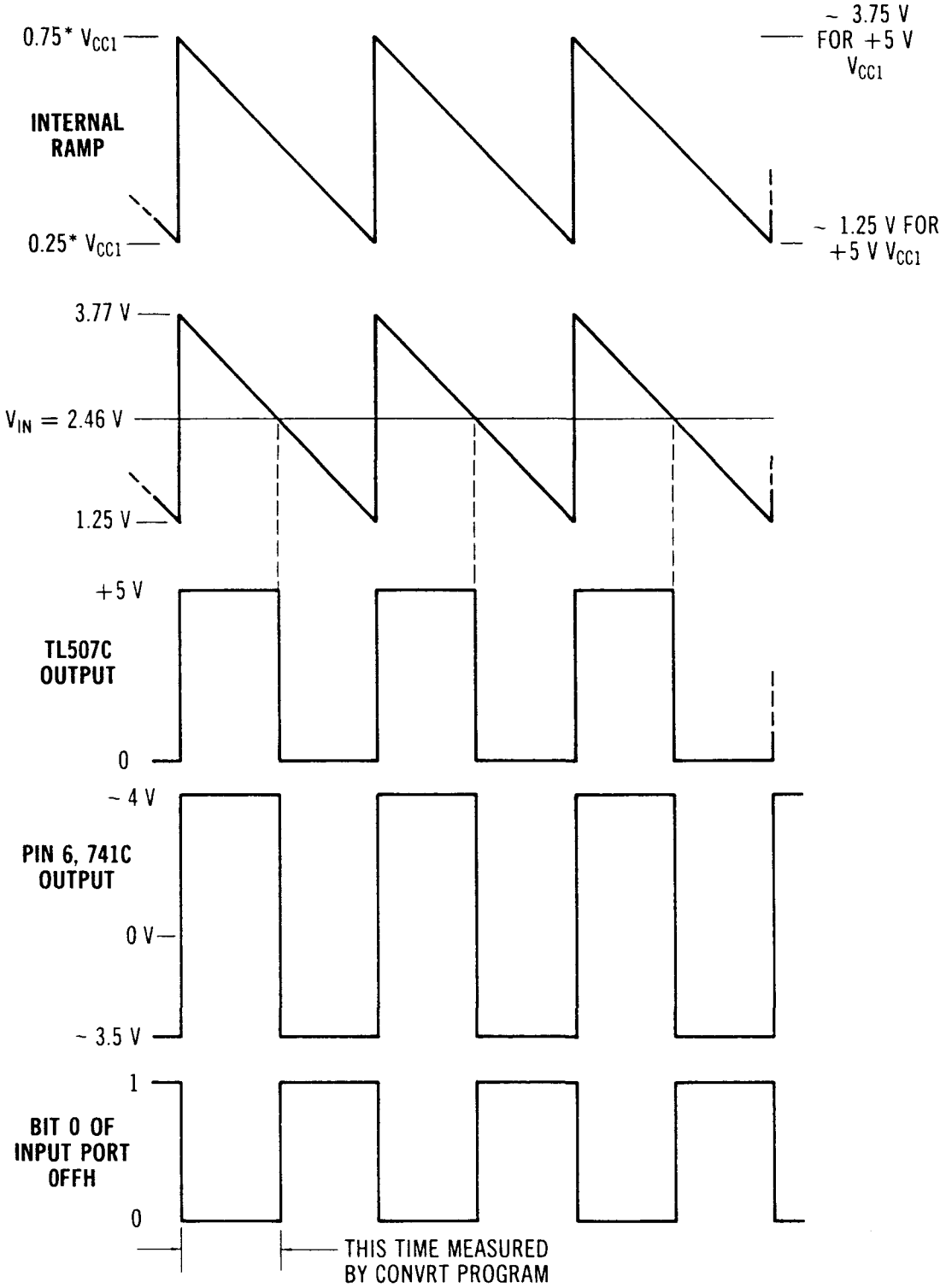


Fig. 5–8. Adc waveforms.

approach would be used in a strictly hardware implementation of the TL507C circuit; that is, measurement of the duty cycle of the output waveform.

An alternative approach, however, is to increase the ramp voltage by outputting a single clock pulse, comparing the comparator output, pulsing the clock again, and so forth until the comparator output changes. If we know how many clock pulses we have output, we know the duty cycle. The total number of clock pulses in a ramp cycle is 128 and the duty cycle will be:

$$\text{Duty Cycle (\%)} = \frac{\text{No. pulses before comparator switch}}{128} \times 10$$

This is the scheme we use in our implementation of the adc, controlling the clock pulse output and testing the comparator output.

Getting back to the TL507C internal diagram, there are a few remaining points to consider. The adc is set up so that an analog input of less than 200 mV disables the output. This level of voltage is considered to be an invalid input signal. Secondly, there's a general enable input for the device called ENABLE. If this input is grounded, the device OUTPUT will remain high (1). So, we keep the ENABLE input permanently active by tying it to V_{cc}.

The truth table for the TL507C is shown in Table 5–1. Under normal circumstances, the only conditions we'll be working with would be the last two entries, for analog input voltages greater than 25% and less than 75% of V_{cc}.

Table 5–1. Truth Table for TL507C

Analog Input Condition	Enable	Output
X	L†	H
$V_i < 200$ mV	H	L
$V_{ramp} > V_i > 200$ mV	H	H
$V_i > V_{ramp}$	H	L

†Low level on enable also inhibits the next function.
H = high level, L = low level, X = irrelevant

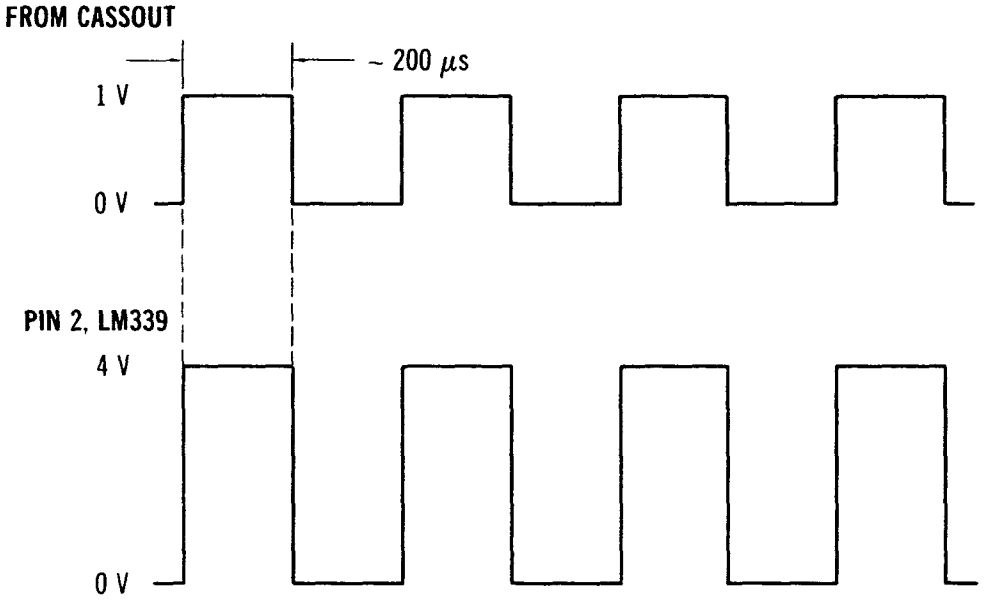


Fig. 5–10. CASSOUT signal.

of the LM339 is shown in Fig. 5–10; it is identical to the CASSOUT signal except that it has a greater voltage swing.

The CASSOUT signal is the clock to the TL507C. Each pulse will be (arbitrarily) set to about 200 μs wide. The counter in the TL507C changes on the negative-going portion of the clock pulse. A complete ramp will occur over 128 clock periods.

The CASSIN Signal

The cassette input circuitry in the Model III responds to a negative voltage level. The output voltage of the open collector OUTPUT line from the TL507C swings from about 0 volts to about +5 volts and, therefore, must be converted to a waveform that swings both positive and negative. The 741C accomplishes this by comparing the OUTPUT voltage with a voltage of about 1.2 volts at the junction of the 3900- and 1200-ohm resistors. The output of the 741C follows the OUTPUT of the TL507C as shown in Fig. 5–8.

Other Connections

The ENABLE input of the TL507C is tied to +5 volts, making the chip always active; V_{CC2} is not connected. RESET is tied to ground. The

THE CHEAP AND EASY CIRCUIT

The circuit for the Model III adc is shown in Fig. 5–9. The heart of it, of course, is the TL507C. The LM339 comparator is used for level conversion of the Model III cassette output signal, while the 741C is used for level conversion to a Model III compatible cassette input signal.

The CASSOUT Signal

The clock input to the TL507C is derived from the Model III cassette output. This would normally be the signal that goes to the AUX input of the cassette recorder during a write tape operation. The CASSOUT signal is shown in Fig. 5–10. It is a 0 to 1-volt square wave.

The clock input to the TL507C must be at least 2.5 volts, and this calls for some level conversion of the CASSOUT signal. The LM339 accomplishes this by comparing the CASSOUT signal to a voltage of about 0.25 volt at the junction of the 6800-ohm and 330-ohm resistors. The output

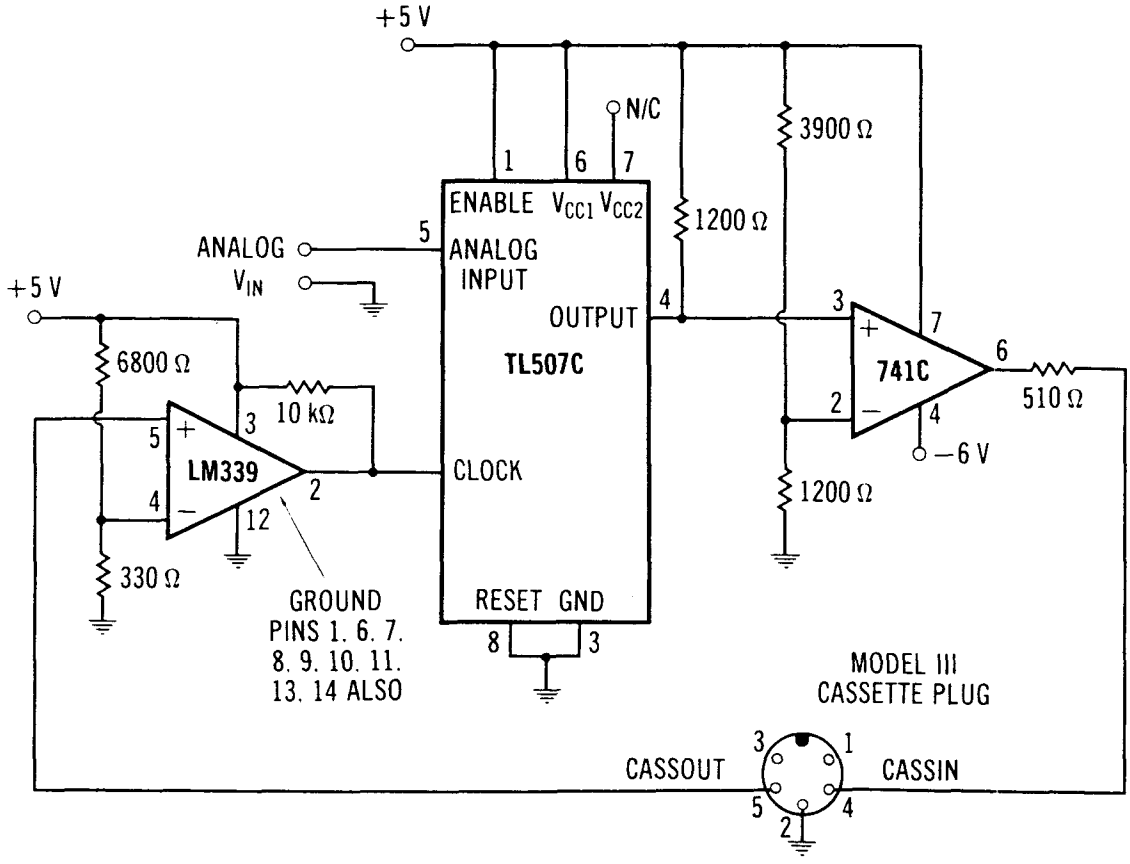


Fig. 5–9. Adc detailed circuit.

counter will be at some meaningless value when power is first applied, but thereafter will repeat modulus 128. The ANALOG INPUT is referenced to ground.

CONSTRUCTING THE ADC

The adc circuit was built up on a Radio Shack 276-175 prototype board and is shown in Fig. 5–11. The prototype board has two bus columns, one each for power and ground on the left and right sides of the board. There are two sets of 23 rows used for connecting integrated circuits.

The drawing in Fig. 5–11 is meant as a general guide for interconnections; use the logic diagram of Fig. 5–9 as the definitive circuit. Solid lines in the construction figure represent solid bus wires; these can be routed under or over components. Note the keying of the IC chips; pin 1 on all chips is at the upper left corner.

After the board has been wired up, recheck the wiring and prepare the power cables. The two leads on top go to a +5-volt power supply. Radio Shack has an inexpensive +5-volt supply kit (277-125) which you can use for this purpose. The −6-volt leads connect to the −6-volt supply for the 741. You may substitute a +6-volt battery in place of a large power supply with no problem, as the 741C draws negligible amounts of current. The positive lead to the battery or power supply attaches to the ground bus of the prototype board.

The three leads on the bottom go to the cassette input jack on the Model III. The proper pin numbering is shown in Fig. 5–9. For testing purposes, simply clip test leads to the existing Model III cassette cable as shown in Fig. 5–12. The two leads for the analog input signal connect to the voltage to be measured. A simple voltage divider using one 10K potentiometer and a 10K resistor is shown in Fig. 5–13.

THE SOFTWARE

The flowchart in Fig. 5–14 shows the basic scheme for measuring the duty cycle by counting clock pulses. The CASSIN signal is read from bit 0 of I/O port 0FFH. This bit is a 1 when the analog input voltage is higher than the ramp voltage. If the input is a 0 initially, the ramp voltage is above the analog input. In this case, via CASSOUT, clock pulses are output until the input bit goes to a 1 (ramp below). If the input is a 1,

clock pulses are output until the ramp voltage rises above the analog input. At this point, a count is set to 0. The maximum count will be 128, and this can be held in one byte. Now, clock pulses are output until CASSIN goes to 1. The count is incremented for each clock pulse output. When CASSIN goes to 1, the ramp voltage has fallen below the analog input voltage.

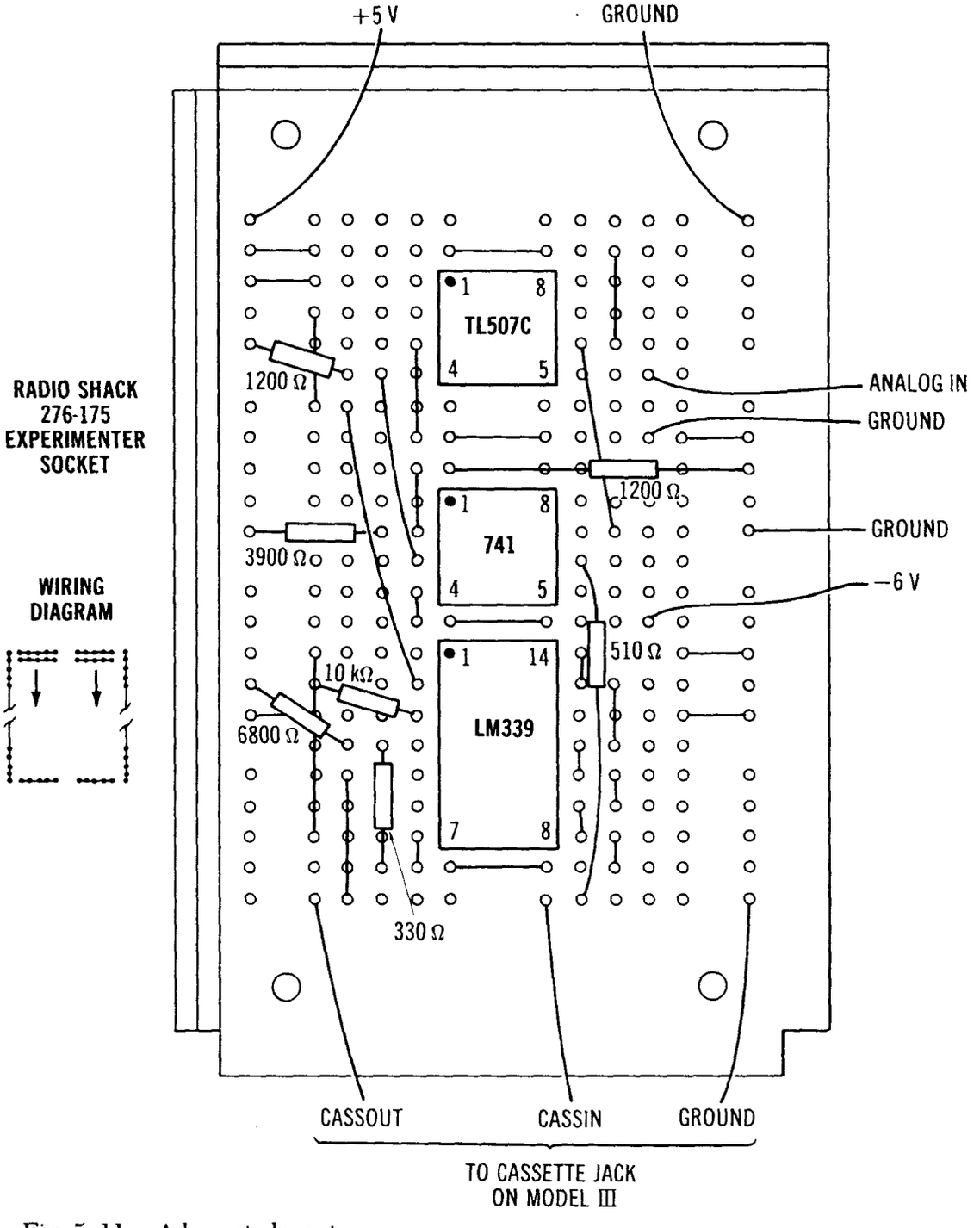


Fig. 5–11. Adc parts layout.

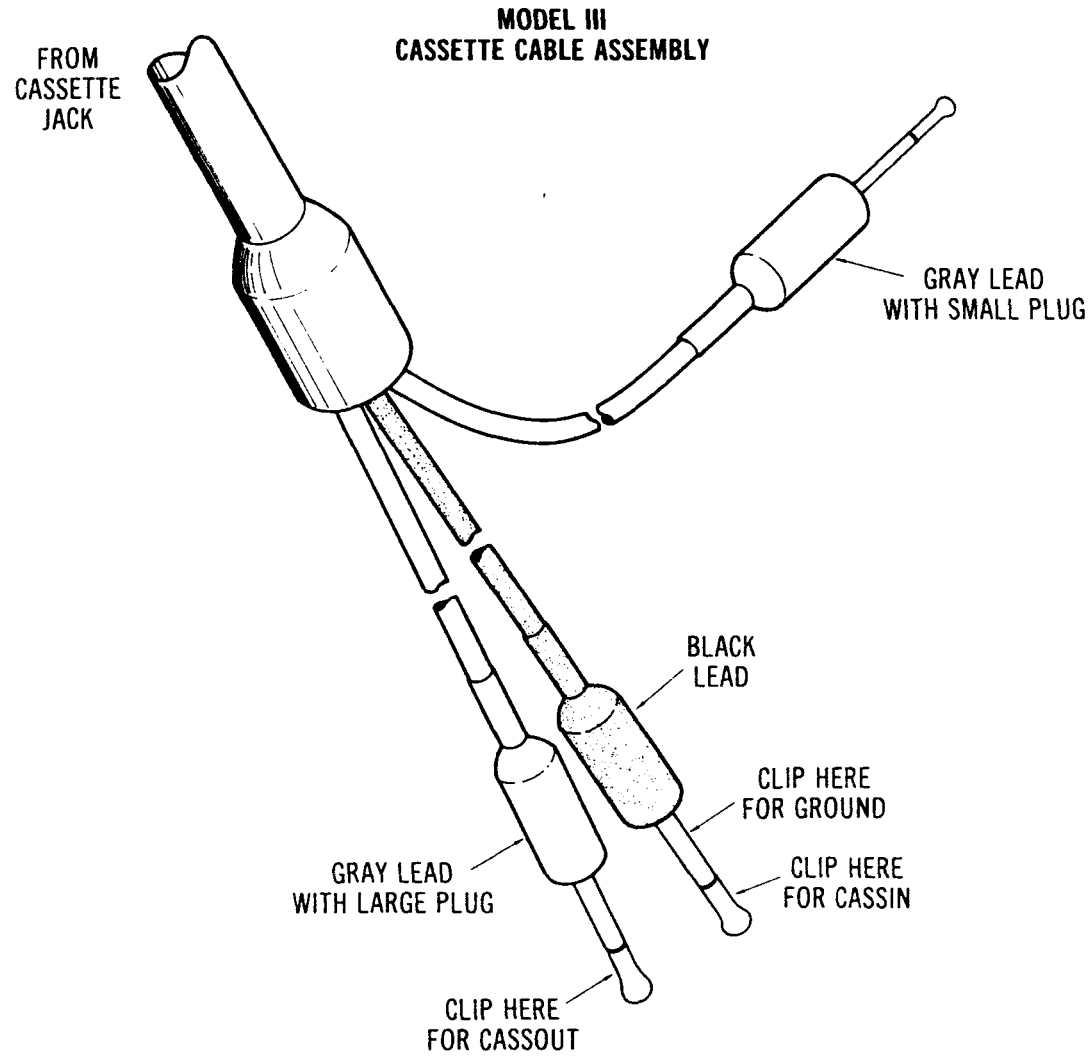


Fig. 5–12. Cassette cable structure.

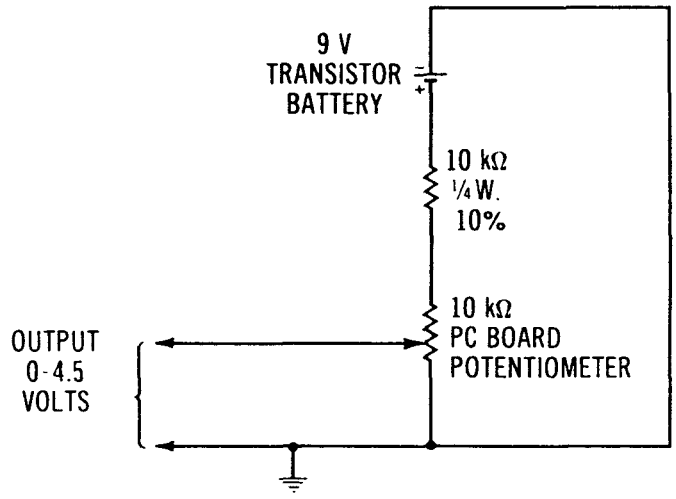


Fig. 5–13. Voltage divider for testing.

This algorithm is implemented in the assembly language program shown in Fig. 5–15. Three subroutines are used: TSTIN, OUTCLK, and DELAY. DELAY simply delays a fixed amount of time, about 200 μs. It's called by OUTCLK to create a fixed-width clock pulse of 200 μs. OUTCLK outputs one complete clock cycle to CASSOUT. The cassette toggle is first set one way by outputting a 2 to port 0FFH. DELAY is then called. Next, the cassette toggle is set to the opposing voltage level by an output of 1. DELAY is called again.

The TSTIN subroutine is called by the main CONVRT program to test the state of the CASSIN line. The state of CASSIN is returned in the Z

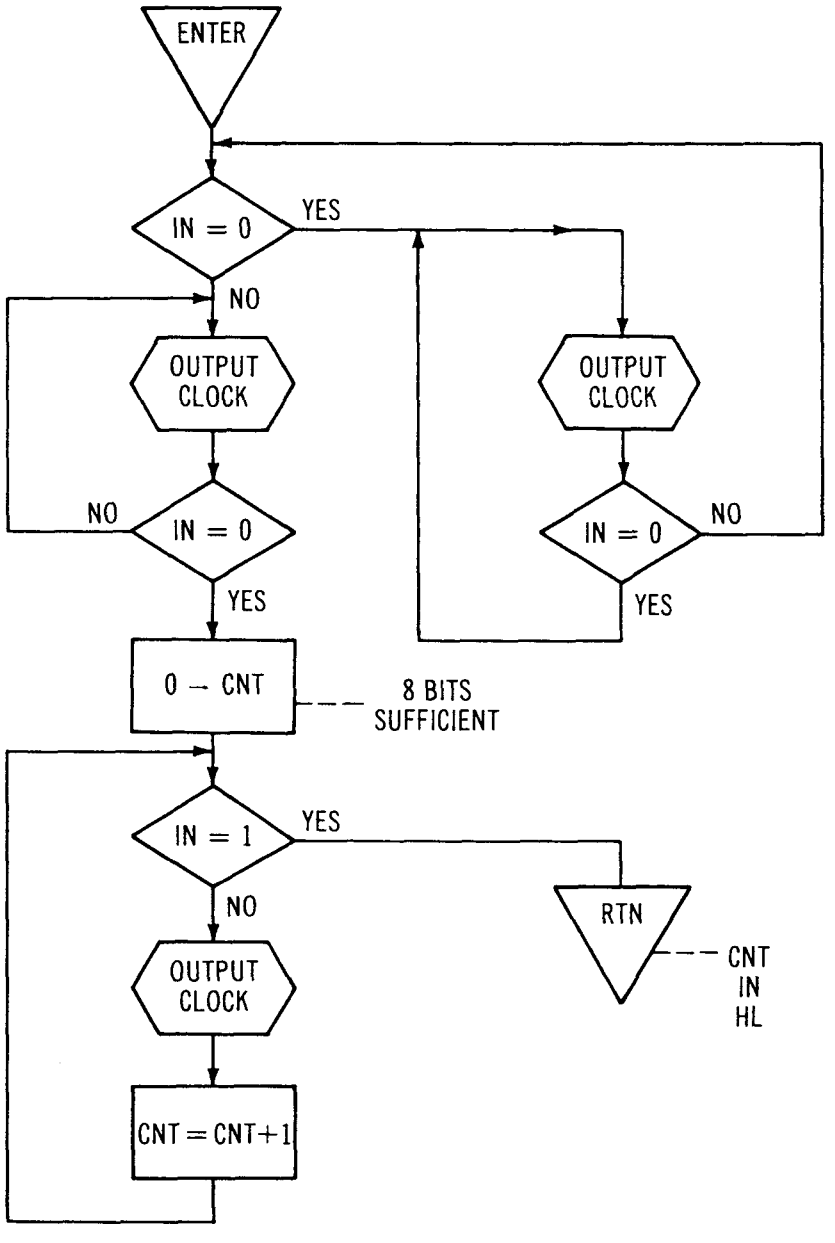


Fig. 5–14. Flowchart for the adc.

```
7F00            00100           ORG     7F00H           ;CHANGE THIS AS REQD
                00110 ;*********************************************************
                00120 ;* A TO D CONVERSION PROGRAM FOR MODEL III                *
                00130 ;*     INPUT: NONE                                        *
                00140 ;*     OUTPUT:(HL)=CONVERSION COUNT OR -1 IF TIME OUT     *
                00150 ;*********************************************************
7F00 110000     00160 CONVRT  LD      DE,0            ;TIME OUT COUNT
7F03 CD2B7F     00170 CON005  CALL    TSTIN           ;LOOK FOR 1
7F06 200A       00180         JR      NZ,CON020       ;GO IF 1
7F08 CD3B7F     00190 CON010  CALL    OUTCLK            ;0, WAIT 'TIL 1
7F0B CD2B7F     00200         CALL    TSTIN             ;TEST
7F0E 28F8       00210         JR      Z,CON010          ;GO IF STILL 0
7F10 18F1       00220         JR      CON005            ;RESTART
7F12 CD3B7F     00230 CON020  CALL    OUTCLK            ;NOW LOOK FOR 0
7F15 CD2B7F     00240         CALL    TSTIN             ;TEST
7F18 20F8       00250         JR      NZ,CON020         ;GO IF STILL 1
7F1A 210000     00260         LD      HL,0            ;INITIALIZE CONVERT CNT
7F1D CD2B7F     00270 CON030  CALL    TSTIN             ;NOW LOOP 'TIL 1
7F20 2006       00280         JR      NZ,CON090         ;GO IF 1
7F22 CD3B7F     00290         CALL    OUTCLK            ;PULSE
7F25 23         00300         INC     HL                ;BUMP COUNT
7F26 18F5       00310         JR      CON030            ;TRY AGAIN
7F28 C39A0A     00320 CON090  JP      0A9AH           ;RETURN WITH ARG
7F2B 13         00330 TSTIN   INC     DE              ;BUMP TIME OUT CNT
7F2C 7A         00340         LD      A,D             ;TEST FOR DE=0
7F2D B3         00350         OR      E
7F2E 2006       00360         JR      NZ,TST010       ;GO IF NO TIME OUT
7F30 21FFFF     00370         LD      HL,-1           ;SET ERROR FLAG
7F33 D1         00380         POP     DE              ;RESET STACK
7F34 18F2       00390         JR      CON090          ;RETURN
7F36 DBFF       00400 TST010  IN      A,(0FFH)        ;READ CASS IN
7F38 E601       00410         AND     1               ;GET 1500 BAUD BIT
7F3A C9         00420         RET                     ;RETURN
7F3B 3E02       00430 OUTCLK  LD      A,2             ;ONE WAY
7F3D D3FF       00440         OUT     (0FFH),A        ;OUTPUT
7F3F CD4A7F     00450         CALL    DELAY           ;DELAY ABOUT 200 MICS
7F42 3E01       00460         LD      A,1             ;OPPOSITE WAY
7F44 D3FF       00470         OUT     (0FFH),A        ;OUTPUT
7F46 CD4A7F     00480         CALL    DELAY           ;DELAY 200 MICS
7F49 C9         00490         RET                     ;RETURN
7F4A 0619       00500 DELAY   LD      B,25            ;ABOUT 200 MICS
7F4C 10FE       00510 DEL010  DJNZ    DEL010
7F4E C9         00520         RET                     ;RETURN
0000            00530         END
00000 Total errors
```

Fig. 5–15. Analog-to-digital conversion program.

flag; Z is set (Z) if CASSIN is 0, or reset (NZ) if CASSIN is 1. TSTIN also tests a time-out count in DE. The count is incremented, and if 0, has been incremented 64K (65,536) times. In this case, TSTIN resets the stack and returns to the calling (BASIC) program. Time-out occurs when CASSIN does not change state in a reasonable amount of time. Time-out is indicated by returning a count of −1.

The main driver portion of CONVRT implements the logic shown in the flowchart by calling TSTIN and OUTCLK. At the end of CONVRT, the count of clock pulses is held in the HL register pair and returned to BASIC by the standard return of JP 0A9AH, which returns the HL register contents to a BASIC variable.

The assembly language version of CONVRT (see Fig. 5–15) executes at location 7F00H. Fig. 5–16 shows a BASIC driver program that calls CONVRT and also incorporates the machine-language code of CONVRT

```
100 ' MODEL III TL507C A TO D CONVERSION DRIVER
110 DATA 17,0,0,205,43,127,32,10,205,59,127,205,43,127,40,248
120 DATA 24,241,205,59,127,205,43,127,32,248,33,0,0,205,43,127
130 DATA 32,6,205,59,127,35,24,245,195,154,10,19,122,179,32,6
140 DATA 33,255,255,209,24,242,219,255,230,1,201,62,2,211,255
150 DATA 205,74,127,62,1,211,255,205,74,127,201,6,25,16,254,201
160 FOR I=32512 TO 32590
170 READ A: POKE I,A
180 NEXT I
190 DEFUSR0=&H7F00
200 CLS: I=0
210 A=USR0(0)
220 IF A=-1 THEN PRINT @ 512+20,"OUT OF RANGE                    ":GOTO 210
230 I=I+1
240 A=(.75*5)-(A-1)*(2.5/128)
250 PRINT @ 512+20,I,INT(A*100)/100;"
260 GOTO 210
```

Fig. 5–16. BASIC driver for the conversion program.

as a series of DATA statement values. The code is moved to the 7F00H area before execution of the BASIC program.

The BASIC driver for CONVRT converts the count returned from CONVRT to a voltage level, predicated upon a +5-volt supply for V_{cc}. The iteration count is displayed in the middle of the screen along with the voltage value. When V_{cc} is exactly +5 volts, the allowable analog voltage input can be +1.25 volts through +3.75 volts, for a total range of 2.5 volts. As there are 128 steps, each step represents 2.5/128 volts or about 0.01953 volt. The actual voltage read from the analog voltage input is:

$$V = (+3.75) - (CNT - 1) \times 0.01953$$

and this value is calculated and displayed on the screen.

USING THE ADC

Connect all power leads to the adc and turn on the +5 volts and −6 volts dc. Make the obvious "smoke tests;" none of the chips should feel hot to the touch. Connect a voltage source to the analog voltage input; this can be the circuit shown in Fig. 5–13 or simply a 1.5-volt battery connected between ground and the input lead. Connect the cassette leads either by clip leads or specially wired cable to the cassette DIN jack on the rear of the Model III.

Protect the 7F00H RAM area and enter the BASIC program. Double-check the DATA statements for correct values. Execute the BASIC

program. You should see a slight pause as the machine-language code is moved from the DATA statements to the 7F00H area, the screen should clear, and you should then see the iteration count followed by the voltage value on the screen. If you do not see a conversion voltage immediately, recheck the power and wiring connections. If the analog input voltage is not properly connected, you should see the OUT OF RANGE message.

The adc components and timing are not critical. If you experience trouble, chances are it's in the wiring or machine-language code. You may troubleshoot the circuit by outputting pulses by the following BASIC code:

```
100 OUT 255,1
110 OUT 255,2
120 PRINT @ 512+20,INP(255) AND 1
130 GOTO 100
```

This code outputs clock pulses at a very low data rate, but one still sufficient to cycle the adc through the ramp voltages in a few seconds. The input should alternate between 0 and 1, depending upon the analog input voltage.

HOW DOES IT WORK?

Wonderfully! All kidding aside, this is one of those projects that works extremely well, thanks to the specifications of the TL507C chip. The values in Table 5–2 show the output obtained from various analog voltage inputs. Voltage inputs above +3.75 and below +1.25 are out of range, but all other inputs are very close to what they should be.

The number of samples (conversions) per second is about 6. Actually, each sample should take anywhere from about 2 clock pulses (1 ms) to about a worst case of about 256 clock pulses (128 ms), corresponding to 1000 to 7.8 samples per second. The BASIC program overhead reduces the number of samples per second to about 6, however, regardless of the analog input voltage. This conversion rate could be "cranked up" by increasing the clock frequency output on CASSOUT. The accuracy of the conversion is unaffected by such system functions as real-time-clock interrupts, as the program directly counts each clock pulse.

Table 5–2. Adc Test Data

Digital Voltmeter	Screen Value
<1.26	OUT OF RANGE
1.30	1.30–1.31
1.40	1.42
1.50	1.50–1.52
1.60	1.58–1.59
1.70	1.69–1.70
1.80	1.79–1.80
1.90	1.89–1.91
2.00	1.99–2.01
2.10	2.09–2.11
2.20	2.18–2.19
2.30	2.28–2.29
2.40	2.40
2.50	2.48–2.49
2.60	2.59
2.70	2.67–2.69
2.80	2.79
2.90	2.89–2.90
3.00	2.98–2.99
3.10	3.08–3.09
3.20	3.18
3.30	3.28–3.29
3.40	3.37–3.38
3.50	3.47–3.49
3.60	3.57–3.59
3.70	3.69
3.74	3.73–3.75
3.76	3.75
>3.76	OUT OF RANGE

ADC APPLICATIONS

In earlier chapters we've seen how some devices can generate a voltage that is an analog of real-world quantities such as temperature and light intensity. There are many different types of transducers that convert other physical quantities into voltage analogs. This cheap and easy adc can be used to monitor the outputs of these devices and send them to the Model III for recording and processing. In the last two chapters of this book we look at cheap and easy transducers and show you how to use adcs for the Models I, III, and Color Computers.

Section II

Using the RS-232-C Port on the Models I, III, and Color Computers

chapter 6

RS-232-C Communications

The RS-232-C port of the Color Computer, Model I, or Model III communicates with the outside world of RS-232-C (serial) devices. These devices, line printers, modems, and others, use voltage levels and signals defined by the RS-232-C interface structure, an industry standard. The RS-232-C ports can also be used to communicate with special-purpose devices designed for real-world inputs, however. In this section we discuss the RS-232-C standard and the internal logic of the RS-232-C interface in the three systems.

STANDARD ASYNCHRONOUS FORMAT

The RS-232-C Standard defines the format for data communication; hence, serial communication is often called *RS-232-C* or simply RS-232 communication. A system *port* designed for data communication is known as a *serial port* or *RS-232 port.*

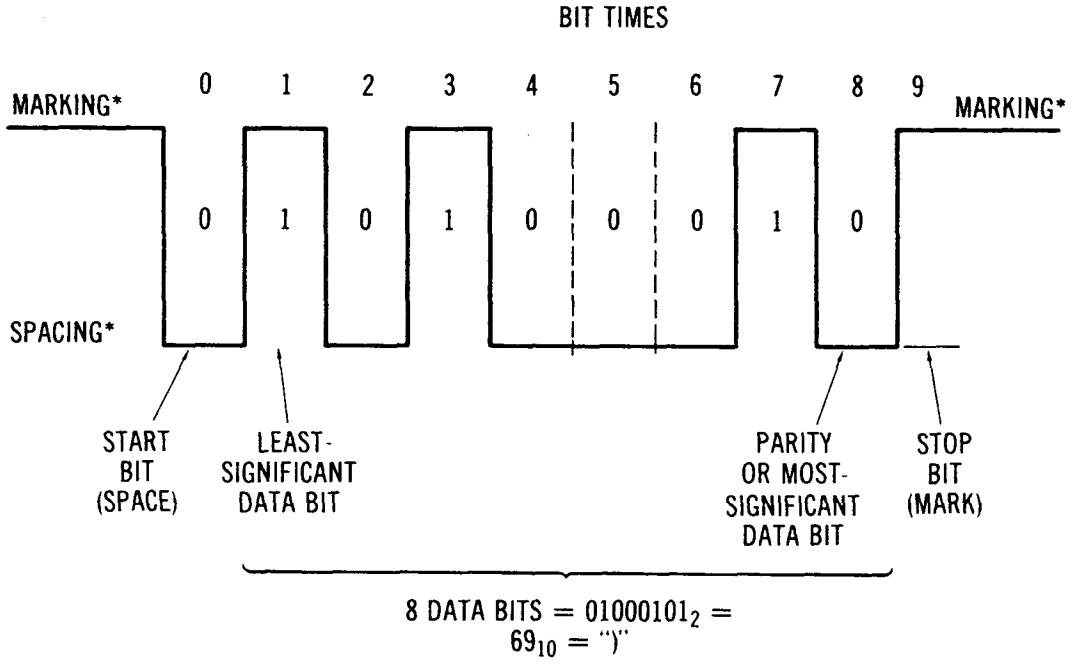
The standard format for asynchronous communication is shown in Fig. 6–1. In this type of data communication, 8 bits make up a byte of data transmitted to or from the computer system and another computer system or input/output device. The data is *asynchronous* or occurring at unpredictable times rather than at regularly spaced intervals. A good example of asynchronous data is a character transmitted from a keyboard to a communications system such as the Source or Micronet. The bulletin board system does not know when to expect the next character; it may occur within 1/10 second or 10 seconds. Each character must be detected and handled on an individual basis.

The computer system or device receiving an asynchronous character can read either a logic 1 or logic 0 on the RD or *receive data* line. Initially,

the RD line is called *marking* or logic 1. The system or device expects this high condition at first. After some time, the transmitting system sends a character by bringing the RD line to a *space* or logic 0 for one "bit time." A character is made up of 8 bits, each occupying one bit time. The length of the bit time depends on the data transmission rate and may vary from about 9 ms (0.009 s) to 0.1 ms (0.0001 s).

The receiving system or device detects the 0, delays 1/2 bit time, and then reads in the remaining bits at evenly spaced intervals of one bit time. The leading start bit is followed by 8 data bits (least-significant to most-significant), and one or two stop bits (logic 1). The stop bits ensure that the line will be marking prior to transmission of the next byte of data. The number of data bits may vary from 5 to 8, the number of stop bits from 1 to 2, and the most-significant data bit may represent parity or a checksum, depending upon the computer system.

You can see that the key to asynchronous transmission is evenly spaced bit times and short lengths of data to prevent skew of the time each bit is read (middle of each bit time). Asynchronous transmission would not work for long strings of data, as the middle of the bit time would be harder and harder to determine accurately, as there is no self-contained clock pulse reference within the data.



*ON RS-232-C LINES REVERSE LOGIC (MARKING IS −12 V, SPACING IS +12 V)

Fig. 6–1. Asynchronous data format.

The baud rate is the data transmission rate of all data. A typical baud rate for the Color Computer is 300 baud, representing 300 bit times per second or 30 characters per second. Each character at this baud rate is made up of one start bit, 8 data bits, and one stop bit, a total of 10 bits. The bit time is 1/300 second, or 3.33 ms.

The standard logic levels for RS-232-C communication are voltages above 3 volts and below −3 volts. Voltages above 3 volts represent logic 0 and below −3 volts represent logic 1. In the Color Computer, Model I, and Model III, the voltages used are 12 volts for logic 0 and −12 volts for logic 1. See Fig. 6–2.

COLOR COMPUTER SERIAL INTERFACE

The complete serial interface of the Color Computer is shown in Fig. 6–3. There are four lines in the serial interface, and they go to a 4-pin DIN plug on the rear of the system. These four lines are a subset of the 25 lines normally used in other computer systems, such as the Radio Shack Models I and III. Asynchronous serial communication can be performed with only three lines, and the Color Computer uses this approach to handle both input and output of serial data.

The Color Computer serial interface is made up of two receive comparators and one transmit operational amplifier (op amp) used as a comparator. A comparator simply compares two voltages. If the positive (+)

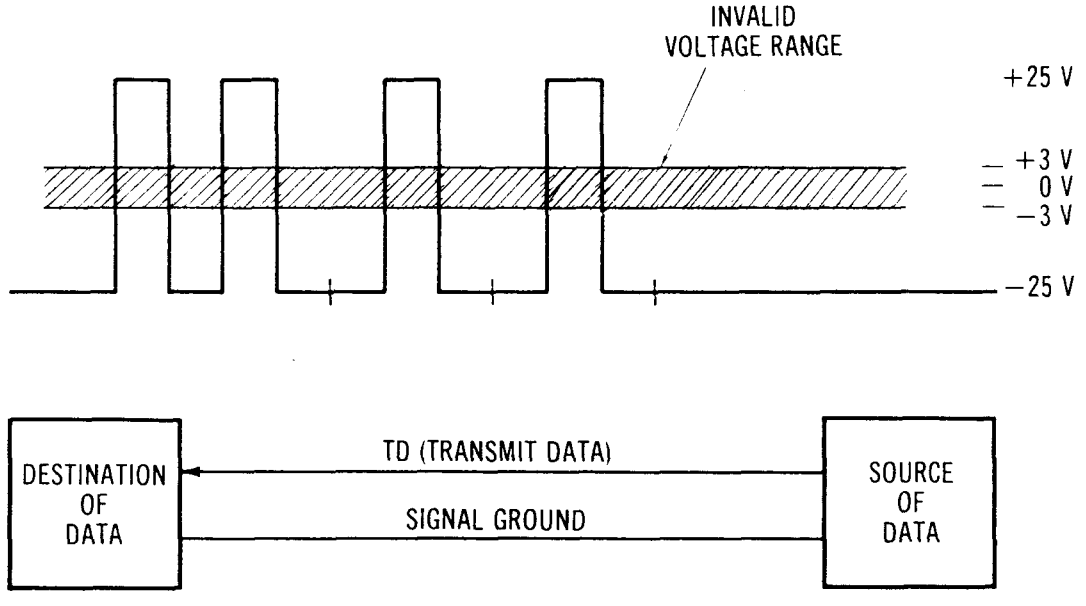


Fig. 6–2. RS-232-C voltage levels.

input is a higher voltage than the negative (−) input, the output of the comparator is a logic 1. If the positive (+) input is a lower voltage than the negative (−) input, the output of the comparator is a logic 0, as shown in Fig. 6–4.

The 741C op amp comparator compares the negative (−) input from bit 1 of PIA address $FF20 with the positive (+) input from the voltage divider R23/R24. The positive input is a constant voltage of +1.38 volts. A logic 1 from PIA $FF20 bit 1 will generate −12 volts on the TD line; a logic 0 from PIA $FF20 will generate +12 volts on the TD line. TD stands for *transmit data* and is the serial data output line from the Color Computer. The Color Computer BASIC interpreter drives the $FF20 PIA to output serial data over the PIA line. User assembly language programs can also output data over the TD line.

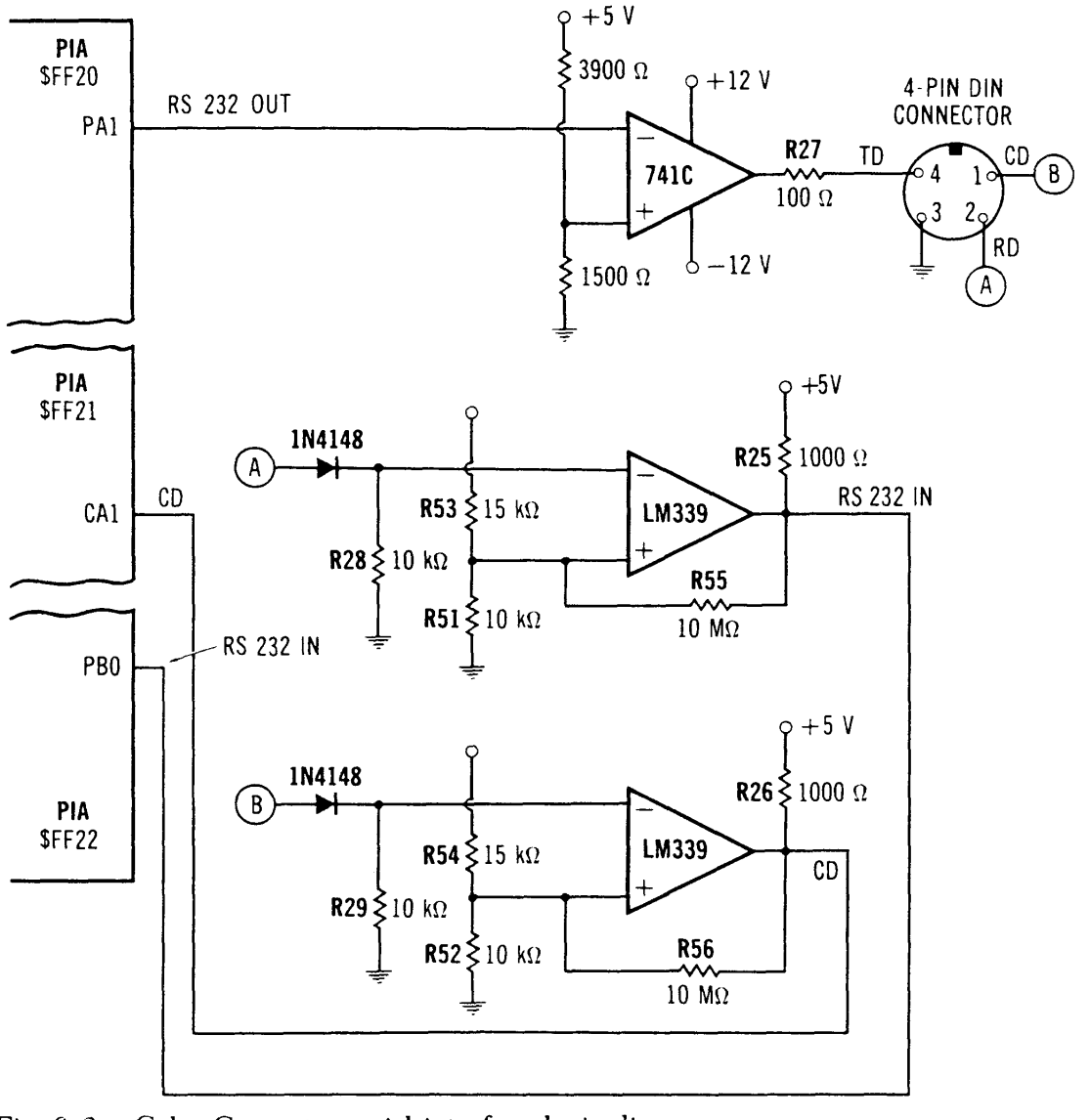


Fig. 6–3. Color Computer serial interface logic diagram.

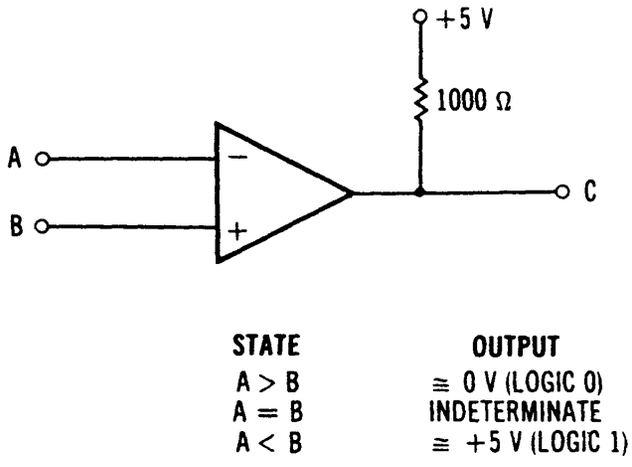


Fig. 6–4. RS-232-C comparator circuit.

The two LM339 comparators input serial data. Both have the same configuration. One is connected to the RS232IN line of the $FF22 PIA (bit 0). The second is connected to the CD line of the $FF21 PIA. The CD line generates an interrupt and is not used in this project.

The positive input of the RS232IN comparator is biased at +2 volts by the R54/R52 resistor divider. The negative input is connected via a diode to the RD or receive data line of the serial port. Normally, the input on this line will be serial data, represented by a positive or negative voltage. If the level is at +2 volts plus about 0.6 volt, the output of the comparator is a logic zero or about 0 volt. If the input is negative, the diode will not conduct, and the output of the comparator is a logic one or about +5 volts. The RS232IN signal to the $FF22 PIA, therefore, follows the serial data on the RD line and can be tested by reading the PIA bit 0 with assembly language code. The third line used in serial communication is ground, connected to pin 3 of the DIN connector.

MODELS I AND III RS-232-C PORTS

The Models I and III use about the same logic for generation of RS-232-C signals, with some minor differences. Fig. 6–5 shows the RS-232-C logic.

The heart of the RS-232-C port is a Western Digital TR1602B UART or *universal asynchronous receiver/transmitter.* UARTs, in spite of the tongue-twisting acronym, are simply devices to receive and transmit serial data, with programmable baud rates and slight format differences.

Because the data is serial, transmission may be over a single pair of wires, as shown in Fig. 6–6. For two-way transmission, another wire is added, as shown in Fig. 6–7. One wire is labeled *receive data* or RD and the other is *transmit data.* The common wire is *signal ground.* In the half-duplex mode, transmission is one direction or the other, but not both simultaneously. In the full-duplex mode, transmission may be in both directions simultaneously.

While it's possible to send data over the 3-wire system of Fig. 6–7, the system is limited. There is no handshaking other than a response from the

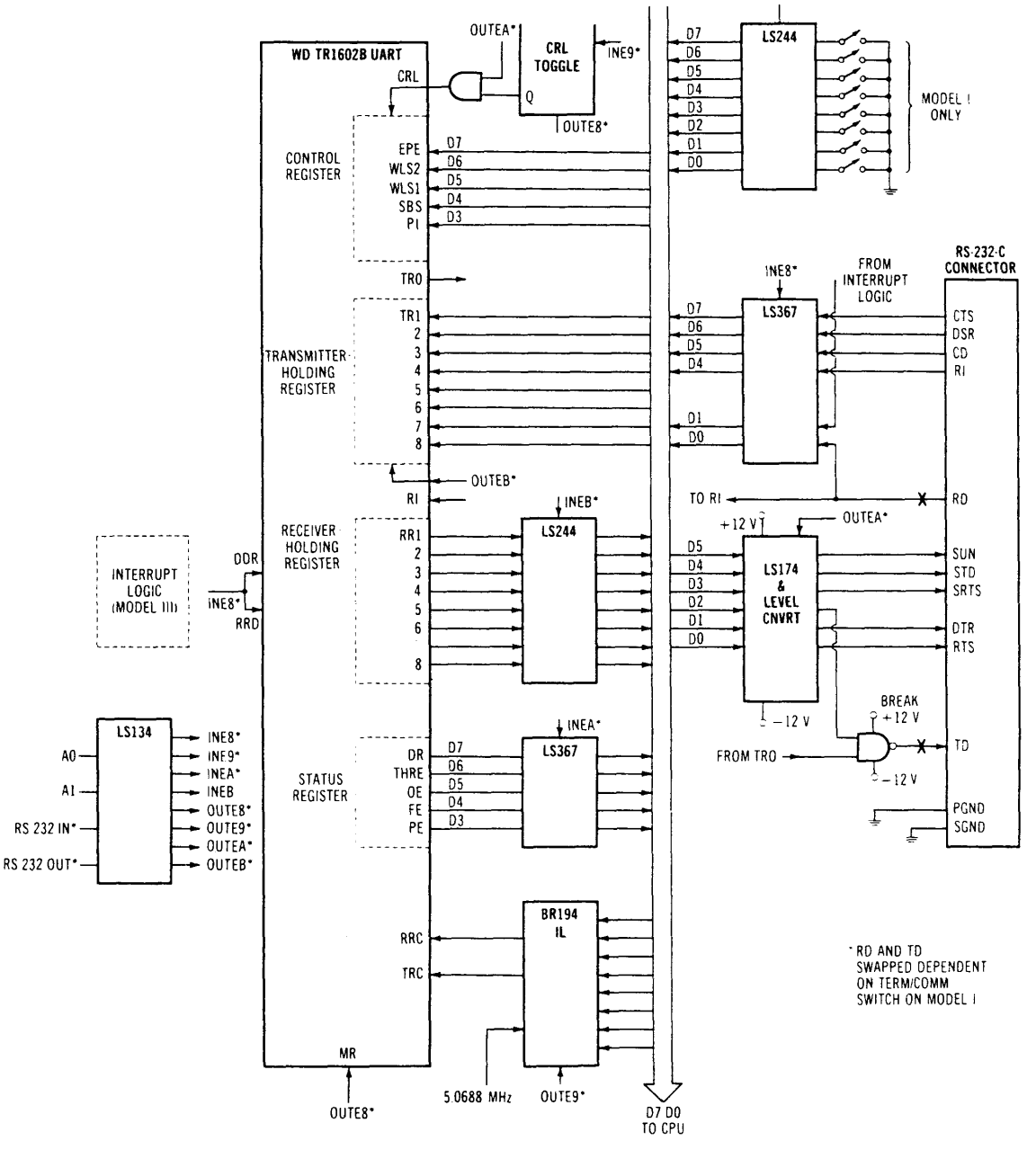
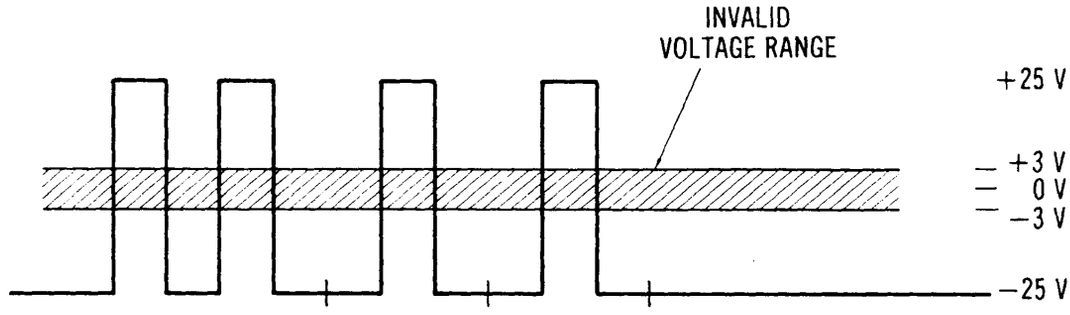


Fig. 6–5. Models I and III RS-232-C logic diagram.



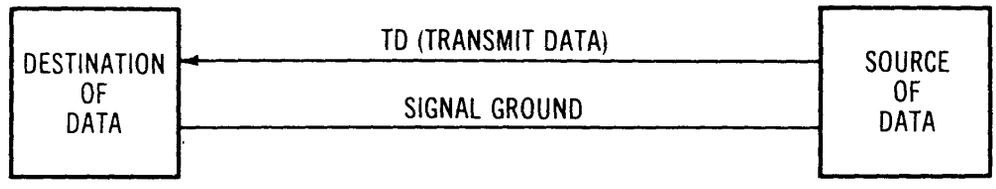


Fig. 6–6. One-way serial transmission.

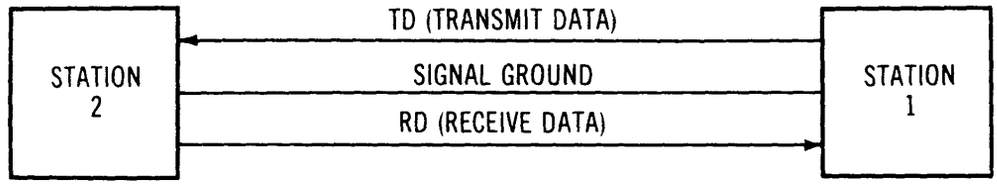


Fig. 6–7. Two-way serial transmission.

other end that might be encoded into a character string. The full RS-232-C asynchronous system, therefore, includes other meaningful signals, as shown in Table 6–1. For example, DSR tells the terminal that the data set is ready, powered up and set to communicate. DTR is the opposite response from the data terminal, indicating that it is ready. In practice, only a subset of the signals are used for many applications, such as serial printers and modems. The RS-232-C signals used in the Models I and III are marked in the table with an asterisk; pin numbers remain the same.

THE WESTERN DIGITAL TR1602B

The Western Digital TR1602B, used in both the Models I and III, is a large-scale chip that handles variable transmission rates, word formats, full-duplex operations, parity, and just about every phase of most RS-232-C communications.

Table 6–1. RS-232-C Signals

Pin No.	Symbol	Description
1°°	PGND	Protective ground
2°°	TD	Transmitted data
3°°	RD	Received data
4°°	RTS	Request to send
5°°	CTS	Clear to send
6°°	DSR	Data set ready
7°°	SGND	Signal ground
8°°	CD	Carrier detect
9	–	Reserved for data set testing
10	–	Reserved for data set testing
11	–	Unassigned
12	–	Secondary received line signal detector
13	SCTS	Secondary clear to send
14°	STD	Secondary transmitted data
15		
16	SRD	Secondary received data
17		
18°	SUN	Secondary unassigned
19°	SRTS	Secondary request to send
20°°	DTR	Data terminal ready
21	–	Signal quality detector
22°°	RI	Ring indicator
23		
24		
25	–	Unassigned

°Model III only
°°Model I and III

As you can see from Fig. 6–5, there are 4 registers in the TR1602B: the control register, the transmitter-holding register, the receiver-holding register, and the status flags.

The Control Register

The control register is normally loaded once, before any communications activities. There are 5 control bits associated with the control register, shown in Fig. 6–8. The WLS1 and WLS2 bits determine the word length of the transmission, as shown in the figure. This is the number of data bits per word, exclusive of the parity bit. If the PI or parity-inhibit

bit is a 1, no parity bit will be generated with each character. If the PI bit is a 0 and the EPE bit is a 1, even parity will be generated; if the PI bit is a 0 and EPE is a 0, odd parity will be generated. If the SBS bit is a 1, two stop bits will follow the last data bit or the parity bit. If SBS is a 0, one stop bit will follow.

The Transmitter-Holding Register

The transmitter-holding register is an 8-bit register that holds the byte to be transmitted. This byte is loaded from the data bus and stored in the transmitter-holding register. As soon as it is loaded, the start bit is sent out over the TRO (transmitter register output), followed by the 5 or 8 bits in the transmitter-holding register, least-significant bit first. The transmission rate is determined by the TRC or transmitter clock input. The transmitter-holding register, therefore, performs a parallel-to-serial conversion of the character to be transmitted.

The Receiver-Holding Register

The counterpart to the transmitter-holding register is the receiver-holding register. It accumulates the incoming data bits from the RD line, performing a serial-to-parallel conversion. The receiver-holding register is read after all data bits have been received to recover the parallel form of the received character.

The Status Register

The status register is a collection of 5 bits representing the TR1602B status, as shown in Fig. 6–9. If THRE is a 1, the transmitter-holding register is empty and has performed its parallel-to-serial conversion and sent the data out over the TD line. A new character can now be stored in the transmitter-holding register. If the DR bit is a 1, a new character has been received and is in the receiver-holding register; it can now be read from the TR1602.

The OE, FE, and PE are error indications. If one or more are 1s, an error has occurred. PE is parity error, indicating that the received parity bit does not match the "parity" of the received data bits. One or more data bits have been erroneously received. FE indicates that no stop bit was found in the received character. The OE bit indicates that a received

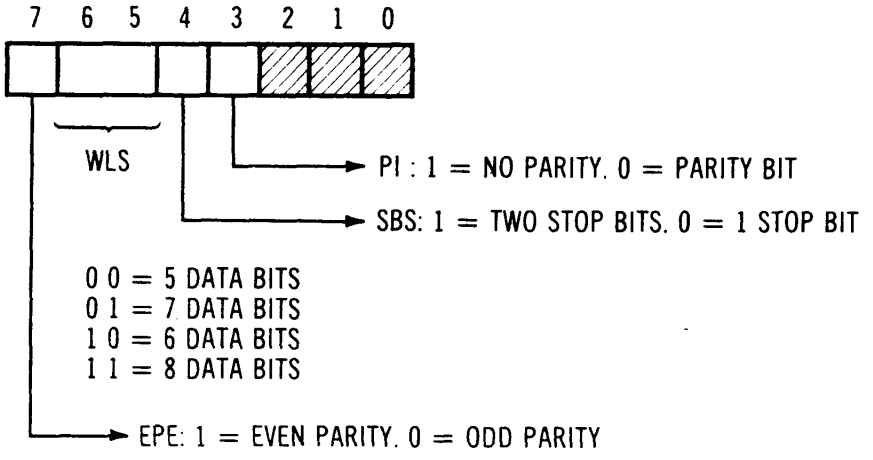


Fig. 6–8. Control register bits in the TR1602B.

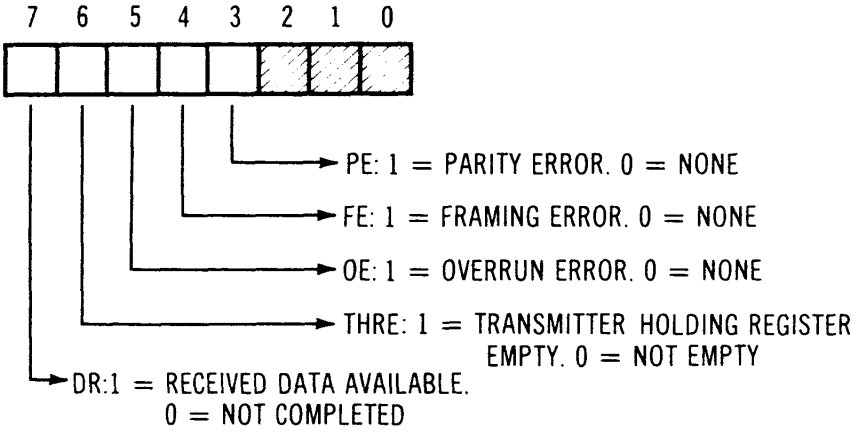


Fig. 6–9. Status register in the TR1602B.

character was not read (by the computer) fast enough to avoid overwriting by the next incoming character. Both the receiver and transmitter are double buffered. An OE error will occur if two characters are received before the computer performs a read.

The Baud-Rate Generator

The second large-scale part used in the RS-232-C interface is the BR194IL chip. This is a baud-rate generator chip that determines the bit time used in data communications. In the Model I, the clock input is from a 5.0688-MHz crystal oscillator. The Model III uses a system timing signal of 5.0688 MHz. This clock reference is divided down to the proper transmitter and receiver clock frequency and sent to the TR1602B via the RRC and TRC inputs. These inputs are 16 times the bit time rate to

enable centering during the middle of a bit time and to increase the resolution during high baud rates.

The baud-rate generator is loaded with two 4-bit codes that represent the frequencies to be used for the receiver and transmitter clock. This normally is done at about the same time the control register is loaded. The baud rates shown in Fig. 6–10 are determined by the code sent to the BRG.

MODELS I AND III INTERFACE LOGIC

The interface logic for the Models I and III are almost identical. Fig. 6–5 shows part numbers associated with the Model III, but similar logic is used on the Model I. The TR1602B has four addresses associated with it, hexadecimal 0E8H, 0E9H, 0EAH, and 0EBH.

Fig. 6–5 shows a 74LS134 decoder used to decode the RS-232-C address into 4 input and 4 output signals. Address lines A0 and A1 determine the two least-significant bits of the address, while RS232IN* and RS232OUT* are generated by an IN 0EX and OUT 0EX instruction, respectively. The IN or OUT may be a Z-80 machine-language IN or

CODE	BAUD RATE
0000	50
0001	75
0010	110
0011	134.5
0100	150
0101	300
0110	600
0111	1200
1000	1800
1001	2000
1010	2400
1011	3600
1100	4800
1101	7200
1110	9600
1111	19.200

Fig. 6–10. Baud rates in the TR1602B.

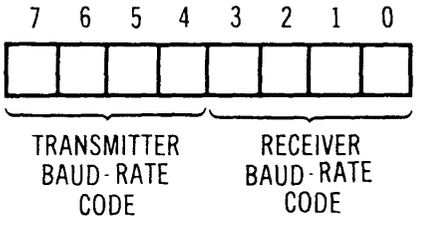


Table 6–2. Models I and III In/Out Actions

Address	IN	OUT
0E8H	Read modem status register (CTS, DSR, CD, RI)	Master reset
0E9H	Read sense switches (Model I); toggle CRL (Model III)	Load BRG with baud-rate codes
0EAH	Read UART status register	Load UART control register and set break, DTR, RTS, or set SUN, STD, SRTS, BREAK, DTR, RTS
0EBH	Input character from receiver-holding register	Output character to transmitter-holding register

OUT, but the same effect is achieved by a BASIC INP or OUT. Table 6–2 shows the actions that occur for the 4 addresses for either reads or writes (INs and OUTs).

The Model I RS-232-C Switches

The Model I differs from the Model III in that it has 8 RS-232-C switches that can be read by an IN 0E9H. These switches are a manual way to define the RS-232-C parameters; no action is taken in the TR1602B when the switches are read. The switch data is simply used in lieu of defining the baud rate, word length, parity, and stop bits via a BASIC or assembly language input.

The switches do not exist in the Model III. An IN 0E9H causes a different action. It toggles the CRL signal so that an OUT EAH sets three additional signals in the Model III: SUN, STD, and SRTS. These signals are secondary signals (secondary undefined, TD, and RTS, respectively) and are not used in normal Model III communications programs. Each consecutive OUT 0EAH toggles the load from control register to secondary. A master reset of OUT 0E8H resets the toggle to a normal load of the control register.

Initialization of the RS-232-C

The first action to take before doing any data communications is to initialize the TR1602B and baud-rate generator chip. The sequence is:

1. Do an OUT (0E8H),A in assembly language or a BASIC OUT 232,0. The value in A doesn't matter as no actual data are sent. This action resets the TR1602B by the MR input. It also resets the "data received" and disconnects the receiver-holding register by inputs DRR and RRD, respectively.
2. Define the serial parameters (word length, etc.). This is a 5-bit code as shown in Fig. 6–11. The remaining 3 bits define the break, DTR, and RTS outputs. The configuration of these depends upon the device on the other end of the line, but at this point the break bit must be set to enable the output of serial data on the TD line. Output the parameters to the control register by an OUT (0EAH),A or a BASIC OUT 234,A. This sets up the control register for the data communications format to be used. The data are read into the control register from the data bus on the output. The lower 3 bits are latched into the 74LS174 and converted to the RS-232-C levels of +12 and −12 volts, respectively.
3. Define the baud rate to be used. Refer to Table 6–2 for this, as the BRG uses special codes. Output the baud rate to the BRG by an OUT (0E9H),A or OUT 233,A. The 0E9H address strobes in the baud rate data from the data bus into the BR194IL chip.

At this point, the TR1602B is initialized. The preceding actions should not have to be repeated, unless the baud rate or other parameters are changed, something which does not ordinarily happen.

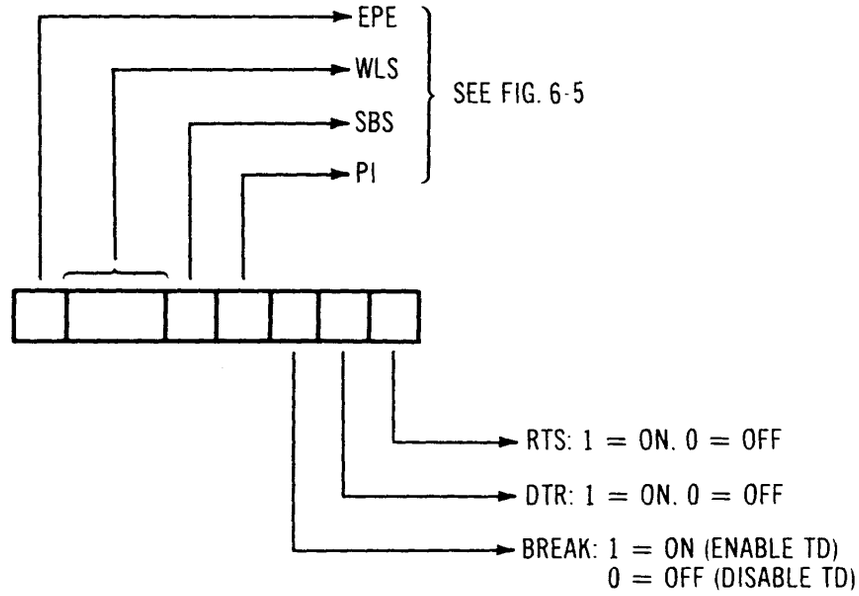


Fig. 6–11. Serial data parameters.

Writing Data

Suppose that you have a serial line printer attached to a Model I or III. The control register has been loaded with the proper format parameters to match the line printer and the BRG has also been initialized. From this point on, it's simply a matter of reading the status, testing to see if the transmitter-holding register is empty, and, if it is, loading it with the next character to be transmitted. This loop is done for every character and goes like this:

Input the status by an IN A,(0EAH) in assembly language or an A = INP(234) in BASIC. Test bit 6 (THRE) of the status. If this bit is a 0, loop back to the test. If this bit is a 1, output the character to be transmitted by an OUT (0EBH),A" or OUT 235,A. Of course, your main driver program must know when the last character to be transmitted comes up and must stop the loop.

Reading Data

Reading data is just about as easy. The read loop consists of reading the status to see if a new character has come in. If it has, the character is read from the receiver-holding register. This is done until some terminating character has been reached, or (usually) the read and write are interspersed in the loop so that you can respond to a prompt question, as from a data communications network. The loop goes like this:

INPUT the status by an IN A,(0EAH). Test bit 7 to see if DR is a 1. If it is not, loop back to the status input. If DR is a 1, read in the assembled character by an IN A,(0EBH) or A = INP(235). Repeat this loop or go to a test for a new character to be transmitted.

Now that we have some background in RS-232-C format, we're ready to tackle some related projects. Chapter 7 describes a real-time clock for the Color Computer which could also be used on the Models I and III, as the interfacing signals are identical. Chapter 8 describes a data communications plugboard for the Models I and III that will make interfacing to the RS-232-C ports somewhat easier.

chapter **7**

A Half-Year Clock for the Color Computer

The ideal clock for a computer system, to our mind, would be an inexpensive, compact, accurate unit with a self-contained power supply that could be easily interfaced to the computer system. This chapter describes a clock for the Color Computer that meets all of the goals above, at the expense of some software complexity, by interfacing directly to the RS-232-C port of the Color Computer.

The half-year clock described here can provide the real-time resolved to 10 seconds or better over a total elapsed time of one-half year. It is powered by a self-contained 9-volt battery. It can be disconnected from the Color Computer at any time, set aside, plugged in at a later time, and continue reporting the time. It is a compact unit measuring 5½ inches by 3½ inches by 1½ inches.

The half-year clock (HYC) is a construction project that uses seven integrated circuits plus some discrete components. The project is built using wire-wrapping techniques. If you've never tried wire-wrapping or feel that this is a little more complex a project than you'd care to handle, fear not. Following are detailed instructions that you can follow.

The HYC uses the serial interface of the Color Computer (Color BASIC or Extended Color BASIC version); we'll start the description of the project there so that you can understand the interfacing aspects of the device.

HYC DESIGN

A block diagram of the HYC design is shown in Fig. 7–1. It interfaces to the Color Computer via the RD, TD, and GROUND lines of the serial port. (Internal operations of the Color Computer serial or RS-232-C port

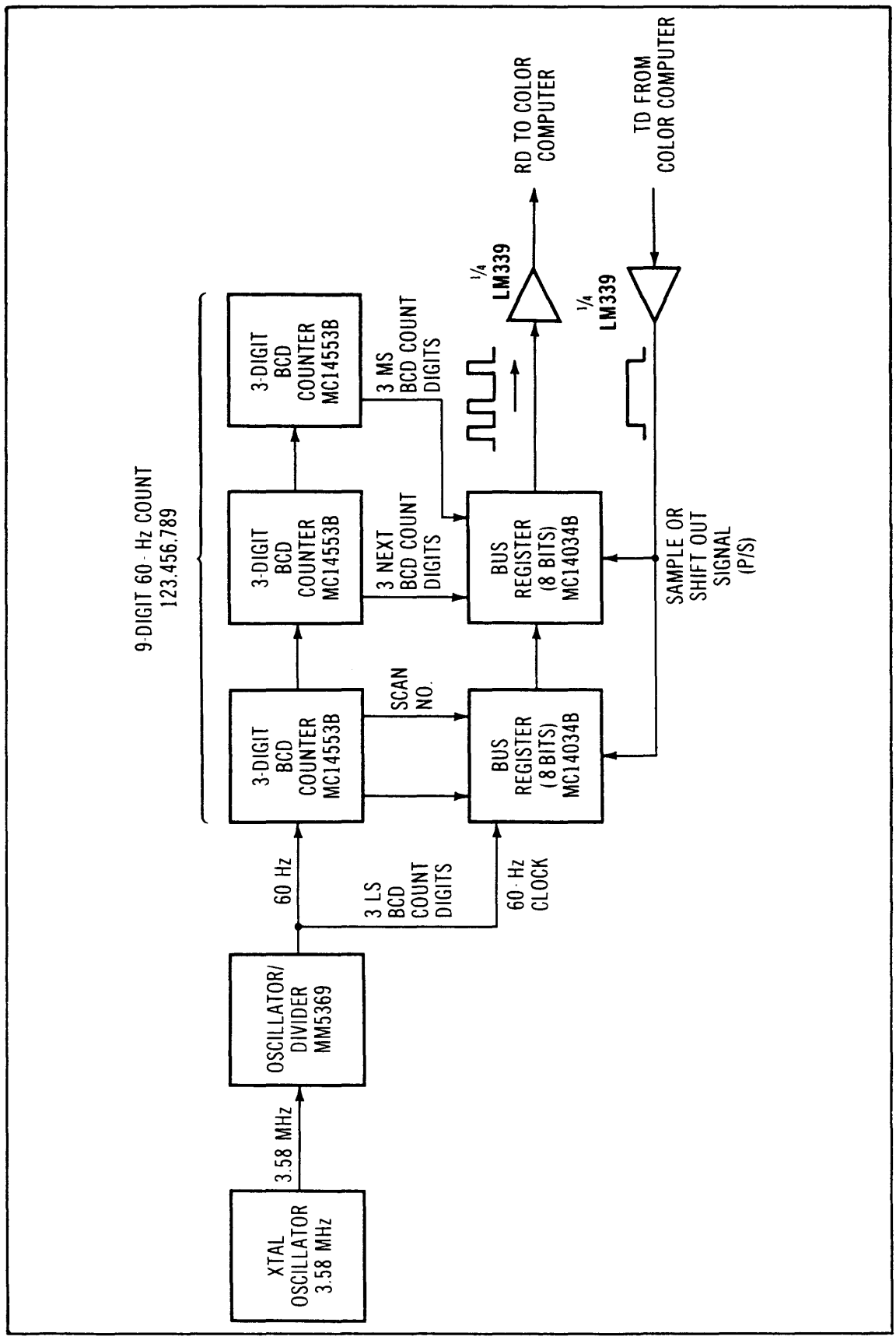


Fig. 7–1. Half-year clock block diagram.

are described in the previous chapter.) The clock count is sent via the RD line after a prompt by the Color Computer from the TD line.

The heart of the HYC is the section made up of the three bcd counter chips. Each of these chips accumulates a 3-decimal digit count of 0–999. The set of three chips accumulate a count of up to 999,999,999.

The input to the three counters is a 60-Hz (60 times per second) signal from the oscillator/divider chip. This chip takes a 3.58-MHz signal from a color burst crystal and divides it down to a 60-Hz signal. At any given time, the count in the counter chips represents the number of 60-Hz pulses received. The maximum count of 999,999,999 represents 16,666,666 seconds, about 192 days worth of time.

The two universal bus register chips record 16 bits from the counters upon command from the Color Computer. The bits are then shifted out to the RD line at a rate of one every 1/60th of a second. Three complete transfers (48 bits total) represent the current time when decoded by the HYC software. The detailed logic diagram of the HYC is shown in Fig. 7–2.

Counter Chips

The counter chips are Motorola MC14553B chips. Each simply increments by one each time a 60-Hz pulse is received from the 60-Hz line (CLK input). The count output is presented a digit at a time over the Q3 through Q0 outputs. Q3 through Q0 represent a bcd, or binary-coded-decimal digit of 0000 through 1001. If the count in one of the MC14553B chips was 678, for example, the outputs on Q3 through Q0 would be 0110, followed by 0111, followed by 1000, followed by a repeat of the sequence.

The scan rate, the rate at which the three digits appear, is controlled by an external capacitor connected to C1A and C1B. A 1.0-μF capacitor generates a scan rate of about 3 Hz, or a new bcd digit every 333 ms. This scan frequency is applied to all three counters simultaneously, so that the bcd digits of all sets of Q3 through Q0 change at the same time. The scan frequency has no relation to the 60-Hz clock frequency.

The DS3 – DS1 outputs indicate which digit is being displayed on the Q3 through Q0 outputs. If DS3 = 0, the most significant bcd digit is being output; if DS2 = 0, the next bcd digit is being output; and if DS1 = 0, the least significant bcd digit is being output. To read the current count, three reads of the three sets of Q3 through Q0 are done—at DS1 time,

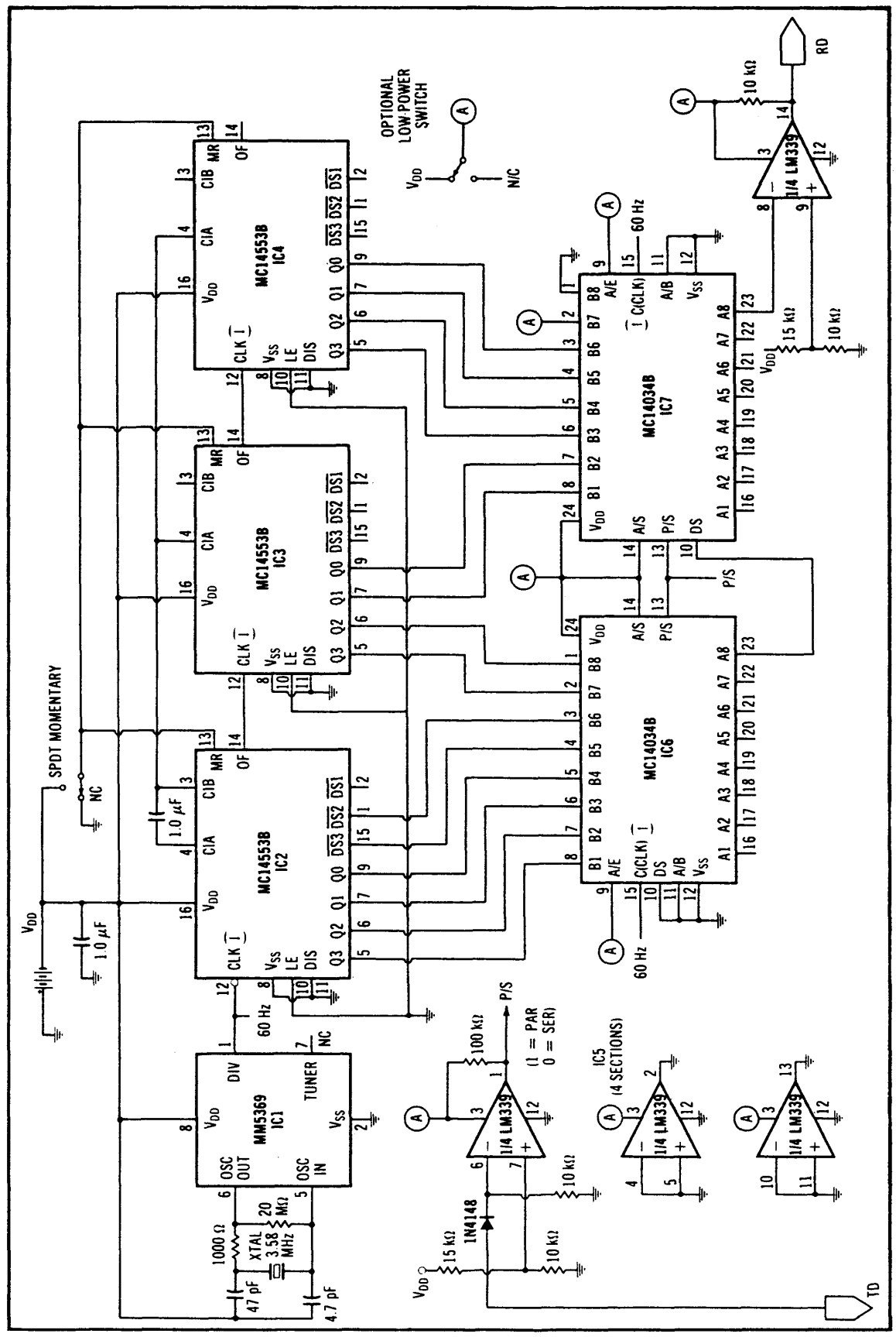


Fig. 7–2. Half-year clock detailed logic diagram.

DS2 time, and DS3 time. When the outputs are shuffled around in the proper order, the 9 digits represent the current count.

The OF output is the output to the next counter. This appears on the 1000th count when the counter recycles to 000. A disable line (DIS) and latch enable line (LE) are not used in this configuration. MR is master reset and is used to reset the counters to when a momentary switch is pulsed.

Bus Register Chips

The bus register chips are Motorola MC14034Bs. These chips contain two 8-bit registers and can operate in a number of different modes, depending upon the configuration of the A/E, P/S, A/B, and A/S inputs. The two modes we are using here are synchronous parallel data input and synchronous serial data input.

In the first mode, parallel data is strobed in or recorded on the clock input. In this case, 14 bits of data from the counter chips are strobed in. Twelve of these bits are the current bcd digits from each counter chip, and two are the DS3 and DS2 scan signals. (The third scan signal, DS1, is not used, since it must be active if either DS3 or DS2 is not active.)

When signal P/S is a logic 1, the bus registers are in the parallel data input mode and the 14 lines are continuously strobed in on the rising edge of every C (CLK) input. As the C input is the 60-Hz signal, the 14 lines are recorded 60 times per second. When signal P/S is a logic 0, the bus registers are in the serial data input mode. This is somewhat of a misnomer, as this mode not only shifts in but shifts out data previously recorded. In this case the 14 bits earlier recorded, in addition to a leading 0 and 1, are shifted out at a 60-Hz rate. The leading 0 and 1 allow synchronization of the serial bit stream.

RS-232 Interface

The RD data line (actually a TD line, viewed from the HYC) is driven by the output of the least-significant bit of the lower order bus register chip. The output is about 0 volts if the data bit is a one or about 5 volts if the data bit is a zero. The bit time of this output is 1/60th of a second, or about 16.66 ms. The RD output goes into the RS232IN bit of PIA $FF22 and is logically equivalent.

The TD data line (actually an RD line, viewed from the HYC) operates identically to the RD comparator in the Color Computer. The P/S output signal changes from 0 to close to 9 volts for a positive or negative input, respectively. The P/S signal is logically equivalent to the RS232OUT bit in the $FF20 PIA.

CMOS Circuitry

All chips except the LM339 chip are CMOS (complementary metal oxide semiconductor) chips. CMOS is characterized by low power consumption. The HYC requires less than 4 mA of current. If the optional power switch for the noncounting chips is used, this current requirement will be about one-half of that amount. The power switch can be used to extend battery life when the HYC is not connected to the system or when the Color Computer is not in use.

A typical alkaline battery has a capacity of about 2 Ah, making the HYC functional for about 250 hours of continuous use in the low-power mode or about 150 hours of continuous use without the optional power switch. This 5–10 day life can be extended by paralleling a number of 9-volt batteries or by using a larger battery such as the NEDA 1603 size, which will not fit in the case used here but will last over 1000 hours in the low-power mode.

CMOS operates from about 3 to 18 volts of supply voltage; the voltage of the supply can be degraded quite far before the HYC will stop operating. The limiting factor is the RD output, which must swing from 0 to at least +2.6 volts for proper Color Computer comparator operation on the RD line.

CONSTRUCTION OF THE HYC

All parts in the HYC are easy to obtain. The 3.58-MHz crystal, oscillator/divider, counter chips, and LM339 are stocked by Radio Shack, and the bus register chips are available from any well stocked IC house (see the ads in any issue of BYTE). Cost of all parts should be under $20. See the parts list in Table 7–1.

The HYC is housed in a project case (Radio Shack 270-219). This plastic case has a built-in compartment large enough to hold a 9-volt

Table 7–1. Parts List for a Half-Year Clock

Amt.	Description
1	IC, Motorola MM5369 oscillator/divider (IC1)*
2	IC, Motorola BCD counter (IC2–IC3)*
1	IC, LM339 comparator (IC5)*
2	IC, Motorola MC14034B universal bus register (IC6–IC7)
1	Resistor, 1K, ¼ W 10%*
4	Resistor, 10K, ¼ W 10%*
2	Resistor, 15K, ¼ W 10%*
1	Resistor, 100K, ¼ W 10%*
1	Resistor, 20 megohm, ¼ W, 10% (two 10-megohm resistors can be used)*
1	Capacitor, 4.7 pF disc*
1	Capacitor, 47 pF disc*
2	Capacitor, 1 μF electrolytic, 25 V*
1	Crystal, 3.58 MHz*
1	Switch, miniature, spdt momentary*
1	Switch, miniature, spst or spdt (optional)*
1	Diode, 1N4000 series (not critical)*
1	Connector, 4-pin male DIN
1	Battery, 9 V, alkaline*
1	Socket, 8-pin wire-wrap*
1	Socket, 14-pin wire-wrap*
3	Socket, 16-pin wire-wrap*
2	Socket, 24-pin wire-wrap
1	Board, pc, prototype*
1	Case*
1	Cable, ribbon, 3- or 4-conductor*
Misc.	Wire-wrap wire, solder, No. 14 wire, battery connectors*

*Available from Radio Shack at time of writing

battery. A 2¾- by 3¾-inch grid board is used to hold the components. Two number 14 bus wires were run down the center of the board. One is used for the V_{DD} bus (+9 volts) and the other is ground. See Fig. 7–3.

The board has circular printed-circuit pads on one side. Mount the IC sockets, resistors, capacitors, and crystal on the side of the board without the etch. Solder the two opposing pins of the IC sockets. Leave the resistor, capacitor, and crystal leads uncut for wire-wrapping.

Wire-wrap the IC pins as shown in Table 7–2. I would recommend buying precut wire-wrap wire in lengths of 1 inch, 2 inches, and 3 inches. It is inexpensive and will cut the wire-wrap time by one-half. Follow the

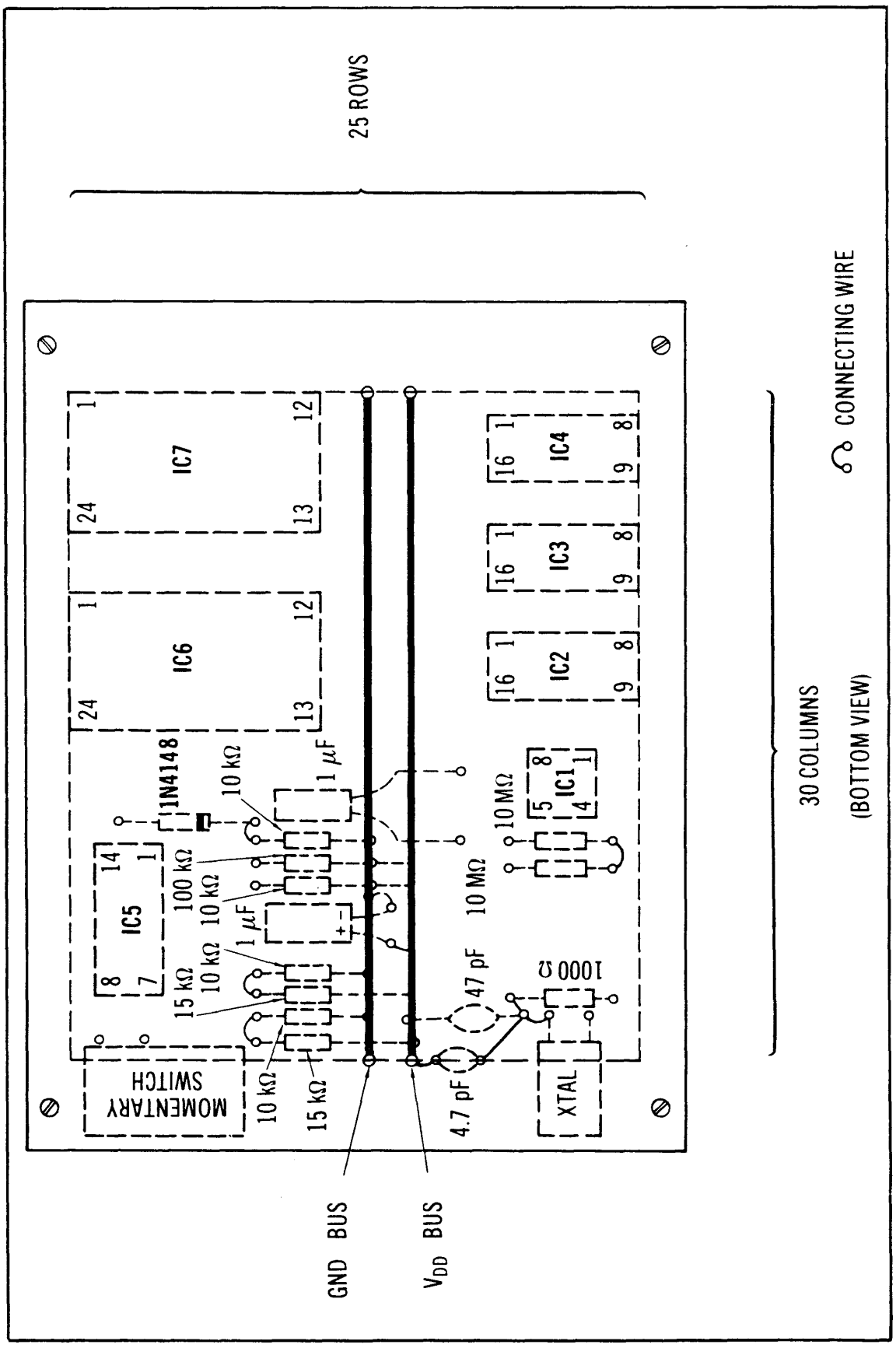


Fig. 7–3. Half-year clock physical layout.

Table 7–2. Half-Year Clock Wire List

MM5369 (IC1)	IC1 – 1 to IC2 – 12 IC1 – 2 to GND IC1 – 5 to 20 M/XTAL IC1 – 6 to 20 M/1K IC1 – 8 to V_{DD}
MC14553B (IC2)	IC2 – 1 to IC6 – 3 IC2 – 3 to 0.47 μF IC2 – 4 to 0.47 μF IC2 – 5 to IC6 – 8 IC2 – 6 to IC6 – 7 IC2 – 7 to IC6 – 6 IC2 – 8 to GND IC2 – 9 to IC6 – 5 IC2 – 10 to IC2 – 11 IC2 – 11 to IC2 – 8 IC2 – 12 to IC6 – 15 IC2 – 13 to SPDT-Center or IC2 – 11 IC2 – 14 to IC3 – 12 IC2 – 15 to IC6 – 4 IC2 – 16 to V_{DD}
MC14553B (IC3)	IC3 – 4 to IC2 – 3 IC3 – 5 to IC6 – 2 IC3 – 6 to IC6 – 1 IC3 – 7 to IC7 – 8 IC3 – 8 to GND IC3 – 9 to IC7 – 7 IC3 – 10 to IC2 – 10 IC3 – 11 to IC3 – 8 IC3 – 13 to IC2 – 13 IC3 – 14 to IC4 – 12 IC3 – 16 to IC2 – 16
MC14553B (IC4)	IC4 – 4 to IC3 – 4 IC4 – 5 to IC7 – 6 IC4 – 6 to IC7 – 5 IC4 – 7 to IC7 – 4 IC4 – 8 to GND IC4 – 9 to IC7 – 3 IC4 – 10 to IC3 – 10 IC4 – 11 to IC4 – 8 IC4 – 13 to IC3 – 13 IC4 – 16 to V_{DD}

Table 7–2—Cont. Half-Year Clock Wire List

Oscillator	20 megohm/IC1 – 6 to 1K 20 megohm/IC1 – 5 to XTAL
RS-232-C Cable	Diode (+) to RD TD to 10K/IC5 – 14 GND to GND
LM339 (IC5)	IC5 – 1 to 100K IC5 – 1 to IC6 – 13 IC5 – 2 to GND IC5 – 3 to V_{DD} IC5 – 4 to IC5 – 2 IC5 – 5 to IC5 – 2 IC5 – 6 to 10K/Diode IC5 – 7 to 10K/15K IC5 – 8 to IC7 – 23 IC5 – 9 to 10K/15K IC5 – 10 to IC5 – 13 IC5 – 11 to IC5 – 13 IC5 – 12 to GND IC5 – 13 to GND IC5 – 14 to 10K/TD
MC14034B (IC6)	IC6 – 9 to IC6 – 24 IC6 – 10 to IC6 – 11 IC6 – 11 to IC6 – 12 IC6 – 12 to GND IC6 – 13 to IC7 – 13 IC6 – 14 to IC6 – 24 IC6 – 23 to IC7 – 10 IC6 – 24 to V_{DD}
MC14034B (IC7)	IC7 – 1 to IC7 – 11 IC7 – 2 to IC7 – 9 IC7 – 9 to IC7 – 24 IC7 – 11 to IC7 – 12 IC7 – 12 to GND IC7 – 14 to IC7 – 24 IC7 – 15 to IC6 – 15 IC7 – 24 to V_{DD}
Miscellaneous connections	SPDT – NC to GND SPDT – NO to V_{DD} GND Bus to GND V_{DD} Bus to 9 V

wire-wrap connections in the table. All detailed connections are shown, but obvious power supply connections are not indicated.

The V_{DD} and ground leads can be connected directly to the two buses. No switch is used for main power. The three leads to the serial port can be implemented with 3-conductor ribbon cable. The ribbon cable can be routed out into the battery compartment of the case and out between the compartment cover as shown in Fig. 7–4. The opposite end of the ribbon cable connects to the 4-pin male DIN plug, as shown in Fig. 7–5.

TESTING THE HARDWARE

When you've assembled the board, test the interconnections from Table 7–1 and Fig. 7–2. Invariably, there will be one or two miswires. (I once wired about 10 chips in mirror image fashion: pin 1 to 24, 2 to 23, etc.; you should be in better shape than this!) Use two common straight pins, clip leads, and an ohmmeter or continuity tester to check all connections.

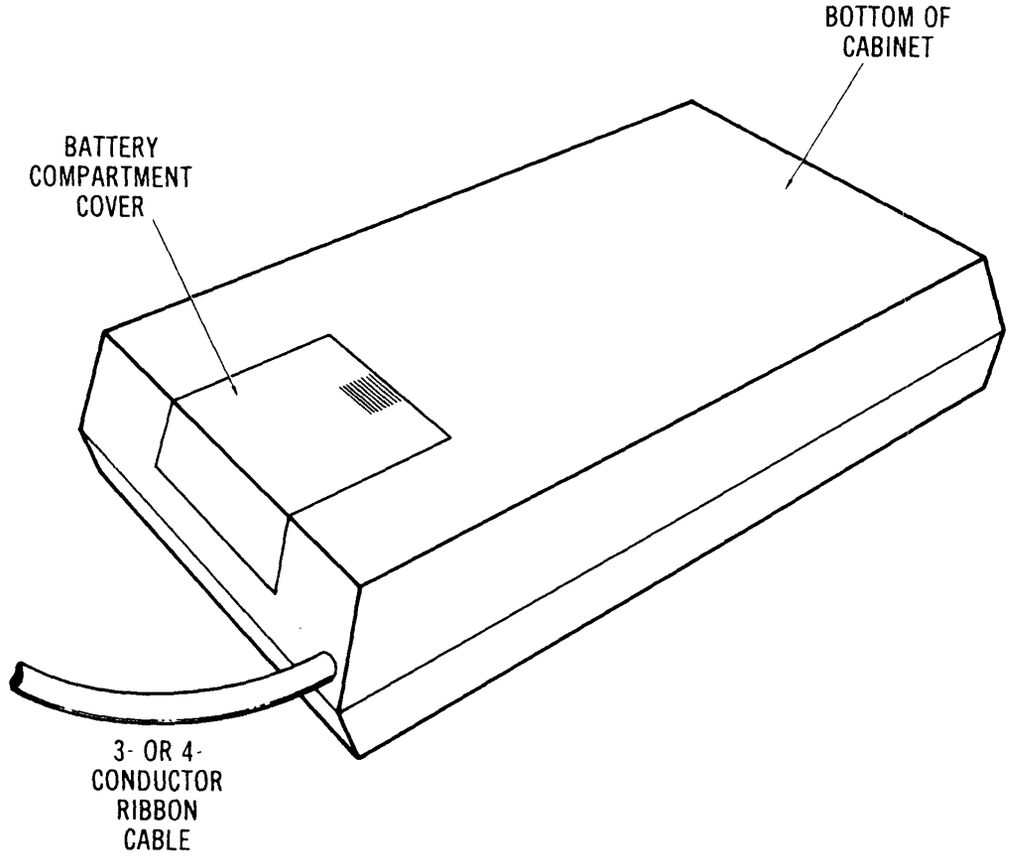


Fig. 7–4. Compartment cover.

When you're confident all the connections are proper, plug in the ICs. CMOS is not as intolerant of static electricity as it once was (the dark days of assembly people standing on antistatic mats with ground straps on their wrists!), but avoid handling the chips more than necessary.

Plug in the 9-volt battery and you should be in operation. If you have an oscilloscope, check between pin 3 of IC4 and ground. You should see the scan clock, operating at about 3 Hz. If it's running less than about 3 Hz, try different values for the 1.0-μF capacitor connected between pins 3 and 4 of IC2. Also check the oscillator/divider output. You should see a clean 60-Hz square wave. If you do not have an oscilloscope, recheck those connections!

HYC SOFTWARE

The HYC, in keeping with the traditions of the Color Computer system, is largely software dependent. The program shown in Fig. 7–6 is a

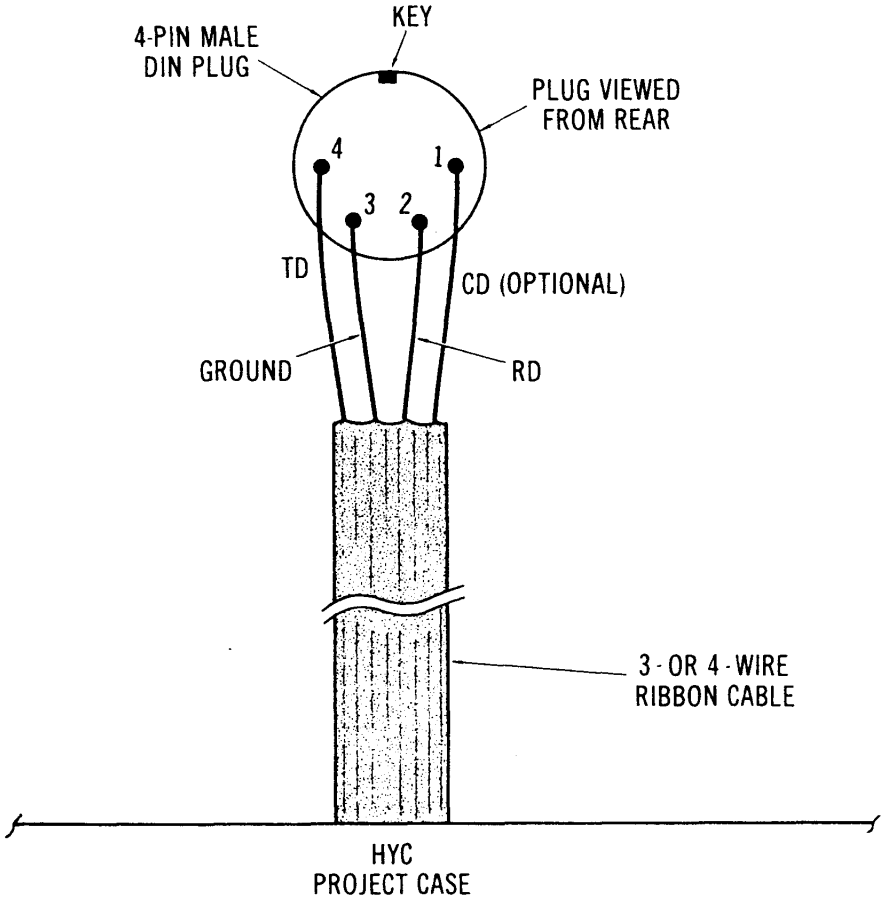


Fig. 7–5. DIN connections.

```
3F00                00100          ORG     $3F00
           3FF0     00110 BUFFER   EQU     $3FF0
                    00120 *************************************************
                    00130 * CLOCK I/O HANDLER.  READS IN 3 CLOCK COUNTS      *
                    00140 *************************************************
3F00 108E 0003      00150 CLOCK    LDY     #3      SCAN #
3F04 CE   3FF0      00160          LDU     #BUFFER
3F07 86   02        00170 CLK010   LDA     #2        2 TO A
3F09 B7   FF20      00180          STA     $FF20     STROBE IN NEXT COUNT
3F0C 86   26        00190          LDA     #38       1.9 CYCLE COUNT
3F0E 8D   47        00200          BSR     DELAY     DELAY 1.9 60 HZ CYCLES
3F10 B7   FF20      00210          STA     $FF20     START SERIAL
3F13 8D   25        00220          BSR     GETSER    GET SERIAL IN
3F15 26   F0        00230          BNE     CLK010    GO IF INVALID
3F17 86   02        00240          LDA     #2        DELAY COUNT
3F19 8D   3C        00250          BSR     DELAY     DELAY 1/10 CYCLE
3F1B 8D   1D        00260          BSR     GETSER    GET SERIAL IN BIT
3F1D 26   E8        00270          BNE     CLK010    GO IF INVALID
3F1F 8D   19        00280 CLK020   BSR     GETSER    GET SERIAL IN BIT
3F21 27   FC        00290          BEQ     CLK020      GO IF 0
3F23 86   1E        00300          LDA     #30       1+1/2 CYCLE COUNT
3F25 8D   30        00310          BSR     DELAY     DELAY 1+1/2 CYCLE
3F27 8D   17        00320          BSR     INPUT     READ CLOCK COUNT
3F29 33   5E        00330          LEAU    -2,U      RESET PNTR
3F2B 1F   20        00340          TFR     Y,D       SCN # NOW IN B
3F2D EB   41        00350          EORB    1,U       TEST SCN #
3F2F C4   03        00360          ANDB    #3        MASK OUT DIGIT
3F31 26   D4        00370          BNE     CLK010    GO IF NOT PROPER SCN #
3F33 33   42        00380          LEAU    2,U       POINT TO NEXT 2 BYTES
3F35 31   3F        00390          LEAY    -1,Y      DECREMENT SCN #
3F37 26   CE        00400          BNE     CLK010    GO IF NOT DONE
3F39 39             00410          RTS             RETURN
3F3A B6   FF22      00420 GETSER   LDA     $FF22   GET SERIAL BIT
3F3D 84   01        00430          ANDA    #1      TEST
3F3F 39             00440          RTS             RETURN
                    00450 *************************************************
                    00460 * INPUT.  INPUTS 14 BITS AND STORES IN TWO BYTES    *
                    00470 *  ENTRY:  BUFFER ADDRESS IN U                      *
                    00480 *  EXIT:   2 BYTES STORED IN BUFFER, U UPDATED      *
                    00490 *************************************************
3F40 8D   00        00500 INPUT    BSR     INPUA   GO TWICE
3F42 4F             00510 INPUA    CLRA            0 TO A
3F43 A7   C4        00520          STA     ,U      CLEAR BYTE
3F45 C6   08        00530          LDB     #8      FOR 8 BITS
3F47 B6   FF22      00540 INP010   LDA     $FF22     GET BIT
3F4A 44             00550          LSRA              OUT TO C
3F4B 66   C4        00560          ROR     ,U        MERGE IN USER STACK
3F4D 86   14        00570          LDA     #20       1 CYCLE COUNT
3F4F 8D   06        00580          BSR     DELAY     DELAY 1 CYCLE
3F51 5A             00590          DECB              DECREMENT ITERATION COUNT
3F52 26   F3        00600          BNE     INP010    GO IF NOT 8
3F54 33   41        00610          LEAU    1,U     POINT TO NEXT STACK BYTE
3F56 39             00620          RTS             RETURN (AGAIN OR CLOCK)
                    00630 *************************************************
                    00640 * DELAY. DELAYS IN MULTIPLES OF .83333 MS           *
                    00650 * ENTRY:  COUNT IN A                               *
                    00660 * EXIT:   AFTER DELAY                             *
                    00670 *************************************************
3F57 8E   005C      00680 DELAY    LDX     #92     FINAGLE FACTOR
3F5A 30   1F        00690 DEL010   LEAX    -1,X      DECREMENT X
3F5C 26   FC        00700          BNE     DEL010    GO IF NOT 0
3F5E 4A             00710          DECA            DECREMENT MAJOR COUNT
3F5F 26   F6        00720          BNE     DELAY   GO IF NOT 0
3F61 39             00730          RTS             RETURN
           0000     00740          END
```

Fig. 7–6. Basic clock I/O handler.

6809 assembly language program that reads data from the HYC. It resides in the upper 256 bytes of RAM in a 16K Color Computer system. Protect this area by a CLEAR 200,&H3EFF when running the program with BASIC.

HOW THE HYC PROGRAM WORKS

The clock I/O handler is divided into three parts. The main loop CLOCK, the INPUT subroutine, and the DELAY subroutine. The DELAY subroutine delays in multiples of 0.8333 second. One 60-Hz pulse has a duration or period of 16.666 ms. This subroutine can be conveniently used for delaying in multiples or submultiples of one 60-Hz bit time.

The INPUT subroutine makes a single read of 16 bits of data from the counters. The BSR at INPUT calls INPUA, resulting in the code from INPUA through the RTS being executed twice. Eight bits of data are read each pass through the code.

The loop at INP010 reads in 8 data bits from the $FF22 PIA. The single bit in bit position 0 is shifted right into the carry condition code and then rotated into the byte pointed to by the user stack pointer. The byte is initially cleared to 0. At the end of the second pass through INPUA, two bytes of data have been stored in the user stack, representing one complete read of three digits.

The main loop at CLOCK performs consecutive calls of the INPUT subroutine until three samplings of DS1, DS2, and DS3 have been compiled in the 6 bytes of the buffer. First, a 1 is output to bit 1 of the $FF20 PIA, bringing P/S to a 1. A delay of 1.9 cycles is then done so that the data can be clocked into the bus registers.

After the delay, the serial output is started by outputting a 0 to bit 1 of the $FF20 PIA, bringing P/S to a 0. Immediately after the output, the serial data is checked by reading PIA $FF22, bit 0. If the data is a 1, the first clock occurred too close to the initialization of the process and the process is repeated from CLK010. If the data is a 0, a delay of 1/10 cycle is done and a test for 0 is done again. If the data is not 0, the clock occurred within 1/10 cycle and the process is repeated.

If the first bit is a 0, the loop at CLK020 delays until the appearance of a 1. At this point, the second clock has (just) occurred. A delay of 1½ cycles is then done to position the next read in the middle of the third data bit time. The INPUT subroutine is then called to read in the next 16 bits. The last two of these will be zeros.

Now the scan number of the first 16 bits is tested. If not equal to binary 11 or DS1, the process is repeated from CLK010. If equal to DS1, the user stack pointer is adjusted, the scan number is adjusted to 2, and another read from CLK010 is done to read the DS2 cycle. A third itera-

tion reads the last cycle, DS3. The short subroutine at GETSER gets the serial bit and tests it, changing the Z condition code to zero or nonzero.

RUNNING THE PROGRAM

Key in the 97 bytes of the program or use POKEs in your BASIC program. The program is relocatable with the exception of 2 bytes. Change the second and third bytes of the second instruction (locations $3F05,6 in Listing 1) if you relocate the program. These 2 bytes should hold the address of a 6-byte buffer. The program can be keyed in any protected area of memory or reassembled at any desired location.

A simple BASIC test program is shown in Fig. 7–7. This program defines the location of the program by the DEFUSR0 statement. (Change this statement if you have relocated the machine-language code.) The assembly language program is called by the USR0 call and returns with 6 bytes of the current clock count in locations $3FF0 through $3FF5. These six locations represent the bcd digits and scan numbers as shown in Fig. 7–8. Sample outputs are shown in Fig. 7–9.

```
100 DEFUSR0=&H3F00
110 A=USR0(0)
120 FOR I=&H3FF0 TO &H3FF5
130 PRINT PEEK(I),
140 NEXT I
150 PRINT:PRINT:GOTO 110
```

Fig. 7–7. BASIC test program.

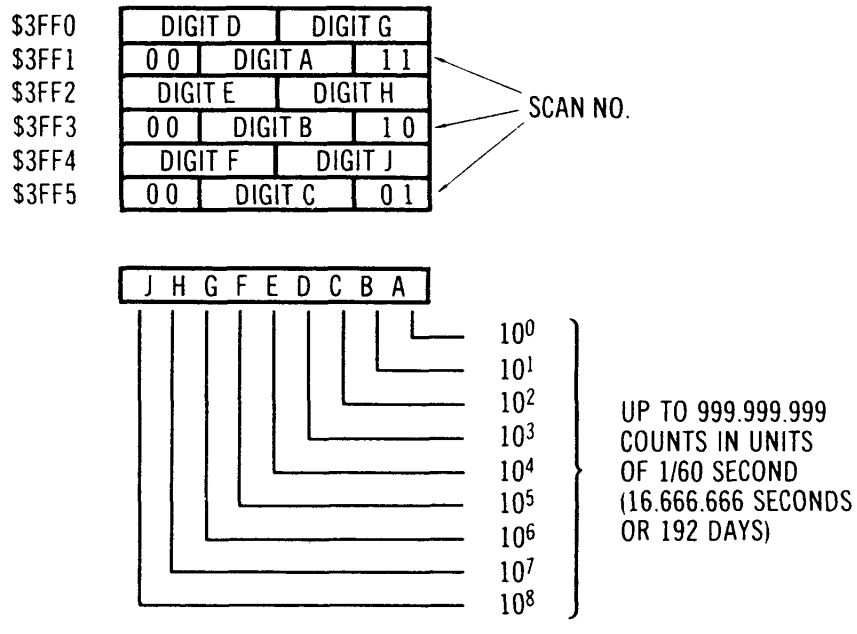


Fig. 7–8. Bcd digits and scan numbers.

A general-purpose BASIC driver is shown in Fig. 7–10. This program displays the actual number of days, hours, minutes, and seconds represented by the count in the HYC. This count can be held in a Color Computer BASIC variable, which allows 9 decimal digits of precision.

A "bias" count may be input to the program before sampling of the count. This bias may be positive or negative to adjust the current count to a previous starting point or to "trim" the time. The bias is in one-sixtieth

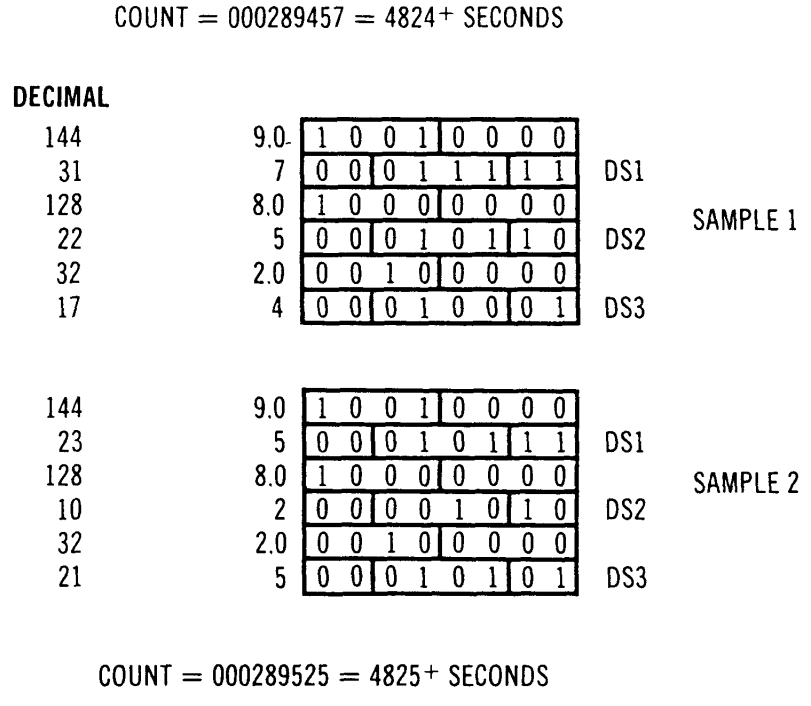


Fig. 7–9. Sample outputs for HYC.

```
100 INPUT "TIME IN 60THS";T2
110 CLS
120 DEFUSR0=&H3F00
130 T0=-1
140 A=USR0(0)
150 A=INT(PEEK(&H3FF1)/4)
160 B=INT(PEEK(&H3FF3)/4)
170 C=INT(PEEK(&H3FF5)/4)
180 D=INT(PEEK(&H3FF0)/16)
190 E=INT(PEEK(&H3FF2)/16)
200 F=INT(PEEK(&H3FF4)/16)
210 G=(PEEK(&H3FF0) AND 15)
220 H=(PEEK(&H3FF2) AND 15)
230 J=(PEEK(&H3FF4) AND 15)
240 T1=(J*100000000+H*10000000+G*1000000+F*100000+E*10000+D*1000+C*100+B*10+A)
242 IF T0=-1 THEN T0=T1
250 IF T1<T0 THEN GOTO 140 ELSE IF T1-T0>999 THEN GOTO 140 ELSE T0=T1
260 T3=T1+T2: IF T3>999999999 THEN T3=T3-999999999
270 T3=INT(T3/60)
280 D=INT(T3/86400):H=INT((T3-D*86400)/3600):M=INT((T3-D*86400-H*3600)/60):S=T3-
D*86400-H*3600-M*60
290 PRINT @ 256, "DAY";D;H;"HOURS";M;"MINS";S;"SECS"
300 GOTO 140
```

Fig. 7–10. BASIC driver for HYC.

of a second units. Use a value of 60 for every second, 3600 for every minute, 216000 for every hour, or 5184000 for every day. The RESET switch is a momentary switch that resets the entire count to 0.

The tests of the current count (T1) with the previous count (T0) require some explanation. In most cases, the sampling process will read count data in a nonchanging state. However, because the scan clock occurs at unpredictable times, the count may be sampled in the middle of a scan clock edge, yielding invalid data. Because of this, T1 (current) is compared to T0 (old). If T1 is less than T0, T1 is invalid and another sample is made. If T1 is less than T0 by 999 counts (16.65 seconds), T1 is considered invalid and another sample is made. The typical display generated by the program shows the time changing every 2 seconds, with occasional lapses of up to 4 seconds. Tests run over days showed less than one (detected) invalid read per minute with a maximum delay of 5.5 seconds. No invalid times appeared.

If the HYC is to be called at random times, make three calls and test for ascending counts with a difference of less than 10 seconds or so. If this is done, the resulting time will be accurate to within 10 seconds of the actual time.

The crystal used should be an excellent time base. It may be fine tuned, however, by substituting a 5–50-pF trimmer capacitor in place of the 4.7-pF capacitor that connects to the crystal.

In operation, the HYC can be disconnected from the system at any time (without powering the Color Computer down) and left running. It can be reconnected at any later time. (BREAK the BASIC program above if this is done to prevent a hangup from T1–T0 greater than 999.)

Had Rip Van Winkle owned a Color Computer, he would have loved the half-year clock. If you have nothing to do for the next 192 days, why not check this project out with your Color Computer to test the accuracy? Or count clock pulses instead of sheep: 999,998,767; 999,998,768; 999,998,769 . . .

chapter **8**

A Data Comm Plugboard

This chapter describes a data communications plugboard that will help in hooking up RS-232-C devices to the Models I and III. Typically, computer users experience a great deal of trouble in connecting these devices for a simple reason: Although the RS-232-C standard rigidly defines the signals involved in data communications, there is a great deal of variation about which signals are used in any given piece of equipment.

THE DATA COMM PLUGBOARD IDEA

The plugboard (Fig. 8–1) interrupts or "breaks" a 25-line RS-232-C cable, routing the lines to a prototype board. To continue each line through the board, a short length of 20-gauge solid wire connects the two sides, as shown in Fig. 8–2. To transpose RD and TD, for example, simply criss-cross pins 2 and 3 as shown in the figure. Any other lines may be connected by short lengths of wire. To test the state of any line, an LED may be connected to signal ground, pin 7, and the line in question, as shown in Fig. 8–2. Lines may be "dummied up" in lieu of wiring up a special RS-232-C plug simply by adding a patch between an active line and a line to be dummied.

To construct the plugboard, use the smaller version of the Radio Shack prototype board (276-175). The back has a sticky paper cover. Peel this off to expose the interconnecting strips, as shown in Fig. 8–3. The rows are numbered from 1 to 23 on the front of the board.

You'll need two RS-232-C connectors. Get the solder type, not the insulation displacement type. (Radio Shack sells the latter type.) Use the proper configuration for the equipment you'll be interfacing. Modems,

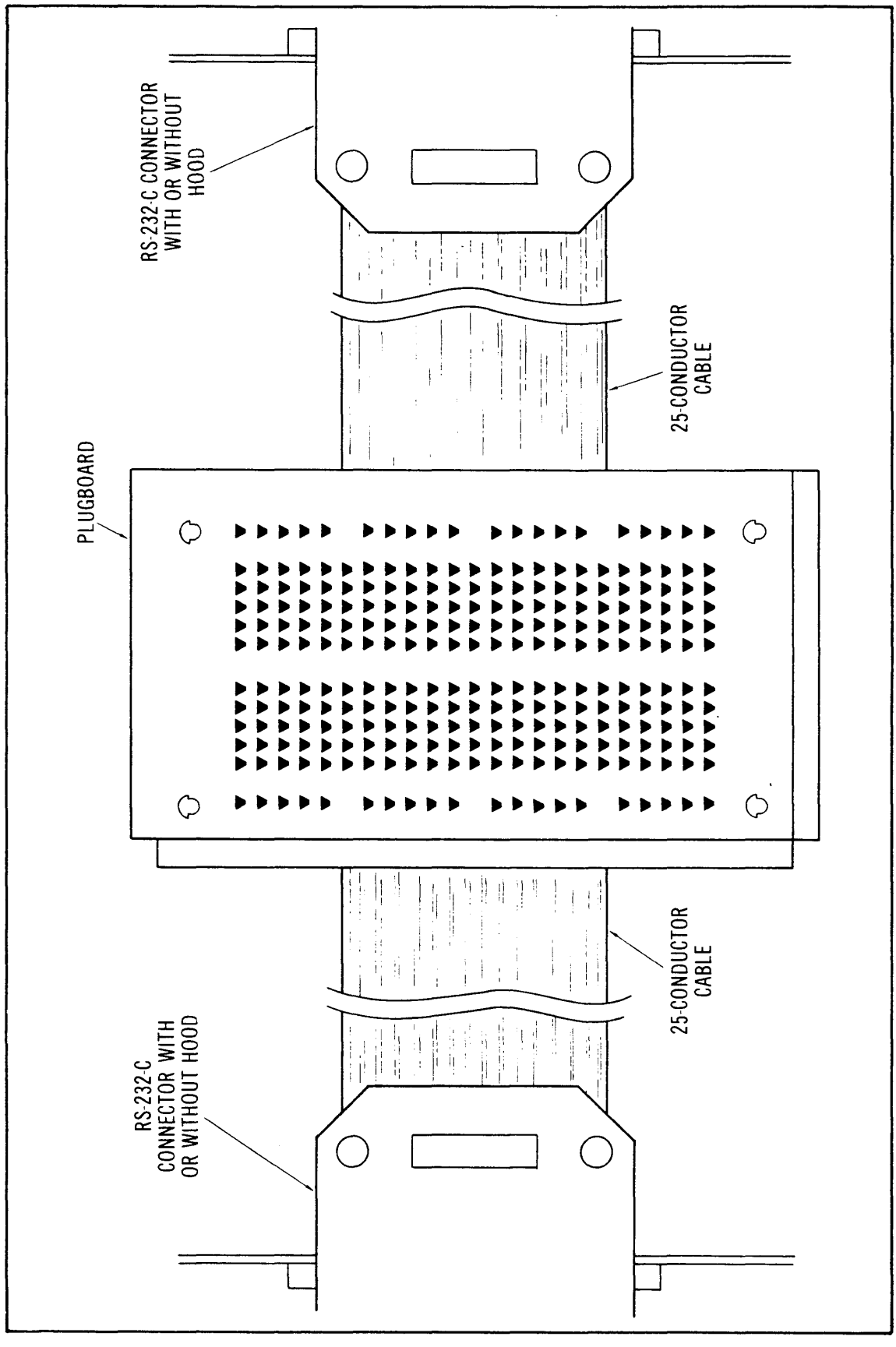


Fig. 8–1. Data comm plugboard layout.

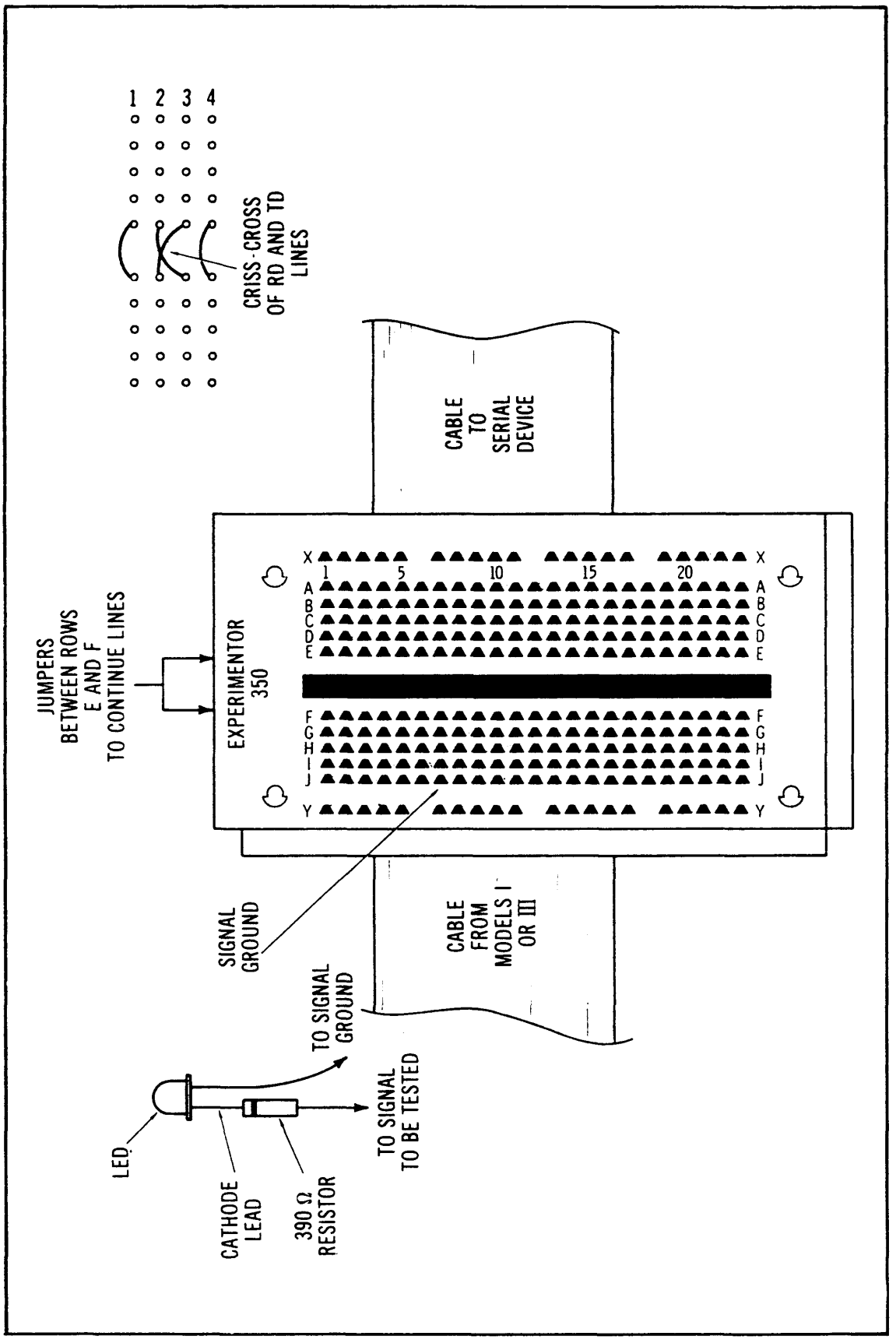


Fig. 8–2. Typical use of plugboard.

for example, generally have a female RS-232-C connector and require a male connector on the cable.

The connectors' pins are numbered; the numbering on many connectors is almost impossible to see unless you hold the connector at the right

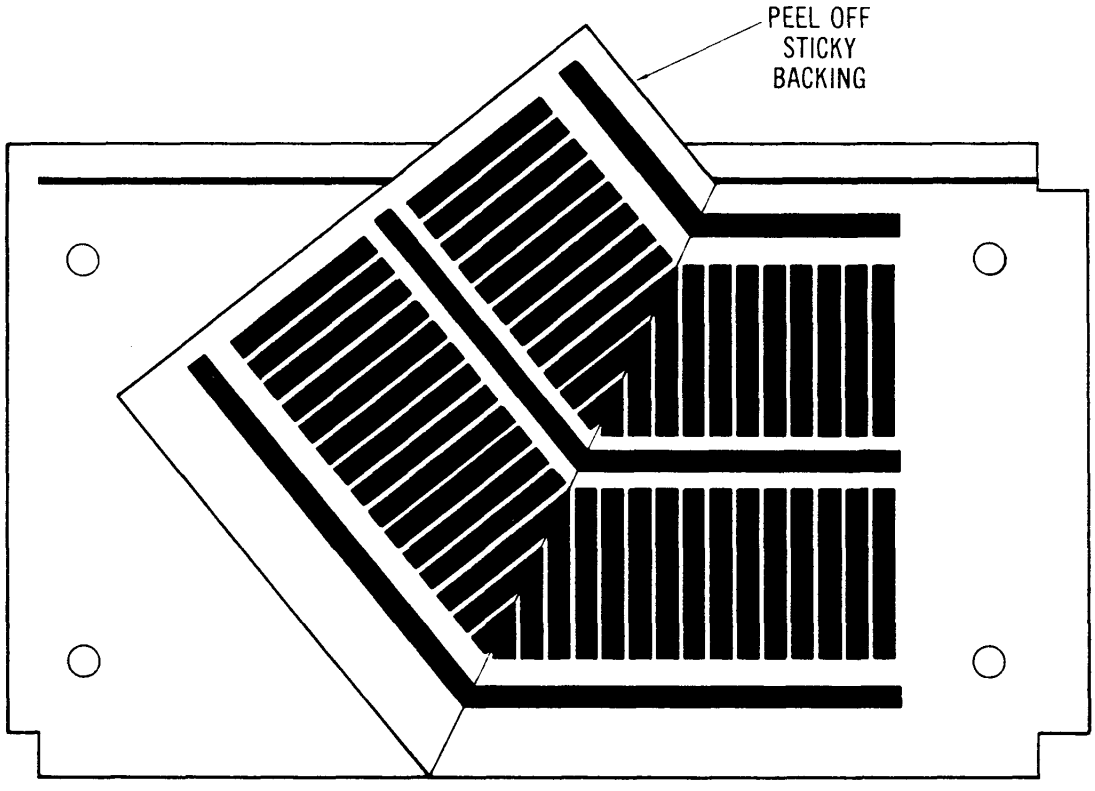


Fig. 8–3. Back of prototype board.

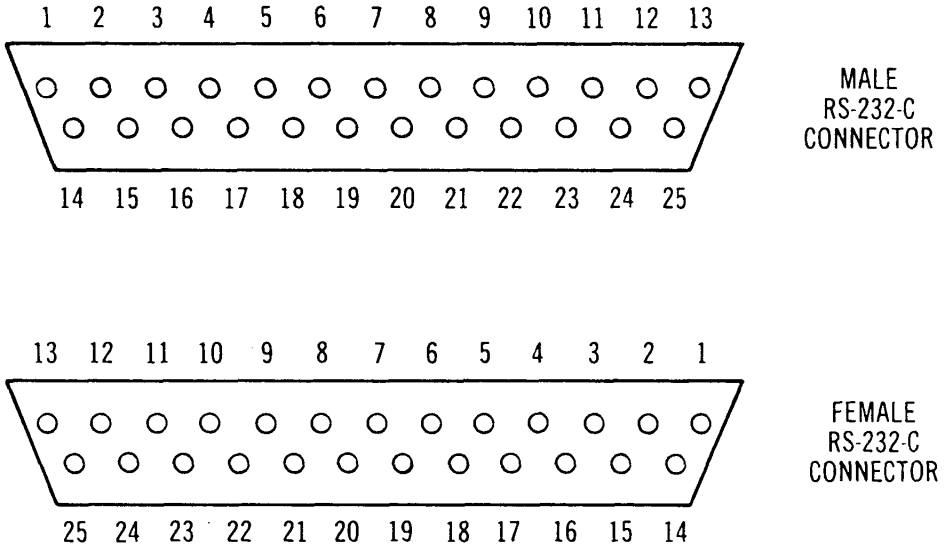


Fig. 8–4. Standard RS-232-C connector numbering.

angle with the right light. Standard numbering for RS-232-C connectors, looking into the connector, is shown in Fig. 8–4.

Obtain a 25-wire ribbon cable (RS 278-771 is a 40-conductor cable that may be split). Wire the ribbon cable so that the wire positions correspond to the RS-232-C pin numbers on both RS-232-C connectors. This will involve separating the cable into two halves, as shown in Fig. 8–5.

Cut the RS-232-C cable in half. Separate the wires on both cut ends of the cable and strip each wire with a wire stripper about 1/16 inch. Solder 23 wires onto each of the 23 rows of the board. Match the row number with the RS-232-C connector pin number. Solder the 24th wire (pin 24) to the vertical strip on each side of the board. Cut off the 25th wire. This signal is undefined in the RS-232-C specification. See Fig. 8–6.

Jumpers are made of 20-gauge wire, about the best gauge for the grip connectors of the board. Jumper opposing pins to continue the lines unchanged. Only the Models I and III signals shown in Table 6–1 need to be jumpered.

You're now set to experiment with the programming examples below, or to use the plugboard to help in connecting serial devices to your system.

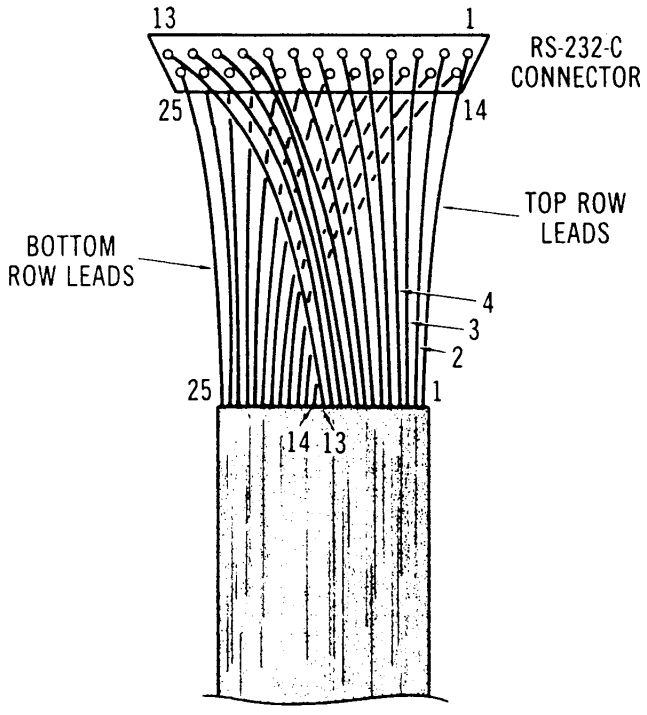


Fig. 8–5. Cable to RS-232-C connector wiring.

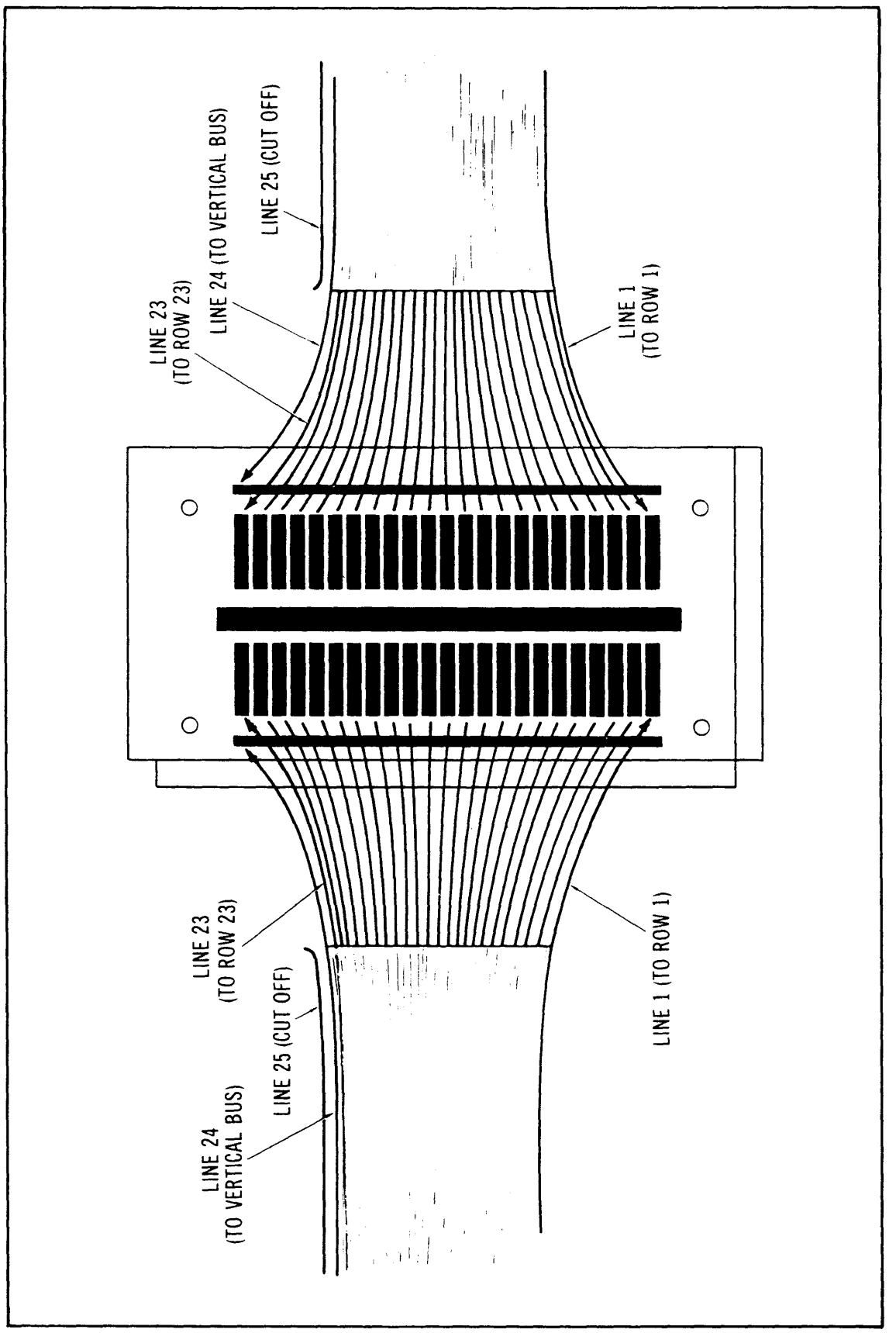


Fig. 8–6. Cable to prototype board wiring.

RS-232-C PROGRAMMING EXAMPLES

The examples below are in BASIC. They can be converted to Z-80 assembly language by substituting INs and OUTs for the BASIC INPs and OUTs. We don't have the space to show you complete serial printer or communications drivers, but what we're attempting to do is to take the mystery out of the actual interfacing to the Models I and III RS-232-C.

Setting the RTS, DTR, and Break Lines

The RTS and DTR lines are outputs from the RS-232-C controller that carry request-to-send and data-terminal-ready signals. The RTS line (pin 4) is set by

```
100 OUT 232,0           'initialize RS-232-C
110 OUT 234,xxxxxxx1    'binary value with lsb set
```

The "xxxxxxx1" indicates that bit 0 is set for the RTS. The DTR line (pin 20) is set similarly:

```
100 OUT 232,0           'initialize RS-232-C
110 OUT 234,xxxxxx1x    'binary value
```

If both RTS and DTR are to be set, the binary value would be xxxxxx11.

The "break" does not come out on the RS-232-C connector, but enables or disables the TD line. Use a 1 to enable the TD line:

```
100 OUT 232,0           'initialize RS-232-C
110 OUT 234,xxxxx1xx    'enable TD
```

Use the above code and experiment with the plugboard by connecting an LED and 390-ohm resistor between pin 7 of the plugboard and pin 4 or 20. The TRUE states of the lines are −12 V and the FALSE states are +12 vdc.

Reading the CTS (pin 5), DSR (pin 6), CD (pin 8), and RI (pin 22) Lines

These lines are inputs to the RS-232-C controller that transmit clear-to-send, data-set-ready, carrier-detect, and ring-indicator signals. To read the lines do:

```
100 OUT 232,0          'initialize RS-232-C
110 A=INP(232)         'read lines
```

The A variable will be a binary value corresponding to CTS, DSR, CD, RI, X, X, X, and X, where the Xs are "don't care" bits.

Again, a 1 line is −12 V and a 0 line is +12 V. You can experiment by first setting RTS or DTR and jumpering on the plugboard between the RTS or DTR pins to the four input lines. This jumpering is a common way to "dummy up" a signal, either by connector wiring or, in this case, on the plugboard.

Setting the SUN (pin 10), STD (pin 14), and SRTS (pin 19) Lines

These lines are the secondary lines not normally used in communications programs. They can be set by:

```
100 OUT 232,0          'initialize RS-232-C
110 A=INP(233)         'toggle CRL flip-flop
120 OUT 234,xxxxxxxx   'set SUN, STD, SRTS
130 A=INP(233)         'toggle CRL flip-flop
```

Set the binary value in line 120 to X, X, SUN, STD, SRTS, X, X, X. The INP (233) toggles the control register load flip-flop so that the control register is not loaded. Again, these lines are −12 V for a 1 or +12 V for a 0.

Outputting on TD (Pin 2)

The BASIC code shown in Fig. 8–7 provides a continuous output of a specified character. If you have an oscilloscope, you can connect the scope between pin 7 (SGND) and pin 2 (TD) and observe the output.

```
90 ' SERIAL DATA OUT EXERCISER
100 OUT 232,0
105 PRINT "INPUT EP, WLS, SSB, PI"
110 INPUT EP,WL,SS,PI
120 WD=EP*128+WL*32+SS*16+PI*8+4
130 OUT 234,WD
135 INPUT "BAUD CODE";BA
150 OUT 233,BA
160 INPUT "CHARACTER CODE";CH
165 I=0
170 A=INP(234)
180 IF (A AND 64)=0 THEN GOTO 170
185 I=I+1
190 OUT 235,CH
200 GOTO 170
```

Fig. 8–7. BASIC driver program for continuous output.

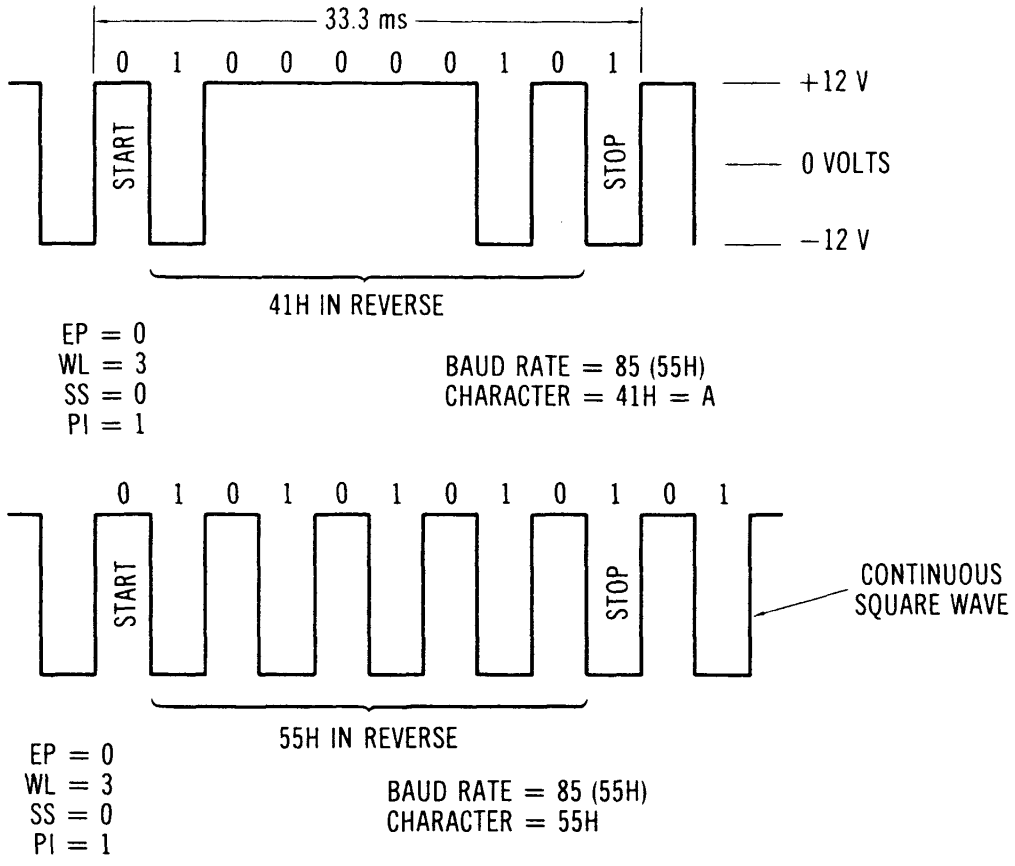


Fig. 8–8. Typical oscilloscope waveforms for CH=65 and CH=85.

The EP, WL, SS, and PI inputs define even/odd parity, word length, number of stop bits, and parity inhibit, respectively. This format data is sent to the control register by the OUT 234,WD. The baud-rate code is sent to the BRG by the OUT 233,BA.

The character value (CH) is the decimal equivalent of the character to be sent. The value must be in a range of 0 through 255. ASCII character A, for example, is decimal 65 (041H). The loop at 170 through 200 continuously checks the THRE status and, if the transmitter-holding register is empty, outputs the character to the THR by an OUT 235,CH.

Fig. 8–8 shows scope waveforms for CH=65 and CH=85 with no parity bit, 1 stop bit, and 8 data bits at a 300-baud rate. This BASIC loop keeps up quite well with the 30-character-per-second rate. Setting the baud rate to 600 (BA=102) during tests resulted in about 41 characters per second because of the BASIC overhead. It is feasible to drive a line printer in BASIC!

The last application is shown in Fig. 8–9. This is a loop-back where the TD line output is jumpered back to the RD (pin 3) line input. The

```
90 ' LOOP BACK EXERCISER
100 OUT 232,0
105 PRINT "INPUT EP, WLS, SSB, PI"
110 INPUT EP,WL,SS,PI
120 WD=EP*128+WL*32+SS*16+PI*8+4
130 OUT 234,WD
135 INPUT "BAUD CODE";BA
150 OUT 233,BA
160 INPUT "CHARACTER CODE";CH
165 I=0
170 A=INP(234)
180 IF (A AND 64)=0 THEN GOTO 170
185 I=I+1
190 OUT 235,CH
200 A=INP(234)
210 IF (A AND 128)=0 THEN GOTO 170
220 A=INP(235)
230 PRINT A
240 GOTO 170
```

Fig. 8–9. BASIC loop-back program.

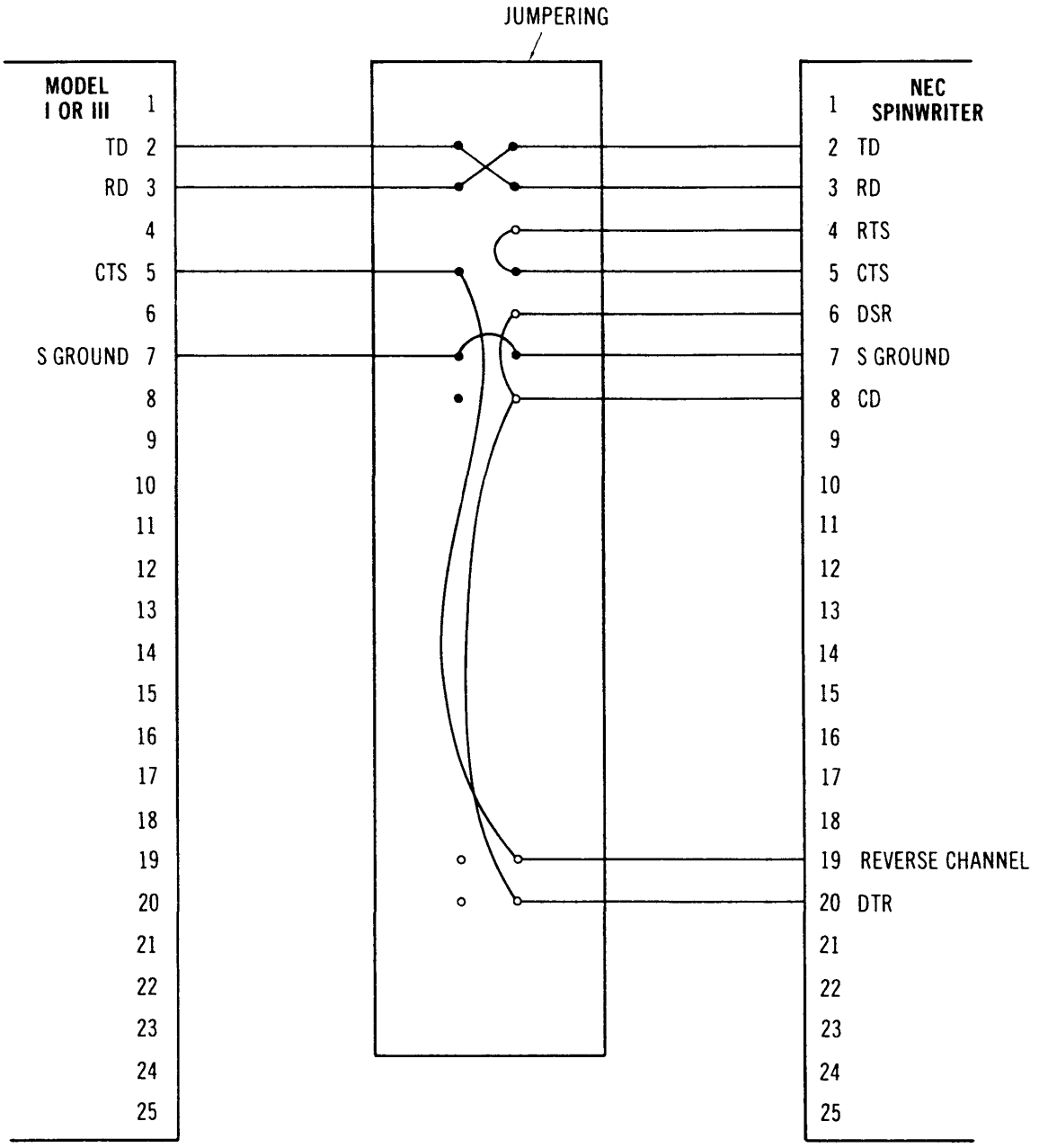


Fig. 8–10. Typical plugboard interconnections.

character sent out comes right back in on the RD line. This loop-back technique is commonly used for testing an RS-232-C interface "locally" and eliminating problems caused by malfunctioning communications equipment. Jumper the two lines by a short wire between pins 2 and 3 on the plugboard.

CONNECTING SERIAL DEVICES

There are so many serial devices that it's hard to generalize about proper RS-232-C cable configurations to drive the devices. A typical configuration which can be set up by the plugboard is shown in Fig. 8–10. This connects the NEC Spinwriter to the Model I or III and allows transfer rates of up to 1200 baud.

You can greatly facilitate connection of serial equipment if you read the interfacing requirements for the serial device to be used with your Model I or III, use the plugboard to test line conditions, and possibly even use some of the code provided above.

Section III

Using the Cassette Output Port on the Models I and III

chapter 9

Models I and III Cassette Output Circuitry

There's always an advantage in using existing hardware in interfacing external devices—there's no need to perform address decoding, hook up to a multiline bus, or to design and implement controller functions. The cassette port is the most rudimentary input/output port in the Models I and III. It was originally designed to interface to a cassette recorder so that BASIC and machine-language programs and data could be saved. The cassette port, however, can be used for a variety of other uses.

In this section we look at cassette output circuitry and associated projects for the Models I and III. Though the logic of the cassette port is covered briefly in Chapter 5, we reiterate it here for those readers who aren't interested in the analog-to-digital converter covered there.

The next three chapters feature three different projects that use the cassette port output. All projects will work with a Model I system without an expansion interface or with a Model III system. The projects are a tone generator with volume control, a telephone dialer, and an RS-232-C driver.

CASSETTE LOGIC IN THE MODELS I AND III

The Models I and III both use about the same logic in the cassette output circuitry, as shown in Fig. 9–1. The REM output to turn on the recorder is slightly different in address decoding between Models I and III, but in both cases it simply closes a relay. Two normally open relay contacts go to pins 1 and 3 of the cassette jack, a 5-pin DIN connector. We won't be using the relay output for these projects, as earlier relays were prone to sticking (especially when used to drive the ac supply for milling machines). What we will be using is the output that normally goes

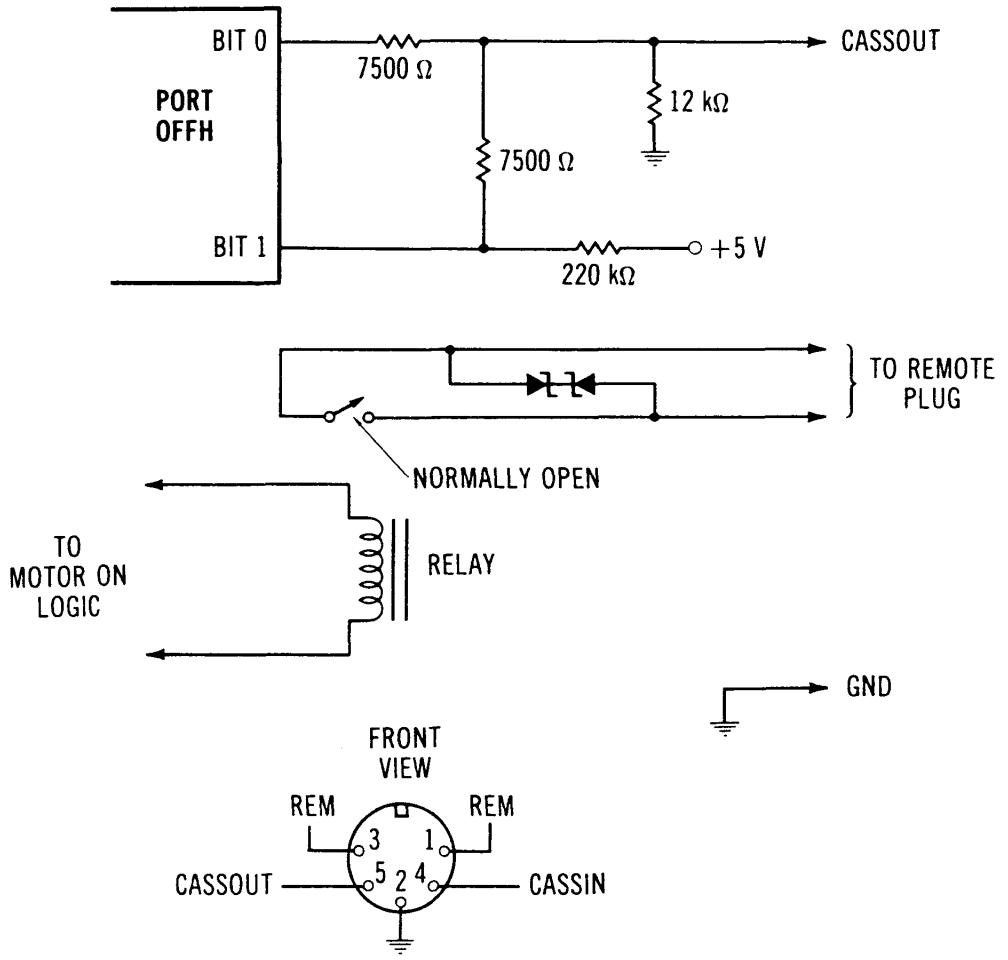


Fig. 9–1. Cassette output circuitry for the Models I and III.

to the AUX input of the cassette to write data on the tape. This is a single line connected to pin 5 of the DIN connector. This line is driven by 2 bits at I/O address 0FFH in both the Models I and III.

There are three voltage levels that can be output to the CASSOUT line, depending upon the configuration of the two least-significant bits of port 0FFH. See Table 9–1. A bit configuration of 01 binary produces

Table 9–1. Cassette Output

Port 0FFH BITS		
Bit 1	Bit 0	CASSOUT Voltage
1	0	≃0 V
0	0	≃0.4 V
0	1	≃0.8 V
1	1	≃0.4 V

about 0 volts, 00 produces about 0.4 volt, and 10 produces about 0.8 volt. Bit configuration 11 is redundant as it generates 0.4 volt again. The three signal levels are used to write cassette data in the 500-baud mode as shown in Fig. 9–2. A single square-wave cycle generates a clock pulse. A following cycle is output for a 1 data bit, or no output indicates a 0 data bit.

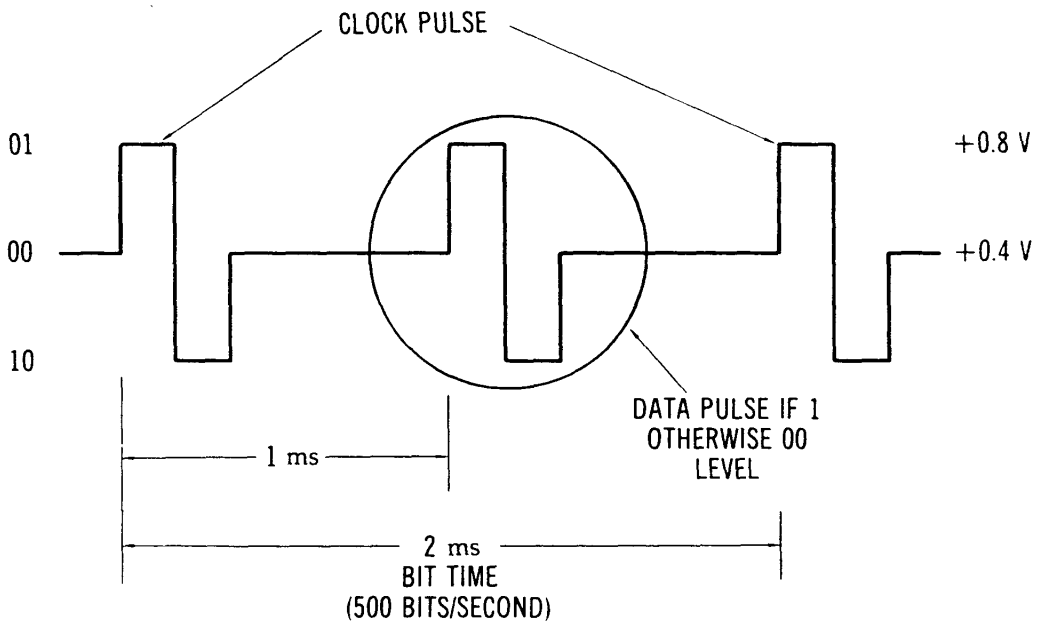


Fig. 9–2. 500-baud cassette output signals.

The Model III also has 1500-baud capability. In this case continuous frequency-shift keying is used to produce 1320-Hz or 2680-Hz tones to represent data. Only the 0-volt and 0.8-volt levels are used for this scheme. In both cases, the major part of the logic is in the ROM firmware. The electronics really just consists of the two output latches and a few resistors.

In the following projects, we use those 2 bits to generate square waves for musical tones, telephone dialing, and RS-232-C output. The majority of the design effort, as in the TRS-80 cassette functions, is in the software. The hardware consists of three simple circuits with a minimum of parts.

chapter **10**

A Musical Tone Generator

This first project that uses the cassette output port produces six octaves of notes representing the first six octaves on the piano keyboard. The notes are square waves, rich in odd harmonics. Two volume levels can be output, one using the 0- and 0.4-volt levels and a second using the 0- and 0.8-volt levels. The circuit is shown in Fig. 10–1.

TONOUT CIRCUIT

The circuit uses the CASSOUT output as an input to an LM386 audio amplifier. The LM386 requires only a capacitor and an 8-ohm speaker to implement a complete audio amplifier. A miniature 10K potentiometer is used at the input for volume control. The power supply for the LM386 may be any convenient voltage from +4 to +12 volts. A 6-volt battery works fine for the power supply, or Radio Shack sells a low-priced power supply kit.

TONOUT Software

TONOUT (Fig. 10–2) is an assembly language program that drives the

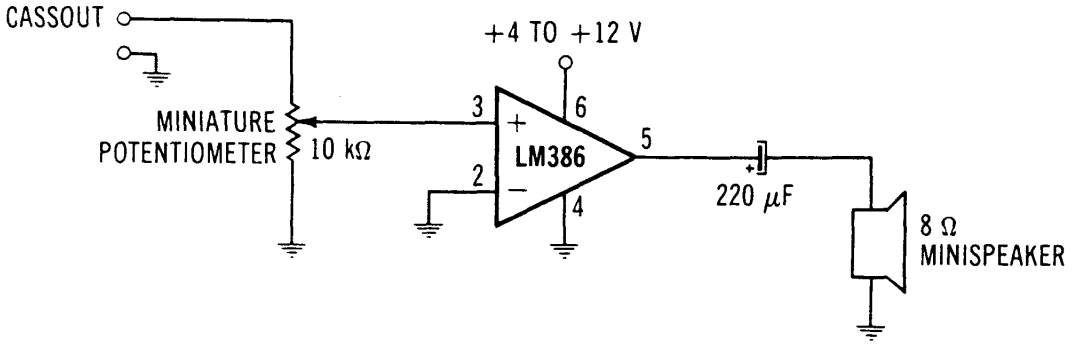


Fig. 10–1. TONOUT circuit.

circuit to produce square waves from about 20 Hz to over 10,000 Hz. The low and high frequencies won't come out very well (or at all) in the LM386, but for the most part tones sound fine. You might consider using

```
9000            00100           ORG     9000H
                00110 ;*********************************************************
                00120 ;* TONE OUTPUT. OUTPUTS TONE THROUGH CASSETTE PORT.      *
                00130 ;*      ENTRY: HL=> PARAMETER BLOCK                      *
                00140 ;*             PARAM+0=DURATION CNT IN 2 BYTES           *
                00150 ;*                 +2=FREQ CNT*(18.04 MICROSECS MOD I    *
                00151 ;*                    15.79 MICROSECS MOD III), 2 BYTES  *
                00160 ;*                 +4=LEVEL: 2=LOW,3=HIGH, ONE BYTE      *
                00170 ;*      EXIT:  AFTER TONE HAS SOUNDED                    *
                00180 ;*********************************************************
                00190 ;
9000 CD7F0A     00200 TONOUT  CALL    0A7FH           ;GET HL
9003 E5         00210         PUSH    HL              ;TRANSFER TO IX
9004 DDE1       00220         POP     IX
9006 DD4E04     00230         LD      C,(IX+4)        ;PUT LEVEL IN C
9009 DD6603     00240         LD      H,(IX+3)        ;MSB FREQUENCY
900C DD6E02     00250         LD      L,(IX+2)        ;LSB FREQUENCY
900F 7C         00260         LD      A,H             ;GET MSB FREQ CNT
9010 B7         00270         OR      A               ;TEST FOR ZERO
9011 2022       00280         JR      NZ,LOWFRE       ;GO IF LOW FREQUENCY
                00290 ; HIGH FREQUENCY HERE
9013 45         00300         LD      B,L             ;GET FREQUENCY COUNT
9014 DD6601     00310         LD      H,(IX+1)        ;MSB OF DURATION
9017 DD6E00     00320         LD      L,(IX+0)        ;LSB OF DURATION
901A 2B         00330         DEC     HL              ;FOR JR NC
901B 11FFFF     00340         LD      DE,-1           ;FOR FREQ LOOP
901E 79         00350 HI010   LD      A,C               ;LEVEL TO A (4)
901F EE02       00360         XOR     2                 ;NOW 00 OR 01 (7)
9021 D3FF       00370         OUT     (0FFH),A          ;TURN ON (11)
9023 78         00380         LD      A,B               ;GET FREQ COUNT (4)
9024 3D         00390 HI020   DEC     A                   ;ON LOOP (4)
9025 20FD       00400         JR      NZ,HI020            ;LOOP TIL 0 (12/7)
9027 79         00410         LD      A,C             ;DUMMY (4)
9028 3E02       00420         LD      A,2             ;NOW 10 (7)
902A D3FF       00430         OUT     (0FFH),A        ;TURN OFF (11)
902C 78         00440         LD      A,B             ;GET FREQ COUNT (4)
902D 3D         00450 HI030   DEC     A                 ;OFF LOOP (4)
902E 20FD       00460         JR      NZ,HI030          ;LOOP TIL 0 (12/7)
9030 19         00470         ADD     HL,DE           ;DECREMENT DUR CNT (11)
9031 38EB       00480         JR      C,HI010         ;GO IF NOT 0 (12/7)
9033 1823       00490         JR      LOW090          ;RETURN TO BASIC
                00500 ; LOW FREQUENCY HERE
9035 E5         00510 LOWFRE  PUSH    HL              ;FREQ COUNT TO DE
9036 D1         00520         POP     DE
9037 CB83       00530         RES     0,E             ;MAKE EVEN
9039 DD4600     00540         LD      B,(IX+0)        ;DURATION CNT TO B
903C 79         00550 LOW010  LD      A,C               ;GET LEVEL (4)
903D EE02       00560         XOR     2                 ;NOW 00 OR 01 (7)
903F D3FF       00570         OUT     (0FFH),A          ;TURN ON (11)
9041 62         00580         LD      H,D               ;GET FREQ COUNT (4)
9042 6B         00590         LD      L,E               ;(4)
9043 2B         00600 LOW020  DEC     HL                  ;DECR FREQ CNT (6)
9044 2B         00610         DEC     HL                  ;(6)
9045 7C         00620         LD      A,H                 ;TEST HL (4)
9046 B5         00630         OR      L                   ;(4)
9047 20FA       00640         JR      NZ,LOW020           ;GO IF NOT 0 (12/7)
9049 79         00650         LD      A,C             ;DUMMY (4)
904A 3E02       00660         LD      A,2             ;NOW 10 (7)
904C D3FF       00670         OUT     (0FFH),A        ;TURN OFF (11)
904E 62         00680         LD      H,D             ;GET FREQ COUNT (4)
904F 6B         00690         LD      L,E             ;(4)
9050 2B         00700 LOW030  DEC     HL                  ;DECR FREQ CNT (6)
9051 2B         00710         DEC     HL                  ;(6)
9052 7C         00720         LD      A,H                 ;TEST HL (4)
9053 B5         00730         OR      L                   ;(4)
9054 20FA       00740         JR      NZ,LOW030           ;GO IF NOT 0 (12/7)
9056 10E4       00750         DJNZ    LOW010          ;GO IF D CNT NOT 0 (13/8)
9058 C9         00760 LOW090  RET                     ;RETURN TO BASIC
0000            00770         END
00000 Total errors
```

Fig. 10–2. TONOUT program.

CASSOUT as an input to an amplifier with better fidelity if you're a purist. TONOUT is designed to interface to a BASIC program. It is completely relocatable (more about that later) and requires three parameters from the BASIC code: a frequency count, a duration count, and a level.

The frequency count is a value from 1 to 65,535 that is used as a timing-loop count. Each count delays 18.04 μs for the Model I and 15.79 μs for the Model III. The delay is on the on-and-off portions of the square wave, as shown in Fig. 10–3; therefore, the frequency of the square wave produced is 1/(36.08E−6) or 1/(31.58E−6).

The duration count of 1–65,535 determines the length of time that the tone is played. In fact, the duration count is the number of cycles of the tone. The length of time that the tone plays is also dependent upon the frequency. To play quarter notes the duration count would be 25 for a 100-Hz tone, 50 for 200 Hz, and so forth. The duration count is (1/frequency) times the fraction of a second the tone is to be played. The third parameter is level. A value of 2 is a low level and 3 is a high level. The level parameter is in one byte.

The basic problem in TONOUT is how to get the tightest possible loop to toggle the 0FFH bits on and off and still allow for longer duration low-frequency notes. The approach used here is to split up TONOUT into

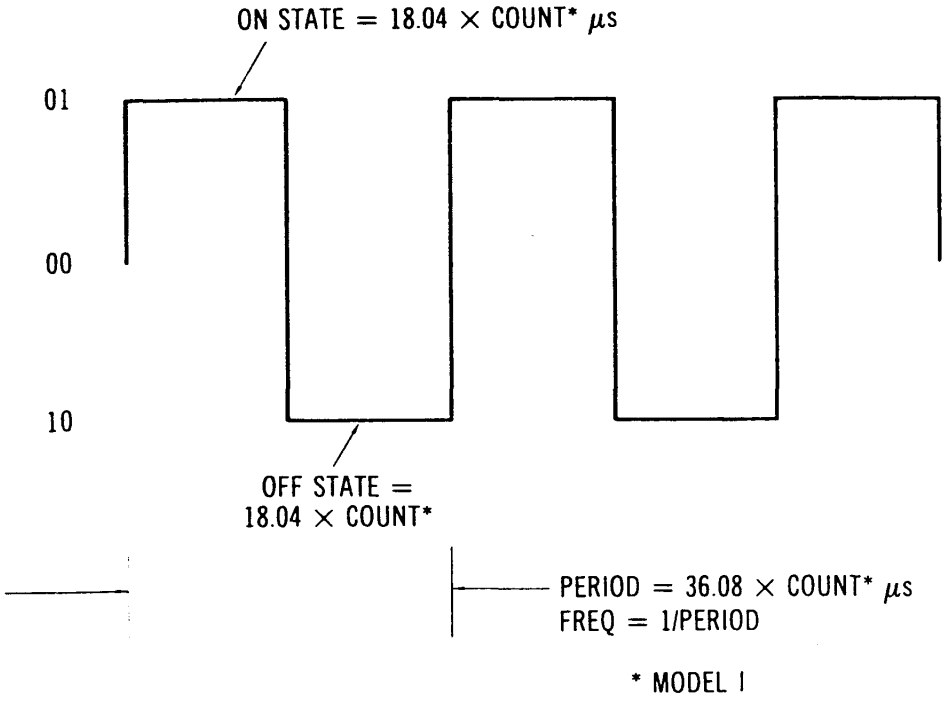


Fig. 10–3. TONOUT output waveform.

two segments of code, one for high-frequency notes and one for low-frequency notes.

TONOUT is entered from BASIC by a DEFUSR call. The CALL 0A7FH gets the argument from BASIC and puts it into the HL register pair. The argument in this case is a pointer to a parameter block of the three arguments in 7 bytes (see Fig. 10–4). This pointer is transferred to the IX register.

The level parameter is put into the C register and the frequency count is put into HL. Next, the frequency count is tested for magnitude. If the H register is nonzero, the frequency count is greater than 255 and a low-frequency note will be played. If the frequency count is less than 256, the high-frequency segment is executed. The single byte of the frequency count is transferred to the B register and the 2 bytes of the duration count are transferred to HL and decremented by one for the JR C loop. (C will decrement below 0 before the loop is terminated.) The DE register pair is loaded with −1 for a tight timing loop.

The output portion of the loop consists of two almost identical segments. Lines 350 through 400 are the on portion that turns on the top of the square wave. Lines 410 through 460 turn off the output. Both decrement the frequency count in a timing loop that determines the frequency. The level for the on is determined by an XOR of 10 and the level parameter to produce either a 00 (low) or 01 (high). After one complete cycle, the duration count in HL is decremented by an ADD HL,DE. If the result is not negative, another cycle is generated.

The code from lines 510 through the end is a similar routine for low-frequency notes. In this case, the frequency count is held in HL and decremented twice. The frequency count is first made even by a RES 0 instruction for a test of decrementing down to zero. The duration count is assumed to be 254 or less and held in B for a DJNZ instruction.

Using TONOUT with BASIC

TONOUT can be used to generate tones other than musical notes. The precise frequencies generated are:

1. Freq = 1/((42.29 + 18.04 × count) × 1E−6) for high-frequency tones and
2. Freq = 1/((41.15 + 18.04 × count) × 1E−6) for low-frequency tones.

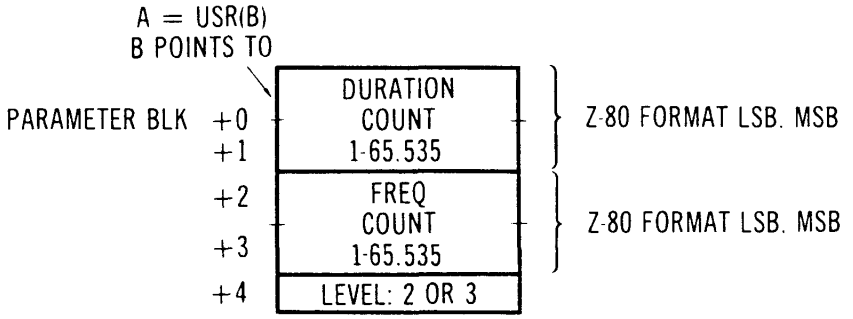


Fig. 10–4. TONOUT argument passing formats.

```
20 'PROGRAM TO FIND BEST FIT FOR 8 OCTAVES
40 DIM NT$(11)
60 A$="A A#B C C#D D#E F F#G G#"
80 FOR J=0 TO 11
100 NT$(J)=MID$(A$,J*2+1,2)
120 NEXT J
140 FOR I=0 TO 7
160 RESTORE
180 LPRINT "OCTAVE ";I+1
200 FOR J=0 TO 11
220 LPRINT NT$(J);"=";
240 N=(27.5*2↑I)*2↑((J)/12):LPRINTN,
260 CT=((1/N)-36.5E-6)/15.79E-6
280 LPRINT "F CNT=";CT
300 NEXT J
320 NEXT I
```

Fig. 10–5. BASIC program for pitch conversion.

```
OCTAVE  1
A = 27.5          F CNT= 2300.64
A#= 29.1352       F CNT= 2171.39
B = 30.8677       F CNT= 2049.39
C = 32.7032       F CNT= 1934.23
C#= 34.6478       F CNT= 1825.54
D = 36.7081       F CNT= 1722.95
D#= 38.8909       F CNT= 1626.12
E = 41.2035       F CNT= 1534.73
F = 43.6535       F CNT= 1448.46
F#= 46.2493       F CNT= 1367.03
G = 48.9994       F CNT= 1290.18
G#= 51.9131       F CNT= 1217.64
OCTAVE  2
A = 55            F CNT= 1149.17
A#= 58.2705       F CNT= 1084.54
B = 61.7354       F CNT= 1023.54
C = 65.4064       F CNT= 965.961
C#= 69.2957       F CNT= 911.616
D = 73.4162       F CNT= 860.321
D#= 77.7817       F CNT= 811.906
E = 82.4069       F CNT= 766.207
F = 87.3071       F CNT= 723.073
F#= 92.4986       F CNT= 682.361
G = 97.9989       F CNT= 643.933
G#= 103.826       F CNT= 607.662
OCTAVE  3
A = 110           F CNT= 573.427
A#= 116.541       F CNT= 541.113
B = 123.471       F CNT= 510.613
C = 130.813       F CNT= 481.825
C#= 138.591       F CNT= 454.652
D = 146.832       F CNT= 429.005
D#= 155.563       F CNT= 404.797
E = 164.814       F CNT= 381.948
F = 174.614       F CNT= 360.381
F#= 184.997       F CNT= 340.025
G = 195.998       F CNT= 320.811
G#= 207.652       F CNT= 302.675
OCTAVE  4
```

Fig. 10–6. Pitch conversion output.

Table 10–1. Integer Counts for TONOUT

	Model I A, A#, B, C, C#, D, D#, E, F, F#, G, G#
OCTAVE 1	2013, 1900, 1793, 1692, 1597, 1508, 1423, 1343, 1268, 1196, 1129, 1065
2	1005, 949, 896, 845, 798, 753, 710, 670, 632, 597, 563, 532
3	502, 473, 447, 421, 398, 375, 345, 334, 315, 297, 280, 265
4	250, 235, 222, 210, 198, 186, 176, 166, 154, 147, 139, 131
5	124, 117, 110, 104, 98, 92, 87, 82, 77, 73, 68, 64
6	61, 57, 54, 51, 48, 45, 42, 40, 37, 35, 33, 31
	Model III **A, A#, B, C, C#, D, D#, E, F, F#, G, G#**
OCTAVE 1	2300, 2171, 2049, 1934, 1826, 1723, 1626, 1535, 1448, 1367, 1290, 1218
2	1149, 1085, 1024, 966, 912, 860, 812, 766, 723, 682, 644, 608
3	573, 541, 511, 482, 455, 429, 405, 382, 360, 340, 321, 303
4	286, 269, 254, 240, 226, 213, 201, 190, 179, 169, 159, 150
5	142, 134, 126, 119, 112, 106, 99, 94, 88, 83, 78, 74
6	70, 66, 62, 58, 55, 52, 49, 46, 43, 40, 38, 36

The above formulas are for the Model I. Use (37 + 15.79 × count) and (36 + 15.79 × count) for the Model III. The 18.04 represents the on/off loop times, and the other constant represents the overhead for the frequency and duration timing.

```
40 REM SAMPLE TONOUT DRIVER
50 DATA 205, 127, 10, 229, 221, 225, 221, 78, 4, 221
51 DATA 102, 3, 221, 110, 2, 124, 183, 32, 34, 69
52 DATA 221, 102, 1, 221, 110, 0, 43, 17, 255, 255
53 DATA 121, 238, 2, 211, 255, 120, 61, 32, 253, 121
54 DATA 62, 2, 211, 255, 120, 61, 32, 253, 25, 56
55 DATA 235, 24, 35, 229, 209, 203, 131, 221, 70, 0
56 DATA 121, 238, 2, 211, 255, 98, 107, 43, 43, 124
57 DATA 181, 32, 250, 121, 62, 2, 211, 255, 98, 107
58 DATA 43, 43, 124, 181, 32, 250, 16, 228, 201,
60 FOR I=36864-65536 TO 36952-65536
62 READ A:POKE I,A
64 NEXT I
100 DEFUSR0=&H9000
110 INPUT D,F,L
120 POKE &HA001,INT(D/256):POKE &HA000,D-(INT(D/256))*256
130 POKE &HA003,INT(F/256):POKE &HA002,F-(INT(F/256))*256
140 POKE &HA004,L
150 A=USR0(&HA000)
160 GOTO 110
```

Fig. 10–7. Sample TONOUT driver program.

BASIC can easily be used to build up a table of values for matching frequency counts to musical notes. Fig. 10–5 shows a Model III BASIC program for converting to American Standard pitch. The Model I version is identical except for constants. In this scheme there are 12 notes per octave, A, A=, B, C, C#, D, D#, E, F, F#, G, and G#, each calculated by raising 2 to successive 1/12th powers. The notes of each octave double over the preceding octave. The first portion of output from the program of Fig. 10–5 is shown in Fig. 10–6. Table 10–1 shows the suggested integer counts for the notes for the Models I and III.

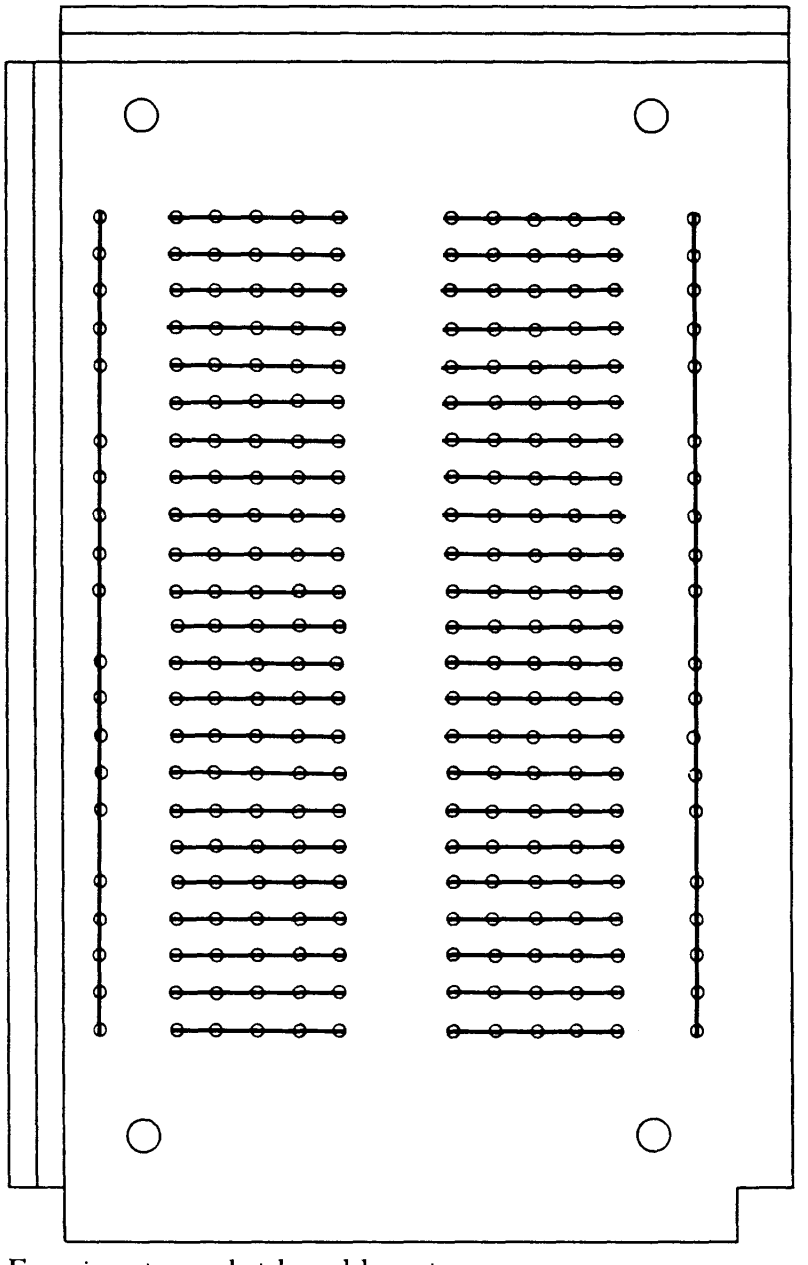


Fig. 10–8. Experimenter socket board layout.

INTERFACING TONOUT TO BASIC

Fig. 10–3 shows the TONOUT program incorporated into a BASIC program as DATA values. The DATA values are the machine-language bytes of TONOUT. TONOUT is relocatable and can be moved anywhere in RAM. The program in Fig. 10–7 moves the bytes to the &H9000 area by a series of POKEs and then INPUTs a duration count, frequency count, and level value for experimentation.

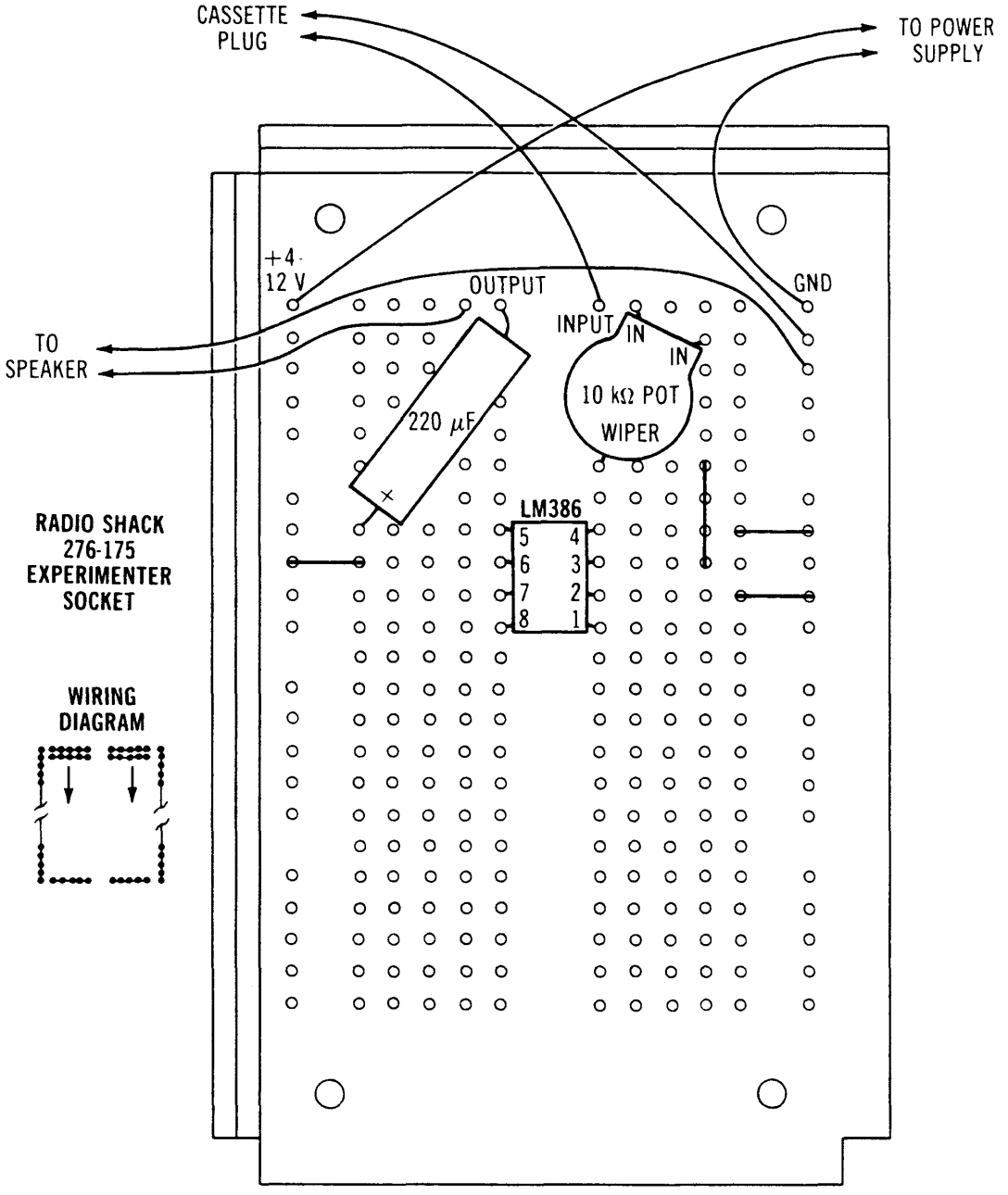


Fig. 10–9. TONOUT physical board layout.

CONSTRUCTING THE TONOUT ELECTRONICS

All three projects in this section use a similar construction method. Radio Shack carries an experimenter socket project board, which is a matrix of 23 rows, each with two halves, as shown in Fig. 10–8. Each of the 46 row segments are connected electrically. Two buses run down the board on the extreme right and left.

Components can be plugged into the board with a minimum of fuss. The interconnections for the TONOUT electronics are shown in Fig. 10–9, along with power supply, speaker (any 8-ohm), and cassette plug connections. Make the connections to the 5-pin DIN plug as shown in Fig. 10–10.

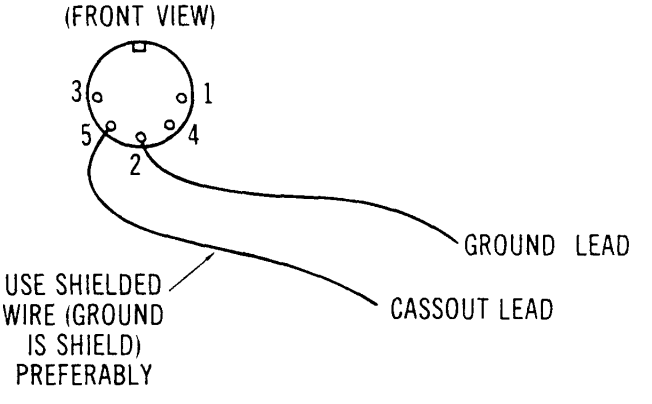


Fig. 10–10. DIN connections.

KILL THOSE INTERRUPTS!

To get precise frequencies for TONOUT, it's a good idea to disable the real-time clock interrupts in the Models I and III. If the real-time clock is running (and it may be, even without a display), the timing on tones may be off by 4% or so, and there may be some modulation on the tone. Add a disable interrupt instruction (243 decimal) at the beginning of TONOUT and an enable-interrupt instruction (251 decimal) right before the 201 decimal for the RET; modify the POKE loop accordingly. The DI and EI were not included here to give the user some flexibility in using TONOUT in different configurations.

chapter **11**

A Telephone Dialer

The second project that uses cassette output is a pulse-type telephone dialer. Most telephone lines, even those using tone dialing, will accept dialing by a series of pulses, spaced at defined intervals. A rotary dial phone simply makes and breaks the phone line, as shown in Fig. 11–1.

TELDIL CIRCUIT

The circuit for TELDIL is shown in Fig. 11–2. Three or four wires come from the telephone jack. Break the red wire (standard coding) and route it to the normally closed contacts of a relay. (As usual, I will disavow any knowledge of this project if you are confronted by the phone

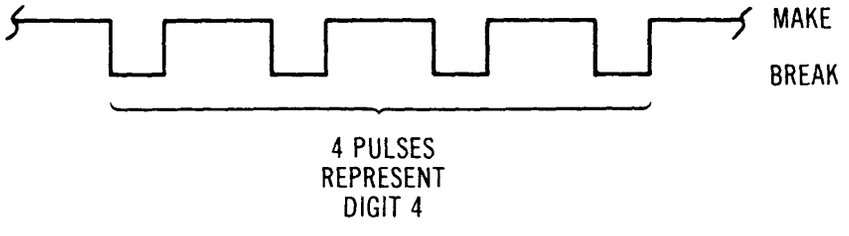


Fig. 11–1. Telephone pulse dialing.

company.) The relay is driven by an LM3900 operational amplifier (op amp), which, in turn, is driven by the CASSOUT line. Whenever the CASSOUT output level is other than 0 volts, enough current flows through the 220-ohm resistor to turn on the LM3900, bringing the output on pin 5 to 0. This closes the relay and breaks the phone line. The diode across the relay contacts is necessary and prevents high-voltage spikes from the inductive load of the relay coil.

TELDIL SOFTWARE

The software for TELDIL is again a relocatable assembly language program that interfaces to BASIC (see Fig. 11–3). Although the version shown uses a delay loop for the Model I, there is enough "slop" in the constants to use the same code for the Model III as well. The make/break rate for digits is 10 pulses per second. The line is broken for about 38.5 ms for each pulse and then made for 61.5 ms, as shown in Fig. 11–4. Interdigit delay is about 830 ms. These delays can be adjusted for faster dialing on an experimental basis.

BASIC passes a pointer to TELDIL in the USR call. The pointer points to a string of ASCII decimal digits, such as 17145551212. Any number of digits can be used. The last byte of the string is a non-ASCII digit, such as CHR$(0). See Fig. 11–5.

TELDIL uses two loops. The outer loop from line 250 through 480 picks up the ASCII digit from the string, tests it for valid ASCII decimal codes of 0 through 9, converts the digit to 1 through 10 pulses, pulses the line, and then increments the string pointer. The inner loop, at lines 330 through 430, pulses the line for each digit. The line is broken by outputting binary 01 to port 0FFH and delaying 38.5 ms. The line is then reconnected for 61.5 ms. The number of pulses is held in the B register, and the DJNZ repeats the loop for the number of pulses required. The outer loop also delays 830 ms for the inter-digit delay.

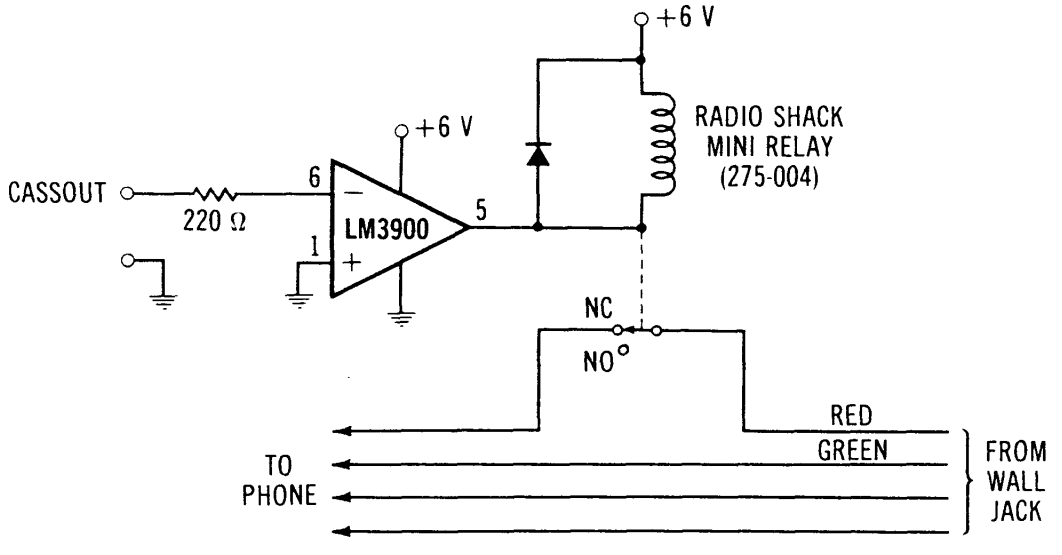


Fig. 11–2. TELDIL circuit.

```
9000            00100          ORG     9000H
                00110 ;*****************************************************
                00120 ;* TELEPHONE DIALER. DIALS ANY NUMBER OF DIGITS FOR     *
                00130 ;* ROTARY-TYPE PHONE THROUGH CASSETTE PORT.             *
                00140 ;*     ENTRY: HL=> STRING OF ASCII DECIMAL DIGITS, TERM- *
                00150 ;*            INATED BY NON-ASCII                       *
                00160 ;*     EXIT:  AFTER NUMBER HAS BEEN DIALED              *
                00170 ;*****************************************************
                00180 ;
0604            00190 PULSE    EQU     1540            ;38.5 MS PULSE
099C            00200 WAIT1    EQU     2460            ;10 PULSES PER SECOND
7814            00210 WAIT2    EQU     30740           ;830 MS INTERDIGIT
9000 CD7F0A     00220 TELDIL   CALL    0A7FH           ;GET ARGUMENT IN HL
9003 E5         00230          PUSH    HL              ;MOVE TO IX
9004 DDE1       00240          POP     IX              ;IX POINTS TO STRING
9006 DD7E00     00250 TEL005   LD      A,(IX)            ;GET NEXT DIGIT
9009 D630       00260          SUB     30H               ;CONVERT TO BINARY
900B 2002       00270          JR      NZ,TEL010         ;GO IF NOT 0
900D 3E0A       00280          LD      A,10              ;0-USE 10 PULSES
900F 3828       00290 TEL010   JR      C,TEL090          ;RTN IF LT 30H
9011 FE0B       00300          CP      11                ;TEST FOR GT 10
9013 3024       00310          JR      NC,TEL090         ;RETURN IF GT 10
9015 47         00320          LD      B,A               ;# OF PULSES IN B
9016 3E01       00330 TEL020   LD      A,1                 ;ON CODE
9018 D3FF       00340          OUT     (0FFH),A            ;TURN ON RELAY
901A 210406     00350          LD      HL,PULSE            ;DELAY CONSTANT
901D 0E00       00360          LD      C,0                 ;RETURN FLAG
901F 1819       00370          JR      DELAY               ;DELAY
9021 3E02       00380 TEL030   LD      A,2                 ;OFF CODE
9023 D3FF       00390          OUT     (0FFH),A            ;TURN OFF RELAY
9025 219C09     00400          LD      HL,WAIT1            ;BETWEEN PULSE DELAY
9028 0E01       00410          LD      C,1                 ;RETURN FLAG
902A 180E       00420          JR      DELAY               ;DELAY
902C 10E8       00430 TEL040   DJNZ    TEL020              ;LOOP IF MORE PULSES
902E 211478     00440          LD      HL,WAIT2          ;BETWEEN DIGITS DELAY
9031 0E02       00450          LD      C,2               ;RETURN FLAG
9033 1805       00460          JR      DELAY             ;DELAY
9035 DD23       00470 TEL050   INC     IX                ;BUMP STRING POINTER
9037 18CD       00480          JR      TEL005            ;LOOP FOR NEXT DIGIT
9039 C9         00490 TEL090   RET                     ;RETURN TO BASIC
                00500 ; DELAYS 24.81*CNT IN MICROSECS + OVERHEAD
903A 2B         00510 DELAY    DEC     HL                ;DECREMENT DELAY COUNT
903B ED5F       00520          LD      A,R               ;TIME WASTER
903D ED5F       00530          LD      A,R               ;TIME WASTER
903F 7C         00540          LD      A,H               ;TEST HL
9040 B5         00550          OR      L
9041 20F7       00560          JR      NZ,DELAY            ;LOOP IF NOT DONE
9043 CB49       00570          BIT     1,C             ;TEST FOR RTN PNT 2
9045 20EE       00580          JR      NZ,TEL050       ;GO IF RTN PNT 2
9047 CB41       00590          BIT     0,C             ;TEST FOR RTN PNT 1
9049 20E1       00600          JR      NZ,TEL040       ;GO IF RTN PNT 1
904B 18D4       00610          JR      TEL030          ;RTN PNT 0
0000            00620          END
00000 Total errors
```

Fig. 11–3. TELDIL program.

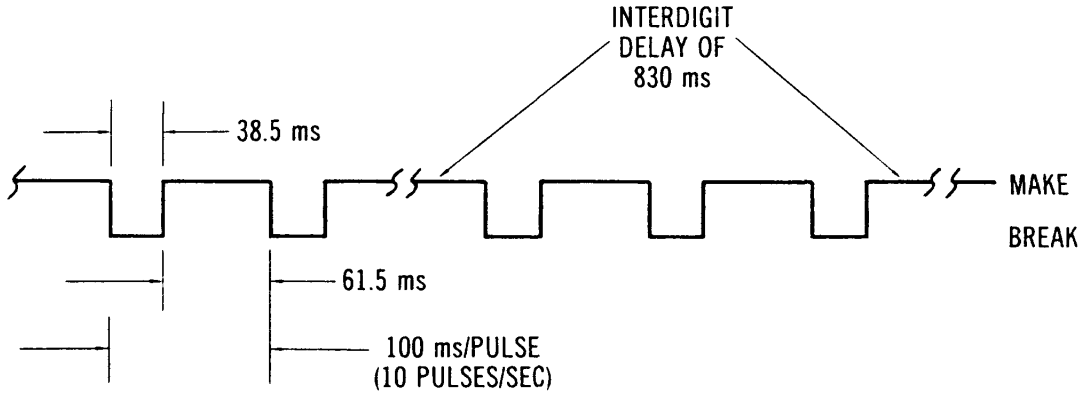


Fig. 11–4. Dialing pulse parameters.

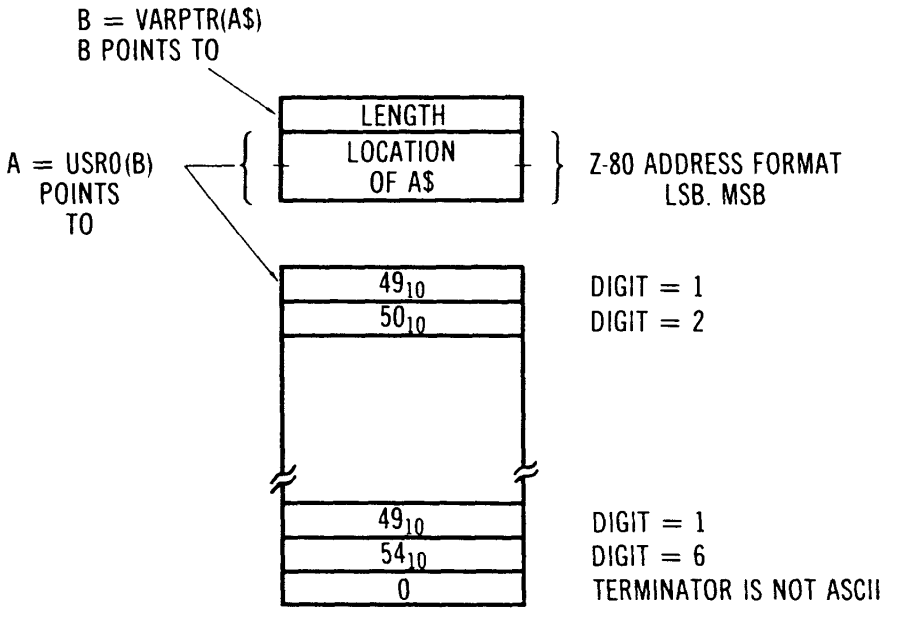


Fig. 11–5. TELDIL parameter passing.

DELAY is a simple time delay routine that delays 24.81 μs times the HL count. To keep the code relocatable, DELAY is not called by a CALL, which would have a nonrelocatable address, but is called with a code for the three return points.

```
40 REM SAMPLE TELDIL DRIVER
60 A=0:B=0:C=0:A$=""
80 DATA 205, 127, 10, 229, 221, 225, 221, 126, 0, 214
100 DATA 48, 32, 2, 62, 10, 56, 40, 254, 11, 48
120 DATA 36, 71, 62, 1, 211, 255, 33, 4, 6, 14
140 DATA 0, 24, 25, 62, 2, 211, 255, 33, 156, 9
160 DATA 14, 1, 24, 14, 16, 232, 33, 20, 120, 14
180 DATA 2, 24, 5, 221, 35, 24, 205, 201, 43, 237
200 DATA 95, 237, 95, 124, 181, 32, 247, 203, 73, 32
220 DATA 238, 203, 65, 32, 225, 24,212
240 CLEAR 500
260 DEFUSR0=&H9000
280 FOR I=36864-65536 TO 36940-65536
300 READ A:POKE I,A
320 NEXT I
340 INPUT A$: A$=A$+CHR$(0)
360 B=VARPTR(A$)
380 C=PEEK(B+1)+(PEEK(B+2))*256
400 A=USR0(C-65536)
420 GOTO 340
```

Fig. 11–6. TELDIL embedded in BASIC.

INTERFACING TELDIL TO BASIC

Fig. 11–6 shows the machine-language code of TELDIL incorporated within a BASIC program. In this case it is moved to the &H9000 area, but it could be relocated to any convenient area in RAM. The ASCII

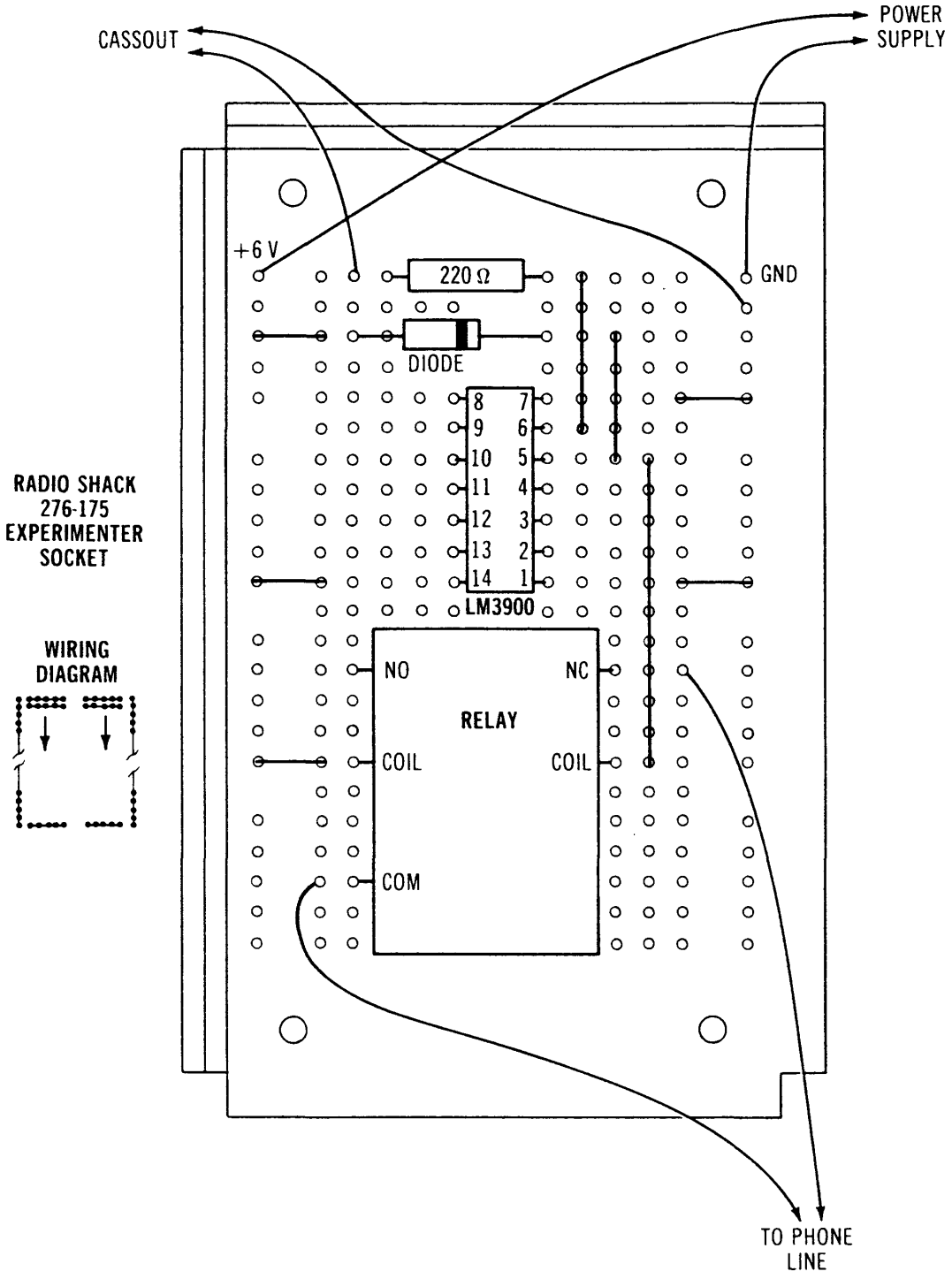


Fig. 11–7. TELDIL physical layout.

string is INPUT and a CHR$(0) is concatenated to the string for the terminating character.

VARPTR is used to find the string location. Make certain that VARPTR is used directly before the USR call as string variables move.

The C-65536 adjusts for addresses in the upper 32K of RAM. For a 32K or 48K system this would normally be the area in which string variables would be located. String variables within a BASIC program line, however, have addresses that represent the location of the program line, and the argument in the USR call must be adjusted accordingly.

CONSTRUCTING THE TELDIL ELECTRONICS

TELDIL uses the project board discussed in Chapter 10. The components are connected as shown in Fig. 11–7. Power supply voltage should be over 6 volts; the relay shown will not work well with a +5-volt supply. The cassette plug is connected as shown in Fig. 10–10.

Phone line connections may be made with the help of standard phone plug hardware that can be obtained from your neighborhood Radio Shack store.

chapter **12**

A Serial Driver

The third project using the cassette output is an RS-232-C output port that can be used to drive a serial printer, modem, or other serial device. Standard baud rates of 300, 600, 1200, and 2400 can be selected with 10 bits per character.

RS-232-C signals appear as shown in Fig. 12–1. A voltage level below −3 volts represents a 1 bit, while a voltage level above +3 volts represents a 0 bit. Although the number of bits in a transmission varies, a common convention used with the TRS-80 is shown in Fig. 12–1.

Each byte to be transmitted occurs at asynchronous or irregular times. The line is normally at a 1 level. A start bit of 0 leads the output and signals the receiving device that data is coming in. Eight data bits follow, least-significant bit first. A stop bit of 1 puts the line in the 1 level after transmission in preparation for the next character.

The spacing for the 10 bits depends upon the baud rate. Baud rates of 300, 600, 1200, and 2400 represent bit times of 3.333, 1.666, 0.833, and 0.416 ms, respectively.

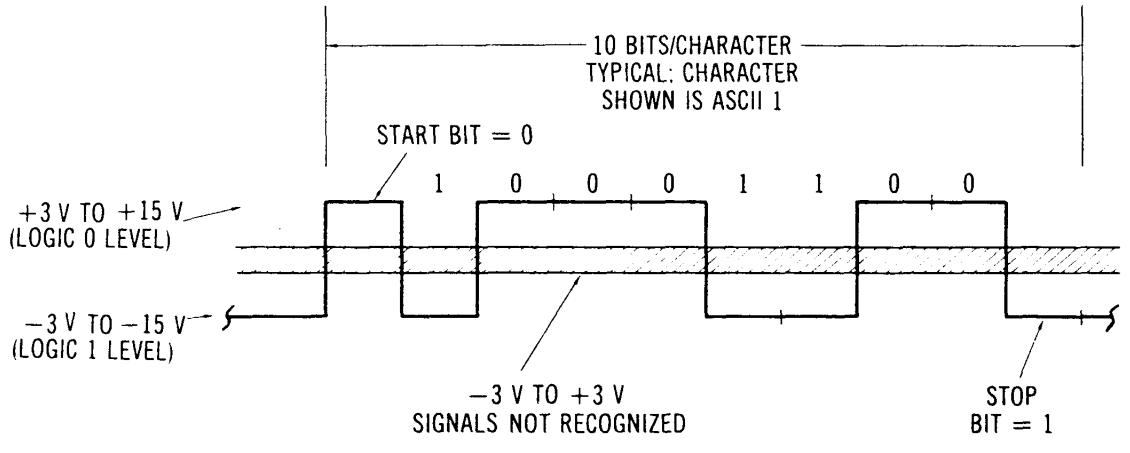


Fig. 12–1. RS-232-C signals for SEROUT.

SEROUT CIRCUIT

The chief problem in SEROUT is to convert the low voltage levels of CASSOUT into two RS-232-C voltages. This is done with an LM741 comparator shown in Fig. 12–2. The voltages used with the comparator are +6 to +12 volts and −6 to −12 volts. Batteries will work fine, and the voltages are not critical.

The voltage divider input to the positive (+) input of the LM741 is biased at about (220/15000) × V+, where V+ is the positive voltage level. This puts the positive (+) input at about 0.1 volt for a +6-volt supply, or about 0.05 volt for a +12-volt supply. The output of the 741 will be −6 to −12 (1 level) whenever the CASSOUT input is greater than the positive (+) input level and +6 to +12 (0 level) whenever the CASSOUT input is less than the positive (+) input level.

BASIC initializes CASSOUT to the binary 00 level (0.44 volt), so the TD (transmit data) line at reset is normally −6 to −12 volts. SEROUT toggles the CASSOUT line at the appropriate baud rate to generate the RS-232-C signals.

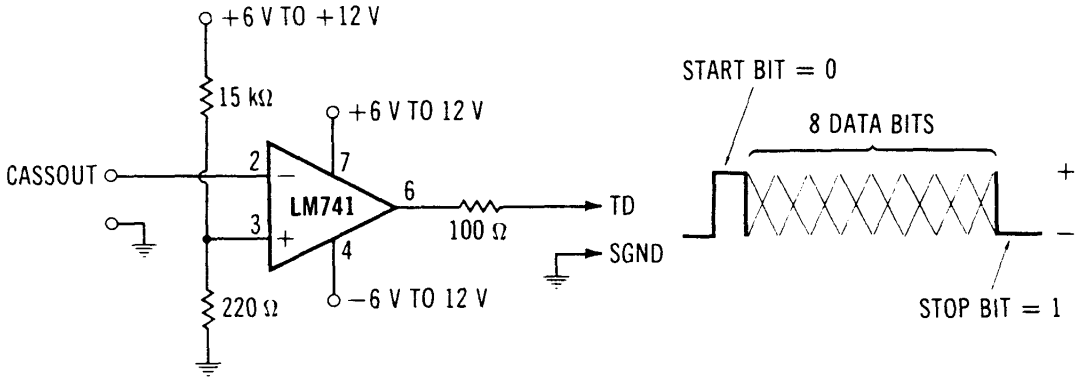


Fig. 12–2. SEROUT level conversion circuit.

SEROUT SOFTWARE

Again, SEROUT is a relocatable assembly language program called from BASIC by a USR call (see Fig. 12–3). Two parameters are passed, the byte to be transmitted and the baud rate to be used. The byte may be

```
9000            00100          ORG     9000H
                00110 ;*****************************************************
                00120 ;* SERIAL OUT THROUGH CASSETTE PORT.                 *
                00130 ;*     ENTRY:  H=BYTE TO BE TRANSMITTED              *
                00140 ;*             L=BAUD RATE: 300=201,600=100,1200=51, *
                00150 ;*                          2400=23 (MOD I)/ 230,115,*
                00160 ;*                          58,26 (MOD III)          *
                00170 ;*     EXIT:   AFTER OUTPUT                          *
                00180 ;*****************************************************
                00190 ;
9000 CD7F0A     00200 SEROUT   CALL    0A7FH           ;GET PARAMETERS
9003 54         00210          LD      D,H             ;MOVE DATA TO D
9004 2600       00220          LD      H,0             ;HL NOW HAS DELAY CNT
9006 E5         00230          PUSH    HL              ;TRANSFER CNT TO IY
9007 FDE1       00240          POP     IY
9009 3E02       00250          LD      A,2             ;START BIT
900B D3FF       00260          OUT     (0FFH),A        ;OUTPUT
900D FDE5       00270          PUSH    IY              ;DELAY CNT TO HL
900F E1         00280          POP     HL
9010 0E00       00290          LD      C,0             ;FLAG FOR RTN
9012 1820       00300          JR      DELAY           ;DELAY ONE BIT TIME
9014 0608       00310 SER040   LD      B,8             ;SETUP DATA BIT LOOP
9016 3E02       00320 SER050   LD      A,2               ;0 BIT TO A
9018 CB3A       00330          SRL     D                 ;SHIFT OUT DATA BIT
901A 3001       00340          JR      NC,SER055         ;GO IF DATA BIT=0
901C 3D         00350          DEC     A                 ;DATA BIT=1
901D D3FF       00360 SER055   OUT     (0FFH),A          ;OUTPUT
901F FDE5       00370          PUSH    IY                ;DELAY CNT TO HL
9021 E1         00380          POP     HL
9022 0E01       00390          LD      C,1               ;FLAG FOR RTN
9024 180E       00400          JR      DELAY             ;DELAY
9026 10EE       00410 SER060   DJNZ    SER050            ;LOOP IF MORE BITS
9028 3E01       00420          LD      A,1             ;STOP BIT
902A D3FF       00430          OUT     (0FFH),A        ;OUTPUT
902C FDE5       00440          PUSH    IY              ;DELAY CNT TO HL
902E E1         00450          POP     HL
902F 0E02       00460          LD      C,2             ;FLAG FOR RTN
9031 1801       00470          JR      DELAY           ;DELAY
9033 C9         00480 SER070   RET                     ;RETURN TO BASIC
                00490 ; DELAYS 14.6*CNT IN MICROSECS (MOD I)  + OVERHEAD
9034 2B         00500 DELAY    DEC     HL                ;DECREMENT DELAY COUNT
9035 7C         00510          LD      A,H               ;TEST HL
9036 B5         00520          OR      L
9037 20FB       00530          JR      NZ,DELAY          ;LOOP IF NOT DONE
9039 CB49       00540          BIT     1,C             ;TEST FOR RTN PNT 2
903B 20F6       00550          JR      NZ,SER070       ;GO IF RTN PNT 2
903D CB41       00560          BIT     0,C             ;TEST FOR RTN PNT 1
903F 20E5       00570          JR      NZ,SER060       ;GO IF RTN PNT 1
9041 18D1       00580          JR      SER040          ;RTN PNT 0
0000            00590          END
00000 Total errors
```

Fig. 12–3. SEROUT program.

```
100 REM SAMPLE SEROUT DRIVER
110 DATA 205, 127, 10, 84, 38, 0, 229, 253, 225, 62
120 DATA 2, 211, 255, 253, 229, 225, 14, 0, 24, 32
130 DATA 6, 8, 62, 2, 203, 58, 48, 1, 61, 211
140 DATA 255, 253, 229, 225, 14, 1, 24, 14, 16, 238
150 DATA 62, 1, 211, 255, 253, 229, 225, 14, 2, 24
160 DATA 1, 201, 43, 124, 181, 32, 251, 203, 73, 32
170 DATA 246, 203, 65, 32, 229, 24, 209
180 FOR I=36864 TO 36930
190 READ A:POKE I-65536,A
200 NEXT I
210 DEFUSR0=&H9000
220 INPUT "RATE?";RT
230 CH=48
240 B=USR0(CH*256+RT)
250 GOTO 240
```

Fig. 12–4. Sample BASIC driver for SEROUT.

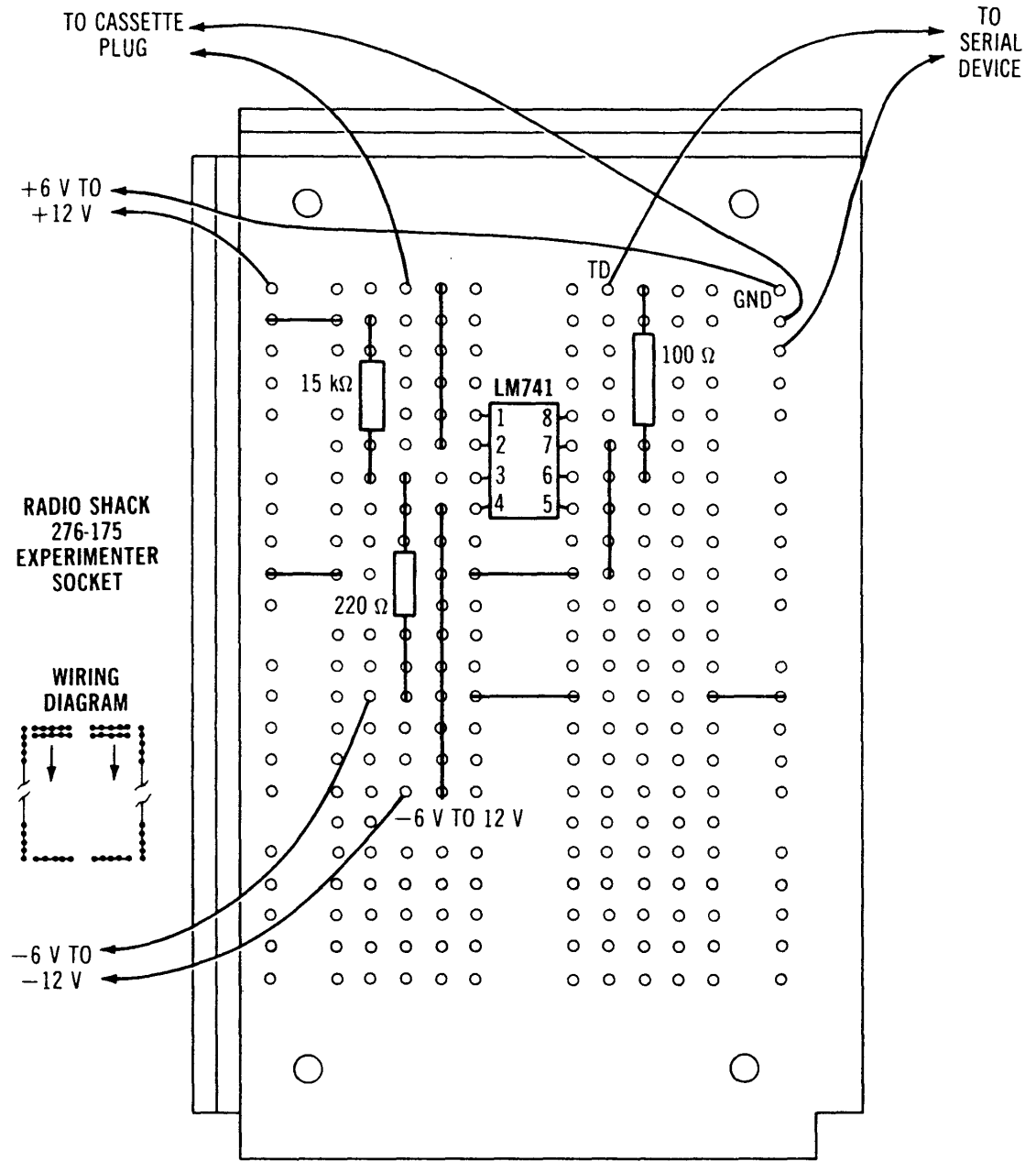


Fig. 12–5. SEROUT physical circuit layout.

any value from 0 through 255. If 7 data bits are to be transmitted (as in data communications applications), make the eighth bit 0 for parity.

SEROUT first picks up the baud rate and puts it into HL. The baud rate is a delay count for the DELAY subroutine. The byte to be transmitted is moved to the D register. Line 260 turns on CASSOUT to generate a start bit. A delay of one bit time is then done. The loop from line 320 through 410 outputs the 8 data bits, from least significant to most signifi-

cant. A one bit is generated by a 01 level and a zero bit by a 10 level. A delay of a bit time is done for each bit. A stop bit is generated in line 430 after the 8 data bits. This leaves the TD line in the 1 level condition in preparation for the next start bit. The DELAY subroutine is called by JRs with a return flag to keep the code relocatable.

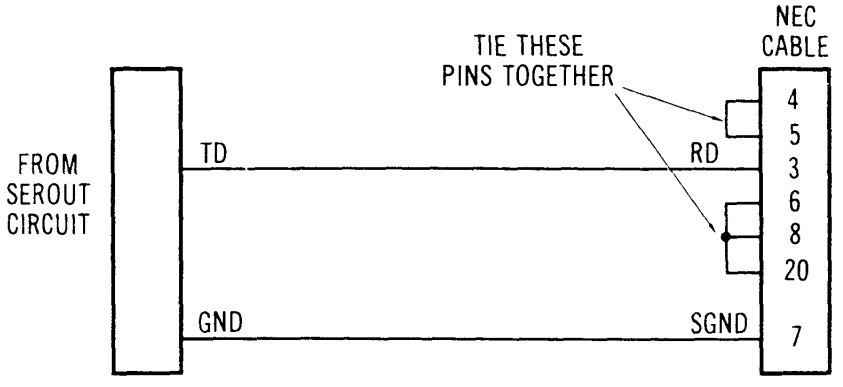


Fig. 12–6. Typical NEC spinwriter connections.

INTERFACING SEROUT TO BASIC

Fig. 12–4 shows a sample BASIC driver that contains the machine code as DATA statements. The code is relocated to the &H9000 area. An ASCII 0 is continually output at a user-specified baud rate.

The actual BASIC code to be used depends a great deal upon the application. If you are using SEROUT as a printer driver, then you'll have to make certain that the printer can accept characters at the rate you'll be transmitting. This is usually not a problem except on carriage return/line feeds where the print mechanism is busy for relatively long times as the carriage returns. If a character is output during this busy condition, it may be disregarded and lost.

The baud rate delay times shown are values obtained by trial and error with the real-time clock active. You may have to adjust these on an experimental basis, depending upon your system. Output a line of characters continuously with different baud rate values. Find the high and low values at which you lose characters and choose a midpoint value for your standard baud rate value. For high baud rates, turn off the interrupts by a DI and EI as described in TONOUT.

CONSTRUCTING THE SEROUT ELECTRONICS

Fig. 12–5 shows the project board layout for SEROUT. Two sets of power supply leads connect to the positive and negative supplies. The TD line and ground (called SGND in RS-232-C nomenclature) connect to the serial device. The serial device may require other signals to be tied high to simulate a ready condition. Again, this depends upon the device, and can't be detailed here. A typical connection to an NEC Spinwriter is shown in Fig. 12–6.

The projects in the last three chapters show what can be done with the cassette output port on even a 16K Model I system without expansion interface. The opposite direction—CASSIN—is covered in the next section.

Section IV

Using the Cassette Input on the Model III and the Color Computer

chapter **13**

Discrete Inputs for the Model III and the Color Computer

In Section 3, we discussed using the cassette output of the Models I and III as a single discrete output line to drive a music synthesizer, telephone dialer, and serial port. In this section, we look at the inverse—how to implement discrete (binary) inputs on the Color Computer and the Model III. Unfortunately, the schemes we use are not applicable on the Model I, so it gets short shrift in this section.

Of course, it is possible to implement dozens of discrete input lines to the Model I, Model III, or Color Computer by using a *peripheral interface adapter* or *peripheral I/O* device, such as the 8255 semiconductor chip. This method requires four or five integrated circuits in addition to the PIO or PIA.

The approaches in this chapter, however, involve using few additional components other than sensors. This cheap and dirty approach can be used to detect remote switch closures, such as burglar alarms and fire detectors, or even remote data transmission devices, such as pulses generated by a telephone-type rotary dial. Another use of the discrete line inputs is as a frequency counter. With the proper sensors and software, we can implement a low-frequency counter that can easily measure thousands of counts per second; the software can handle "switch bounce," too. The next chapter describes this application.

As an example of a practical application of this discrete line input, we show how to construct an anemometer that will measure windspeeds from 2½ to over 60 miles per hour. Believe it or not, this device costs less than $10.00 and can be made by "hackers" without opposing thumbs. See Chapter 15.

WHERE ARE THE DISCRETE INPUTS?

Looking at the Color Computer, we can find plenty of potential discrete inputs. There are two joystick jacks, a cassette jack, and an RS-232-C jack.

Joystick Switch Inputs

The left and right joysticks have four analog channels that could be used as discrete inputs. Even more promising, however, are the joystick switch inputs. The joystick switches are shown in Fig. 13–1. They are normally open switches that close to ground. The output of each switch goes to bits 0 (right joystick) and 1 (left joystick) of PIA address $FF00. As you can see from Fig. 13–2, the switch inputs to the PIA are shared by two keyboard rows; normally you wouldn't be using both the keyboard

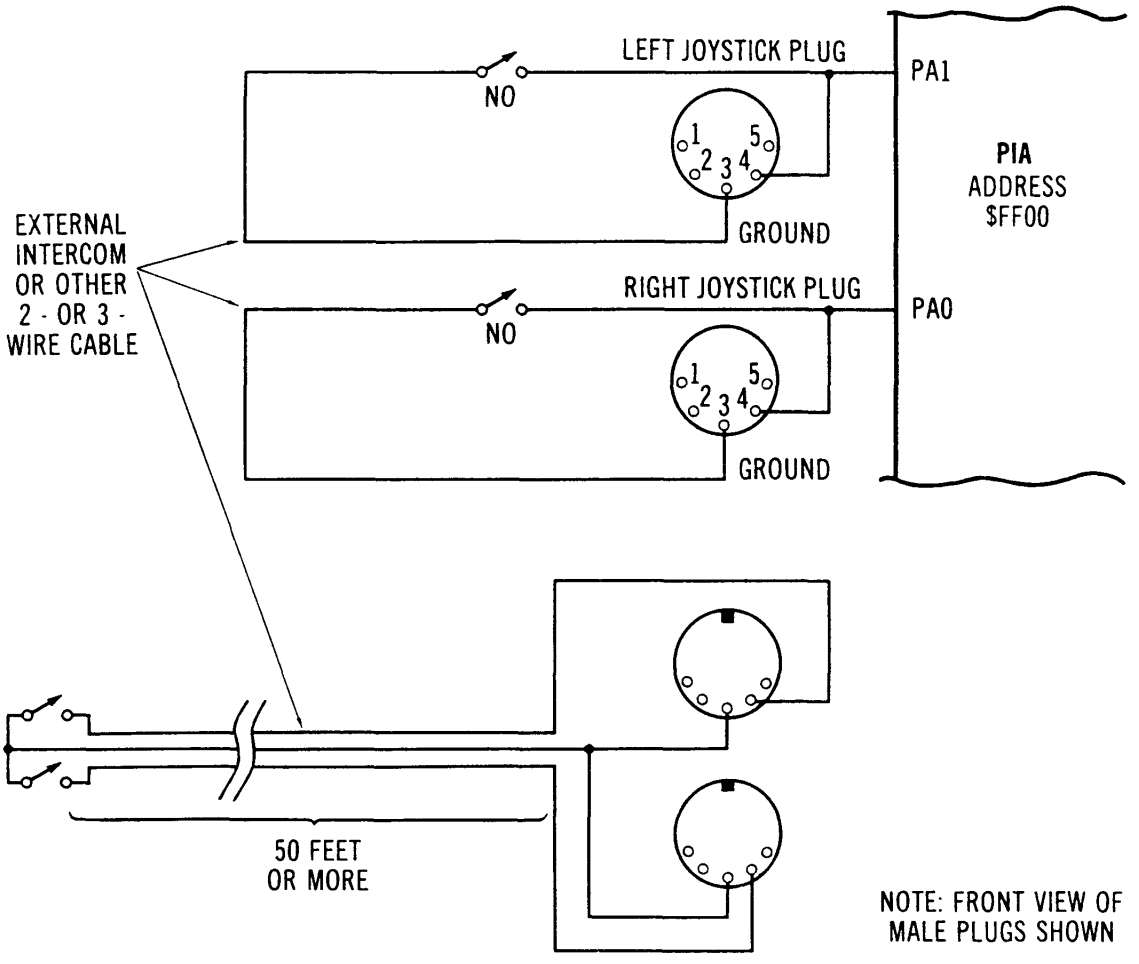


Fig. 13–1. Joystick switch logic wiring.

and joystick switches at the same time. The joystick switches connect to the PIA through a small filter made up of a choke and bypass capacitor as shown in the figure; this eliminates some input noise.

If an external switch or switches are substituted for the joystick switches, a cable can be run 50 feet or more to a remote location. This is generally not a recommended procedure with an unterminated input such as this, but I experienced no difficulties and no false readings in a home environment with a 60-foot intercom-type cable.

The program used is shown in Fig. 13–3, which simply checks for a 1 or 0 on either joystick input. This Extended Color BASIC program loops at about 30 senses per second, making this scheme fine for switch closures in burglar alarms, fire detectors, microswitches in mailboxes triggered by the weight of the mail, and so forth.

At this point, it's probably well to mention a typical switch that can be used for remote sensing. Radio Shack has "submini" lever switches with or without roller (275-017 and 275-016, respectively), which require about 50 grams to operate. These switches were used in the applications described here, although virtually any spdt switch could be used.

RS-232-C Input

Another possibility for a discrete input on the Color Computer is the RS-232-C RD input. This line is normally used to input serial data. As Fig. 13-4 shows, it connects to an LM339 comparator in the Color Computer. One input to the comparator is a voltage divider made up of a 15K and a 10K resistor. The junction point is a constant +2 volts and goes to the positive (+) input.

The negative (−) input connects to the external RD line via a common diode and a 10K resistor to ground. RS-232-C signals are normally above +3 volts (0 bit) or below −3 volts (1 bit). When the RD line is more positive than about 2.6 volts, the input forward-biases the diode and the negative (−) input is greater than the positive (+) input, producing a 0 comparator output. When the RD line is negative, the diode is reverse-biased and the output of the comparator is 1.

The comparator output goes to PB0 of a PIA whose address is $FF22. Reading bit 0 of PIA $FF22 can be done in similar fashion to the joystick switch read, as shown in Fig. 13–5.

Fig. 13–6 shows the remote connections for the RS-232-C remote input. Tie the normally closed contact of the switch to the positive termi-

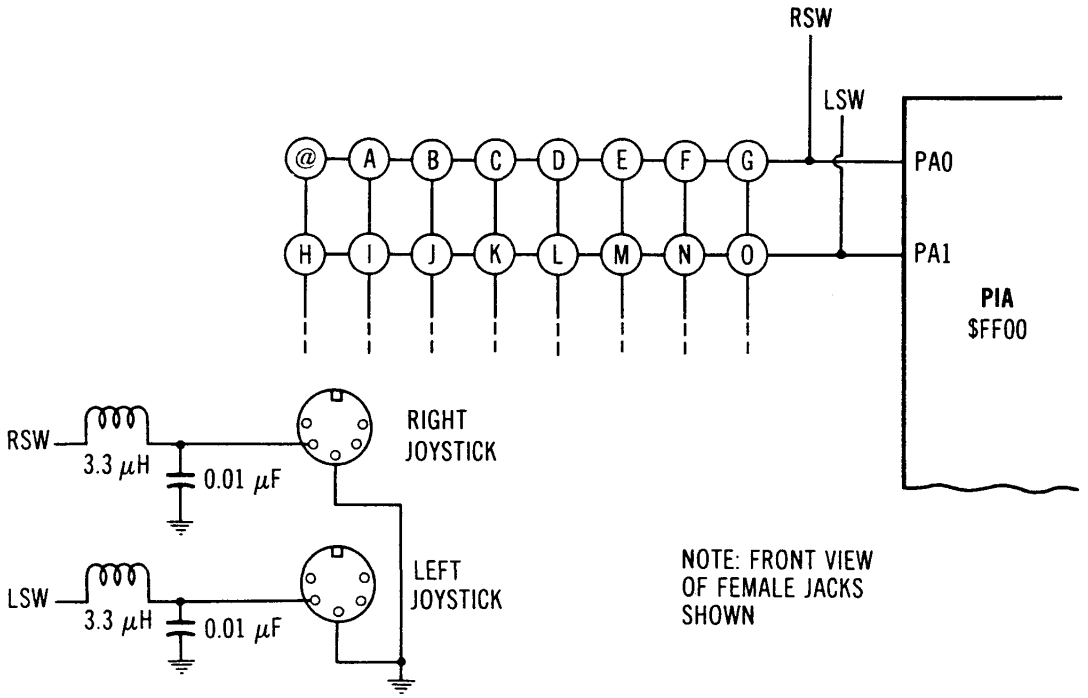


Fig. 13–2. PIA logic for joystick switches.

```
100 ' SWITCH CLOSURE FOR RIGHT AND LEFT JOYSTICK INPUTS
110 INPUT "RIGHT(R) OR LEFT(L) DETECT";A$
120 IF A$="R" THEN M=1 ELSE M=2
130 A=(PEEK(&HFF00) AND M)
140 IF A=M THEN PRINT "OFF" ELSE PRINT "ON"
150 GOTO 130
```

Fig. 13–3. BASIC joystick switch program.

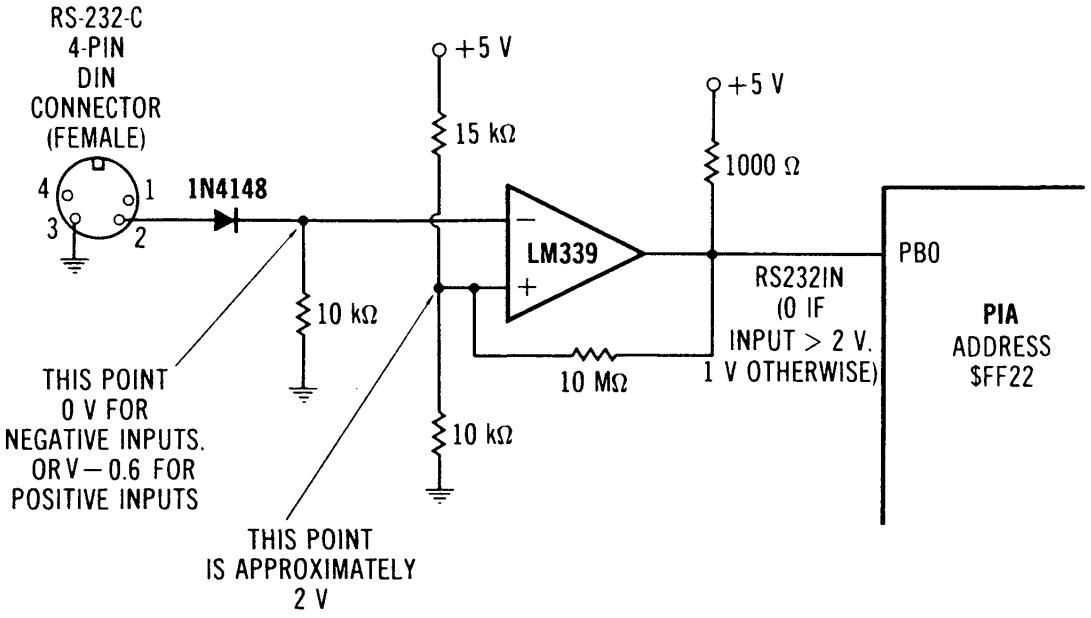


Fig. 13–4. RD line discrete input circuit.

```
100 ' SWITCH CLOSURE FOR RS-232-C RD INPUT
110 A=(PEEK(&HFF22) AND 1)
120 IF A=0 THEN PRINT "ON" ELSE PRINT "OFF"
130 GOTO 110
```

Fig. 13–5. BASIC RD line program.

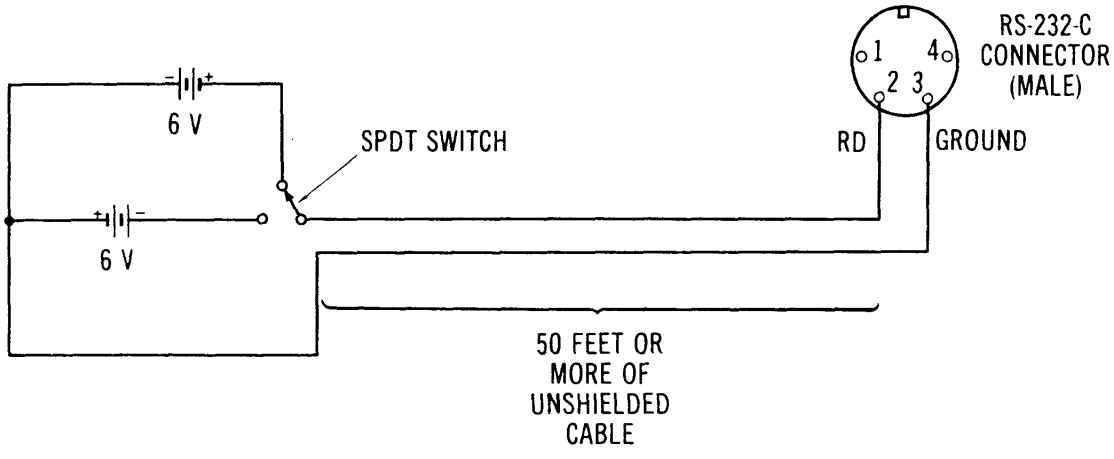


Fig. 13–6. Remote switch connections for RS-232-C.

nal of a small 6-volt battery. Tie the normally open contact of the switch to the negative terminal of a second battery. Tie the opposite ends of the batteries together and to the ground lead of the RS-232-C connector. The common contact of the switch goes to the RD line. There will be some switch bounce when the circuit is broken (on the order of 50 or 60 ms), but this arrangement is fine for slow-speed sensing.

Again, this scheme was exercised using ordinary two-conductor cable without termination and with a 60-foot run in a home environment. No false readings were detected. Twisted pair cable could be used to increase the noise immunity, but as this method is essentially current, rather than voltage, driven, even longer runs should be possible.

Cassette Input

And now for the third method of implementing a discrete input, which forms the thrust of this chapter—using the cassette input. Examination of the Model III and Color Computer shows that the same scheme is used for both the Color Computer and Model III 1500-baud cassette inputs—a comparator input. The Models I and III 500-baud cassette inputs use a different scheme, one of rectifying pulses. This scheme is not as usable for random inputs as the one discussed here.

As a matter of fact, the input circuits in the Color Computer and the Model III 1500-baud cassette logic are identical and are shown in Fig. 13–7.

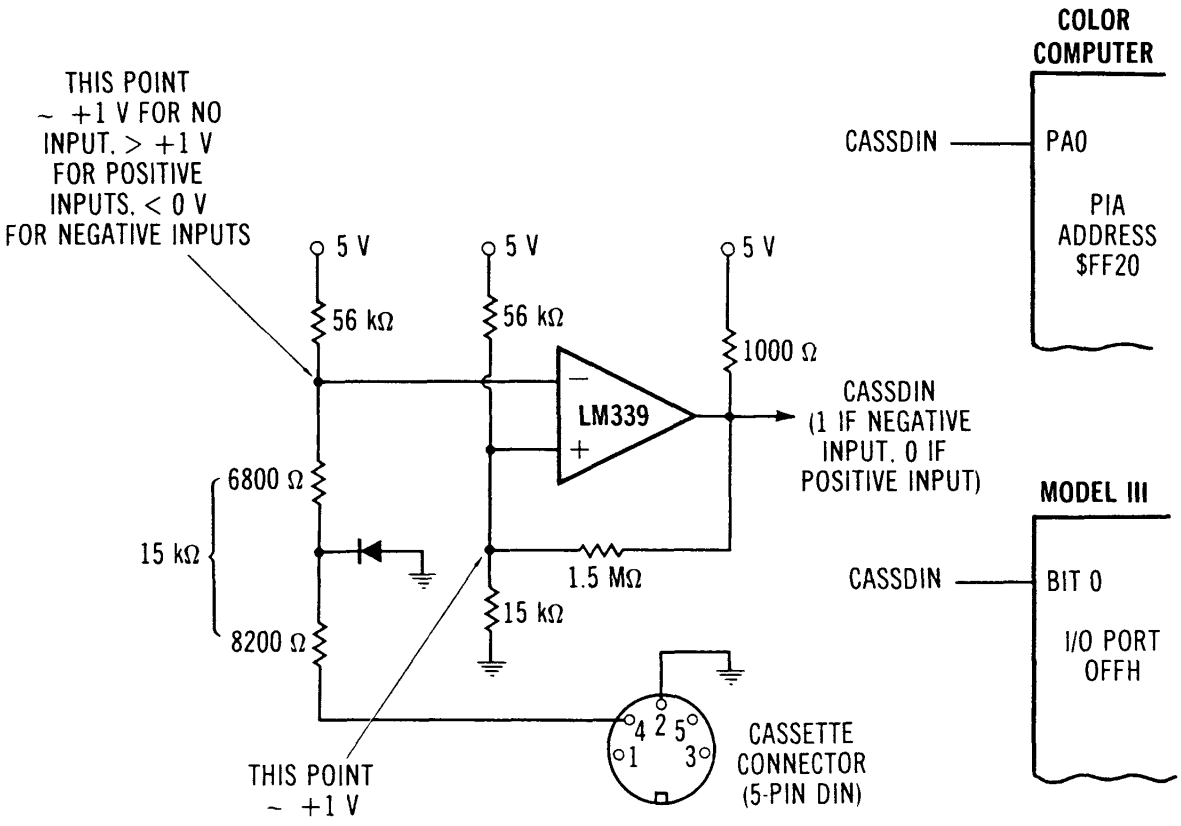


Fig. 13–7. 1500-baud cassette input logic.

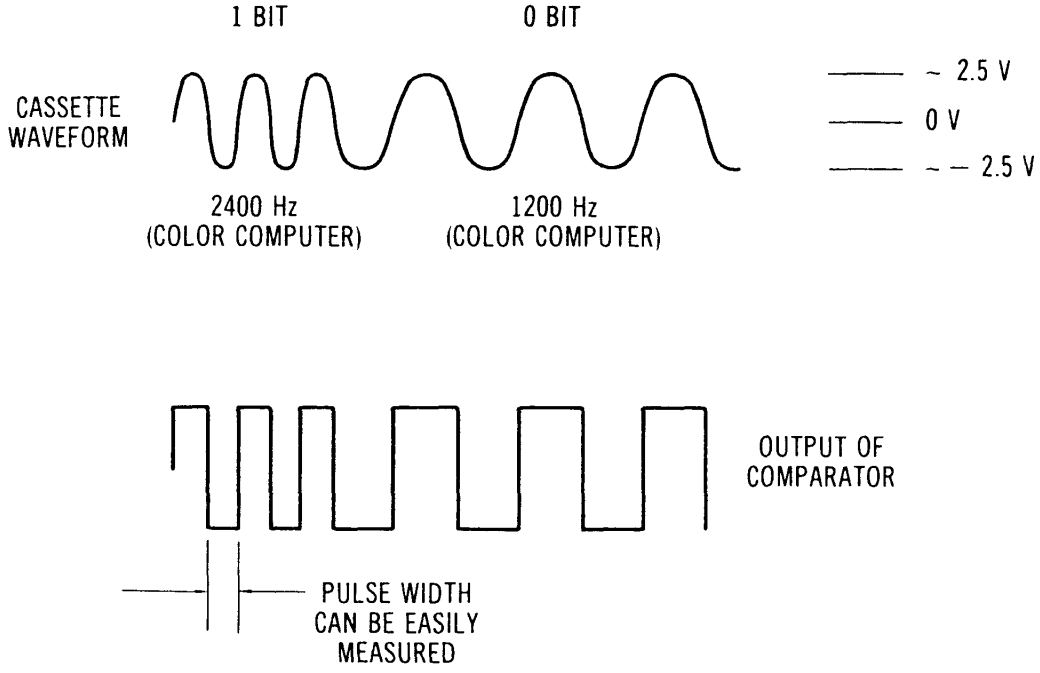


Fig. 13–8. 1500-baud cassette waveforms.

The cassette waveform is sine-wave *frequency-shift* modulated. A different frequency is used for a 0 and 1 bit, as shown in Fig. 13–8. The Model III or Color Computer firmware measures the frequency of the sine wave by looking at the binary output of the LM339 comparator, as shown in Fig. 13–8.

One input to the LM339 is a fixed voltage of about +1 volt from the junction of the 56K and 15K voltage divider. The second input is from a similar voltage divider. In the latter case, however, a diode goes to ground at the junction of the 6.8K and 8.2K resistors.

The input from the cassette recorder is an ac signal, with about 2.5-volt swings on either side of 0. When the cassette signal is positive, the positive input is greater than +1 volt and the comparator output is 0. When the cassette signal is negative, the diode conducts, the negative (−) input is less than 0 volts and the comparator output is 1.

The output of the comparator goes to bit 0 of PIA address $FF20 in the Color Computer or to bit 0 of I/O port address 0FFH in the Model III. Reading either port is a single BASIC instruction (PEEK (&HFF20) or INP(255)), or a comparable machine-language instruction. Fig. 13–9 shows a simple Color Computer BASIC test of the cassette in bit and Fig. 13–10 shows the equivalent Model III test.

A remote-sensing switch can be implemented in identical fashion to the RS-232-C method, as shown in Fig. 13–11. Two batteries produce +3 volts and −3 volts, and these voltages are tied to the NC and NO contacts of the remote-sensing switch. The switch is connected via ordinary two-conductor cable. Again, twisted pair may be used if desired. A 60-foot length of cable was used in a home environment, and no false readings were detected for slow switch closures.

SWITCH BOUNCE

The BASIC programs shown above for the three discrete input methods are fine for slowly changing inputs such as burglar alarms. The resolution of a typical BASIC loop allows sampling at a dozen or so times per second. When the frequency of switch closures is greater, however, we must rely on faster assembly language code. Assembly language code can test the inputs thousands of times per second. In fact, assembly language is so fast that the bounce of switch contacts can cause problems.

The typical switches mentioned above do not close instantaneously. There is a period during which minute movements produce make and break conditions, as shown in Fig. 13–12. There are various hardware schemes to eliminate switch bounce, but we prefer a software solution. The usual software approach is to delay for a fixed interval after detection of the first switch closure.

```
100 ' SWITCH CLOSURE FOR CASSETTE INPUT
110 A=(PEEK(&HFF20) AND 1)
120 IF A=0 THEN PRINT "OFF" ELSE PRINT "ON"
130 GOTO 110
```

Fig. 13–9. BASIC program for the Color Computer cassette input.

```
100 ' SWITCH CLOSURE FOR CASSETTE INPUT
110 A=(INP(255) AND 1)
120 IF A=0 THEN PRINT "OFF" ELSE PRINT "ON"
130 GOTO 110
```

Fig. 13–10. BASIC program for the Model III cassette input.

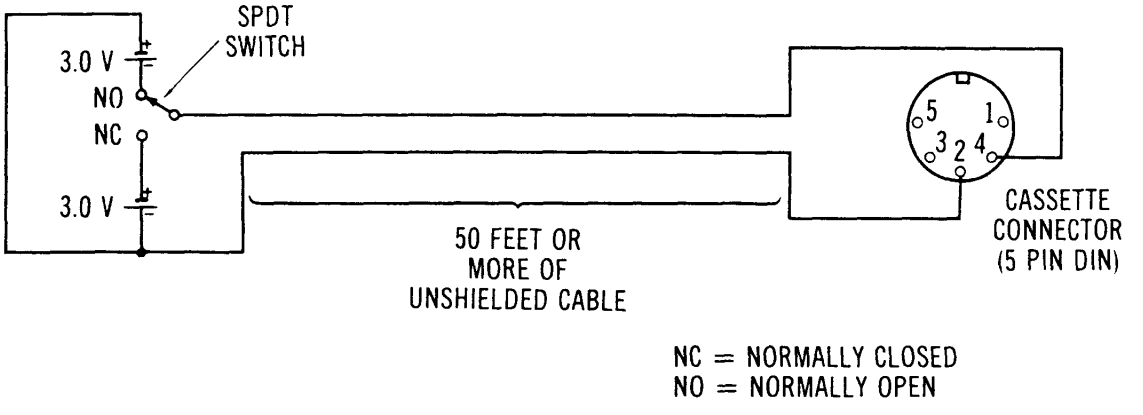


Fig. 13–11. Remote switch connections for cassette operation.

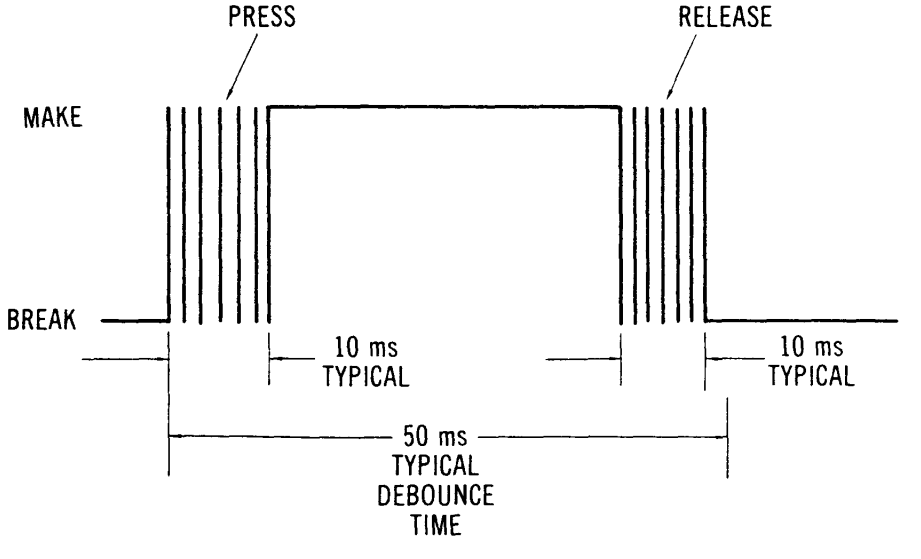


Fig. 13–12. Make-break switch bounce spikes.

The listing below represents the following conditions: The two Radio Shack switches referenced above were pressed rapidly for exactly 10 closures in about two seconds for various *debounce delays* ranging from 10 ms to 100 ms. The count represents the number of switch closures detected. Counts greater than 10 indicate that switch bounces were counted as closures.

Debounce Delay	Count
100 ms	10
90	10
80	10
70	10
60	10
50	17
40	19
30	22
20	33
10	59

The switch bounce delay in software varies with the type of switch and action of the operator. Use these figures as a rough guide only. In the next chapter we look at a low-frequency event counter that automatically debounces the switches that are used to detect the events.

chapter **14**

A Low-Frequency Event Counter

In this chapter we look at a single discrete input from the cassette data, but in this case it's used to measure a more rapidly changing signal.

THE SOFTWARE

Fig. 14–1 shows a low-frequency event counter program for the Color Computer that will measure events that occur thousands of times per second. The discrete input on the cassette input line is designed to interface to a BASIC driver.

Three parameters are stored in high memory (16K system). An interval count is stored in locations $3FFA,B. The interval count may be any number from 1 through 32768 and represents the "window" time that events will be counted, in units of 30.35 μs. An interval count of 1000, for example, represents 30,350 μs, or 30.35 ms. Maximum window time is 32,768 × 30.35 μs, or 0.994508 second.

A debounce delay count in milliseconds is stored in locations $3FFC,D. This delay count will cause the program to close the window for a specified time after each pulse is detected. The number of counts in the interval is returned in locations $3FFE,F.

A similar program for the Model III is shown in Fig. 14–2. The three parameters are passed in locations 7FFAH through 7FFFH and represent the same variables. Operation for the two programs is similar, and both are described in general terms here.

The DELAY subroutine delays for 1 ms by a simple loop. An interval count is adjusted for the 1-ms delay in units of 30.35 or 26.86. The DEBNC subroutine gets the debounce delay parameter and calls DELAY to delay for the debounce time in units of 1 ms.

```
3F00                00100          ORG     $3F00
                    00110 ************************************************
                    00120 * LOW-FREQUENCY EVENT COUNTER WITH DEBOUNCE        *
                    00130 *    INPUT:  $03FFA = INTERVAL CNT IN 30.35 MICROSEC *
                    00140 *            UNITS, 2 BYTES                       *
                    00150 *            $03FFC = DEBOUNCE DELAY CNT IN MS, 2 BY *
                    00160 *            $03FFE = RESERVED FOR COUNT, 2 BYTES    *
                    00170 *    OUTPUT: $03FFE = # OF COUNTS IN INTERVAL, 2 BYT *
                    00180 ************************************************
                    00190 *
3F00 BE   3FFA      00200 LOWFRE  LDX     $3FFA   GET INTERVAL CNT
3F03 108E 0000      00210         LDY     #0      INITIALIZE COUNT
3F07 30   1F        00220 LOW010  LEAX    -1,X        DECREMENT INT CNT (5)
3F09 1F   10        00230         TFR     X,D         NOW IN D (7)
3F0B 4D             00240         TSTA                TEST FOR NEGATIVE (2)
3F0C 2B   0D        00250         BMI     LOW090      GO IF DONE (3)
3F0E B6   FF20      00260         LDA     $FF20       GET PIA BYTE (5)
3F11 84   01        00270         ANDA    #1          GET CASSDIN BIT (2)
3F13 27   F2        00280         BEQ     LOW010      GO IF 0 (3)
3F15 31   21        00290         LEAY    1,Y       1, INCREMENT COUNT
3F17 8D   07        00300         BSR     DEBNC     DEBOUNCE DELAY
3F19 20   EC        00310         BRA     LOW010    CONTINUE INTERVAL
3F1B 10BF 3FFE      00320 LOW090  STY     $3FFE   STORE COUNT
3F1F 39             00330         RTS             RETURN FROM SUBROUTINE
                    00340 *
                    00350 * DEBOUNCE DELAY SUBROUTINE
                    00360 *
3F20 34   10        00370 DEBNC   PSHS    X       SAVE INTERVAL COUNT
3F22 BE   3FFC      00380         LDX     $3FFC   GET DELAY COUNT IN MS
3F25 8D   06        00390 DEB010  BSR     DELAY   DELAY N MS
3F27 30   1F        00400         LEAX    -1,X      DECREMENT DELAY COUNT
3F29 26   FA        00410         BNE     DEB010    GO IF NOT N MS
3F2B 35   90        00420         PULS    X,PC    RETRIEVE INTERVAL COUNT,RTN
                    00430 *
                    00440 * DELAY SUBROUTINE. DELAYS N MS.
                    00450 *
3F2D 34   10        00460 DELAY   PSHS    X       SAVE DELAY COUNT
3F2F 8E   006F      00470         LDX     #111    FINAGLE FACTOR
3F32 30   1F        00480 DEL010  LEAX    -1,X      DECREMENT FINAGLE COUNT
3F34 26   FC        00490         BNE     DEL010    LOOP FOR 1 MS
3F36 AE   64        00500         LDX     4,S     GET INTERVAL COUNT
3F38 30   88 DF     00510         LEAX    -33,X   ADJUST FOR 1 MS DELAY
3F3B AF   64        00520         STX     4,S     RESTORE IN STACK
3F3D 35   90        00530         PULS    X,PC    RETRIEVE COUNT, RTN
          0000      00540         END
00000 TOTAL ERRORS
```

Fig. 14–1. Low-frequency event counter program for the Color Computer.

The "main-line" code is in LOWFRE. The interval count parameter is decremented by one each time through the main loop. When the interval count is decremented beyond zero, the interval is completed, and the subroutine returns to the BASIC program. If the interval count is not completed, the PIA or I/O port bit for CASSDIN is read. If the cassette bit is a 1, the count of pulses is increased by one and the DEBNC subroutine is called for the debounce delay.

These programs detect a 1 pulse, or a negative voltage input. The input signal can be generated by any switch closure occurring at rates up to thousands of times per second. A typical example is a roller switch on a rotating cam shaft. A longer interval can be created by repetitively calling the subroutine from BASIC.

BASIC drivers for both versions are shown in Figs. 14–3 and 14–4. The machine-language forms of the program are contained within the BASIC program in DATA statements, and the programs are relocated to high RAM by the BASIC code.

```
7F00            00100          ORG     7F00H            ;THIS SR NON-RELOCATABLE
                00110 ;*****************************************************
                00120 ;* LOW-FREQUENCY EVENT COUNTER WITH DEBOUNCE          *
                00130 ;*    INPUT:  7FFAH = INTERVAL COUNT IN 26.86 MICROSEC  *
                00140 ;*            UNITS, 2 BYTES                         *
                00150 ;*            7FFCH = DEBOUNCE DELAY CNT IN MS, 2 BYTES *
                00160 ;*            7FFEH = RESERVED FOR COUNT, 2 BYTES    *
                00170 ;*   OUTPUT: 7FFEH = # OF COUNTS IN INTERVAL, 2 BYTES  *
                00180 ;*****************************************************
                00190 ;
7F00 F3         00191 LOWFRE  DI                       ;DISABLE INTERRUPTS
7F01 DD2AFA7F   00200         LD       IX,(7FFAH)      ;GET INTERVAL COUNT
7F05 01FFFF     00210         LD       BC,-1           ;FOR DECREMENTS
7F08 FD210000   00220         LD       IY,0            ;INITIALIZE COUNT
7F0C DD09       00230 LOW010  ADD      IX,BC               ;DECREMENT INT CNT(15)
7F0E D21F7F     00240         JP       NC,LOW090           ;GO IF DONE (10)
7F11 DBFF       00250         IN       A,(0FFH)            ;GET I/O BYTE (11)
7F13 E601       00260         AND      1                   ;GET CASSDIN BIT (7)
7F15 CA0C7F     00270         JP       Z,LOW010            ;GO IF 0 (10)
7F18 FD23       00280         INC      IY                ;INCREMENT COUNT
7F1A CD257F     00290         CALL     DEBNC             ;DEBOUNCE DELAY
7F1D 18ED       00300         JR       LOW010            ;CONTINUE INTERVAL
7F1F FD22FE7F   00310 LOW090  LD       (7FFEH),IY      ;STORE COUNT
7F23 FB         00311         EI                       ;ENABLE INTERRUPTS
7F24 C9         00320         RET                      ;RETURN FROM SR
                00330 ;
                00340 ; DEBOUNCE SUBROUTINE
                00350 ;
7F25 FDE5       00360 DEBNC   PUSH     IY              ;SAVE INTERVAL CNT
7F27 FD2AFC7F   00370         LD       IY,(7FFCH)      ;GET DEBOUNCE DELAY
7F2B CD357F     00380 DEB010  CALL     DELAY             ;DELAY N MS
7F2E FD09       00390         ADD      IY,BC             ;DECREMENT DELAY CNT
7F30 38F9       00400         JR       C,DEB010          ;GO IF NOT N MS
7F32 FDE1       00410         POP      IY              ;RETRIEVE INT COUNT
7F34 C9         00420         RET                      ;RETURN FROM SR
                00430 ;
                00440 ; DELAY SUBROUTINE. DELAYS N MS.
                00450 ;
7F35 215E00     00460 DELAY   LD       HL,94           ;FINAGLE FACTOR
7F38 09         00470 DEL010  ADD      HL,BC             ;DECREMENT FINAGLE CNT
7F39 DA387F     00480         JP       C,DEL010          ;LOOP FOR 1 MS
7F3C 11D8FF     00490         LD       DE,-37          ;ADJUSTMENT CONSTANT
7F3F DD19       00500         ADD      IX,DE           ;ADJUST FOR  1 MS DELAY
7F41 C9         00510         RET                      ;RETURN FROM SR
0000            00520         END
00000 Total errors
```

Fig. 14–2. Low-frequency event counter program for the Model III.

```
100 ' LOWFRE DRIVER
110 DATA 190,63,250,16,142,0,0,48,31,31
120 DATA 16,77,43,13,182,255,32,132,1,39
130 DATA 242,49,33,141,7,32,236,16,191,63
140 DATA 254,57,52,16,190,63,252,141,6,48
150 DATA 31,38,250,53,144,52,16,142,0,111
160 DATA 48,31,38,252,174,100,48,136,223,175
170 DATA 100,53,144
180 FOR I=&H3F00 TO &H3F3E
190 READ A: POKE I,A
200 NEXT I
210 DEFUSR0=&H3F00
220 INPUT "INTERVAL, DELAY";IC,DC
230 POKE &H3FFA,INT(IC/256):POKE &H3FFB,IC-INT(IC/256)*256
240 POKE &H3FFC,INT(DC/256):POKE &H3FFD,DC-INT(DC/256)*256
250 A=USR0(0)
260 B=B+PEEK(&H3FFE)*256+PEEK(&H3FFF):PRINT B
270 GOTO 250
```

Fig. 14–3. Color Computer BASIC driver program.

```
100 ' LOWFRE DRIVER
110 DATA 243,221,42,250,127,1,255,255,253,33
120 DATA 0,0,221,9,210,31,127,219,255,230
130 DATA 1,202,12,127,253,35,205,37,127,24
140 DATA 237,253,34,254,127,251,201,253,229,253
150 DATA 42,252,127,205,53,127,253,9,56,249
160 DATA 253,225,201,33,94,0,9,218,56,127
170 DATA 17,219,255,221,25,201
180 FOR I=32512 TO 32577
190 READ A:POKE I,A
200 NEXT I
210 DEFUSR0=&H7F00
220 INPUT "INTERVAL, DELAY"; IC,DC
230 POKE &H7FFA,IC-INT(IC/256)*256:POKE &H7FFB,INT(IC/256)
240 POKE &H7FFC,DC-INT(DC/256)*256:POKE &H7FFD,INT(DC/256)
250 A=USR0(0)
260 B=B+PEEK(&H7FFE)+PEEK(&H7FFF)*256: PRINT B
270 GOTO 250
```

Fig. 14–4. Model III BASIC driver program.

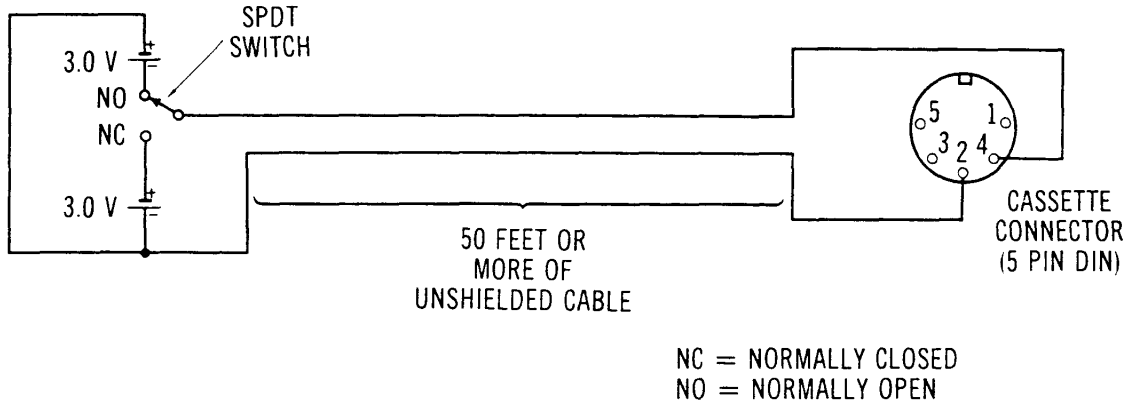


Fig. 14–5. Testing for low-frequency event counting.

RUNNING THE PROGRAM

The BASIC program asks for the interval and debounce delay parameters, POKEs them into the parameter block, and then calls the machine-language subroutine. A running total of all counts is PRINTed after each call. To see how the program works, connect a switch as shown in Fig. 14–5, and then close the switch for various delay and interval times.

In the next chapter we look at a somewhat different version of an event counter, one that measures pulses from the rotation of an anemometer shaft.

chapter **15**

An Anemometer for Measuring Windspeed

To give you a practical example of what can be accomplished with a single discrete input, consider the following plumbing/electronics project, an anemometer. All parts can be purchased at your local hardware store and Radio Shack. The anemometer will measure a wide range of windspeeds; it is easy to construct, and the entire project costs less than $10.00.

THE PLUMBING

The physical appearance of the anemometer is shown in Fig. 15–1. It's constructed with common 1-inch and ½-inch PVC sprinkler fittings, wooden dowels, and plastic cups. A parts list is shown in Table 15–1.

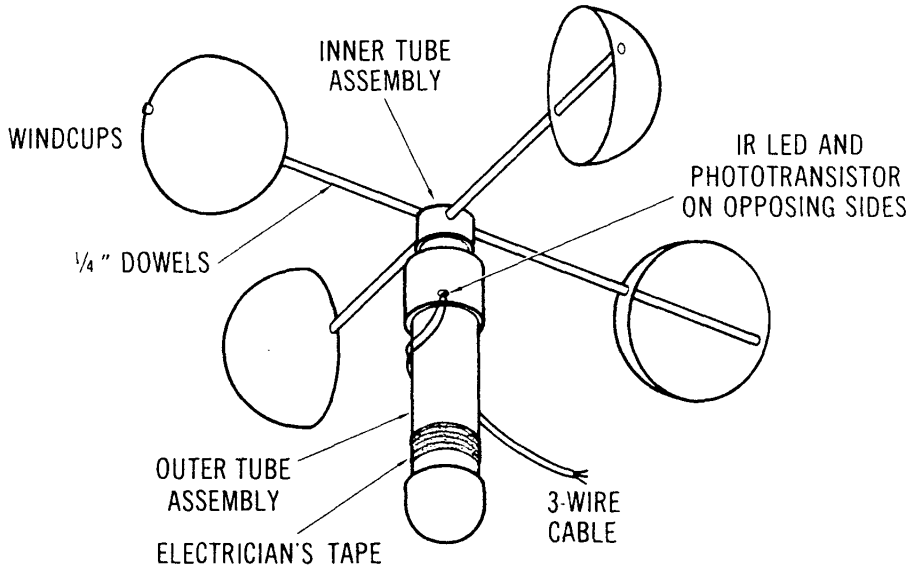


Fig. 15–1. Completed anemometer.

Table 15–1. Anemometer Plumbing Parts List

Qty	Description
1	4-inch piece of 1-inch PVC thickwall tubing
2	1-inch slip cap
2	½-inch slip cap
1	6-inch piece of ½-inch thinwall tubing
1	Roundhead nail or small screw
1	3-foot length of ¼-inch-diameter wooden doweling
4	Plastic low-mass cups (halves of plastic ball or toy)
1	Container of PVC cement
1	Suitable mounting hardware for mast

To assemble the unit, refer to Fig. 15–2, and proceed as follows:

1. Cut a piece of 1-inch PVC thickwall tubing to 4 inches. It cuts easily with any saw. A hacksaw is best for a clean cut.
2. Drill a hole in a 1-inch cap just large enough to pass a ½-inch PVC tube without friction.
3. Cement the cap to the tube with PVC cement. Push the cap firmly down on the tube.
4. Drill a 3/16-inch hole about ½ inch up from the bottom of the cap completely through the cap.
5. File off any projections on the bottom of a ½-inch cap.
6. Drill a small, centered hole in the cap and push in a decorative nail with a rounded head. It should fit firmly.
7. Cement the cap to a 6-inch piece of thinwall ½-inch PVC tubing.
8. Push the ½-inch tubing through the hole in the 1-inch cap. Now cement a second 1-inch cap over the 1-inch tube. (The second cap should have two ⅛-inch drain holes in the bottom.) Do not push the cap on all the way. The inner ½-inch tube should move as freely as possible.
9. After the cement has dried for an hour, drill a 3/16-inch hole through the inner tube, using the existing hole as a guide. Hold the tubes up to the light. The holes in the tubes should match. If not, drill out the inner tube again.
10. Drill two ¼-inch holes completely through a ½-inch cap. The holes should be at right angles to each other and as close to the top of the cap as possible. The bottom hole should clear the path of the top hole.

11. Cut two ¼-inch wood dowels to 14 inches. Push them through the holes in the cap. Center the dowels.
12. Cement the dowels if they do not fit tightly.
13. Mount four ½ plastic spheres (cups) on the four dowels. All four should present the same face to the wind.

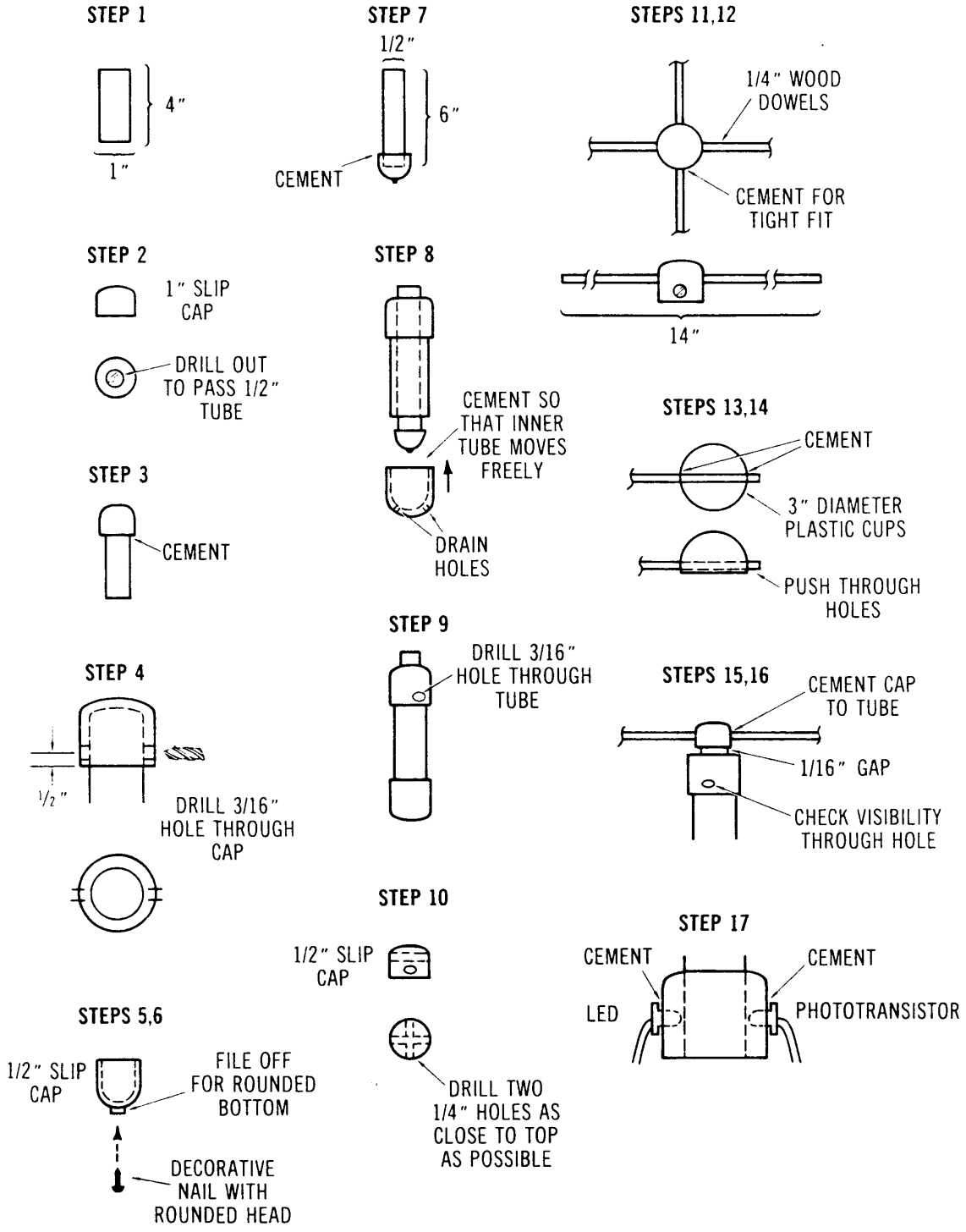


Fig. 15–2. Anemometer assembly details.

14. Align and cement the plastic cups.
15. After the cement has dried, temporarily mount the cup assembly on the inner tube. Cut off enough of the inner-tube so that the bottom of the inner-tube cap is about $\frac{1}{16}$ inch from the top of the 1-inch cap. Cement the cup assembly to the inner tube.
16. Again, check the hole alignment of the inner and outer tubes. Redrill the inner tube if necessary.
17. Press-fit the phototransistor and LED into the two holes. Bring the two leads from each down. Cement the components in place using a bead of PVC cement around the edges.
18. Spin the cup assembly. It should move very freely, even in a light wind. You should be able to spin it by gently blowing at a cup at a distance of about a foot.

THE ELECTRONICS

The electronics assembly is built on a Radio Shack project board. The arrangement of the parts is shown in Fig. 15–3 and a parts list is given in Table 15–2.

Make a cable assembly of four wires and route to the anemometer. Solder the four cable wires to the LED and phototransistor as shown in Fig. 15–4. After soldering the cable wires, wrap a piece of plastic electrical tape around the cable and tubing for strain relief. Put a dab of PVC

Table 15–2. Anemometer Electronics Parts List

Qty	Description
1	Resistor, 150 Ω ¼ W 10% tolerance
1	Resistor, 1000 Ω ¼ W 10% tolerance
1	Resistor, 100,000 Ω ¼ W 10% tolerance
1	Resistor, 4700 Ω ¼ W 10% tolerance
1	Resistor, 560 Ω ¼ W 10% tolerance
1	741C op amp (Radio Shack 276-007)
1	Infrared LED (Radio Shack XC880-A, 276-143)
1	Infrared phototransistor (Radio Shack 276-145)
2	6-V batteries or 8 C cells in assembly
1	Project board (Radio Shack 276-175)
1	DIN plug, 5-pin (Radio Shack 274-003)
Misc.	Wire, cable, solder

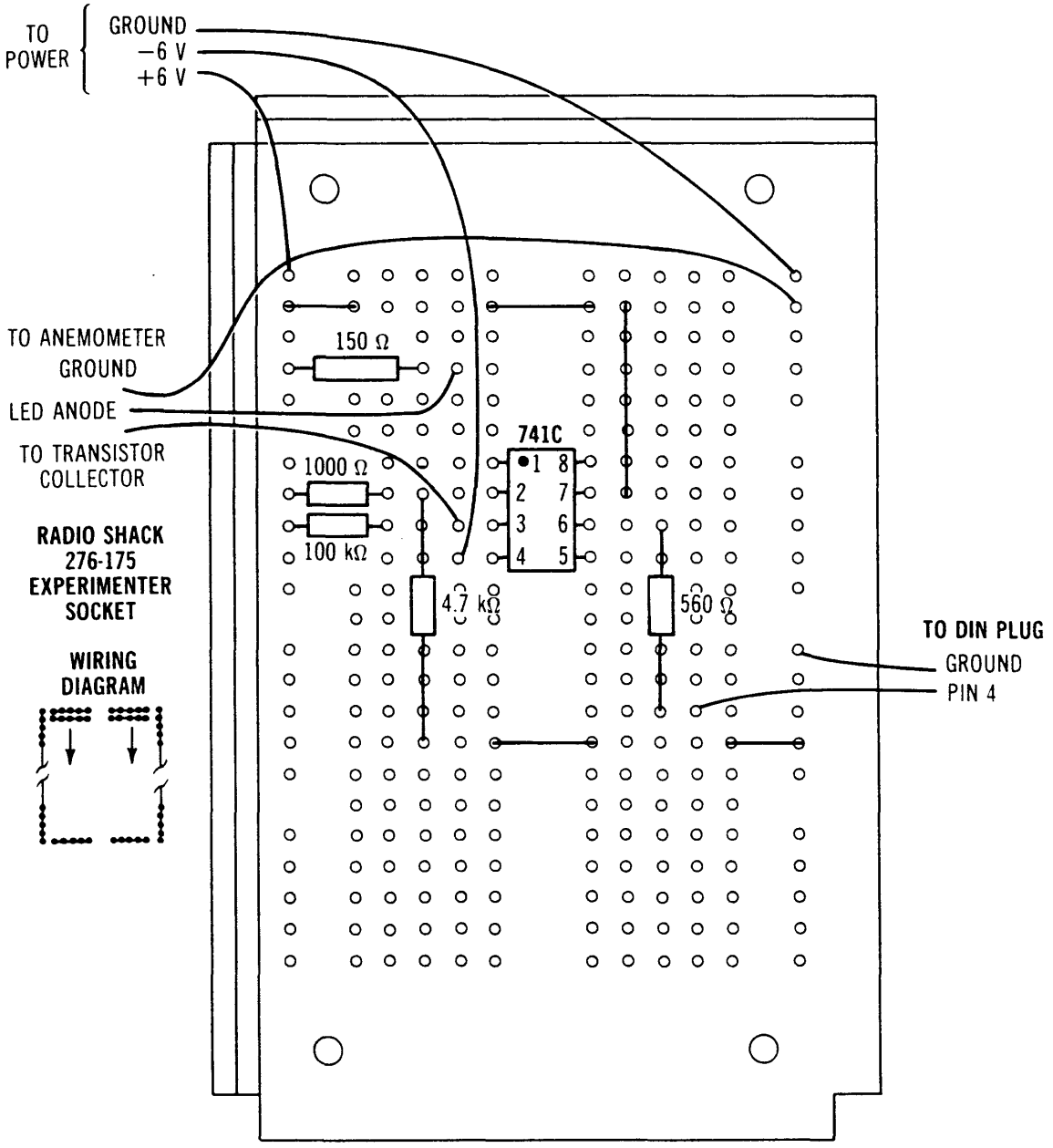


Fig. 15–3. Physical layout of the anemometer electronics.

cement on each solder joint and exposed lead. This will waterproof the connections.

The circuit for the electronics is shown in Fig. 15–5. The electronics produces a +6-volt or −6-volt signal to the cassette input line. The 741C op amp compares a voltage at the negative (−) input that is about 82% of the positive supply voltage. If the input voltage on the positive (+) lead drops below this level, the output of the op amp is −6 volts; otherwise, it is +6 volts.

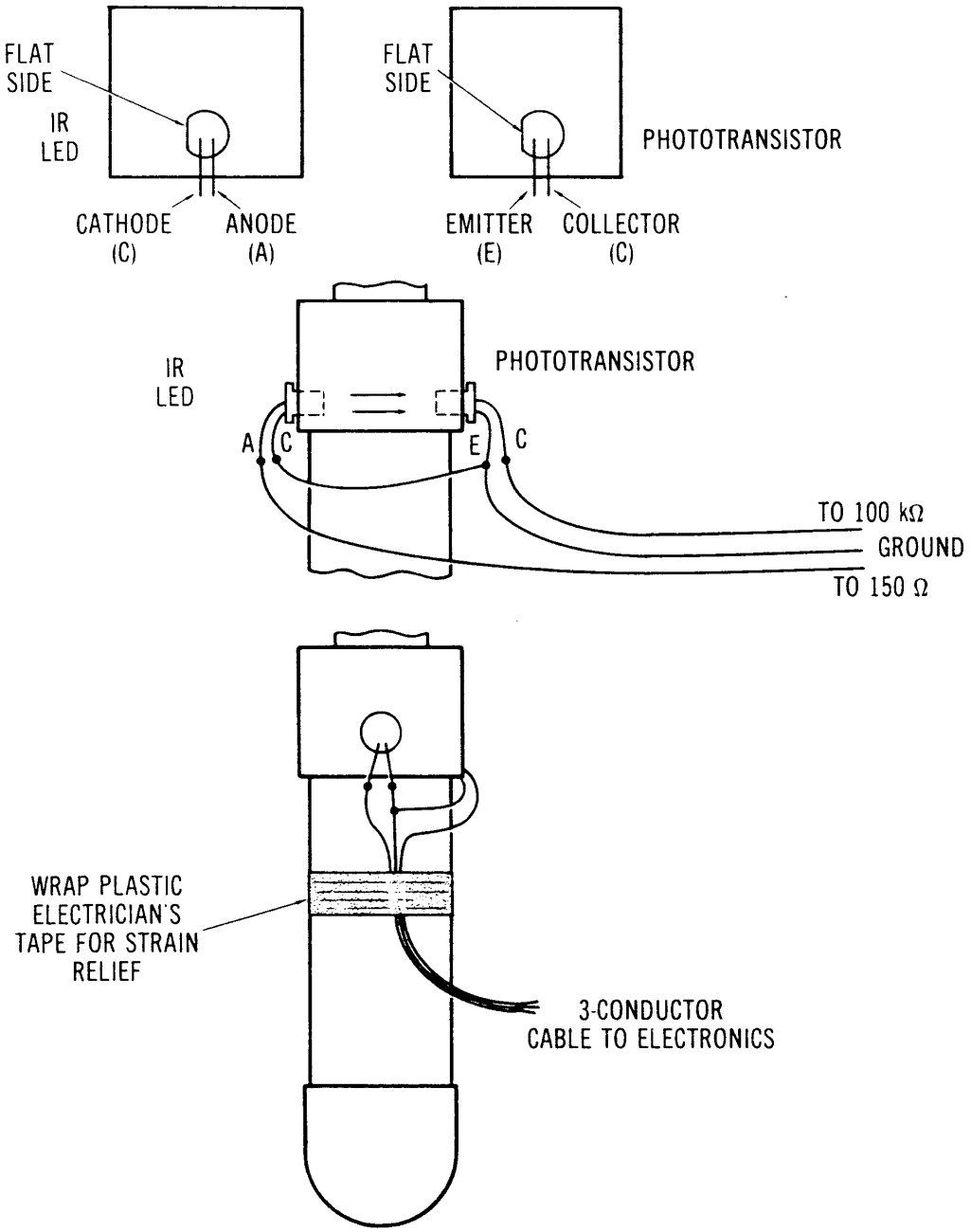


Fig. 15–4. Anemometer shaft electronics.

The LED device is an infrared phototransistor. When the two holes in the anemometer tubing are aligned, the LED infrared light strikes the phototransistor and causes current flow through it. When enough current flows, the positive (+) input drops below the 82% level and the op amp output drops to −6 volts. When no light is striking the phototransistor, no current flows through it, the positive (+) input is +6 volts, and the op amp output is +6 volts.

Locate the anemometer assembly in a shady place exposed to the wind. (It is best not to mount it on a 200-foot tower until after additional testing, however.)

A PERIOD PROGRAM

Figs. 15–6 and 15–7 show PERIOD programs for the Color Computer and Model III, respectively. The PERIOD programs are used with the anemometer, but are also general-purpose programs for measuring the period of any input signal that does not have debounce. The period is measured from the first negative-going transition to the next negative-going transition in 20.23- or 24.33-μs units. The period is passed back to a BASIC driver in locations $3FFE,F (Color Computer) or in HL (Model III). Figs. 15–8 and 15–9 show the PERIOD programs incorporated as DATA statements in BASIC drivers. The machine-language code is relocated to high memory in either case.

USING THE ANEMOMETER

When using the anemometer, protect high memory in the Color Computer by a CLEAR 200,&H3EFF. Protect high memory in the Model III by inputting a MEMORY SIZE of 32511. Load the BASIC anemometer

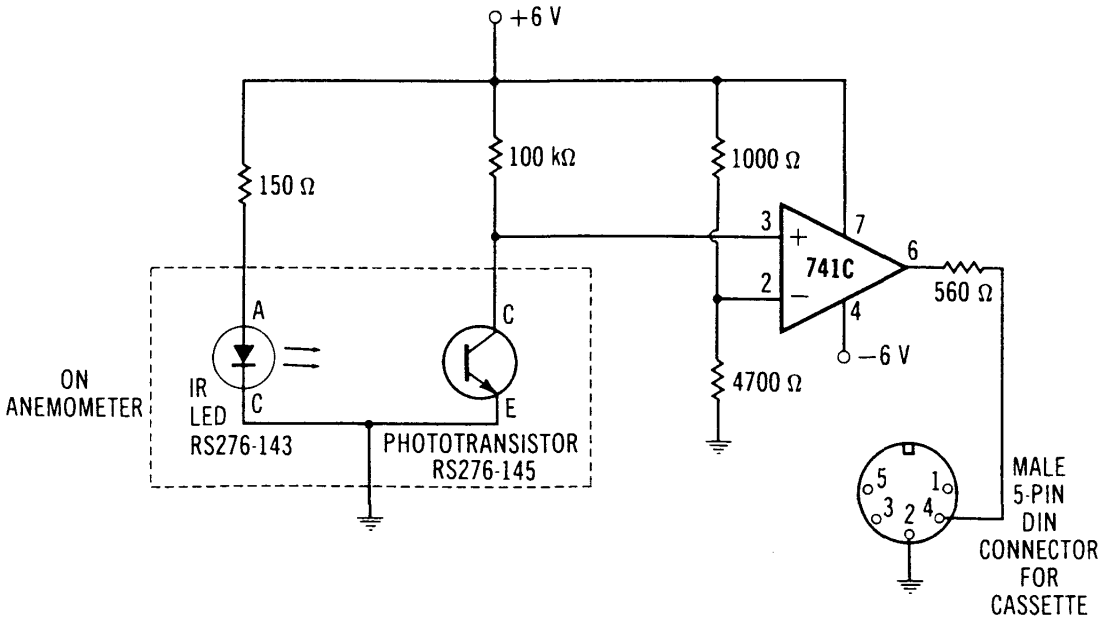


Fig. 15–5. Anemometer electronics logic diagram.

```
3F00                00100          ORG     $3F00
                    00110 ************************************************
                    00120 * SUBROUTINE TO MEASURE PERIOD OF LOW-FREQ SIGNAL   *
                    00130 *    INPUT:  NIL                                   *
                    00140 *    OUTPUT: 3FFEH=PERIOD IN 20.23 MICROSECOND UNITS *
                    00150 *            OR -1 IF TIME OUT, 2 BYTES            *
                    00160 ************************************************
                    00170 *
3F00 8E   0000      00180 PERIOD  LDX     #0        TIME OUT COUNT
3F03 30   01        00190 PER010  LEAX    1,X         INCREMENT TIME OUT CNT
3F05 27   30        00200         BEQ     PER090      GO IF TIME OUT
3F07 B6   FF20      00210         LDA     $FF20       GET PIA BYTE
3F0A 84   01        00220         ANDA    #1          GET CASSDIN
3F0C 26   F5        00230         BNE     PER010      GO IF AT PULSE
3F0E 8E   0000      00240         LDX     #0        INITIALIZE TIME OUT CNT
3F11 30   01        00250 PER020  LEAX    1,X         INCREMENT TIME OUT CNT
3F13 27   22        00260         BEQ     PER090      GO IF TIME OUT
3F15 B6   FF20      00270         LDA     $FF20       GET PIA BYTE
3F18 84   01        00280         ANDA    #1          GET CASSDIN
3F1A 27   F5        00290         BEQ     PER020      GO IF NOT PULSE
3F1C 8E   0000      00300         LDX     #0        INITIALIZE TIME OUT
3F1F 30   01        00310 PER030  LEAX    1,X         INCREMENT TIME OUT CNT
3F21 27   14        00320         BEQ     PER090      GO IF TIME OUT
3F23 B6   FF20      00330         LDA     $FF20       GET PIA BYTE
3F26 84   01        00340         ANDA    #1          GET CASSDIN
3F28 26   F5        00350         BNE     PER030      GO IF STILL PULSE
3F2A 30   01        00360 PER050  LEAX    1,X         INCREMENT TIME COUNT
3F2C 27   09        00370         BEQ     PER090      GO IF TIME OUT
3F2E B6   FF20      00380         LDA     $FF20       GET PIA BYTE
3F31 84   01        00390         ANDA    #1          GET CASSDIN
3F33 27   F5        00400         BEQ     PER050      GO IF NOT END
3F35 20   03        00410         BRA     PER095    NORMAL RETURN
3F37 8E   FFFF      00420 PER090  LDX     #-1       FLAG TIME OUT
3F3A BF   3FFE      00430 PER095  STX     $3FFE     RETURN WITH ARGUMENT
3F3D 39             00440         RTS               RETURN
          0000      00450         END
00000 TOTAL ERRORS
```

Fig. 15–6. Color Computer PERIOD program.

```
7F00            00100          ORG     7F00H           ;RELOCATABLE
                00110 ;*************************************************
                00120 ;* SUBROUTINE TO MEASURE PERIOD OF LOW-FREQ SIGNAL    *
                00130 ;*     INPUT:  NIL                                    *
                00140 ;*     OUTPUT: HL=PERIOD IN 24.33 MICROSECOND UNITS   *
                00150 ;*             OR -1 IF TIME OUT, 2 BYTES             *
                00160 ;*************************************************
                00170 ;
7F00 210000     00180 PERIOD  LD      HL,0            ;TIME OUT COUNT
7F03 010100     00190         LD      BC,1            ;INCREMENT
7F06 09         00200 PER010  ADD     HL,BC             ;INCREMENT TIME OUT CNT
7F07 3829       00210         JR      C,PER090          ;GO IF TIME OUT
7F09 DBFF       00220         IN      A,(0FFH)          ;GET I/O BYTE
7F0B E601       00230         AND     1                 ;GET CASSDIN
7F0D 20F7       00240         JR      NZ,PER010         ;GO IF AT PULSE
7F0F 210000     00250         LD      HL,0            ;REINITIALIZE TIME OUT CNT
7F12 09         00260 PER020  ADD     HL,BC             ;INCREMENT TIME OUT COUNT
7F13 381D       00270         JR      C,PER090          ;GO IF TIME OUT
7F15 DBFF       00280         IN      A,(0FFH)          ;GET I/O BYTE
7F17 E601       00290         AND     1                 ;GET CASSDIN
7F19 28F7       00300         JR      Z,PER020          ;GO IF NOT PULSE
7F1B 210000     00310         LD      HL,0            ;REINITIALIZE TIME OUT
7F1E 09         00320 PER030  ADD     HL,BC             ;INCREMENT TIME OUT
7F1F 3811       00330         JR      C,PER090          ;GO IF TIME OUT
7F21 DBFF       00340         IN      A,(0FFH)          ;GET I/O BYTE
7F23 E601       00350         AND     1                 ;GET CASSDIN
7F25 20F7       00360         JR      NZ,PER030         ;GO IF STILL PULSE
7F27 09         00370 PER050  ADD     HL,BC             ;INCREMENT TIME OUT
7F28 3808       00380         JR      C,PER090          ;GO IF TIME OUT
7F2A DBFF       00390         IN      A,(0FFH)          ;GET I/O BYTE
7F2C E601       00400         AND     1                 ;GET CASSDIN
7F2E 28F7       00410         JR      Z,PER050          ;GO IF NOT END
7F30 1803       00420         JR      PER095          ;NORMAL RETURN
7F32 21FFFF     00430 PER090  LD      HL,-1           ;FLAG TIME OUT
7F35 C39A0A     00440 PER095  JP      0A9AH           ;PASS ARGUMENT BACK
0000            00450         END
00000 Total errors
```

Fig. 15–7. Model III PERIOD program.

```
100 'SAMPLE ANEMOMETER PROGRAM
110 DATA 142,0,0,48,1,39,48,182,255,32
120 DATA 132,1,38,245,142,0,0,48,1,39,34
130 DATA 182,255,32,132,1,39,245,142,0,0
140 DATA 48,1,39,20,182,255,32,132,1,38
150 DATA 245,48,1,39,9,182,255,32,132,1,39
160 DATA 245,32,3,142,255,255,191,63,254,57
170 FOR I=&H3F00 TO &H3F3D
180 READ A: POKE I,A
181 IF INKEY$="" GOTO 181
190 NEXT I
200 DEFUSR0=&H3F00
210 CLS
220 A=USR(0)
230 A=PEEK(&H3FFE)*256+PEEK(&H3FFF)
240 IF A=65535 THEN A=0: GOTO 260
250 A=3.75/(A*20.23E-6)
260 PRINT @262,A;"MPH          "
270 GOTO 220
```

Fig. 15–8. BASIC driver program for the Color Computer PERIOD.

```
100 'SAMPLE ANEMOMETER PROGRAM MODEL III
110 DATA 33,0,0,1,1,0,9,56,41,219
120 DATA 255,230,1,32,247,33,0,0,9,56
130 DATA 29,219,255,230,1,40,247,33,0,0
140 DATA 9,56,17,219,255,230,1,32,247,9
150 DATA 56,8,219,255,230,1,40,247,24,3
160 DATA 33,255,255,195,154,10
170 FOR I=32512 TO 32567
180 READ A: POKE I,A
190 NEXT I
200 DEFUSR0=&H7F00
210 CLS
220 A=USR(0)
230 IF A=-1 THEN A=0:GOTO 250
235 IF A<0 THEN A=65536+A
240 A=3.75/(A*24.33E-6)
250 PRINT @ 532,A;"MPH          "
260 GOTO 220
```

Fig. 15–9. BASIC driver for the Model III PERIOD.

program, connect the cassette input to the electronics, connect the electronics power, and execute the program.

The program measures the period for the rotating anemometer. If no rotation is detected, a -1 is returned as the period; both BASIC programs look for this flag and set the period to 0 in this case.

The windspeed will be displayed in the center of the screen. The windspeed in this case is a rough calculation based on preliminary empirical tests. The tests involved driving madly down city streets while keeping one eye on the road and the other on the rotating anemometer held at arm's length out the open car window. Each rotation of the shaft produces two pulses. A windspeed of 15 miles per hour is approximately 2 revolutions per second; therefore, the 3.75 factor. The rotational speed of the anemometer appears to be linear; 30 mph wind produces 4 revolutions per second, and so forth.

The examples above show what can be achieved without a great deal of additional hardware when using inputs that were not meant to be discrete inputs but were dragged, kicking and screaming, into duty. In the next section we look at a more sophisticated way to interface many lines to the system bus.

Section V

Connecting the System Bus of the Models I and III and Color Computers

chapter **16**

The Color Computer Input/Output Bus

In previous sections of this book we go to some extremes to implement I/O ports by using the Color Computer cassette input and output, the joystick inputs, and the RS-232-C port. In this section we show you the "right way" to connect discrete (on/off) lines to the outside world from the Color Computer. It's not that you can't use the other designs to control and monitor outside world events—the previous implementations work fine. It's just that you can easily build a general-purpose I/O board that can plug into the Color Computer ROM connector or the Model I or Model III input/output bus connector that will provide 24 lines (!) of I/O. Each of the 24 lines can be programmed as either an input or output.

The entire board will cost under $25.00. And, with a few inexpensive components, you can use it to control sprinkling systems and coffee pots and to monitor burglar alarms and doorbells. You can even use an interrupt with the board to run an important real-time foreground task while running a background BASIC or other task!

In this chapter we look into how the Color Computer handles ROM and other input/output operations. In the next chapter we describe a general-purpose input/output board for the Color Computer. The same two topics for the Models I and III are covered in the last two chapters of this section.

COLOR COMPUTER I/O STRUCTURE

Fig. 16–1 shows a logic diagram of the Color Computer I/O. In fact, a large portion of the I/O structure is defined by two chips—the 6809E microprocessor and the *synchronous address multiplexer* or SAM chip.

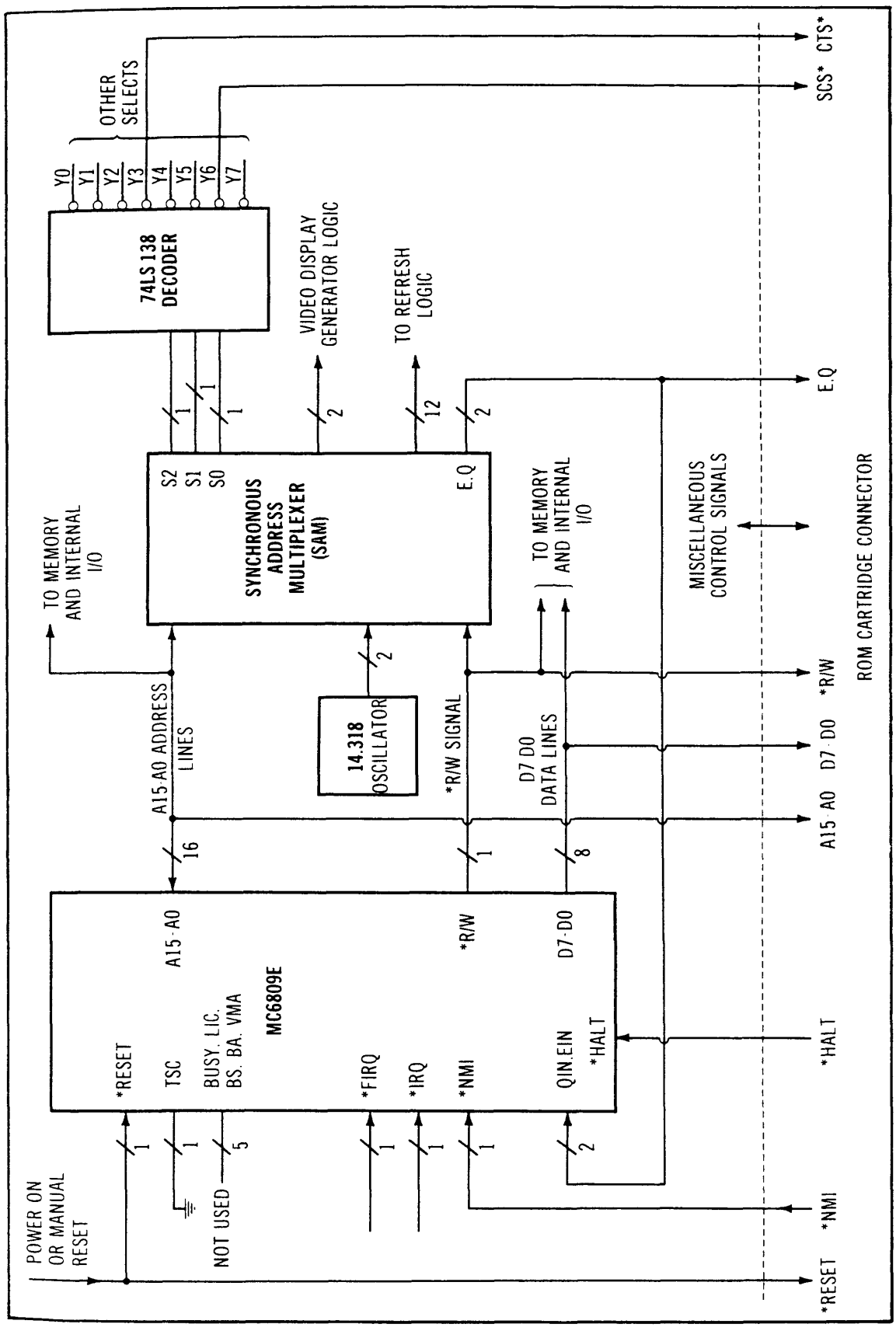


Fig. 16–1. Color Computer I/O block diagram.

The 6809E Microprocessor

The 6809E is closely related to the Motorola 6800 chip. If you want an in-depth understanding of both, I would suggest getting the Motorola *Microcomputer Data Library* reference book, which contains specifications on both the 6800 and 6809E. Some of the basic information is summarized here.

The 6809E is basically an 8-bit microprocessor with some 16-bit processing capability. It has 16 address lines, designated A15 (most significant) through A0 (least significant). The address lines are used to define memory addresses for instruction fetches, access of operands, and reading and writing of I/O data.

There are 8 data lines, designated D7 (most significant) through D0 (least significant). The data lines are used to transfer instruction bytes to the processor during instruction fetch and operand bytes during instruction execution. D7 through D0 are also used to transfer I/O data. All data transfers are done one byte at a time.

The 6809E uses two clock inputs (the E designation of the 6809E specifies an external clock), QIN and EIN. The clock signal is developed from a crystal oscillator signal that is an input to the SAM chip; the SAM chip generates the E and Q clock inputs for the 6809E.

There are three interrupt inputs to the 6809E: *IRQ, *FIRQ, and *NMI. The asterisk prefix indicates that these are *active low* signals that must go to 0 volts for action. The *IRQ is the primary interrupt input to the 6809E. It signals the 6809E that an interrupt has occurred. If the interrupts are enabled (software control), the 6809E will go into a predefined interrupt processing routine at the location defined by the contents of memory location FFF8,9 (BFF8,9 in the Color Computer). The *FIRQ input is an upgrade from the 6800. It is a fast interrupt that saves less of the environment (CPU registers) when an FIRQ occurs. The *NMI is a *nonmaskable* interrupt that cannot be disabled. It is generally used for major conditions that must always be detected, such as a real-time clock pulse or impending power failure.

The *HALT input will halt the CPU at the end of the current instruction. It is an orderly way to stop the CPU and to allow control of the program by an outside source. A typical application might be in single-instruction stepping.

The *RESET input is used to initiate a startup action. This feature is useful at power-up and at those times during a new or routine operation

when the CPU is "hung-up" due to improper programming or I/O protocol.

The *R/W output signal tells the external memory or I/O devices whether a read (high, logic 1, or +5 volts) or write (low, logic 0, or 0 volts) is taking place.

The TSC, BUSY, LIC, BS, BA, and VMA pins are not used in the Color Computer configuration. Many of these signals relate to controlling the address and data bus lines for direct-memory access—independent control of system memory for I/O action.

The lines from the CPU discussed above constitute part of the system bus, which is brought out to the ROM cartridge connector, a 40-pin edge connector on the Color Computer pc board.

The SAM Chip

The 6809E works in conjunction with the synchronous address multiplexer or SAM chip. The SAM is an important chip in the Color Computer.

One task that the SAM handles is *refresh* of the 4116 dynamic memory in the Color Computer. This type of memory must be periodically accessed to retain the voltage charge and, hence, memory data. This refresh is done during times in which no CPU memory addressing is active, so there is no conflict in using the memory address lines.

Another major task of the SAM is to synchronize video display updates and CPU operation. The 6847 video display generator uses RAM memory data to update the video display; it must know when valid data appears from the video display portion of RAM. The SAM chip integrates the CPU and video display memory addressing. Generation of the timing signal is a third SAM function and has been discussed above.

The last function of the SAM is to decode and control the memory mapping of the system. Three signals, S2, S1, and S0, are output from the SAM into a 74LS138 decoder chip. Only one of the 8 outputs of the 74LS138 is active (low) at any time. The one chosen depends upon the state of S2, S1, and S0, which, in turn, depends upon the A15 through A0 inputs.

If Y0 is active, RAM memory from address $0000 through $7FFF is being addressed. If Y1 or Y2 is active, ROM area $8000–$9FFF or $A000–$BFFF is being addressed. If Y3 is active, cartridge ROM at $C000 up is being addressed (CTS*). If Y4 is active, the PIA addresses at

locations $FF00 through $FF1F are being addressed. If Y5 is active, the PIA addresses at locations $FF20 through $FF3F are being addressed. If Y6 is active, memory locations $FF40 through $FF5F are being addressed. These locations are nonexistent in the Color Computer, but come out to a ROM cartridge pin (SCS*) and can be used in external logic. Signal Y7 is not used.

ROM CARTRIDGE SIGNALS

The ROM cartridge connector uses 40 pins with signals from the 6809E CPU, SAM, power supplies, and some additional logic. The ROM cartridge port is more than just a port that enables the Color Computer to execute a program in ROM or EPROM (erasable PROM); it is a general-purpose port that enables interfacing RAM memory or I/O devices of many types.

Table 16–1 lists the ROM cartridge port pins, signal names, source, and description. Signals D7 through D0 and A15 through A0 are the data and address lines from the CPU, respectively. These are essential in connecting memory or I/O devices to the system. Bringing out all 16 address lines allows any of the 65,536 addresses in the 6809E addressing space to be specified.

The *RESET signal to the CPU is also brought out on pin 5 of the ROM cartridge port. A power-on or manual reset can reset an external device with this signal, in addition to causing the CPU reset.

The CART and *HALT CPU inputs are generated from external logic connected to the ROM cartridge port. The EIN and QIN clock outputs from the SAM are also sent to pins 6 and 7 of the ROM cartridge port. We see how these signals work in the cartridge ROM case shortly.

The *R/W signal from the CPU is brought out on pin 18 of the connector. The *R/W signal is necessary to define whether or not a read or write should be done during I/O between an external device and the system.

The *NMI signal to the CPU is generated only by external logic. It can be used to cause a nonmaskable interrupt to the CPU, but is not used in the standard ROM configuration.

The SCS* signal is the select signal from the 74LS138 chip that indicates that an address in the range $FF40 through $FF5F is being used. This spare address is not normally used in Radio Shack software as it needs to be further conditioned by ANDing the E clock signal for external input/output.

Table 16–1. ROM Cartridge Signals

Type	Pin	Name	Source	Description
Power	1	−12 V	CC	
	2	+12 V	CC	
	9	+5 V	CC	
	33	GND	CC	
	34	GND	CC	
Data	10	D0	CC-6809E	
	11	D1	CC-6809E	
	12	D2	CC-6809E	
	13	D3	CC-6809E	Data Bus
	14	D4	CC-6809E	
	15	D5	CC-6809E	
	16	D6	CC-6809E	
	17	D7	CC-6809E	
Address	19	A0	CC-6809E	
	20	A1	CC-6809E	
	21	A2	CC-6809E	
	22	A3	CC-6809E	
	23	A4	CC-6809E	
	24	A5	CC-6809E	
	25	A6	CC-6809E	
	26	A7	CC-6809E	Address Bus
	27	A8	CC-6809E	
	28	A9	CC-6809E	
	29	A10	CC-6809E	
	30	A11	CC-6809E	
	31	A12	CC-6809E	
	37	A13	CC-6809E	
	38	A14	CC-6809E	
	39	A15	CC-6809E	
Clock	6	E	CC-SAM	Clock Signals
	7	Q	CC-SAM	Clock Signals
Select	32	CTS*	CC-74LS138	ROM or I/O Select
	36	SCS*	CC-74LS138	ROM or I/O Select
	40	SLENB*	External	Decode Disable
Other	3	HALT*	External	Halts CPU
	4	*NMI	External	NMI Interrupt
	5	RESET*	CC	Power On or Reset
	8	CART	External	Cartridge Sense
	18	R/W*	CC-6809E	Read/Write Signal
	35	SND	External	External Sound

CC = Color Computer; External = Input to CC

The SLENB* signal is generated by logic connected to the ROM cartridge port. Bringing this input low (0 volts) disables the address decoding by the 74LS138 (Y0 through Y7 remain inactive). This signal enables the device connected to the cartridge port to turn off all internal devices. It is not normally used in Radio Shack software.

The SND input from logic connected to the ROM cartridge port enables an external sound to be routed through the system to television audio.

ROM OPERATION

The layout of a typical ROM plug-in cartridge is shown in Fig. 16–2. The data lines D7 through D0 connect to the data lines of the ROM memory. When any address in the range of $C000 through $DFFF is addressed by the CPU, the CTS* line goes active. The CTS* signal from

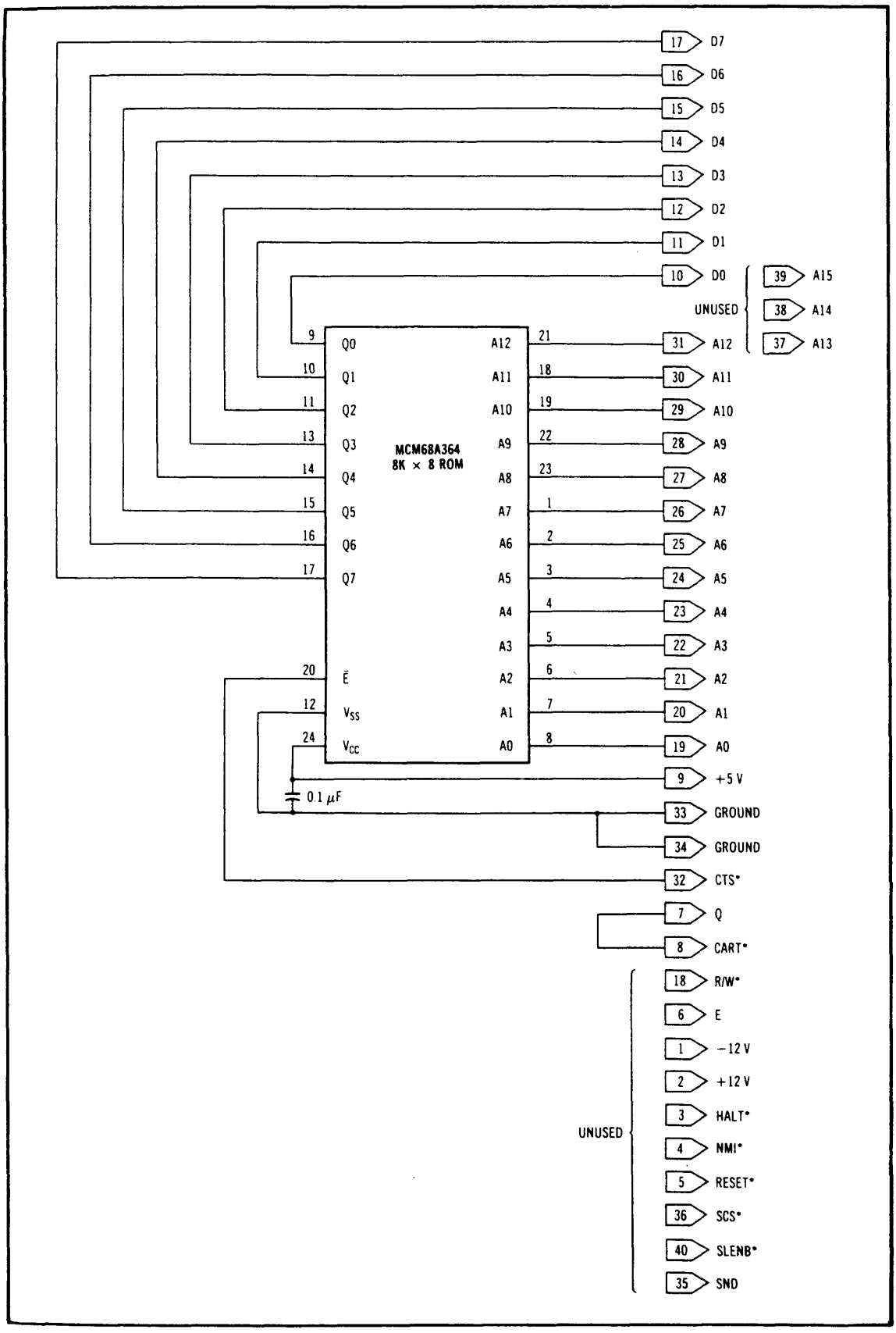


Fig. 16–2. Color Computer ROM logic diagram.

or an I/O device. The programmer must know how the 64K (65,536) memory addresses in the addressing space of the 6809E are mapped; i.e., which addresses are RAM, which are ROM, and which are I/O devices. Writing is handled in similar fashion. An STA ADDRESS can store 8 bits of data either to a memory location or to an I/O device, depending on how the system is mapped. PEEKs and POKEs in BASIC operate in identical fashion to LDAs and STAs; the two commands can read or write data from either memory or I/O devices.

General Input Operation

In general, a read from an external device plugged into the ROM cartridge connector would proceed as follows:

1. An LDA $XXXX or PEEK &HXXXX would be executed by the program, where XXXX is an address in the range $C000 through $BFFF.
2. The CTS* signal would become active and the address lines to the cartridge would contain the entire XXXX address. The CTS* signal is logically ANDed with the CP E clock, necessary for proper input/output operation.
3. The *R/W line would, at a certain point, go to a logic 1, indicating a read.
4. The external I/O controller logic would detect the CTS* and the *R/W and deduce that a read instruction was being executed by the CPU.
5. The controller logic would supply 8 bits of data on the data bus lines.
6. The CPU logic would strobe in the data from the data bus.

In fact, this operation would occur very rapidly, over a portion of one LDA instruction as shown in Fig. 16–3.

Notice that the controller does not use the address lines. All it needs to know is that it is being addressed, and this is apparent by the CTS* line (the controller's address is the address range $C000–$DFFF). We could have used either the CTS* or SCS* signals as the controller's address. In fact, the SCS* is probably better, as the SCS* defines a smaller range of addresses more suitable for an I/O device. In this case the controller's address would have been $FF40 through $FF5F. However, if we use the

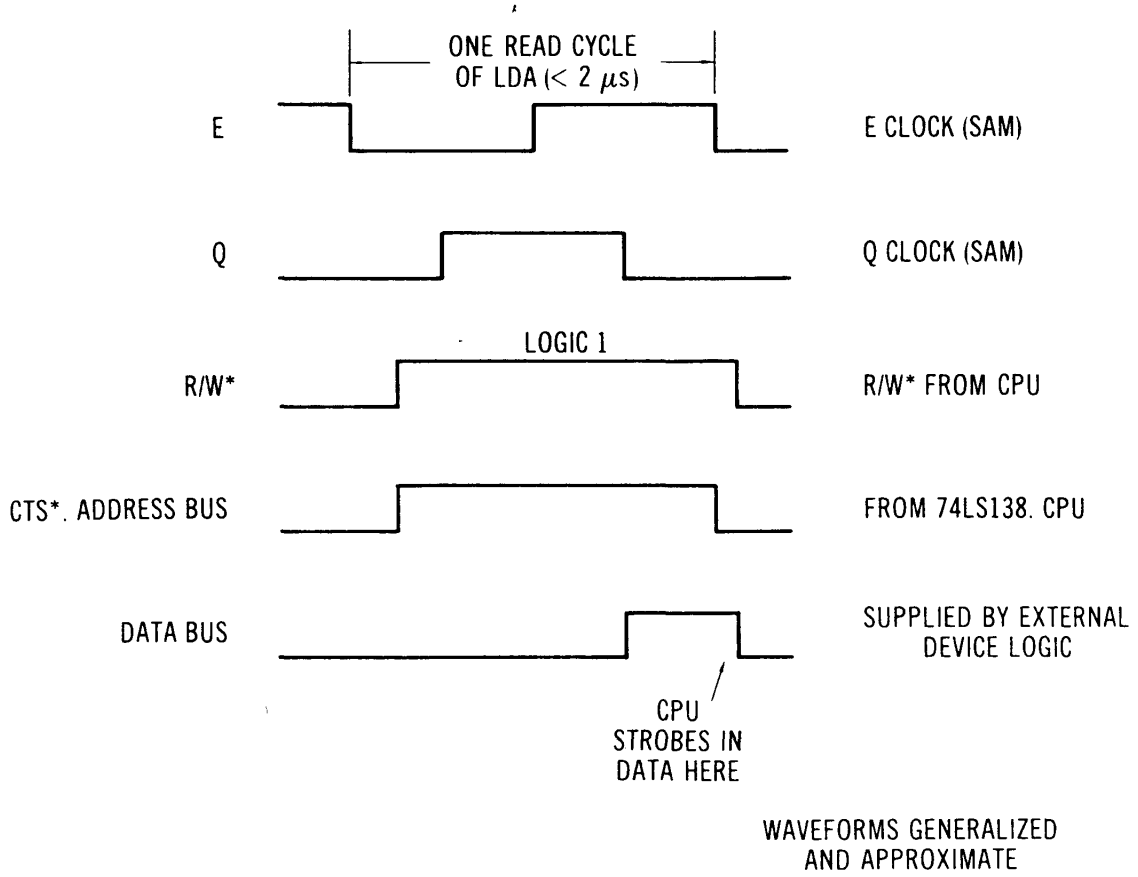


Fig. 16–3. Input timing.

SCS*, we must also AND in the E clock signal as this is not ANDed into SCS* as it is in CTS*.

If the controller had to pass many different types of data, it might well decode all or a portion of the address lines A15 through A0. It depends upon the application. A paper tape reader, for example, might use address $FF40 as the address for reading the next byte of data from paper tape and $FF41 as the address for reading the status of the paper tape (jammed, moving, etc.). It depends upon the complexity of the I/O device.

General Output Operation

In general, a write to an I/O device plugged into the ROM cartridge port would go as follows:

1. An STA $XXXX or POKE &HXXXX,V would be executed, where XXXX is the ROM cartridge port address of $C000 through $DFFF and V is the 8-bit value to be transferred.

2. The CTS* signal would become active and the address lines to the cartridge would contain the entire XXXX address.
3. The *R/W line would, at a certain point, go to a logic 0, indicating a write. The CPU would supply the 8 bits of data on the data bus lines.
4. The external I/O controller logic would detect the CTS* and *R/W and deduce that a write instruction was being executed by the CPU.
5. The controller logic would strobe in the data from the data bus.

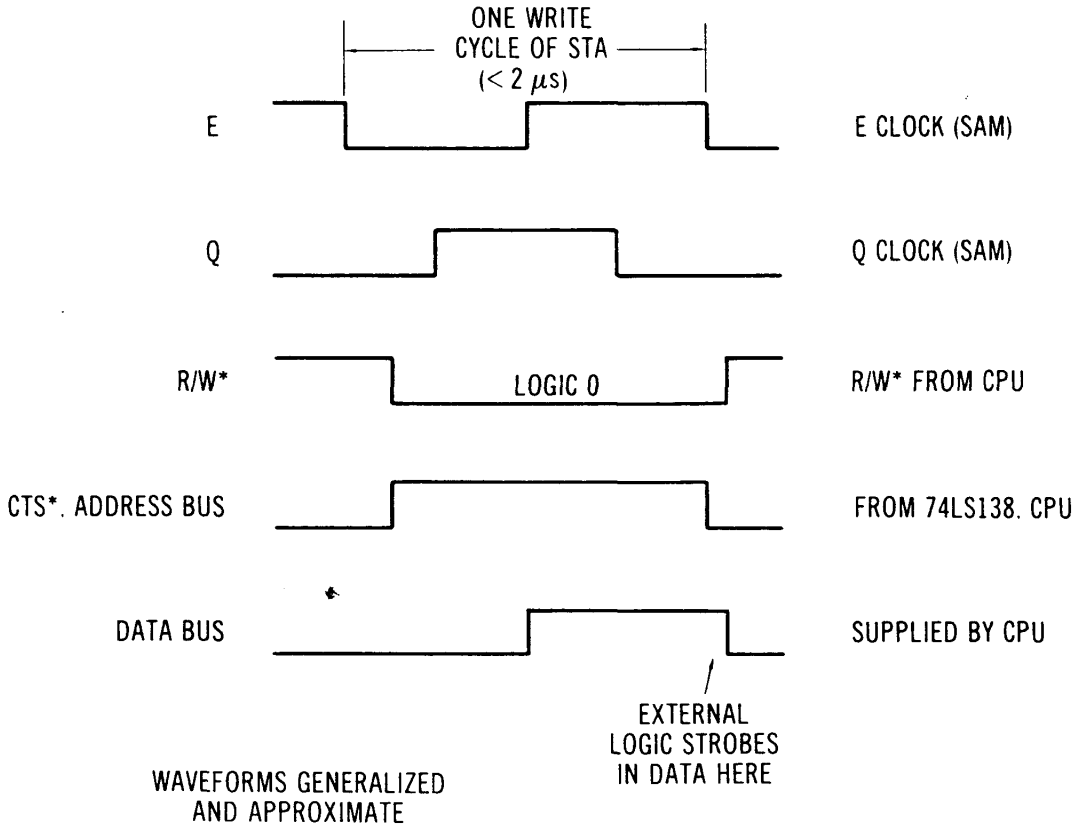


Fig. 16–4. Output timing.

Again, all of this would occur in the space of a single STA instruction (even if a POKE was involved). (See Fig. 16–4.) Again, additional address line decoding might be required. Again, it would be convenient to use the SCS* signal in place of the CTS*.

In the next chapter we look at how to connect to the system bus in the Color Computer with a general-purpose board.

chapter **17**

A General-Purpose I/O Board

The logic diagram of a general-purpose I/O board that plugs into the ROM cartridge connector is shown in Fig. 17–1. It operates according to the rules of the general input/output described in Chapter 16. The board is built around an Intel 8255 programmable peripheral interface (PPI) chip. It uses three 74LS240 *bus buffers* to provide higher current drive capability to 24 lines, 8 of which are inputs and 16 of which are outputs.

DESIGN OF THE BOARD

The 8255 is a general-purpose I/O device that operates in several modes. We've chosen the simplest mode for this application, the mode in which each of three sets of lines can be programmed to be inputs or outputs. In this case we've arbitrarily made the A and B lines the outputs and the C lines the inputs, although the sets could have been either inputs or outputs by simply outputting the proper control byte.

Many of the signals previously described are used in this design. The addresses of the device are $C000, $C001, $C002, and $C003. The CTS* signal enables the 8255 (CS is chip select), and the two address lines A1 and A0 choose the two lower address bits. The 8255 requires a write signal of 0 (WR) and a read signal of 0 (RD), so we've added some additional logic (74LS00) to provide the proper signal from the basic R/W* signal.

The 8255 is cleared by a RESET signal of logic 1; another section of the 74LS00 changes the active low RESET* from the 6809E to an active high signal.

The 24 lines go to three 74LS240 chips. These are octal buffers which provide up to 40 mA of sink current and invert the 8255 signal.

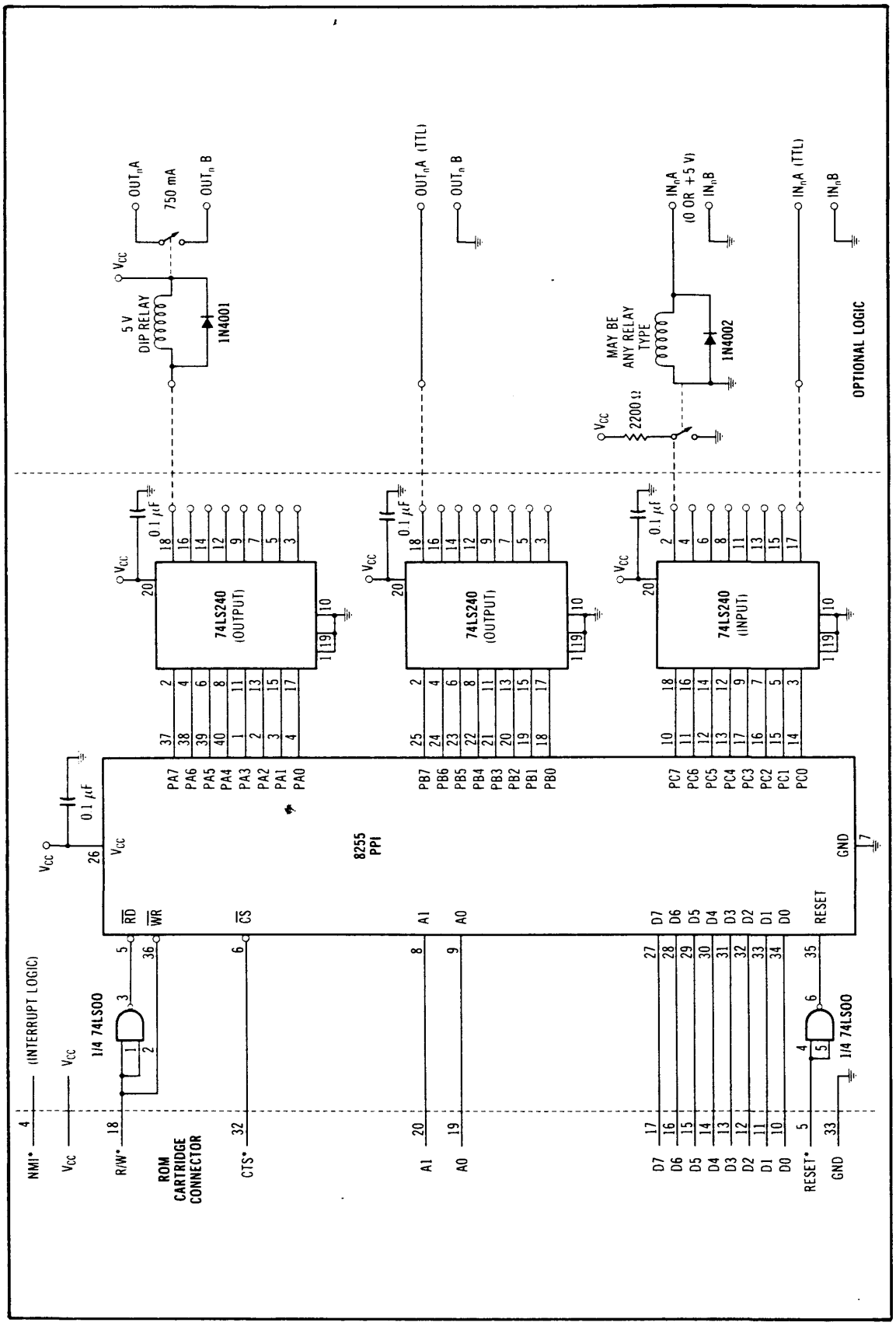


Fig. 17–1. General-purpose I/O board logic diagram.

SOFTWARE FOR THE GENERAL-PURPOSE BOARD

Programming the board is easy. First, the 8255 must be initialized to mode 0, the simplest I/O mode that it can use. This is done by outputting a value of 137 to address $C003 (the 8255 control register), either by a POKE 49155,137 or by an assembly language instruction. This initialization should be done on power-up or after every system reset.

To write out to port A or B, do a POKE 49152,V or a POKE 49153,V with V set to the 8-bit value for lines PX7 through PX0. The value will be latched into the 8255 and remain on the outputs until overwritten by a new value. To set lines PB7, PB6, and PB0, for example, do a

```
POKE 49153,193.
```

The outputs of the 74LS240 will be inverted. To read port C, do a PEEK 49154. The value will be returned as an 8-bit number, corresponding to lines PC7 through PC0. The state of the inputs from the 74LS240 are inverted.

USING THE GENERAL-PURPOSE BOARD

A small reed relay can be driven by a 74LS240 output. The maximum current required for the relay cannot exceed 40 mA. Radio Shack relays (275-228) were used in the prototype, drawing 22.5 mA. These relays will handle up to 750 mA (¾ A) on the contacts and can be used to drive a larger relay or small load at a local or remote location.

On the input side, the Radio Shack relay can be used in reverse. The contact closure pulls down a signal input from logic 1 to 0 as shown in Fig. 17–1. The control voltage can be 5 to 6 volts dc from a remote location.

The output side can also drive any other TTL logic, as long as the length of wire from the 74LS240 output to TTL input is kept shorter than several feet or so. Other devices, such as opto-isolators or solid-state relays, can be driven by the outputs to control virtually any device.

CONSTRUCTION OF THE GENERAL-PURPOSE I/O BOARD

The board is constructed on a Vector 3719-1 DIP plugboard. This board is not available at a Radio Shack store, but is probably one of the most popular prototype boards around. The Vector board comes with a

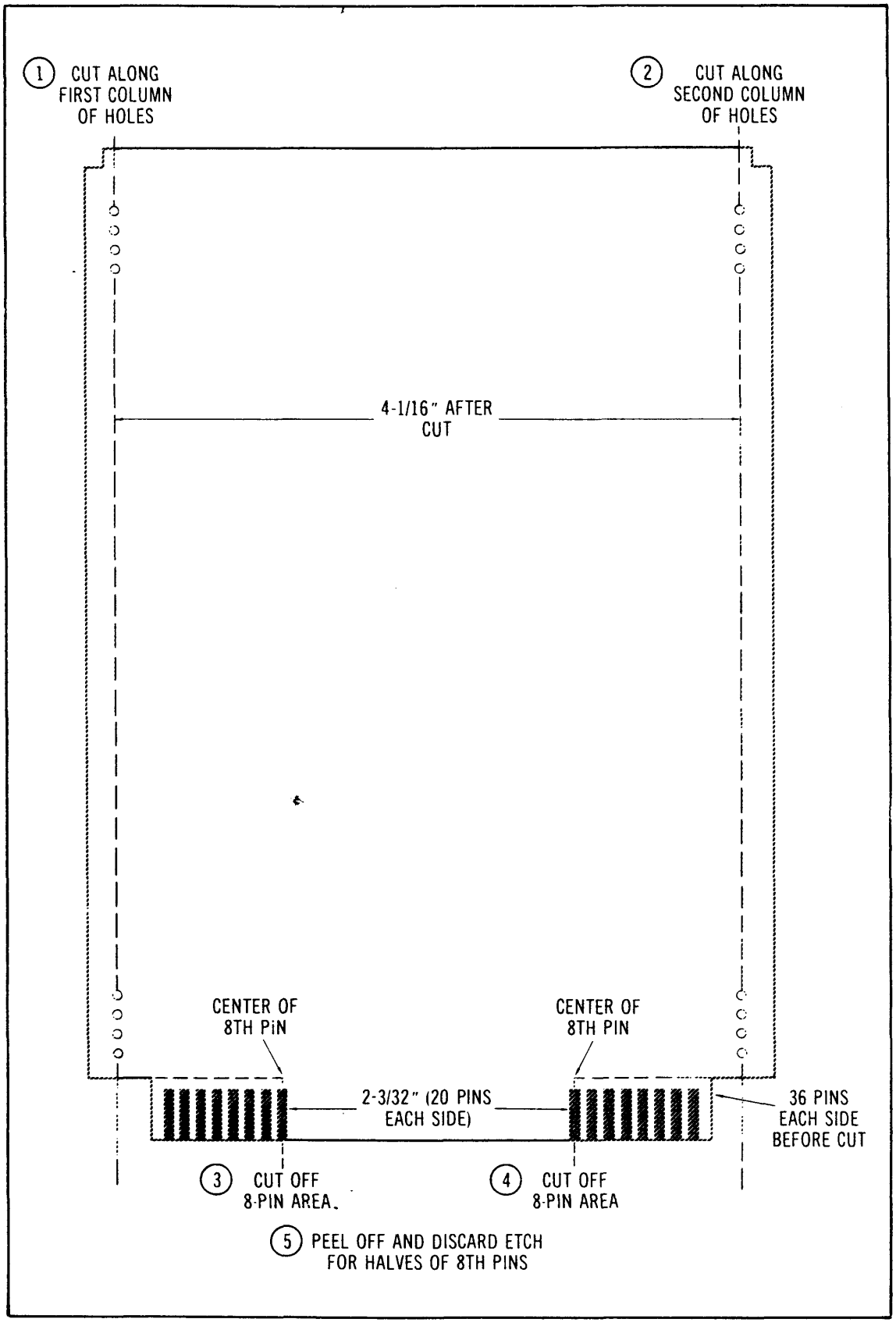


Fig. 17–2. Vector board cutting modification.

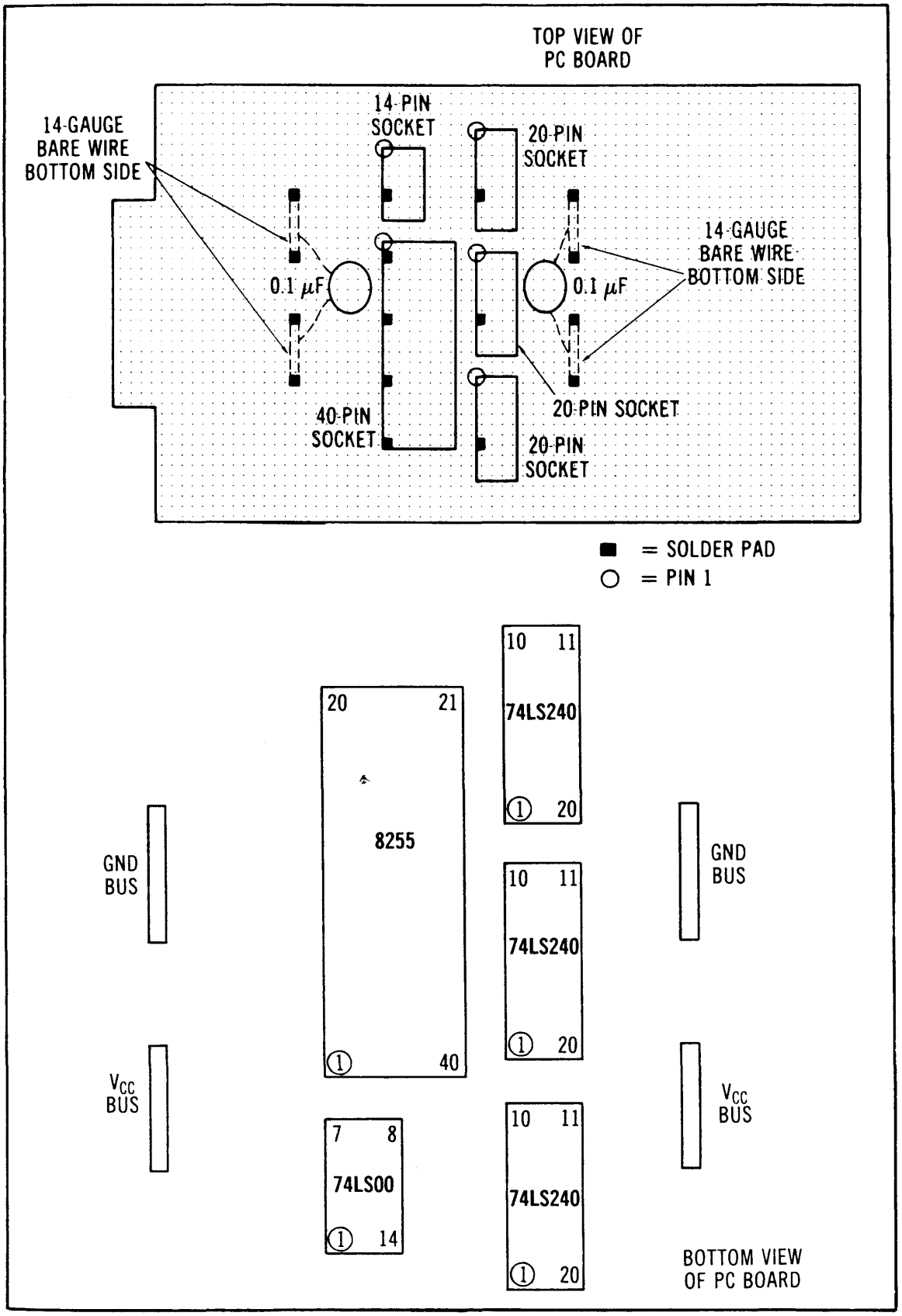


Fig. 17–3. Vector board socket mounting.

36-position edge connector,' as shown in Fig. 17–2. Cut the board as shown in the figure so that it will fit into the ROM cartridge hole and connector.

Once the board is cut, mount five IC sockets as shown in Fig. 17–3. At least one pin of each socket should be soldered to a copper pad on the board. Wiring will provide additional mechanical support for the sockets.

Four buses are constructed out of 14-gauge bare wire as shown in Fig. 17–3. Enlarge the holes slightly to pass the wire and solder the ends to the solder pads on the top side of the board. Two 0.1-μF disc capacitors are mounted between each set of buses.

After mounting the IC sockets and buses, wire the board as shown in Table 17–1. I used wire-wrap wire for all logic connections and larger

Table 17–1. Color Computer GPIO Board Wire List

From	To	Signal	From	To	Signal
CC-18	74LS00-1	R/W*	8255-3	74LS240-15	PA1
74LS00-1	74LS00-2	R/W*	8255-4	74LS240-17	PA0
74LS00-3	8255-5	$\overline{RD}$	8255-25	74LS240-2	PB7
74LS00-1	8255-36	$\overline{WR}$	8255-24	74LS240-4	PB6
CC-32	8255-6	CTS*/$\overline{CS}$	8255-23	74LS240-6	PB5
CC-20	8255-8	A1	8255-22	74LS240-8	PB4
CC-19	8255-9	A0	8255-21	74LS240-11	PB3
CC-17	8255-27	D7	8255-20	74LS240-13	PB2
CC-16	8255-28	D6	8255-19	74LS240-15	PB1
CC-15	8255-29	D5	8255-18	74LS240-17	PB0
CC-14	8255-30	D4	8255-10	74LS240-18	PC7
CC-13	8255-31	D3	8255-11	74LS240-16	PC6
CC-12	8255-32	D2	8255-12	74LS240-14	PC5
CC-11	8255-33	D1	8255-13	74LS240-12	PC4
CC-10	8255-34	D0	8255-17	74LS240-9	PC3
CC-5	74LS00-4	RESET*	8255-16	74LS240-7	PC2
74LS00-4	74LS00-5	RESET*	8255-15	74LS240-5	PC1
74LS00-6	8255-35	RESET*	8255-14	74LS240-3	PC0
CC-9	V_{cc} Bus	V_{cc}	74LS00-14	V_{cc} Bus	
CC-33	GND Bus	GND	8255-26	V_{cc} Bus	
8255-37	74LS240-2	PA7	74LS240-20 (3)	V_{cc} Bus	
8255-38	74LS240-4	PA6	74LS00-7	GND Bus	
8255-39	74LS240-6	PA5	8255-7	GND Bus	
8255-40	74LS240-8	PA4	74LS240-1 (3)	GND Bus	
8255-1	74LS240-11	PA3	74LS240-19 (3)	GND Bus	
8255-2	74LS240-13	PA2	74LS240-10 (3)	GND Bus	

Insert 0.1-μF capacitors between the two V_{cc} and GND Buses and between V_{cc} and GND pins on 5 IC Sockets.

gauge hookup wire for power and bus connections. The wire-wrap wire can be soldered to the IC socket pins, or if you wish you may wire-wrap the pins. Edge connector connections are soldered. Add five additional 0.1-μF disc capacitors between the V_{cc} and ground pins of each socket for noise immunity.

The pin position numbers for the 40 pins of the edge connector are shown in Fig. 17–4. The pin numbering for the board and Color Computer connector is shown in Fig. 17–5.

After wiring, check all connections by "buzzing out" the sockets with a continuity tester. Use two common pins to get into the IC socket holes as shown in Fig. 17–6.

Once the board connections have been checked out, you can add the protective cover. By a strange coincidence, the Color Computer ROM cartridge hole is almost an exact match for a common cassette tape case,

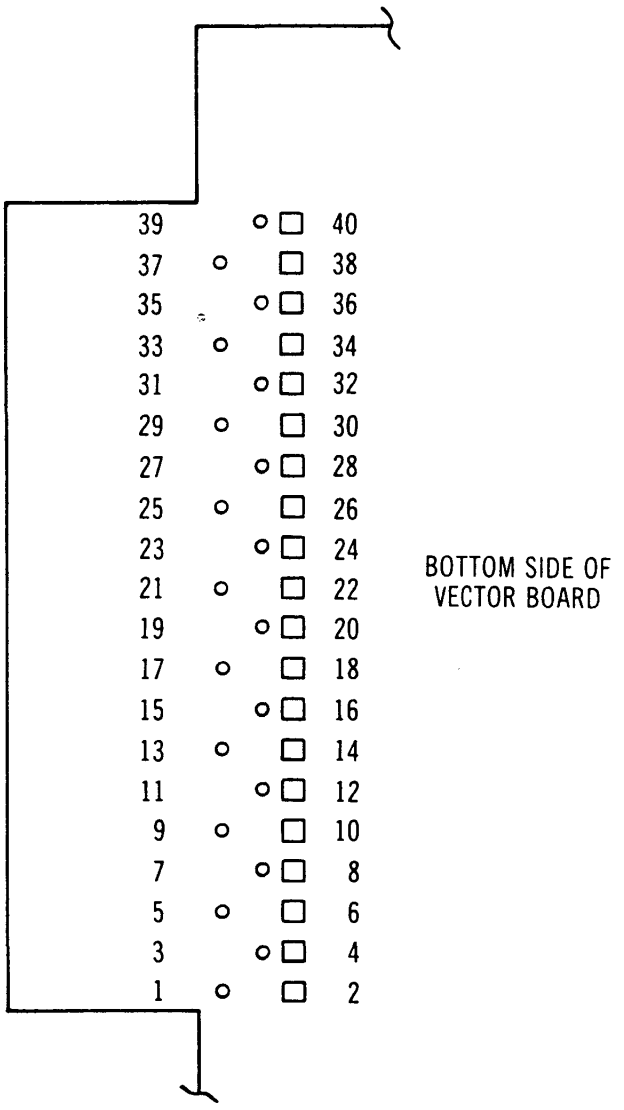


Fig. 17–4. Vector board pin numbering.

inserted widthwise. Two covers can be sacrificed to make a workable cover that will align the board properly in the ROM cartridge hole. See Fig. 17–7. The plastic on the case is easy to work with, and I found that a sharp X-Acto knife blade will cut through the plastic quite easily, albeit with a dozen or so passes. File off the cut edges for aesthetics. Add two screws, as shown in the sketch, to hold the cover in place.

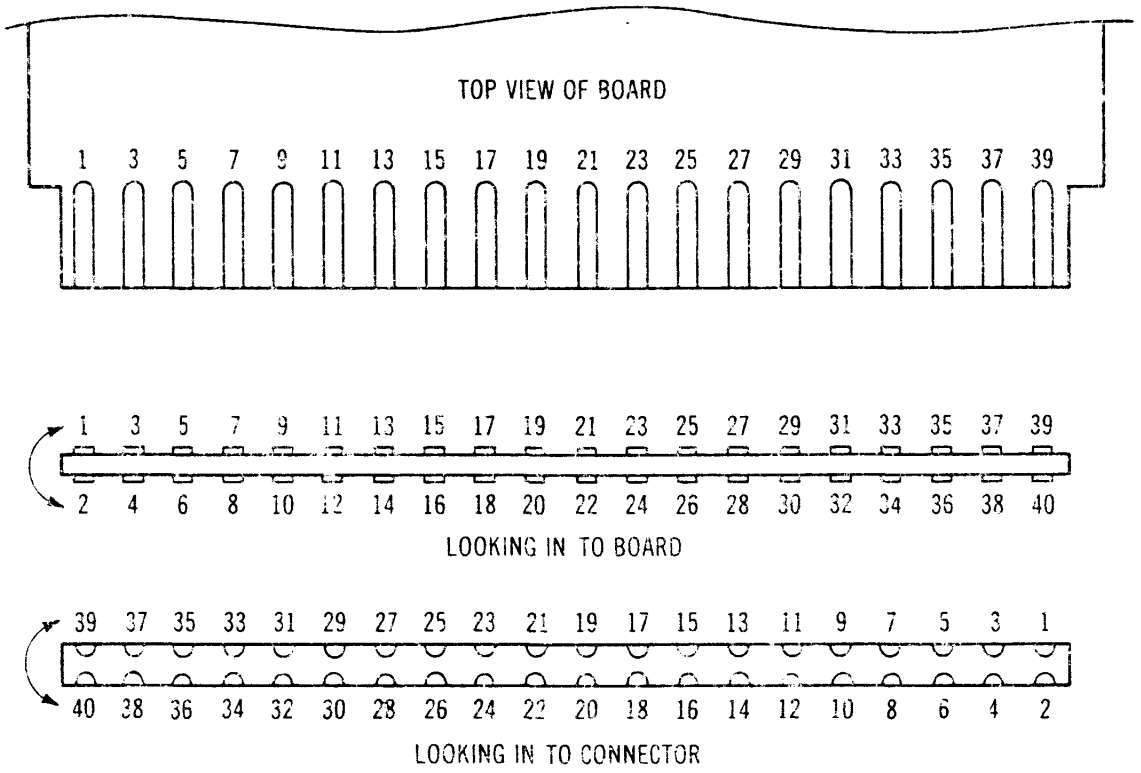


Fig. 17–5. Vector board and socket pin layout.

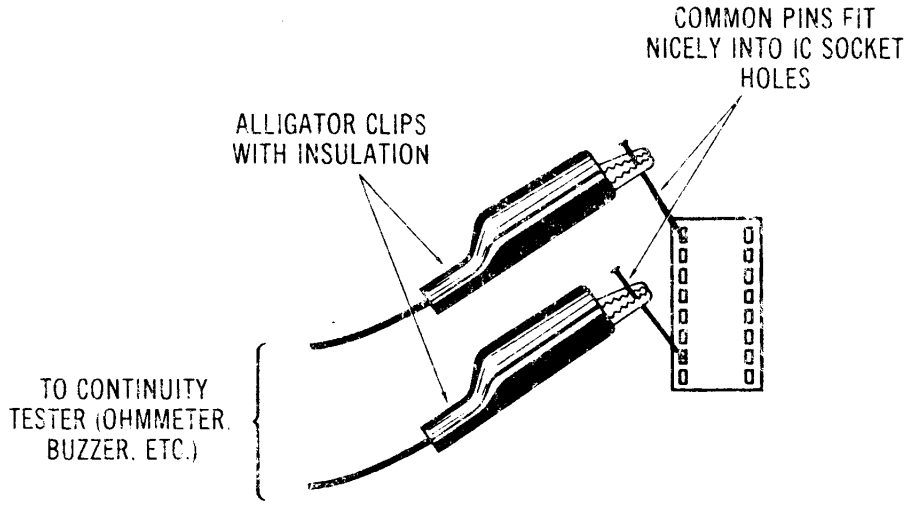


Fig. 17–6. Continuity testing technique.

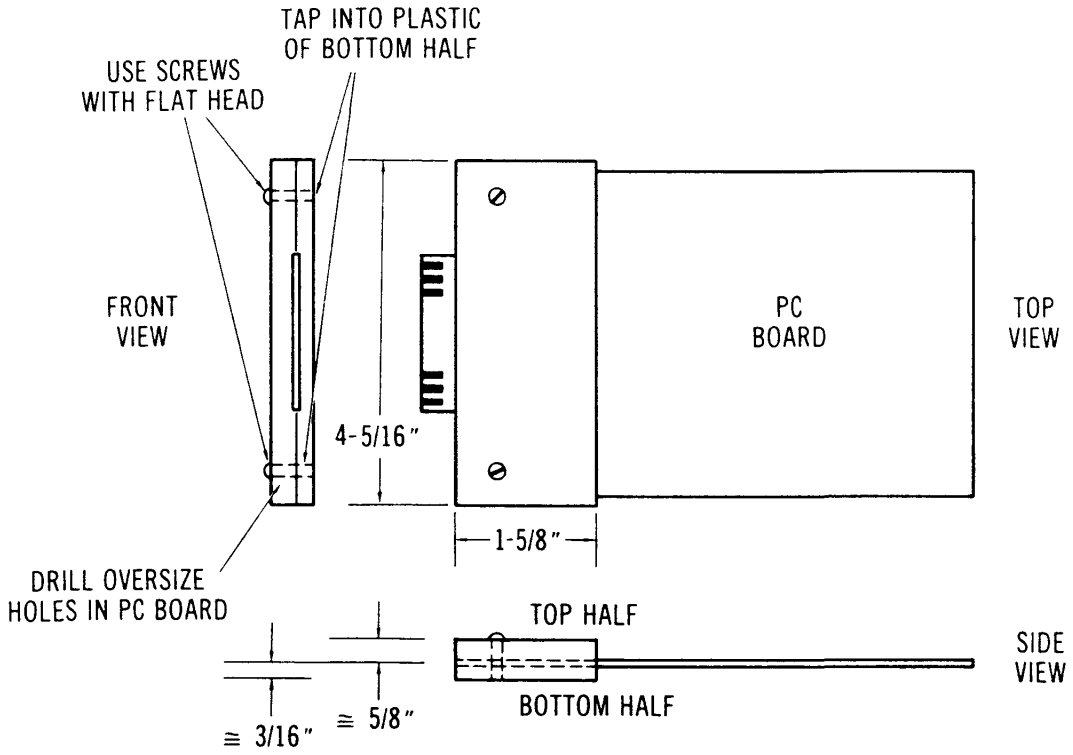


Fig. 17–7. Protective cover for the GPIO board.

```
100 ' TEST DRIVER FOR GENERAL PURPOSE I/O
110 POKE 49155,137
120 POKE 49152,0
130 POKE 49153,0
140 PRINT PEEK(49154)
150 IF INKEY$="" THEN GOTO 140
160 INPUT A,B
170 POKE 49152,A
180 POKE 49153,B
190 GOTO 140
1000 POKE 49155,137
1010 POKE 49152,0
1020 POKE 49153,0
1030 FOR I=0 TO 1000:NEXT I
1040 POKE 49152,255
1050 POKE 49153,255
1060 FOR I=0 TO 1000:NEXT I
1070 GOTO 1010
```

Fig. 17–8. Test driver program.

TESTING THE BOARD

Turn off the Color Computer and insert the board into the cartridge cutout. The pc board should go in easily and the cover should clear the sides with no binding.

Key in the program shown in Fig. 17–8. The first part of this program prints the input lines and allows entry of output data. The second part toggles the output lines on and off at a slow rate. You can test the outputs

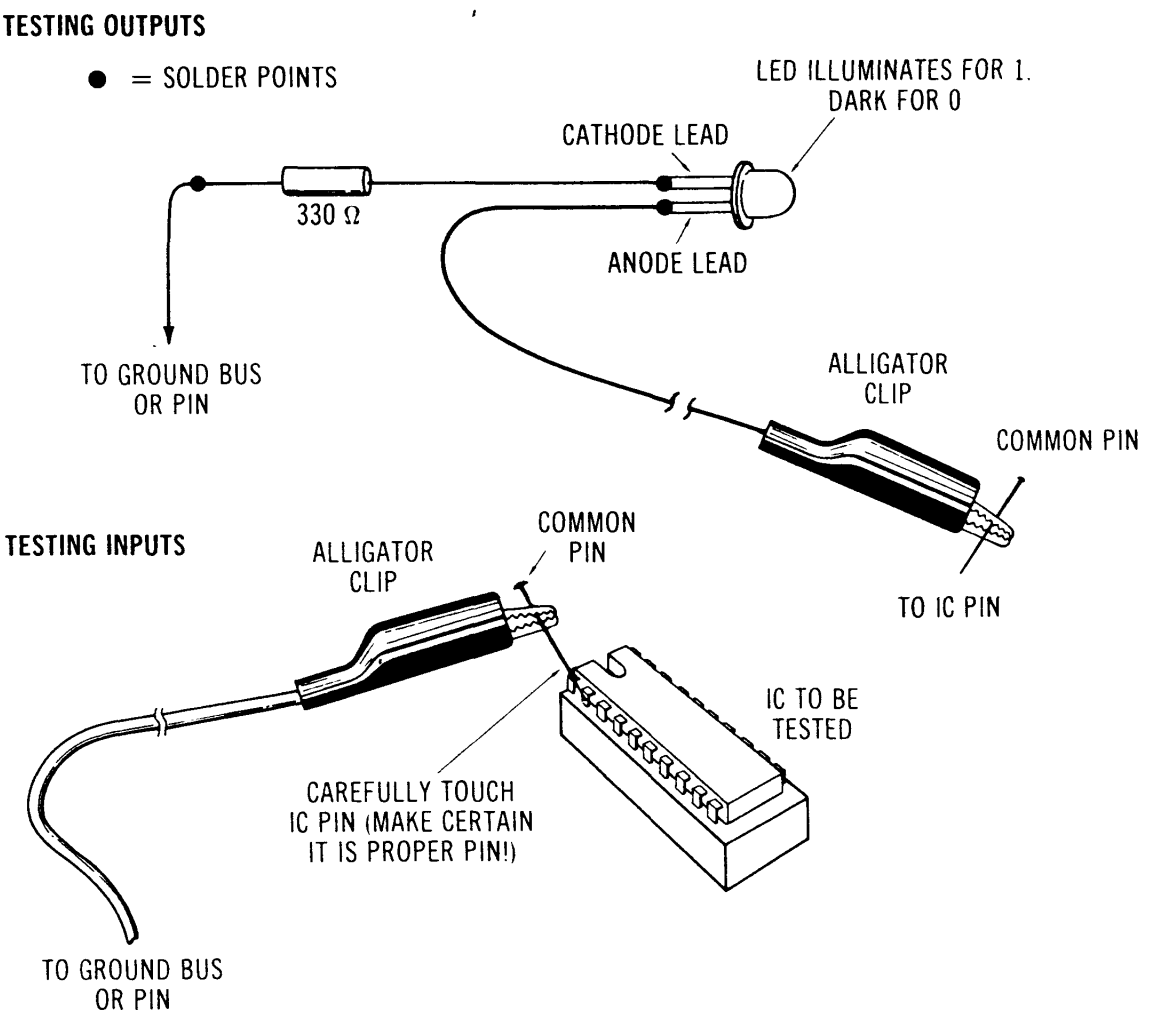


Fig. 17–9. Input and output lines test procedure.

by using a common LED with resistor as shown in Fig. 17–9. Inputs may be carefully grounded as shown in the figure; you should see the values change on the display as you vary the inputs. The relay connections for both input and output are shown in Fig. 17–10.

You may also want to read the last part of Chapter 19 to see how to interface a very similar version of the I/O board to an opto-isolator and LED display.

USING THE NMI INTERRUPT ON THE GPIO BOARD

The NMI* signal on pin 4 can be generated on the GPIO board quite easily by grounding pin 4. When this signal goes to ground (logic 0), it causes the following hardware actions:

1. The CPU completes the current instruction.
2. The CPU saves the contents of the program counter in the stack.
3. If the E bit is currently set in the condition codes, all of the CPU registers are saved in the stack. If the E bit is not set, only the condition codes are saved in the stack.
4. The CPU reads the NMI interrupt vector address from memory location $FFFC,D. It then transfers control to the address found at that location.

In the Color Computer, the NMI vector is actually at location $BFFC,D, and those locations normally hold address $0109. Locations $0109, $010A, and $010B would normally contain a jump instruction to the machine-language NMI interrupt processing routine in RAM mem-

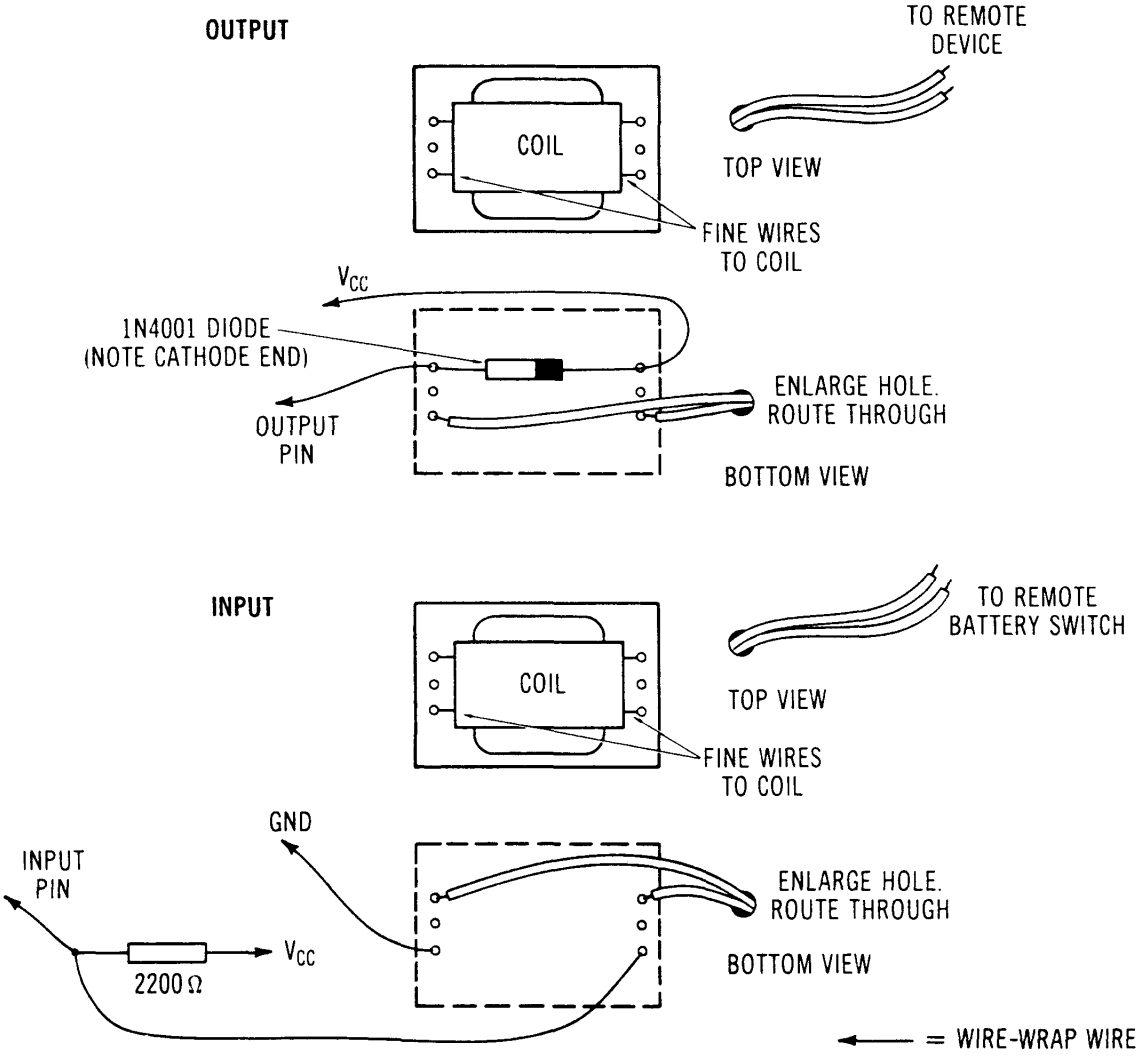


Fig. 17–10. Relay connections.

ory. The NMI interrupt processing routine would handle the NMI interrupt and then, as a last instruction, perform an RTI, or return from interrupt, instruction. This would return processing to the interruption point, which could be in the BASIC interpreter or in a machine-language program in RAM.

We can't provide a complete course in reentrant interrupt processing, but we can give you a flavor of what is involved in using the NMI. The program in Fig. 17–11 shows a simple BASIC program that POKEs a JMP $3FF0 instruction into locations $0109 through $01B, and an INC $3FFF and RTI instruction into locations $3FF0 through $3FF3. This short NMI interrupt processing program will increment location $3FFF for every NMI interrupt.

```
100 ' NMI INTERRUPT EXERCISOR
110 POKE &H109,126:POKE &H10A,63:POKE &H10B,240
120 POKE &H3FF0,124:POKE &H3FF1,63:POKE &H3FF2,255
130 POKE &H3FF3,59
140 POKE &H3FFF,0
150 PRINT PEEK(&H3FFF)
160 GOTO 150
```

Fig. 17–11. NMI interrupt program.

To see the effects of the NMI, connect a short piece of wire from pin 4, with the opposite end loose. Key in the program after first protecting high RAM memory by a CLEAR 200, &H3FEF. Run the BASIC program and ground the NMI pin by grounding the wire on the ground bus or ground pin.

You'll see the count in location $3FFF change rapidly. Each time it changes by one, an NMI interrupt has occurred and the NMI interrupt processing routine at location $3FF0 has been entered. The count changes by more than one because the connection to ground has not been debounced. Many momentary contacts have occurred in connecting the wire to ground, and each one has generated an NMI interrupt.

The NMI* input can indeed be used for real-time processing by providing a single pulse from a logic 1 to a logic 0, and by a relevant NMI processing routine in assembly language code. As the textbooks state, "This will be left up to the student as an exercise!"

In the next two chapters of this section we discuss the Models I and III system bus and show you how to build a similar type of input/output board for these two systems.

chapter **18**

The Models I and III System Bus

In this chapter we describe a counterpart to the general-purpose input/output board for the Radio Shack Color Computer. It's a general-purpose input/output board for the Models I and III. As I imagine that few have both a Color Computer *and* a Model I or III, I'll give you the details on the logic of the board, even though it is very similar to the Color Computer version. We also describe the internal workings of the Models I and III system bus, which are quite a bit different from the Color Computer bus. There are also slight differences between the Models I and III buses insofar as connecting external input/output, and we include notes on that too.

THE SYSTEM BUS: AN OFFSPRING OF THE Z-80

It's true, the Models I and III system bus is very much related to the Z-80 microprocessor signals, although Zilog might not speak of it in mixed company. To describe the Models I and III system bus, therefore, I've got to start with the Z-80 signals. Fig. 18–1 shows a general block diagram of the Model I system bus. We'll talk about the Model I bus and then describe the Model III bus, which has some embellishments.

Model I Bus

The Z-80 has 16 address lines, labeled A15 through A0, most significant to least significant. The address lines are used to address RAM (random access memory), ROM (read only memory), and input/output devices. As there are 16 lines, addresses of 0000000000000000 (0)

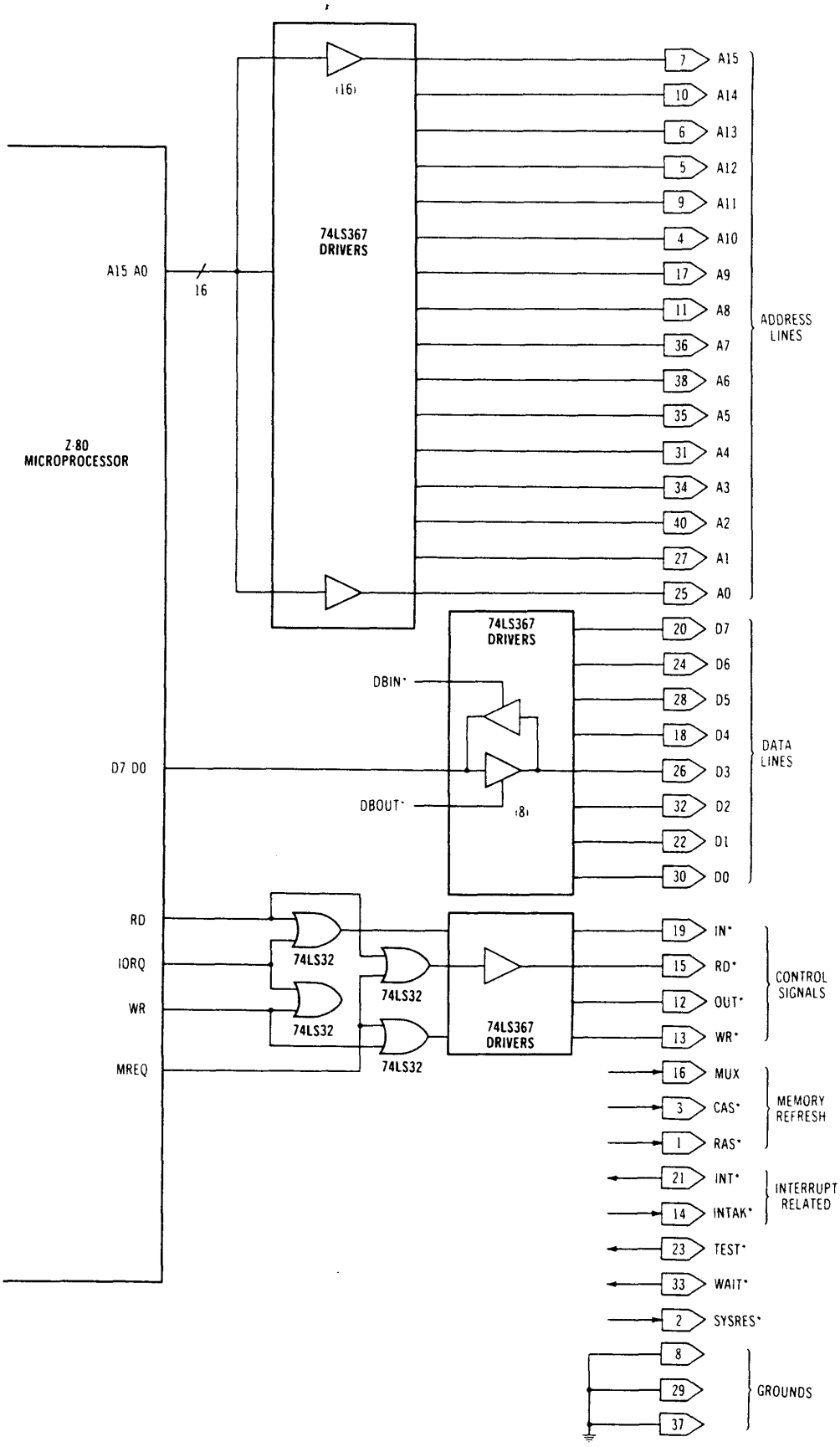


Fig. 18–1. Model I system bus block diagram.

through 1111111111111111 (65,535) may be specified, allowing the Z-80 to address 64K of memory and 256 separate input/output devices.

Perhaps we'd better explain that the Z-80 uses both memory-mapped and input/output mapped I/Os. Input/output mapped I/O means that the Z-80 has separate instructions (IN and OUT) for input/output, allowing all of the 64K addresses to be used for memory addresses if the system designer chooses. Input/output is specified by certain control signals that inform an external I/O device that an IN or OUT machine-language instruction is being executed rather than a memory-reference type instruction.

Memory-mapped I/O is used in such microprocessors as the 6502 and 6809E, where a portion of the 64K addressing space is dedicated to I/O addresses, and there are no control signals that indicate that input/output is being performed. From the standpoint of the CPU, an input/output operation looks just like reading or writing data into memory. Of course, the system designer allocates certain addresses to memory and certain addresses to input/output devices, so that the program is always aware that memory or I/O is being done.

The Models I and III were designed with both memory-mapped I/O and input/output mapped I/O as shown in Fig. 18–2. The keyboard, for example, is memory-mapped at locations 3801H through 3880H; the cassette is I/O mapped at location 0FFH. I/O-mapped operations were separate IN or OUT instructions with I/O addresses of 00000000 (0) through 11111111 (255); these I/O addresses are completely separate from the 64K memory addresses.

The 16 address lines are buffered by 74LS367s to provide higher fan-out, and they go out to all parts of the Model I, including the external 40-pin connector for the system bus. The address lines are unidirectional, that is, they are only outputs from the Z-80.

Back to the Z-80 signals. The next largest set of signals is the data bus, Z-80 signals D7 through D0, most significant through least significant. The data bus is used to pass all data going between Z-80 registers and memory and between Z-80 registers and input/output devices.

Unlike the address bus, the data bus is bidirectional to permit 8-bit transfers in both directions. Because of its bidirectional nature, two sets of 74LS367 bus drivers are used, one controlled by a data bus out signal DBOUT*, and the other controlled by a data bus in signal DBIN*. (The asterisk is used in all Models I and III signals to indicate active low.) The data bus lines also go to the 40-pin system bus connector on the Model I.

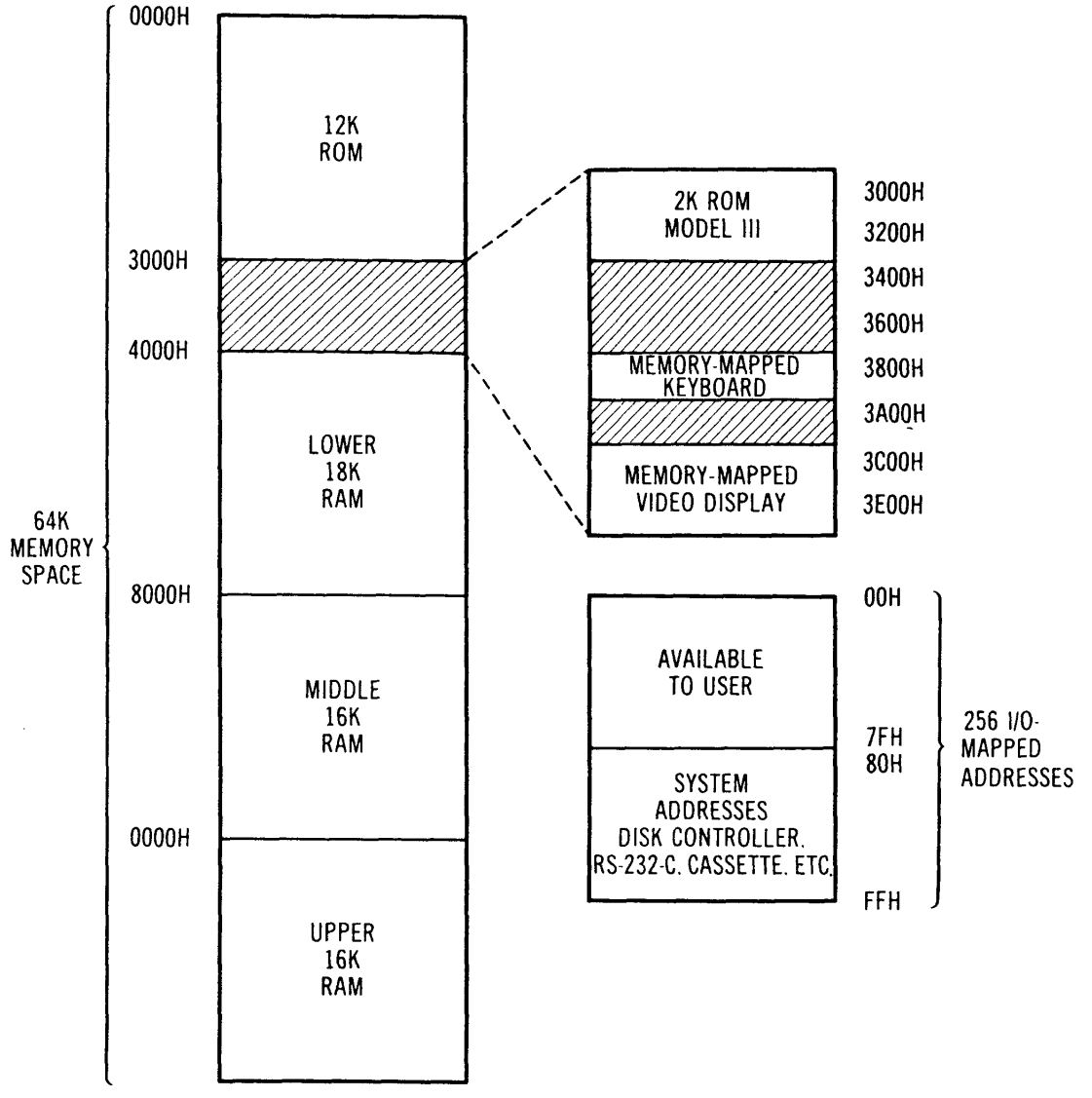


Fig. 18–2. Models I and III memory map.

Looking at the Z-80 once again, we find two sets of control signals, $\overline{\text{MREQ}}$/$\overline{\text{IORQ}}$ and WR/$\overline{\text{RD}}$. Signal $\overline{\text{MREQ}}$ (the bar indicates active low) is used to indicate that a memory operation is in effect and that there is a valid memory address on address lines A15 through A0. $\overline{\text{MREQ}}$ is used with $\overline{\text{RD}}$ to read data from ROM or RAM and with WR to write data to ROM or RAM. $\overline{\text{MREQ}}$ is also used for memory-mapped input/output devices, such as the keyboard.

Signal $\overline{\text{IORQ}}$ is used to indicate that an IN or OUT instruction is being executed and that there is an I/O address on address lines D7 through D0. $\overline{\text{IORQ}}$ is used with RD to read data into the Z-80 A register from an external I/O device, and to write data from the A register to an external

I/O device. IORQ is the primary signal used for all types of I/O mapped input/output.

The signals brought out on the system I/O bus, however, are not MREQ, IORQ, WR, and RD. The RD* signal is active (low) when RD and MREQ are active (memory read). The WR* signal is active (low) when WR and MREQ are active (memory write). The IN* signal is active (low) when IORQ and RD are active (input). The OUT* signal is active (low) when IORQ and WR are active (output). These signals are partially encoded, then, for external memory or input/output.

Other memory-related signals brought out on the system bus for memory refresh are MUX, CAS*, and RAS*. These three signals control memory refresh for the dynamic RAMs used in the Models I and III. Since they're not used in external I/O, I'll avoid any further details.

Other signals brought out on the Model I bus include INT*, INTAK*, TEST*, WAIT*, SYSRES*. INT* is an input, and provides an external I/O interrupt. INTAK* is an interrupt acknowledge signal indicating that the Z-80 received the interrupt. TEST* is a signal that disables all data bus, address bus, and control signals; it is not ordinarily used in Model I operations. WAIT* is an input signal used to interface slow memory or input/output devices, and is not ordinarily used. It dates from the time when memories were significantly slower than the microprocessor. SYSRES* is an output signal indicating power-up or reset (by the RESET button). It, like all of the signals with an asterisk suffix, is active low.

Model III System Bus

Fig. 18-3 shows a general block diagram of the Model III system bus. The Model III system bus differs from the Model I system bus in that it is more isolated from the internal CPU signals. Only 8 address lines are brought out, and a special enable signal gates the data, address, and control lines to the outside world.

The main enable signal is ENEXTIO*, the enable external I/O. This signal is generated by one bit of a 5-bit latch with address 0ECH. When this bit is a 1, signal ENEXTIO* goes low, enabling the 74LS245 (XDB7 through XDB0), the 74LS244 (XA7 through XA0), and the 74LS367 (control lines). If the ENEXTIO bit is a 0, all of these lines are in the high-impedance (disconnected) state.

The XIORQ*, XM1*, IOBUSWAIT*, XRESET*, XOUT*, and XIN*

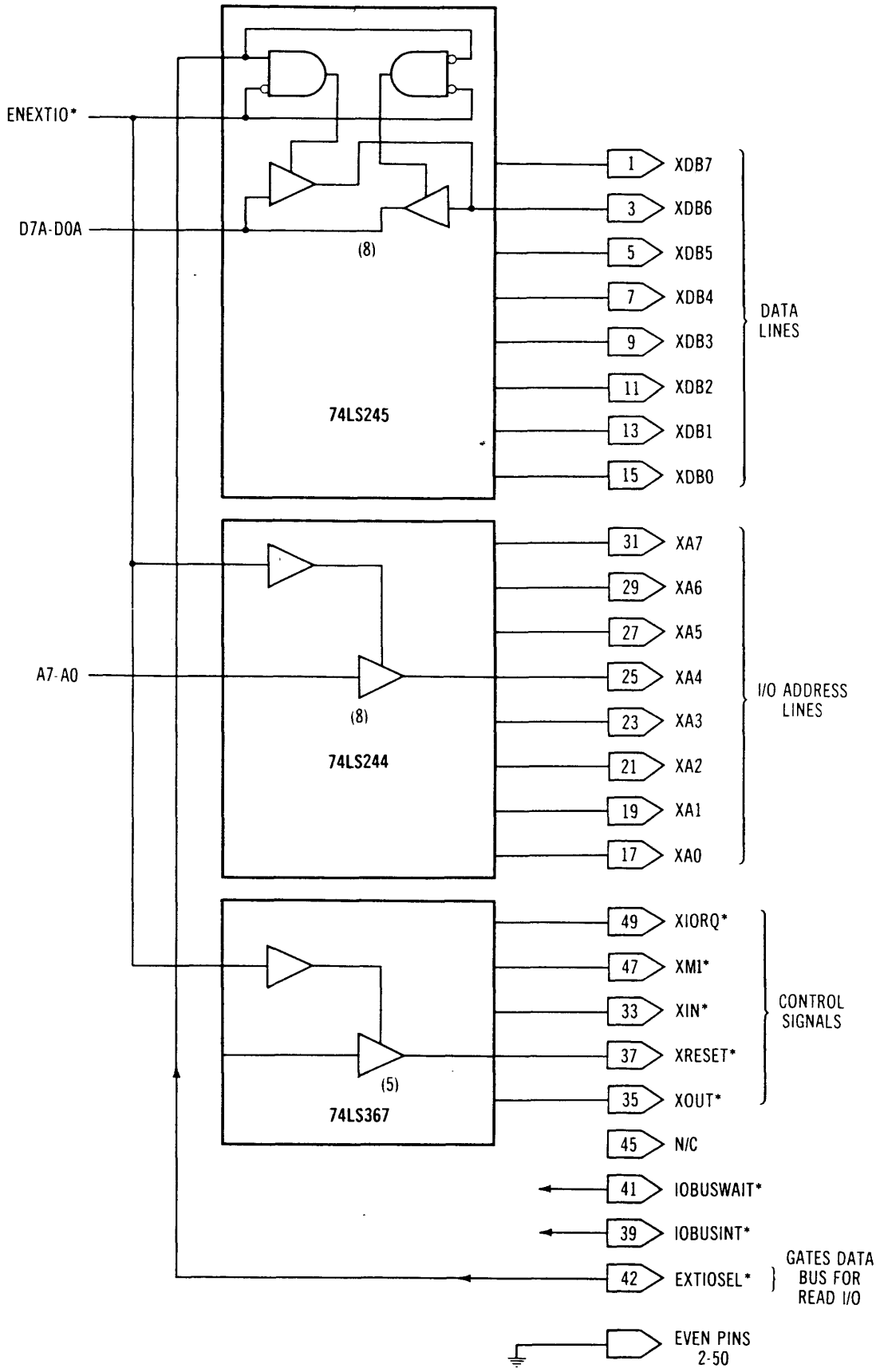


Fig. 18–3. Model III system bus block diagram.

control lines have the same functions as their Model I counterparts, WAIT*, SYSRES*, OUT*, and IN*. (IORQ* and M1* replace the encoded INTACK*.)

The IOBUSINT* is similar to INT* in the Model I, except that an enable signal, ENIOBUSINT (enable I/O bus interrupt), is used to control when an external interrupt will be recognized from the outside world. ENIOBUSINT is made active by writing a 1 to bit 3 of address 0E0H. We won't discuss the external interrupts in this chapter. The 8 address lines are also logically equivalent to their Model I counterparts; they are XA7 through XA0.

The 8 data bus lines XDB7 through XDB0 have a slightly different gating scheme on the Model I than they do on the Model III. Instead of two sets of buffers enabled by RD* or WR*, as in the Model I, there is one bus driver, a 74LS245. The main enable signal used for this chip is ENEXTIO*, which I've already mentioned. Also involved, though, is signal EXTIOSEL*. When EXTIOSEL* is a 1, the 74LS245 routes lines D7A–D0A to the external bus connector. When EXTIOSEL* is a 0, the 74LS245 routes lines from the external bus to lines D7A–D0A. EXTIOSEL* is normally high, so that writes to an external I/O device can be made by simply turning on ENEXTIO* (address 0ECH) and doing an OUT instruction (or a BASIC OUT). If a read of an external device is to be done, however, external logic must put the "bring down" signal EXTIOSEL* at the proper time.

GENERAL SCHEME FOR EXTERNAL I/O

The general scheme for the Model I external I/O is fairly simple.

Output Operation

The procedure for a write of 8 bits to an external device goes like this:

1. The 8-bit value to be written (0 through 255) is put into the A register in the Z-80.
2. A machine-language OUT instruction with an address of 0 through 255 is executed.

The equivalent in BASIC is

```
100 OUT XX,V
```

where XX is the I/O address and V is the value of 0 through 255.

Executing the machine language or BASIC OUT puts the address of the I/O device on address lines A7 through A0 and enables the OUT* signal. The external I/O device decodes the address lines when it receives the OUT* signal. If it recognizes its address, it strobes in the data, which is present on the data bus lines. The entire process is shown in Fig. 18-4.

Output for the Model III is identical, except that the external I/O lines

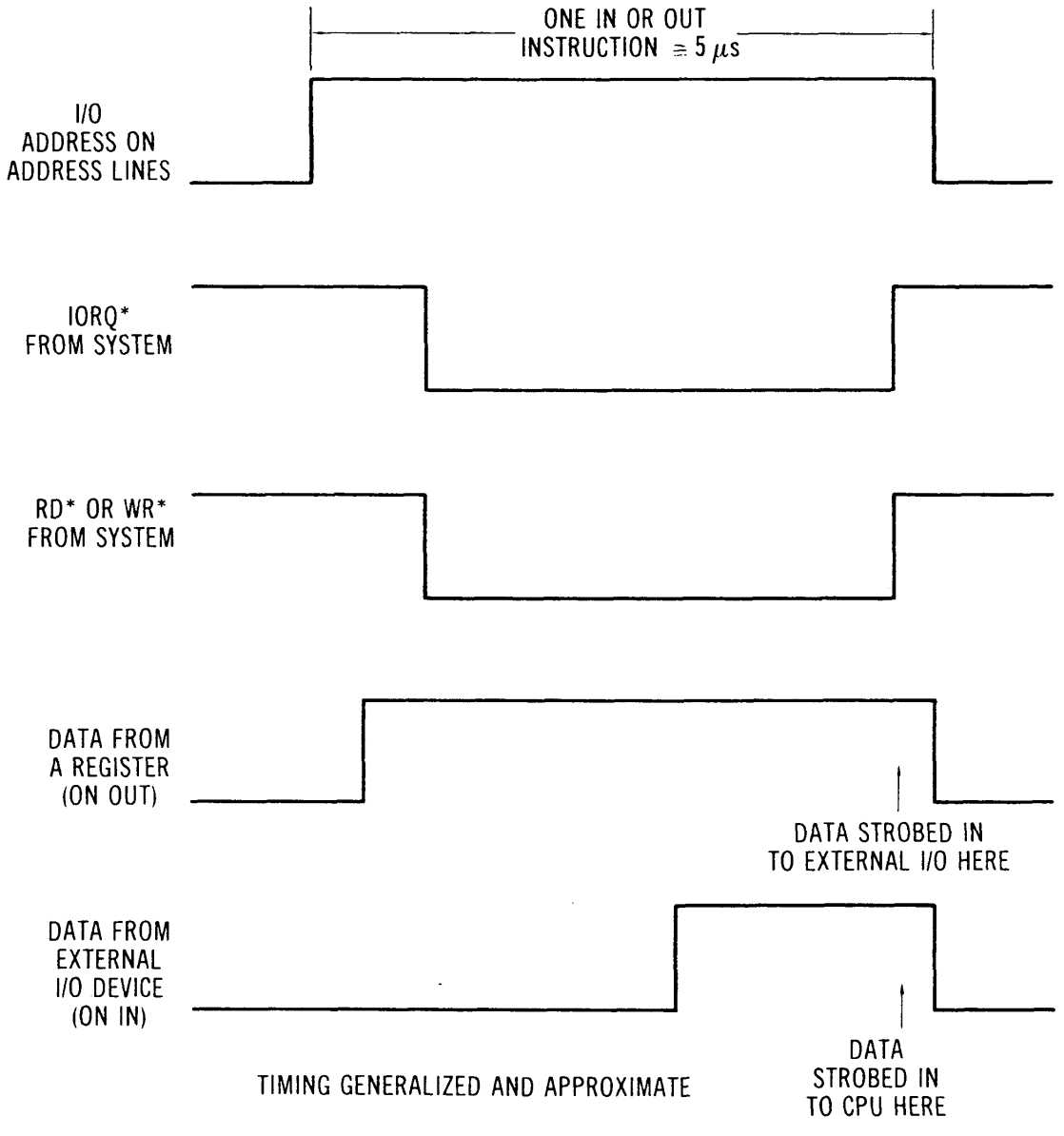


Fig. 18–4. Input/output timing, Models I and III.

must first be enabled by setting bit 4 of address 0ECH to enable signal ENEXTIO.

Input Operation

Input for the Model I goes like this:

1. A machine-language IN or a BASIC INP instruction is executed, with an address of 0 through 255.
2. Data from the I/O device is read into the A register, or into the BASIC variable specified (the BASIC equivalent is 100 A=INP(XX)).

When the machine-language IN or BASIC INP is executed, the address of the I/O device is sent to the address bus lines A7 through A0. Signal OUT* goes low, indicating to the external I/O logic that an I/O address is present on the address lines. If the address is decoded as the address of the I/O device, it responds by gating the 8 bits of data onto the data bus.

Input for the Model III is identical, except that the external I/O logic must also "bring down" signal EXTIOSEL* so that the 74LS245 bus driver switches direction, routing the data to the CPU.

I/O Addresses

In the above description, I've talked about an I/O address. In fact, the I/O address used must be in the range of 0 through 127, as both the Models I and Model III use I/O addresses in the higher range (128 through 255 or 80H through 0FFH) as dedicated system addresses. There are many addresses not used in the higher range, but it's prudent to stay in the nonconflicting 00H through 7FH range for external I/O and can be done without full address decoding logic.

In the next chapter we show you how to interface to the Models I and III system bus signals described above with a general-purpose input/output board.

chapter **19**

A General-Purpose I/O Board for Models I and III

The circuit shown in Fig. 19–1 is a general-purpose I/O board that connects to the Model I or III system bus. It provides 24 input/output lines that can be connected to be either inputs or outputs. The lines can be used to drive relays for input or output as shown in the figure, can implement digital-to-analog or analog-to-digital converters, or can be used for a variety of other applications. In this chapter we describe how the circuit works, give you some construction hints, and then show you typical uses in driving an LED display and detecting a remote input.

HOW THE GPIO WORKS

The GPIO board uses an Intel 8255 programmable peripheral interface. This chip is essentially a programmable controller. The mode that I use in this implementation connects lines PA7 through PA0 as outputs, lines PB7 through PB0 as outputs, and lines PC7 through PC0 as inputs.

The interface to the Model I or III consists of the eight data bus lines, three address lines, the IN* and OUT* lines, and, in the case of the Model III, the EXTIOSEL* line. The IOBUSINT* line is also implemented, but isn't involved in this application.

The address of the GPIO is any four sets of addresses in the "lower" I/O address range of 0 through 127. For convenience, you can look upon the addresses as 0, 1, 2, and 3, but the board will respond to any address with bit 7 equal to 0; address 01111100, for example, will be decoded as the same as address 0.

The address of the latch associated with the PA lines is address 0. Outputting data to address 0 will store the data pattern in the output and

set the lines accordingly. The address of the latch associated with the PB lines is address 1. An identical type of output can be done for this latch. The address of the lines associated with the PC lines is address 2. Inputting data from this address will read the state of the eight lines.

The last address of the GPIO is address 3 (XXXXXX11 address). This is the address of an internal control register in the 8255. Outputting a control word to address 3 "sets up" the 8255 to the input/output configuration desired. Outputting a decimal 137 will set the 8255 to the configuration I've described. This control word output need be done only once, at the beginning of any power-up sequence.

The normal sequence of events for using the GPIO is shown in the following:

```
100 'BASIC DRIVER FOR GPIO
110 OUT 236,16        'Model III only
120 OUT 3,137         'set up 8255
130 OUT 0,XX          'output to PA7–PA0
140 OUT 1,XX          'output to PB7–PB0
150 A=INP(2)          'read PC7–PC0
```

The first command sets the Model III EXTIOSEL*. An important point: This command must be done at the start of any entry to a BASIC program and after any CLS command. When in doubt, issue another EXTIOSEL*! The OUT 3,137 sets the 8255 to the proper input/output configuration. The next two commands output a byte to the two output ports. The next command reads in the configuration of the PC7 through PC0 lines.

The 8255 lines are connected to three 74LS240 line driver chips. These chips provide up to 10 mA of source (+5 V) current, or 40 mA of sink (0 V) current. The top two chips are connected as output drivers and the bottom is an input driver.

GPIO CONSTRUCTION

The board is assembled on a Radio Shack prototype board (276-154). This is a board with a 44-pin connector on one end that mates with a Radio Shack plug (276-1551). The board will be identical for both the Models I and III versions, but the cabling for the plug will be different.

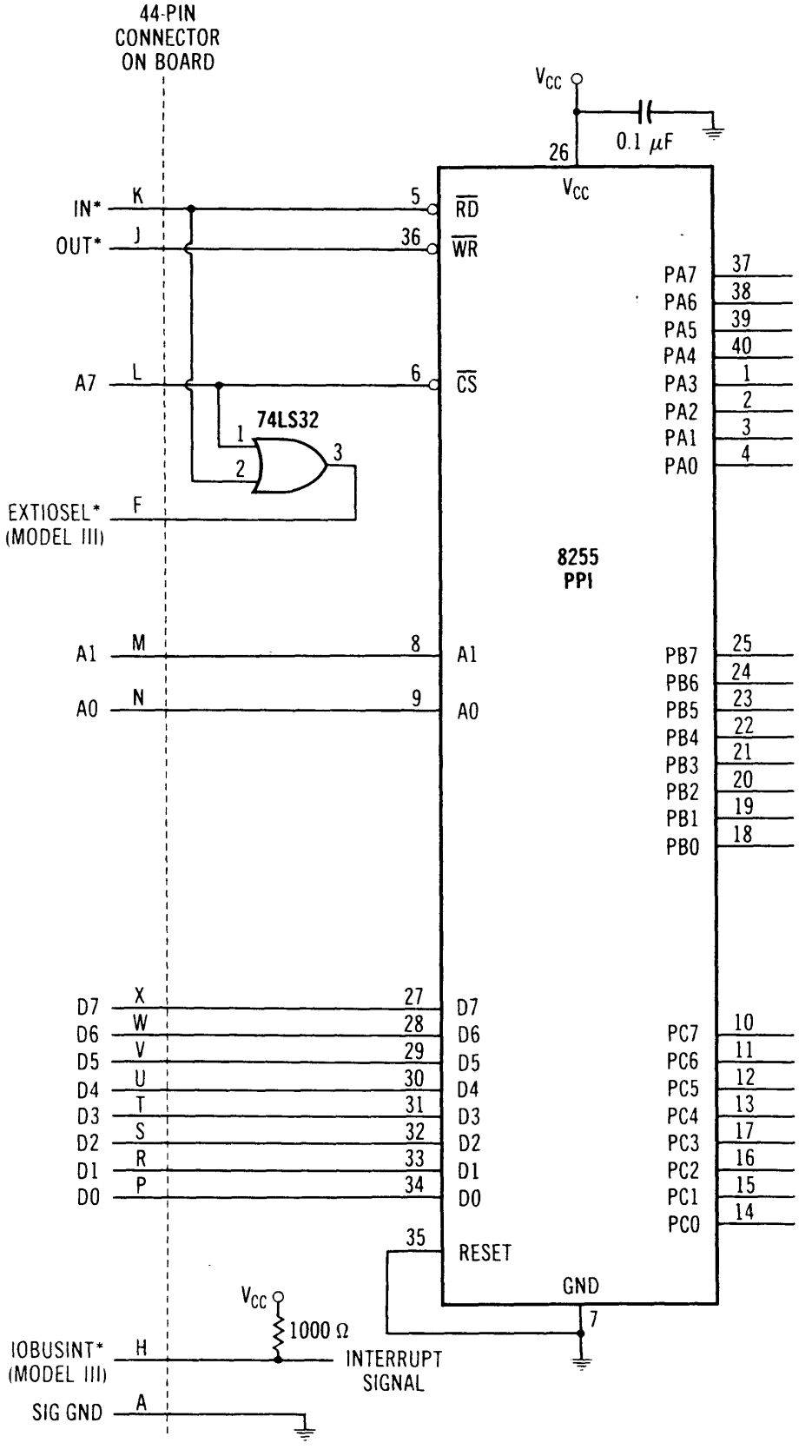
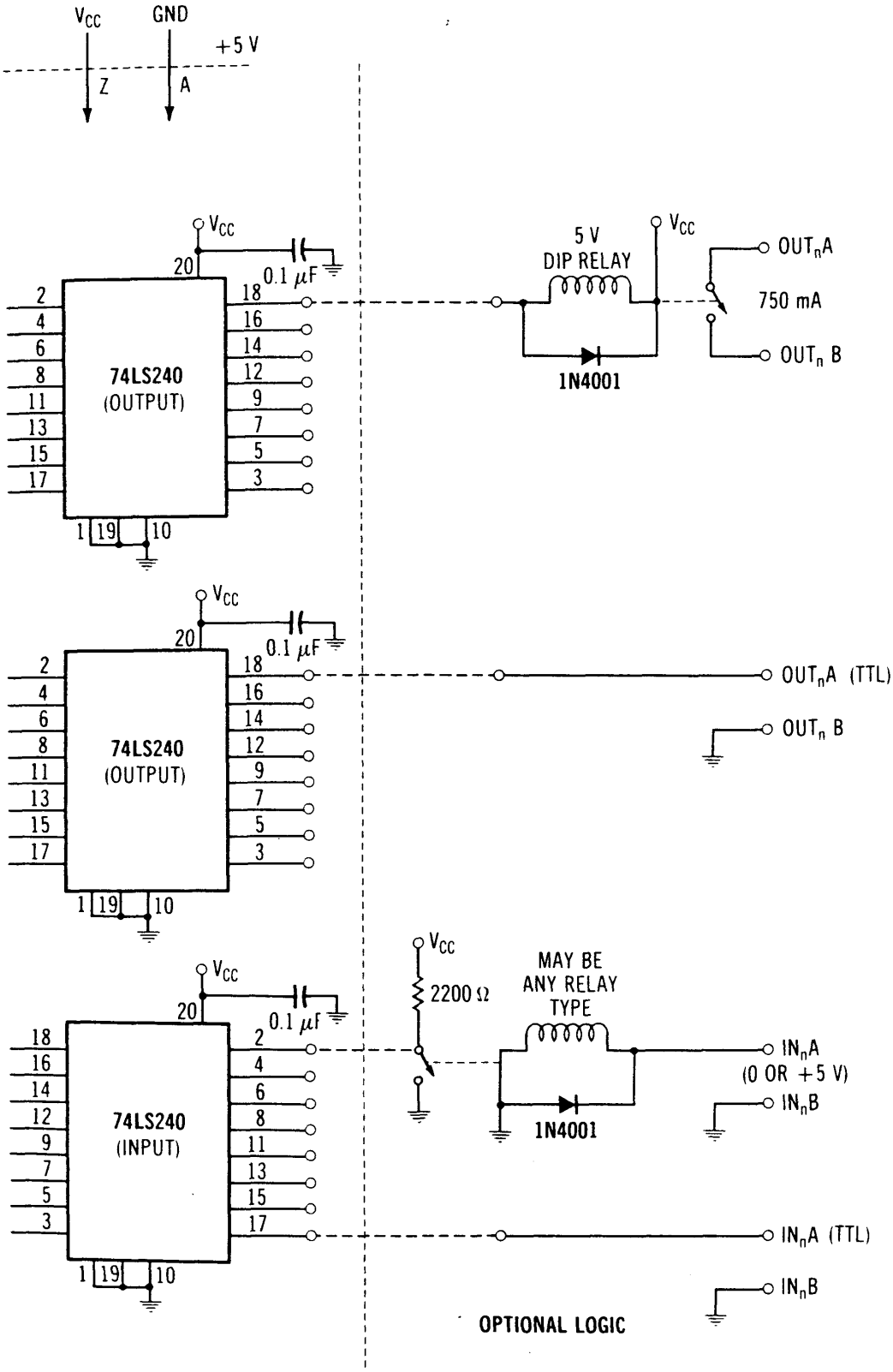


Fig. 19–1. General-purpose I/O board



logic diagram for Models I and III.

Socket Mounting and Wiring

Mount five sockets on the board, as shown in Fig. 19–2. We used wire-wrap sockets for this version. You may use solder-type sockets if you prefer, as there are not a great many interconnections. The sockets should straddle the two etches that represent the V_{cc} (+5 V) and GND buses. Solder opposing socket pins to hold the sockets to the board. Connect 0.1-μF disc capacitors to the V_{cc} and GND buses close to each integrated-circuit socket.

The pins are numbered as shown in Fig. 19–2 and correspond to the connector pin numbering. Use pin A as GND and solder a short wire to

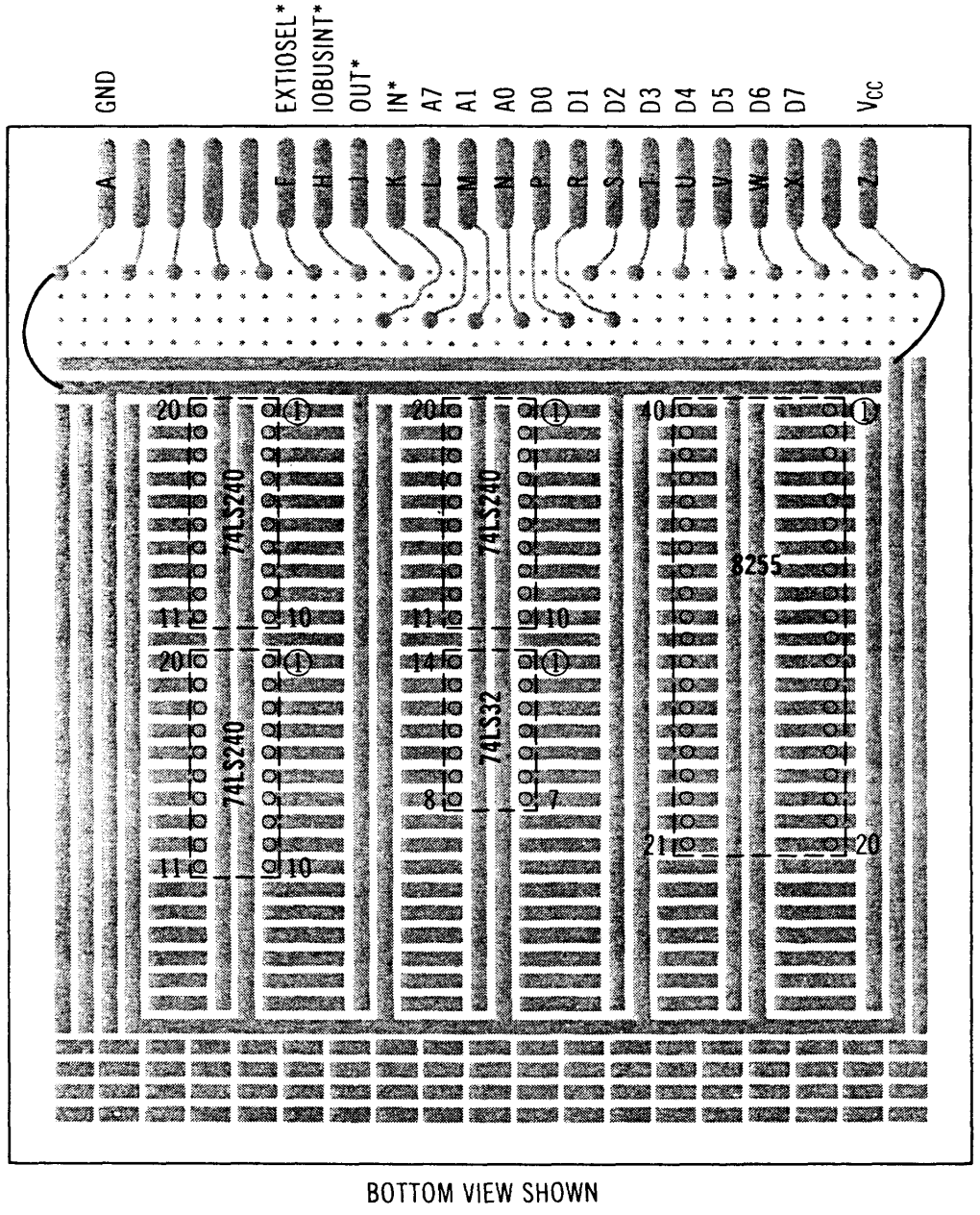


Fig. 19–2. GPIO socket mounting.

the GND bus as shown. Use the pin on the opposite end of the plug, pin Z, as V_{cc} and solder as shown.

Wrap the sockets as shown in Table 19–1. Fig. 19–2 shows the bottom view of the board with correct pin numbering. The connector pins are labeled CON-K, CON-J, etc., in Table 19–1 and are labeled in Fig. 19–2.

Checking the Board Wiring

After you've wired the board, check the wiring before plugging in any integrated circuits. Two common pins fit nicely into the IC socket holes, as shown in Fig. 17–6.

Table 19–1. GPIO Board (Models I/III) Wiring List

From	To	Signal	From	To	Signal
CON-K	8255-5	IN*/$\overline{RD}$	8255-3	74LS240-15	PA1
CON-J	8255-36	OUT*/$\overline{WR}$	8255-4	74LS240-17	PA0
CON-L	8255-6	A7/$\overline{CS}$	8255-25	74LS240-2	PB7
8255-6	74LS32-1	A7/$\overline{CS}$	8255-24	74LS240-4	PB6
8255-5	74LS32-2	IN*/$\overline{RD}$	8255-23	74LS240-6	PB5
CON-F	74LS32-3	EXTIOSEL*	8255-22	74LS240-8	PB4
CON-M	8255-8	A1	8255-21	74LS240-11	PB3
CON-N	8255-9	A0	8255-20	74LS240-13	PB2
CON-X	8255-27	D7	8255-19	74LS240-15	PB1
CON-W	8255-28	D6	8255-18	74LS240-17	PB0
CON-V	8255-29	D5	8255-10	74LS240-18	PC7
CON-U	8255-30	D4	8255-11	74LS240-16	PC6
CON-T	8255-31	D3	8255-12	74LS240-14	PC5
CON-S	8255-32	D2	8255-13	74LS240-12	PC4
CON-R	8255-33	D1	8255-17	74LS240-9	PC3
CON-P	8255-34	D0	8255-16	74LS240-7	PC2
CON-H	1 kΩ Resistor	IOBUSINT*	8255-15	74LS240-5	PC1
1 kΩ Resistor	V_{cc}	IOBUSINT*	8255-14	74LS240-3	PC0
8255-35	8255-7	RESET	74LS32-14	V_{cc} Bus	
CON-Z	V_{cc} Bus	V_{cc}	8255-26	V_{cc} Bus	
CON-A	GND Bus	GND	74LS240-20 (3)	V_{cc} Bus	
8255-37	74LS240-2	PA7	74LS32-7	GND Bus	
8255-38	74LS240-4	PA6	8255-7	GND Bus	
8255-39	74LS240-6	PA5	74LS240-1 (3)	GND Bus	
8255-40	74LS240-8	PA4	74LS240-19 (3)	GND Bus	
8255-1	74LS240-11	PA3	74LS240-10 (3)	GND Bus	
8255-2	74LS240-13	PA2			

Insert 0.1-μF capacitors between V_{cc} and GND pins near IC sockets (5).

Cable Fabrication

If the wiring checks, you're ready to fabricate the cable. There are two cables, one for the Model I and one for the Model III, as shown in Fig. 19–3. Two wires go from the 44-pin connector end of the cable to a +5-volt supply, as shown in Fig. 19–4. At one end of the cable, use the Radio Shack 44-pin connector. Solder the connections. At the other end, use a 40-pin edge connector (Radio Shack 276-1558) for the Model I or a 50-pin edge connector for the Model III. Use the numbering shown in Fig. 19–3 for the edge connectors! It is true that the Model III connector uses the reverse numbering from the Model I connector! Connections may be made using ribbon cable (and "smashing" on the ribbon cable to the connector) or simply by soldering 24-gauge stranded copper wire to the connector pins. If you are using individual wires, use cable ties to band the wire together into a single cable.

PROTOTYPE BOARD CONNECTOR PIN	MODEL I PIN	MODEL III PIN	PROTOTYPE BOARD SIGNAL (PIN)
K	19	33	IN*
J	12	35	OUT*
L	36	31	A7
M	27	19	A1
N	25	17	A0
X	20	15	D7
W	24	13	D6
V	28	11	D5
U	18	9	D4
T	26	7	D3
S	32	5	D2
R	22	3	D1
P	30	1	D0
A	8	8	SIG GND
A	(TO EXTERNAL GROUND)		SIG GND
Z	(TO EXTERNAL +5 V DC)		V_{CC}
H	21	39	IOBUSINT*
F	NC	43	EXTIOSEL*

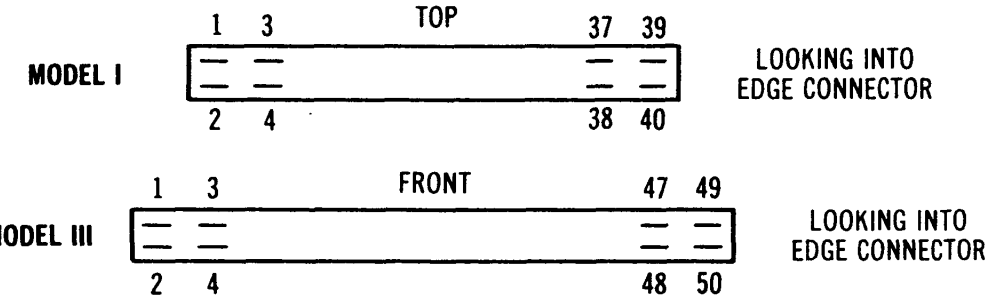


Fig. 19–3. GPIO cabling connections.

Testing the GPIO

When you have "buzzed out" the cable, plug in the integrated circuits, connect the cable to the board, and connect the power supply leads to the +5-volt supply (do not plug the cable into the computer at this point). Make a "smoke test" by momentarily touching the chips. The 8255 should be warm but not hot.

Turn off all power and plug the other end of the cable into the Model I or III. The proper orientation is shown in Fig. 19–5. Execute the BASIC program shown in Fig. 19–6 (eliminate line 110 for the Model I version of the board). This program toggles the outputs of PA7 through PA0 at a low speed rate and also reads lines PC7 through PC0.

Carefully test the outputs of the first 74LS240 by the method shown in Fig. 17–9. Of course, you may use a voltmeter, logic probe, or scope if you have one. You should see the output change from 0 volts to +4 volts or so and back again.

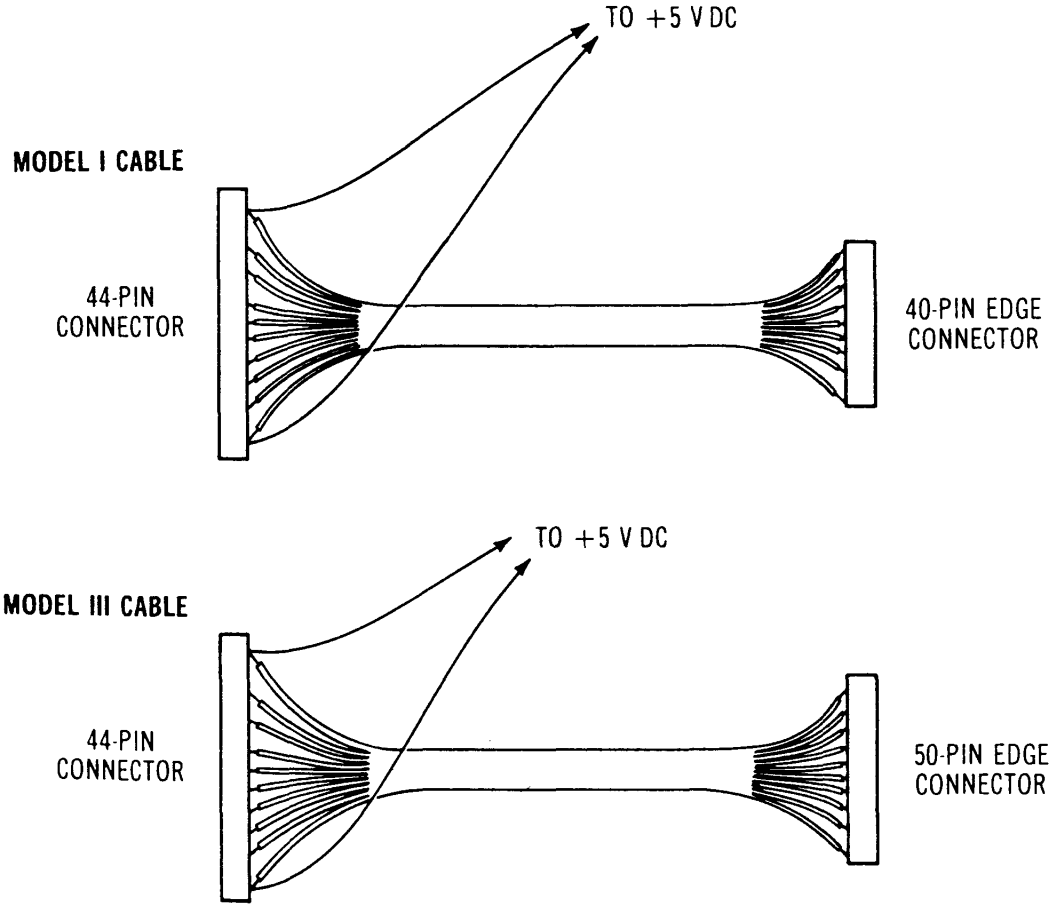


Fig. 19–4. Power supply cabling.

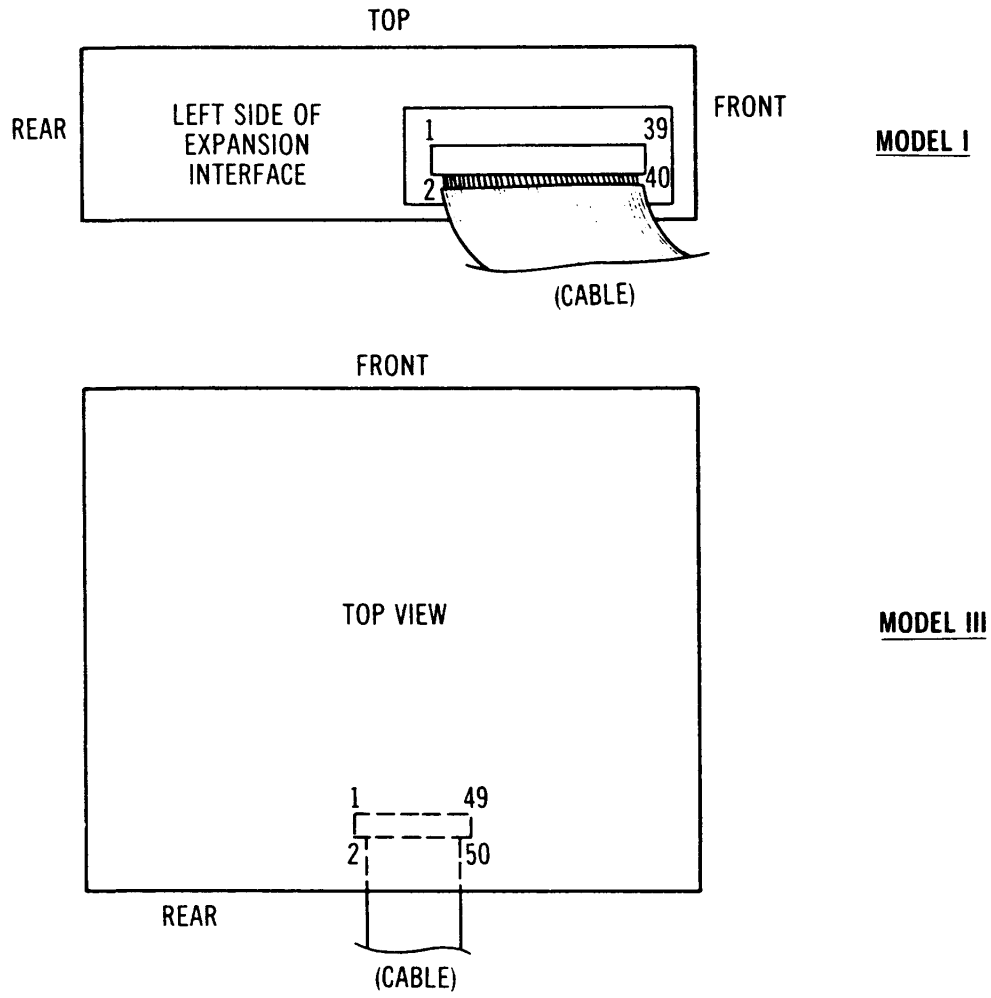


Fig. 19–5. Models I and III system bus connector orientation.

```
100 'DEMO MODEL III PROGRAM FOR OUTPUT AND INPUT
105 CLS
110 OUT 236,16
120 OUT 3,137
140 PRINT @512+32,INP(2)
150 OUT 0,0
160 GOSUB 1000
170 OUT 0,255
180 GOSUB 1000
190 GOTO 140
1000 FOR I=0 TO 100
1010 NEXT I
1020 RETURN
```

Fig. 19–6. GPIO demonstration program.

Inputs may be grounded by connecting the input pins of the third 74LS240 to ground. You should see a 128, 64, 32, 16, 8, 4, 2, or 1 value displayed on the screen, corresponding to the "weight" of the pin grounded. The PB7 through PB0 outputs may be tested by substituting OUT 1,0 and OUT 1,255 for lines 150 and 170, respectively.

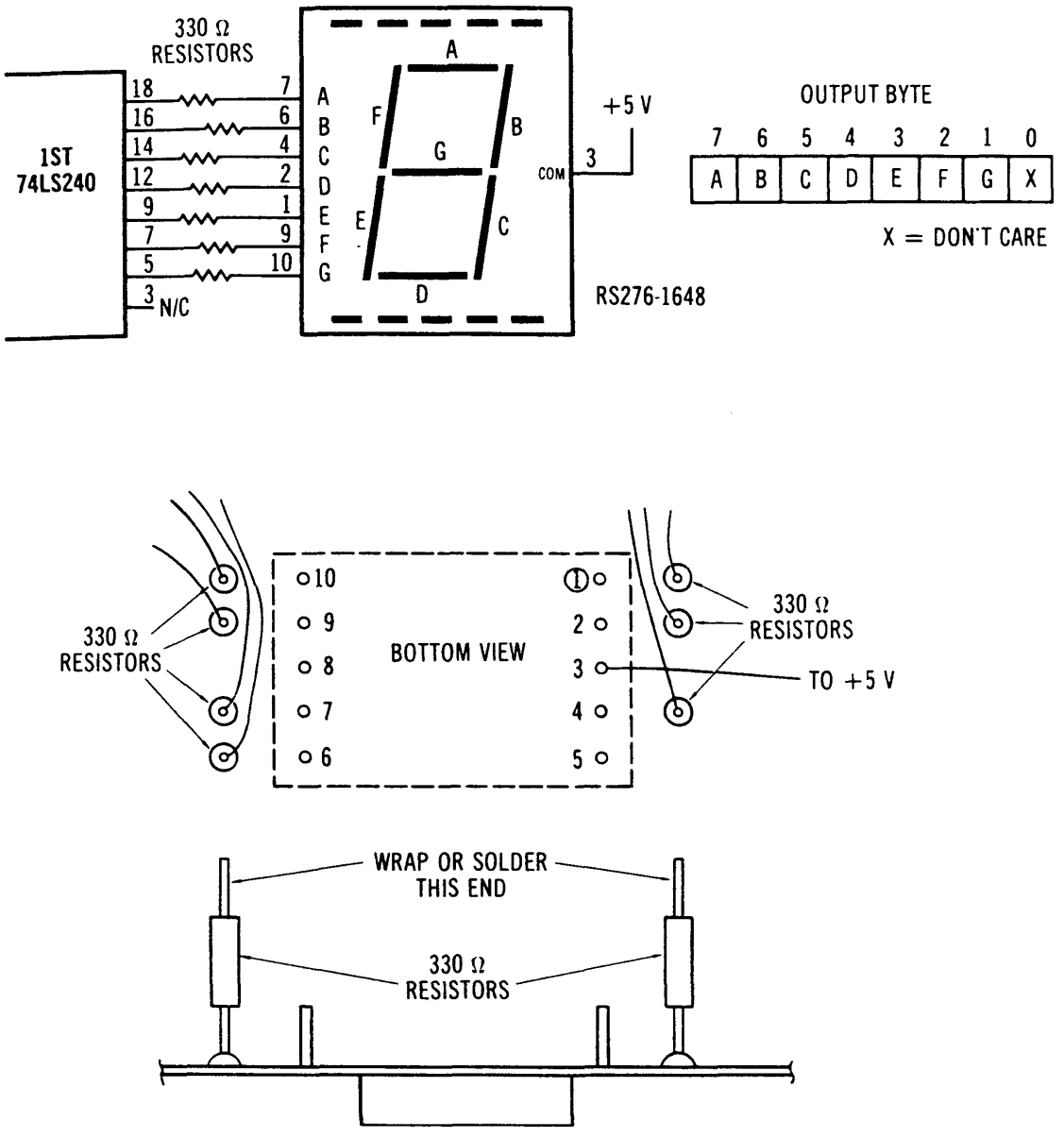


Fig. 19–7. LED display driver layout.

TYPICAL APPLICATIONS FOR THE GPIO

To give you some flavor of how the GPIO board may be used, I've implemented a 7-segment LED display driver as shown in Fig. 19–7. The LED display used is a Radio Shack 276-1648, but any similar common-anode display may be used. Wire the display as shown in Fig. 19–7. The 330-ohm resistors may be stood on end and wire-wrapped on the free end. The current-limiting resistors are connected to the first 74LS240 as

```
' DEMO MODEL I PROGRAM FOR 7-SEGMENT LED OUTPUT
100 OUT 3,137
105 INPUT V
110 OUT 0,0
120 GOSUB 1000
130 OUT 0,V
140 GOSUB 1000
150 IF INKEY$="" THEN GOTO 110 ELSE GOTO 105
1000 FOR I=0 TO 1000
1010 NEXT I
1020 RETURN
```

Fig. 19–8. LED display BASIC driver program.

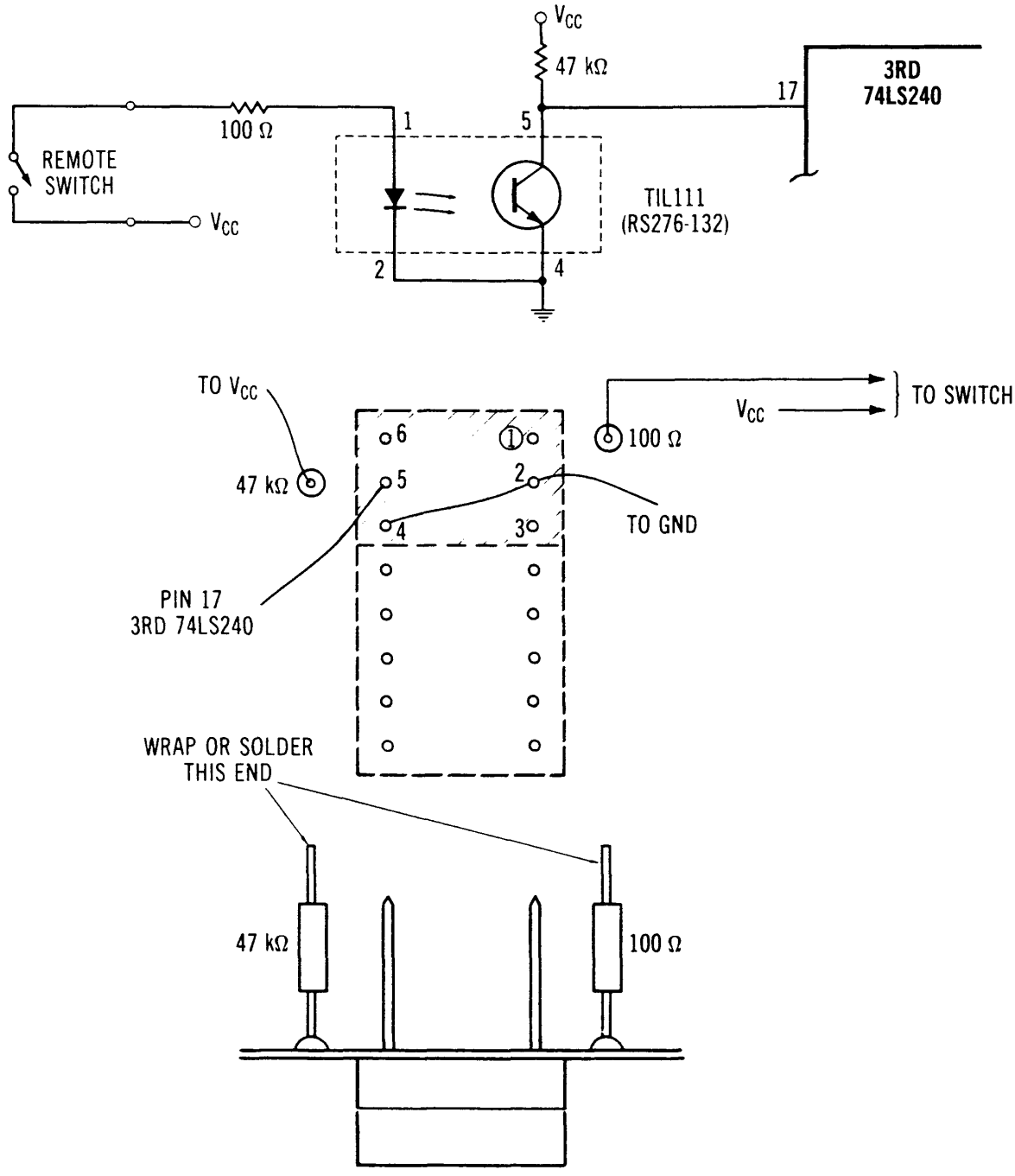


Fig. 19–9. Remote sensing layout.

shown in Fig. 19–7. The Model I BASIC program shown in Fig. 19-8 will drive the LED display and illuminate any combination of the seven segments. Insert a 95 OUT 246,16 for the Model III version.

A second application is a remote sense. Although I've used just one input, up to eight could be used in the GPIO configuration we're using. The remote sense uses a Radio Shack opto-isolator IC. The opto-isolator contains an infrared LED and phototransistor in one package, as shown in Fig. 19–9. Remote switch closure lights the LED and causes the transistor to conduct, bringing the input line to pin 17 of the 74LS240 down to 0.

One advantage of the opto-isolator is that it is a current-driven device. The line to the switch may be any length, as long as the current is sufficient to light the LED and cause the phototransistor to saturate. The opto-isolator eliminates the noise problem associated with TTL type inputs.

The wiring diagram for the remote sense is shown in Fig. 19–9. Again, the resistors may be positioned on end. Use the program shown in Fig. 19–10 (Model I) or Fig. 19–6 (Model III) to test the opto-isolator action.

```
100 'DEMO MODEL I PROGRAM FOR OPTO ISOLATOR INPUT
110 OUT 3,137
120 PRINT INP(3)
130 GOTO 120
```

Fig. 19–10. Remote sense BASIC driver program.

A third application uses relay input or output. The physical layout for both input and output is shown in Fig. 17–10. Radio Shack 275-228 relays (22.5 mA) are used and may be mounted on the board as shown. These relays will handle up to 750 mA (¾ A) on their contacts and can be used to drive a larger load than the 10 or 40 mA output of the 74LS240.

Section VI

Switches and Transducers for the Models I and III and the Color Computer

chapter **20**

General Methods for Inputs and Outputs

In this section we present some ideas on easy ways to monitor real-world physical quantities, such things as temperature, pressure, light intensity, magnetic fields, vibration, water level, shaft position, rotational speed, and others. All of these quantities can be measured with the three computer systems, and most of them can be measured quite easily and with a great deal of accuracy.

HOW TO GET IN AND OUT OF A COMPUTER

Before we discuss the actual circuits and implementation for these devices, let's recap the various ways we can interface the real world to the three computer systems. Various projects and interfacing techniques are covered in previous chapters of this book. However, it will help to have all of the interfacing options present in one place. So we try to summarize all the general techniques in one place in this chapter and reference previous chapters for specifics. In the next chapter we look at practical switches for discrete inputs. In the last two chapters we look at ways to amplify signals for a/d conversion and then describe some practical, inexpensive transducers.

Now, to recap general schemes for interfacing on the three computer systems.

READING SWITCH CLOSURES

When a simple switch must be read for slowly changing real-world events such as burglar alarms, there are a variety of ways to read an on-off condition. Most of the methods simply involve reading a single bit of an

input/output port. One word of caution: The switch may have to be debounced as it rapidly makes and breaks contact before settling down to a steady state.

Method 1 (Model III and Color Computer): Connect a single-pole, double-throw (spdt) switch to +3 and −3 volts as shown in Fig. 20–1. The center lead and ground goes to the cassette input line (the plug that connects to the EAR jack for the cassette input). Read the switch by (INP(255) AND 1) on the Model III and by PEEK(&HFF20) AND 1 on the Color Computer. Good for 50 feet or more of cable.

Method 2 (Color Computer): Connect a single-pole, single-throw (spst) switch between pins 3 and 4 of the right joystick plug of the Color Computer as shown in Fig. 20–2. Read the switch by a PEEK(&HFF00) AND 1. Connect between the same pins on the left joystick plug and

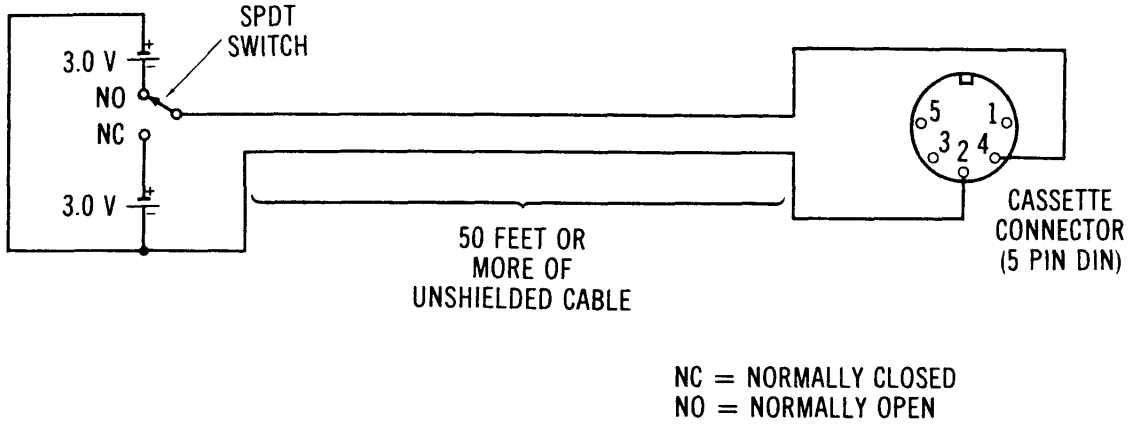


Fig. 20–1. Discrete input using the cassette input.

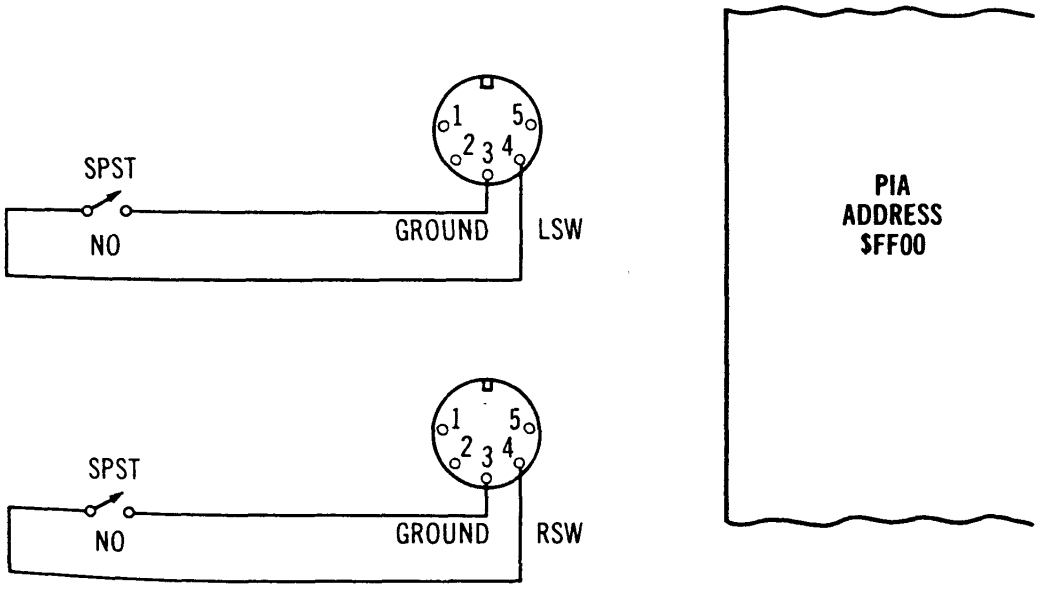


Fig. 20–2. Discrete input using the joystick switch inputs.

read by a PEEK(&HFF00) AND 2. Also good for 50 feet or more of cable.

Method 3 (Color Computer): Connect an spst switch and two resistors in the circuit shown in Fig. 20–3. Four lines may be read by JOYSTK(0), JOYSTK(1), JOYSTK(2), and JOYSTK(3), respectively. An "on" value will be about 0, and an "off" value will be about 32. Test for values greater than or less than 16. Good for 50 feet or more of cable.

Method 4 (Color Computer): Connect an spdt switch and +6 and −6 volts as shown in Fig. 20–4. Connect ground and the center contact of the switch to pins 3 and 2 of the CC RS-232-C jack. Read the switch by a PEEK(&HFF22) AND 1. Good for a distance of 50 feet or more.

Method 5 (Models I and III and Color Computer): Build a general-purpose I/O interface that attaches to the system I/O bus. Two inter-

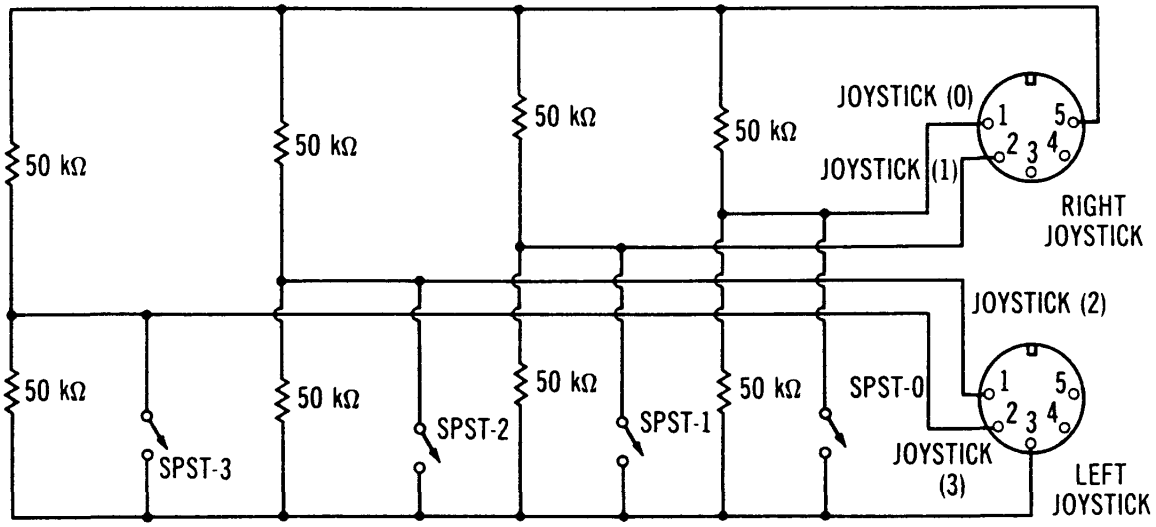


Fig. 20–3. Discrete inputs using the joystick channels.

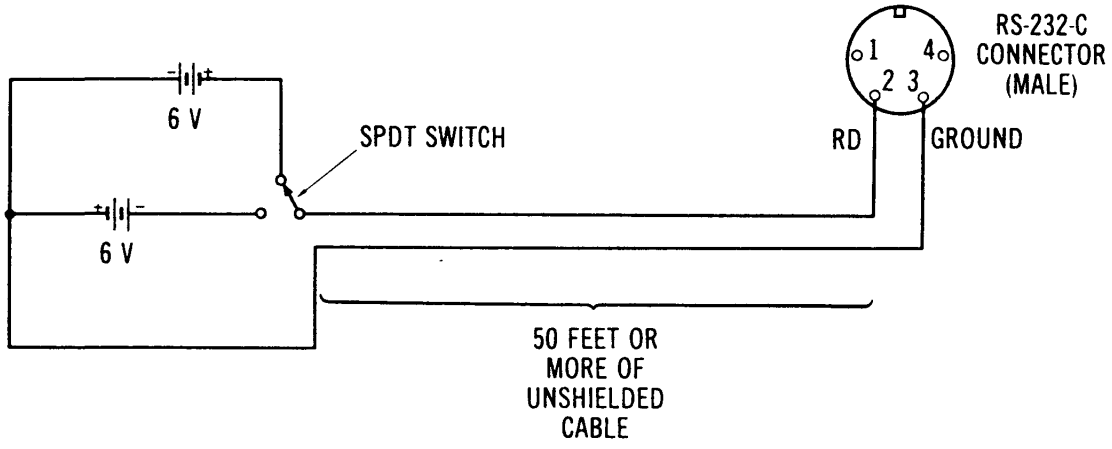


Fig. 20–4. Discrete inputs using the Color Computer RS-232-C inputs.

faces, one for the Models I and III and one for the Color Computer, are described in Chapters 17 and 19, respectively. These interfaces provide up to 24 lines that can be used to read switch closures or other inputs, at the expense of complexity in construction.

Method 6 (Models I and III): Connect an spdt switch between +6 and −6 volts as shown in Fig. 20–5. Connect ground and the center switch lead to one of four inputs on the RS-232-C port. Read the four lines by an OUT 232,0 followed by INP(232) AND 128, 64, 32, or 16. Further details in Chapter 8. Good for 50 feet or more of cable.

Method 7 (Models I and III): Connect an spst switch between pin 2 and pins 21, 23, 25, or 28 of the printer port, as shown in Fig. 20–6. Read the switch by a (PEEK(14312) AND 128, 64, 32, 16) for the Model I or (INP (248) AND 128, 64, 32, or 16) for the Model III. Not recommended for more than a few feet.

CONTROLLING SLOWLY CHANGING EXTERNAL DEVICES

Using the computer to control external devices is more difficult, as power must be provided to switch the devices on and off.

Method 1 (Models I and III and Color Computer): Use the cassette relay to control another relay, as shown in Fig. 20–7. Turn the relay on in

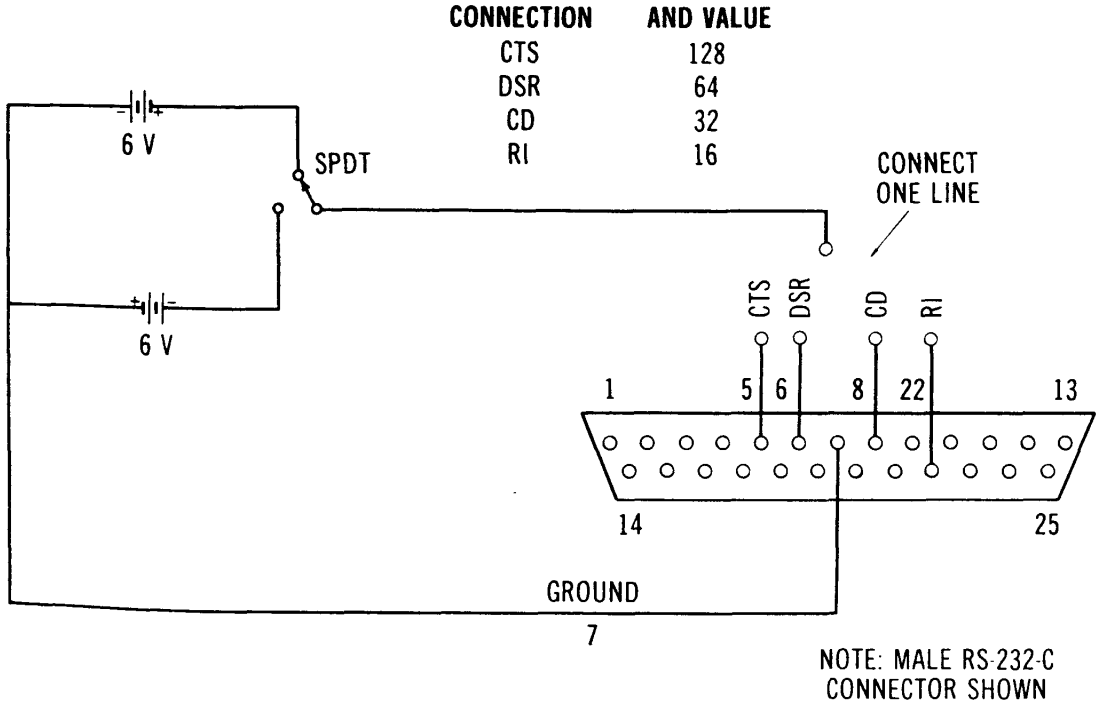
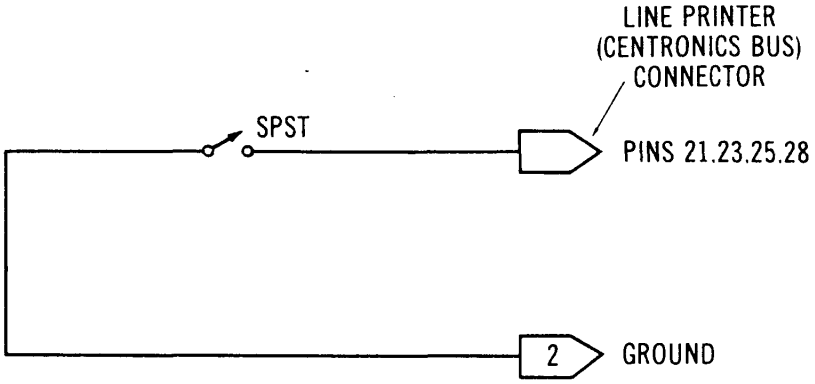


Fig. 20–5. Discrete inputs using the Models I and III RS-232-C inputs.

the Model I by an OUT (255,4), in the Model III by an OUT (236,2), and in the Color Computer by a POKE &HFF21,60. Use a value of 44 to turn the Color Computer off. Do not use for outputs that change more rapidly than once every few seconds or so. The length of either set of lines may be very long.



PIN USED	AND VALUE	SWITCH OFF	SWITCH ON
21	128	1	0
23	64	1	0
25	32	1	0
28	16	1	0

Fig. 20–6. Discrete inputs using the printer port.

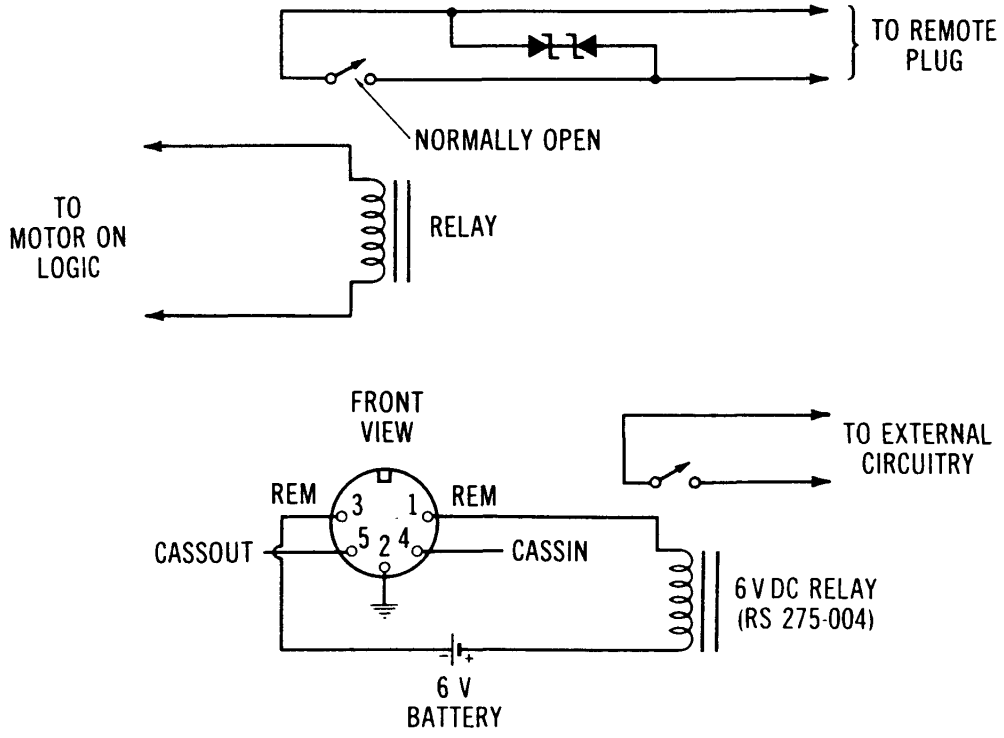


Fig. 20–7. Discrete output using the cassette relay.

Method 2 (Models I and III and Color Computer): Use the cassette relay and an opto-coupler, as shown in Fig. 20–8. Turn the circuit on as in Method 1. Line lengths may be very long.

Method 3 (Models I and III and Color Computer): Use the general-purpose input/output board referenced above to drive up to 24 lines. Use either relays or opto-couplers with each line. See the chapters referenced for further details.

READING IN ANALOG SIGNALS

Real-world quantities such as temperature and light intensity can be converted to electrical analogs such as voltage and resistance. Although we discuss this particular topic in more detail later, here is a look at the general approach:

Method 1 (Color Computer): Read in an analog voltage of 0 through 5 volts by referencing it to ground and connecting the input to one of the four joystick channels, as shown in Fig. 20–3. Use JOYSTK(X) to get the input value in the form 0 through 63. Convert to the proper voltage or real-world equivalent. This method is good for conversions of dozens of times per second. See Chapters 1 and 2. For faster conversion speeds (up to 8K samples per second), see Chapter 3 for a high-speed Color Computer analog-to-digital converter. Lengths of lines to current-driven transducers may be extremely long.

Method 2 (Models I and III): Build the analog-to-digital converter described in Chapter 4. This converter plugs into the printer port and will convert at rates of thousands of samples per second. Lengths of lines with proper transducers may be very long. Two analog channels are provided.

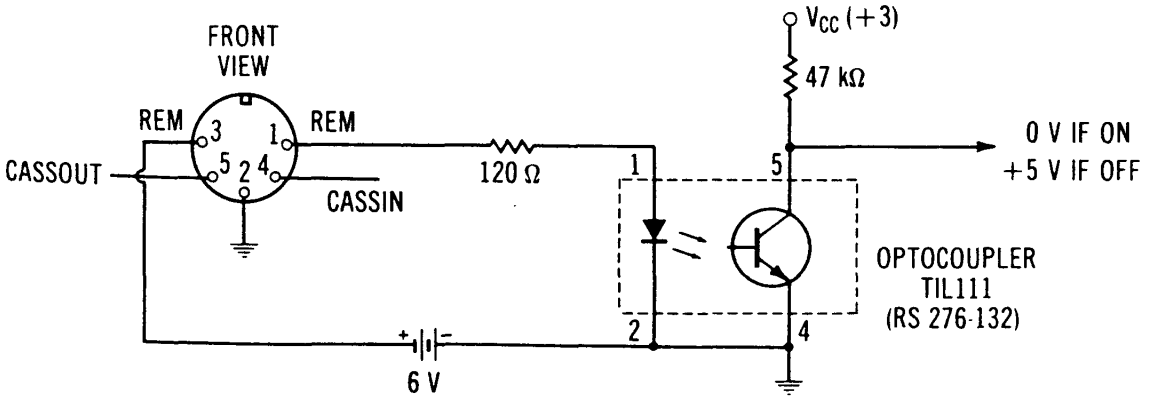


Fig. 20–8. Discrete output using an opto-coupler.

Method 3 (Model III): Build the analog-to-digital converter described in Chapter 5. This converter is extremely accurate but slow (6 samples per second). It allows only one channel and voltage inputs of 1.25 through 3.75 volts.

Method 4 (Models I and III and Color Computer): Use a voltage input to a VCO (voltage-controlled oscillator) such as an LM366 (RS 276-1724) to create a frequency analog that may be measured through the cassette port. This method would be similar to the technique in Method 3, but would measure the frequency of a square wave rather than the duty cycle.

OUTPUTTING ANALOG VOLTAGES

Voltage levels may be used to control dc motors (through dc amplifiers), create music or speech synthesis, or for other real-world functions.

Method 1 (Models I and III): The analog-to-digital converter described in Chapter 4 uses a digital-to-analog converter that will provide output voltages from 0.24 through 4.74 volts in 64 steps. Speed can be tens of thousands of outputs per second with an assembly language driver program.

Method 2 (Color Computer): The Color Computer has a built-in digital-to-analog converter that outputs 0 through 5 volts in 64 steps at speeds of thousands of outputs per second. Output can be routed to the cassette output line.

Method 3 (Models I and III): The cassette output line can be provided with three voltage levels: 0 volts, 0.45 volt, and 0.86 volt by outputting 2, 0, and 1, respectively, to input/output address 0FFH (OUT(255,X)). Output may be done tens of thousands of times per second in assembly language.

READING IN RAPIDLY CHANGING ON/OFF SIGNALS

There are a number of methods to read in frequency analogs of real-world quantities. It's not too difficult to convert a physical parameter to voltage and then convert the voltage to frequency. We can then measure either the period or duty cycle of the incoming signal to work back to the original quantity. In some cases the signal may have to be "conditioned" to eliminate noise or bounce. In general, the greater the frequency, the

shorter the lines must be. Use twisted pair or shielded wire and you may get lengths of 50 feet or greater.

Method 1 (Models I and III): The Model I cassette tape input circuit takes a series of 500-baud pulses, rectifies them, and looks for the dc level at the proper time. It would be possible to input a range of pulses at about 500 to 2000 pulses per second and read them from the cassette port (0FFH, bit 7). The Model III uses the same circuit for 500 baud, but it's best to go to Method 2 below, which is more reliable.

Method 2 (Model III and Color Computer): The 1500-baud cassette logic uses a zero-crossing detector. The incoming waveform should be about 2 to 4 volts peak to peak and must go negative. A dual power supply comparator or ac coupling can be used to generate the waveform. See Chapter 13. The data is read by INP(255) AND 1 (Model III) or PEEK(&HFF20 AND 1) (Color Computer).

Method 3 (Color Computer): The Color Computer RS-232-C port RD line can be used to input a string of pulses and can be read very rapidly by using PEEK(&HFF22) AND 1. The waveform must be in standard RS-232-C format (logic 0 greater than +3, logic 1 less than −3 volts). See Chapters 6 and 7.

Method 4 (Models I and III): Use the four RS-232-C input lines described under *Reading Switch Closures*, Method 6, above. The waveform must be in standard RS-232-C format.

Method 5 (Models I and III and Color Computer): Build the general-purpose input/output board referenced above. This provides up to 24 lines that may be read tens of thousands of times per second. Do not use long runs of cable unless using opto-isolators or current-driven schemes.

OUTPUTTING RAPIDLY CHANGING ON/OFF SIGNALS

There are not as many common real-world applications for this topic, but here are some of the methods:

Method 1 (Models I and III): Use the cassette output line to send square waves of up to 3 kHz or so. Output is accomplished by OUT (255,1) followed by OUT (255,2) in BASIC or by equivalent assembly language code. The waveform will swing between 0 and about 1 volt.

Method 2 (Color Computer): The TD line of the Color Computer can be toggled on and off by writing alternate 0s and 1s to address &HFF20,

bit 1. The waveform will be at standard RS-232-C levels (−12 and +12 volts).

Method 3 (Models I and III): Two RS-232-C signals in the Model I and 5 in the Model III can be toggled on and off. The TD line of the RS-232-C cannot be toggled on and off except by outputting a predefined character. However, some "dummying up" can be done of character bits, and the baud rate controlling the frequency can be varied under program control. The waveforms will be at standard RS-232-C levels. See Chapter 8 for details.

Method 4 (Models I and III and Color Computer): Build a general-purpose input/output board and you can toggle up to 24 separate lines tens of thousands of times per second. Output will be at TTL levels and will swing between 0 and about +4 volts.

These are the general approaches to interfacing the three computer systems. In the next chapter we look at some inexpensive, readily available switches.

chapter **21**

Using Switches for Discrete Inputs

Now that we've looked at the possible interfacing methods, let's present some data on specific devices and methods of measuring real-world quantities. We've tried to use only relatively common devices here, ones that will not cost more than about $15.00 for the most expensive. Most of them can be obtained at Radio Shack or a similar type of electronics parts store. We start with the simple ones and work up into the more exotic.

A WINDOW SENSOR

The first device is the Radio Shack Window Sensor switch, part number 49-516, shown in Fig. 21–1. This is simply a mercury (we assume) switch mounted inside a disc. The device comes apart into two pieces; the back cover has sticky tape that can be used to stick the cover

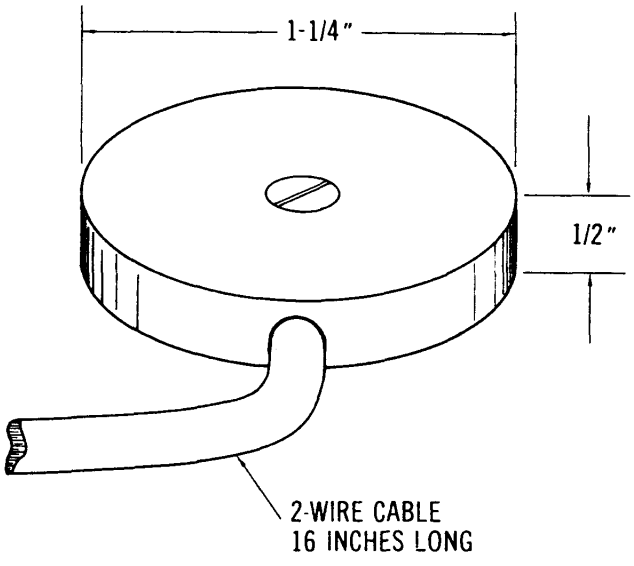


Fig. 21–1. Window sensor switch.

to a window or other smooth surface. The front section, containing the switch, can be rotated around the secured back to any position.

The intended purpose of the device is to act as a window security sensor. The device is stuck to a window and rotated so that the switch is just off or on. If the window is then broken or tapped hard, the switch will toggle as the device rolls or pitches forward or the mercury sloshes around. This simple device also can be made into a workable roll indicator or level sensor as shown in Fig. 21–2. The advantages are that it's modular, comes with the backing, and is supplied with a short cable.

A VIBRATION DETECTOR

The window sensor above is really not very sensitive. Certainly, window breakage will set it off, but in vibration sensing it is ineffectual.

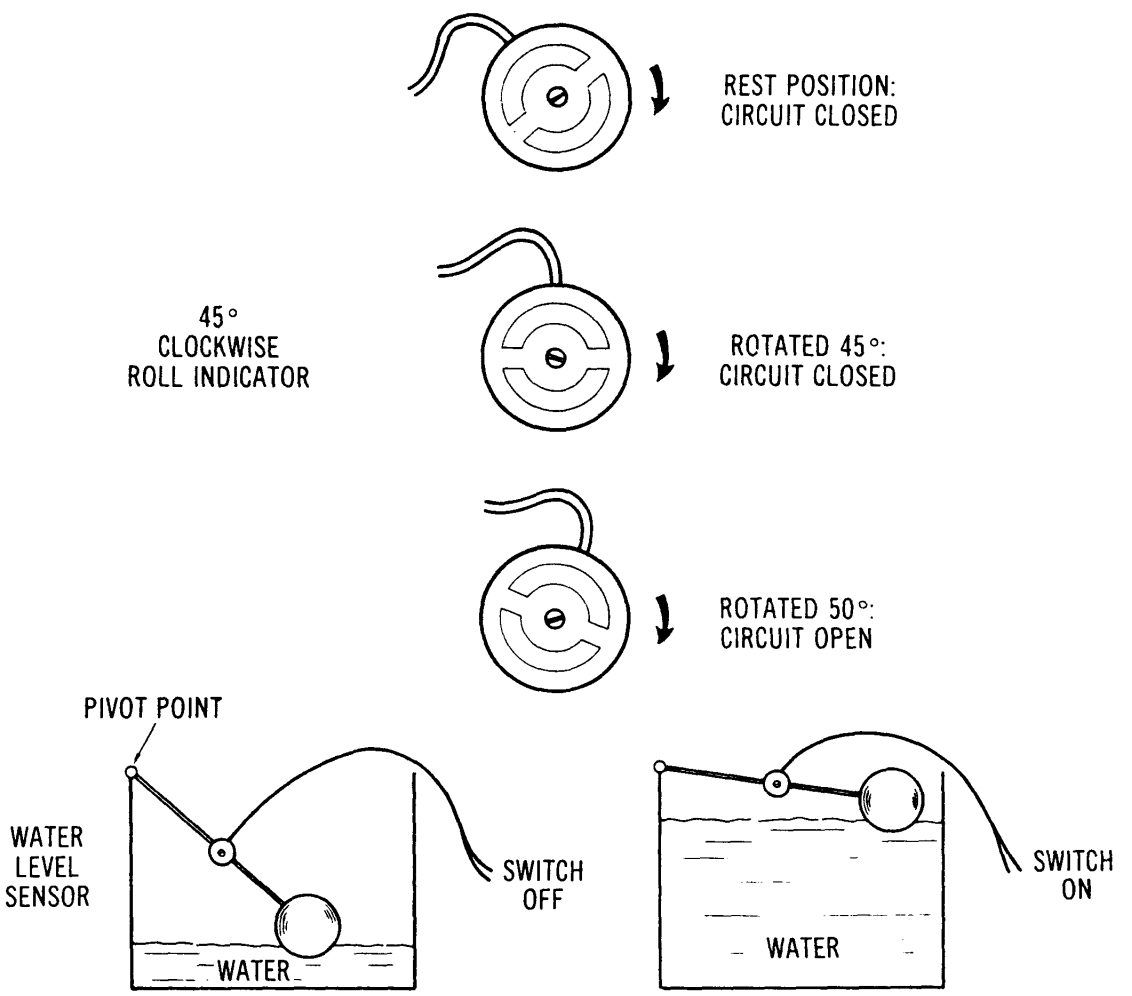


Fig. 21–2. Window sensor switch applications.

The Radio Shack Mini Shock/Vibration Detector (49-521), however, is very sensitive. It is shown in Fig. 21–3. This device is designed to detect vibration from forced entry and to prevent tampering with itself. The side view of the device is shown in Fig. 21–3. The contacts are normally closed and open when vibration moves the device. Evidently the large mass of the upper contact gives it a great deal of inertia, and as it resists movement the contacts open.

How sensitive is it? The specs show settings for 1 to 21 grams as *contact pressure.* This is somewhat difficult to translate into practical effects, but at its most sensitive setting, it will detect a penny dropped from a height of 2 inches 36 inches away from the sensor; the sensor was secured to a wooden table in this test. In fact, that is fairly sensitive. It would certainly make an excellent security sensor or earthquake detector.

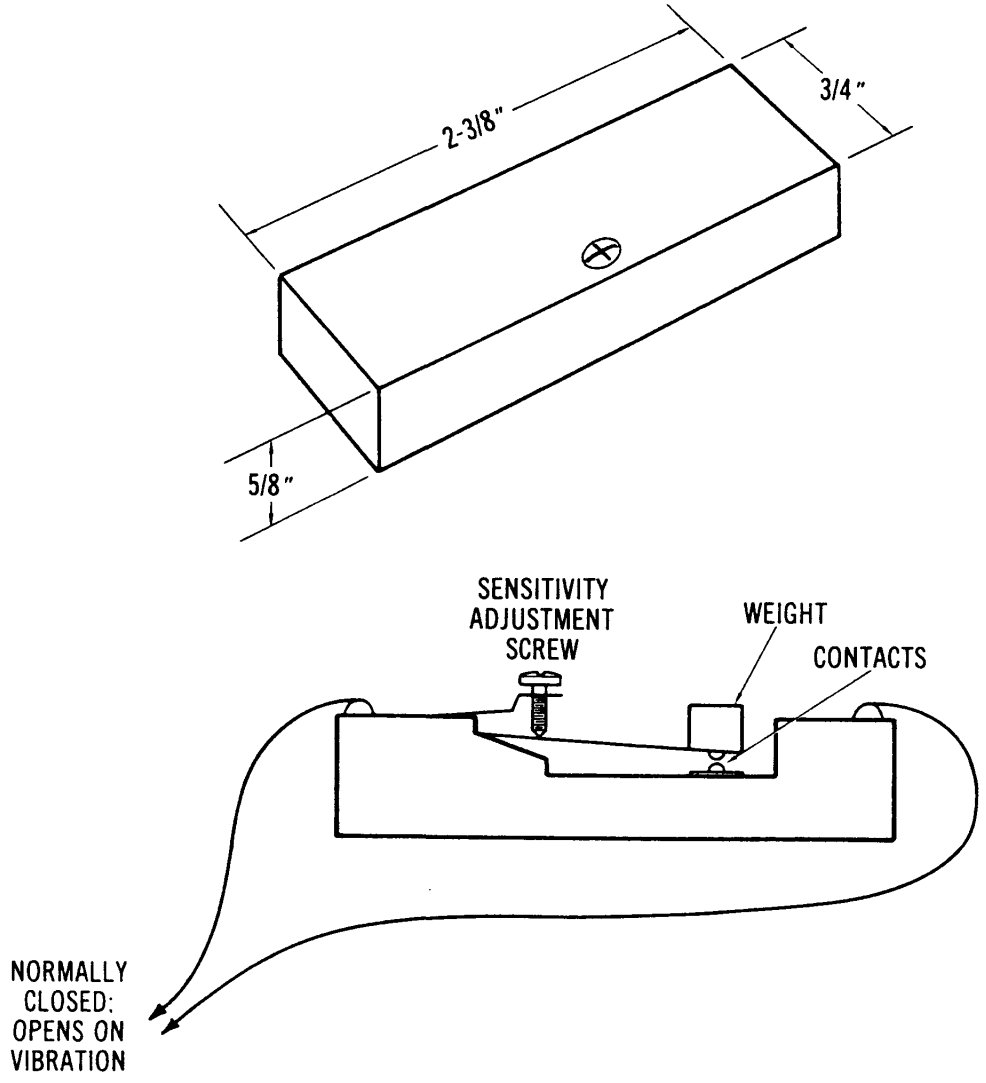


Fig. 21–3. Vibration sensor switch.

If you care to experiment with this device, use the following with a Color Computer

```
100 B=&HFF00
110 IF (PEEK(B) AND 1)=0 THEN GOTO 110
120 SOUND 100,40:GOTO 110
```

to read the right joystick switch and sound an alarm when the sensor breaks contact. The 80-times-per-second sample rate should detect every switch activation.

GLASS REED SWITCHES

A glass reed switch is shown in Fig. 21–4. These switches are glass-enclosed magnetic reeds with axial leads. The contacts on the reeds close when a magnetic field is brought near the switch. The switches are very inexpensive; the Radio Shack version sells for 10 for $1.98 (275-1610).

Glass reed switches can be used as detection devices when no physical

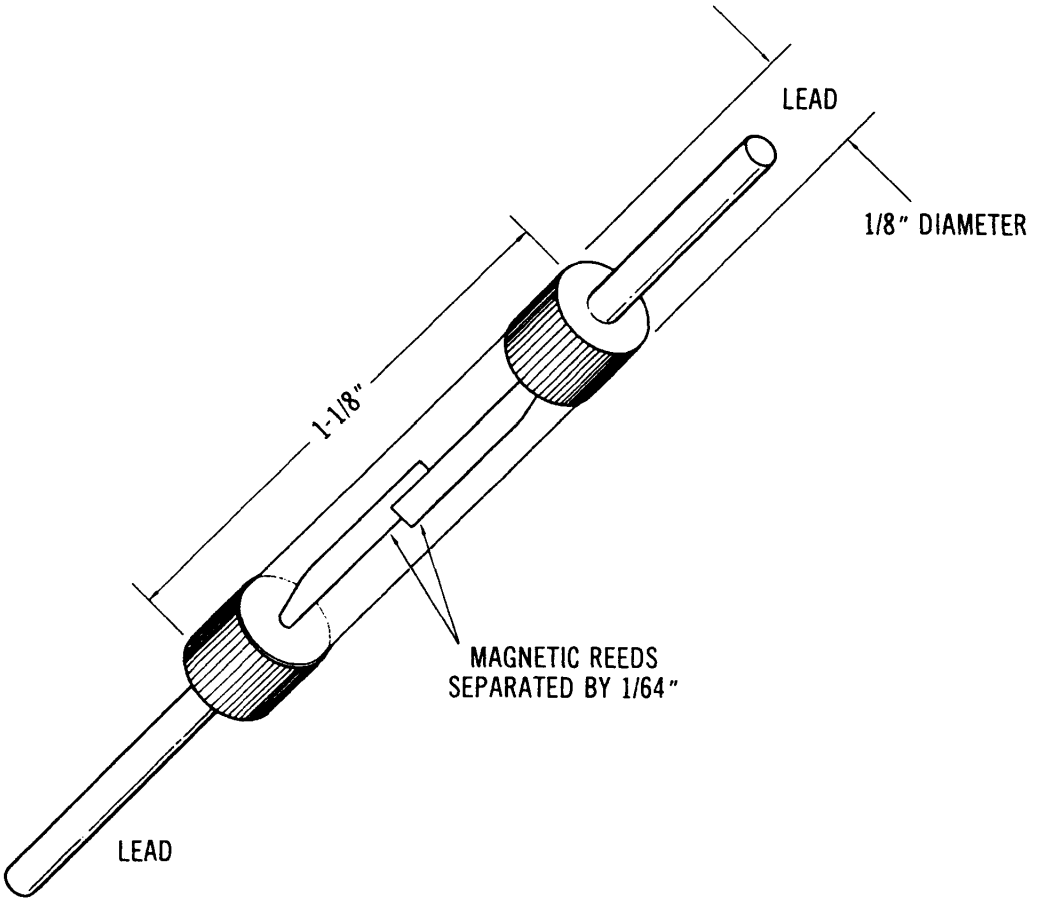


Fig. 21–4. Glass reed switch.

contact is possible. A typical application, for example, might be detection of passage of a model railroad car, as shown in Fig. 21–5. Another good example would be measuring the rotational speed of a shaft by mounting a glass reed switch near the circumference of a disc mounted on the shaft. A magnet mounted on the disc would actuate the switch when it passed nearby on every revolution. The number of revolutions could be easily counted by a computer with built-in debounce.

The obvious question here is, just how sensitive is the reed switch? To answer that, I used Radio Shack ceramic magnets (64-1875). These are rectangular magnets as shown in Fig. 21–6. They can be stacked together. They are not super magnets, but a garden variety, with a lift force of 1/8th pound. (A typical 6-inch bar magnet similar to the one you might have used in high-school physics class has a lift force of about one pound.) The ceramic magnets are ferrite based and very resistant to demagnetization.

The listing below shows the number of magnets required to close a switch at a specific distance and the open-after-close distance for the reed switch described above. You can see that a reed switch/magnet combination could easily lend itself to a variety of computerized sensing applications, especially if a more powerful magnet were used.

Number of Magnets	Close Distance (inches)	Open After Close Distance (inches)
1	3/16	5/8
2	5/16	1 1/4
3	1/2	1 3/4

Of course, it's one thing to talk in generalities about what to do and quite another to do it. To prove to ourselves that it was feasible to

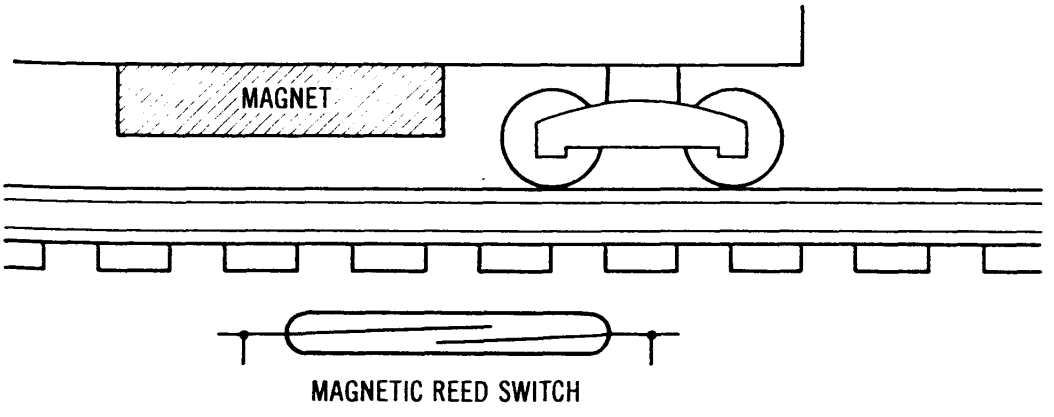


Fig. 21–5. Typical glass reed switch application in a model railroad.

measure rotational speed using reed switches, we rigged up the test setup shown in Fig. 21–7. A small dc motor drove a disc with two magnets, identical to the type we've been talking about. A reed switch was mounted about 1/4 inch away from the circumference of the disc.

The program shown in Fig. 21–8 was then entered into the Color Computer after first performing a CLEAR 200,&H3EFF to protect RAM. This program is identical to the one in Chapter 14, except that the joystick switch port is read instead of the cassette port. We're looking for a 0 in place of a 1, and a short time delay is used before connection of the switch leads. You'll recall that the joystick switches share two of the keyboard rows; connecting the switch lead before execution creates spurious keyboard characters in the switch closed state.

The program in Fig. 21–8 first asks for an interval and time delay. I used an interval of 16474 to correspond to a window of 1/2 second. A time delay of 20 ms was used to debounce the switch closure. With the motor turning at 240 rpm (4 revolutions per second), test results were

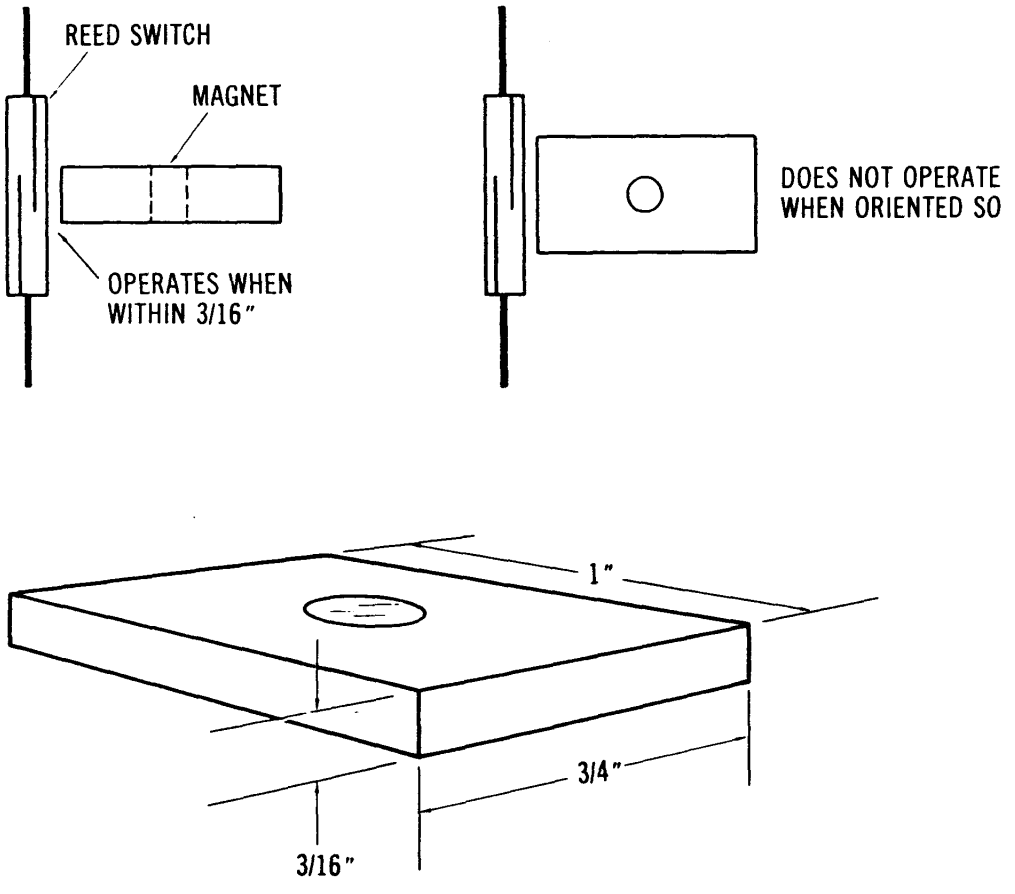


Fig. 21–6. Typical ceramic magnets for magnetic switch applications.

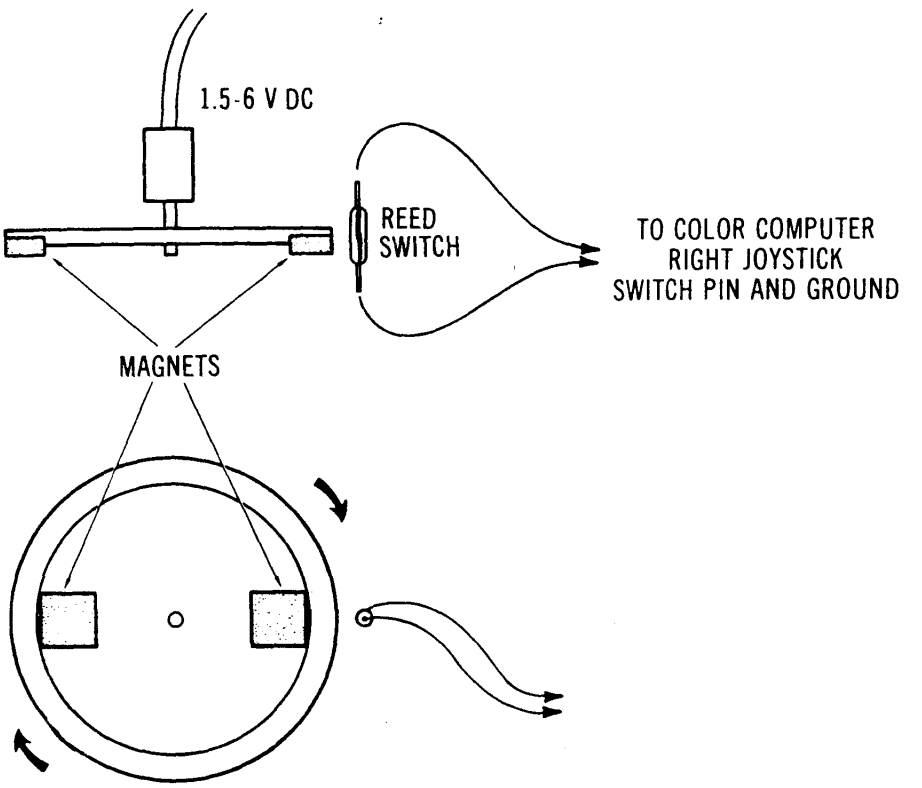


Fig. 21–7. Reed switch rotational-sensing configuration.

```
100 ' LOWFRE DRIVER
110 DATA 190,63,250,16,142,0,0,48,31,31
120 DATA 16,77,43,13,182,255,00,132,1,38
130 DATA 242,49,33,141,7,32,236,16,191,63
140 DATA 254,57,52,16,190,63,252,141,6,48
150 DATA 31,38,250,53,144,52,16,142,0,111
160 DATA 48,31,38,252,174,100,48,136,223,175
170 DATA 100,53,144
180 FOR I=&H3F00 TO &H3F3E
190 READ A: POKE I,A
200 NEXT I
210 DEFUSR0=&H3F00
220 INPUT "INTERVAL, DELAY";IC,DC
225 FOR J=0 TO 3000:NEXT J
230 POKE &H3FFA,INT(IC/256):POKE &H3FFB,IC-INT(IC/256)*256
240 POKE &H3FFC,INT(DC/256):POKE &H3FFD,DC-INT(DC/256)*256
250 A=USR0(0)
260 B=B+PEEK(&H3FFE)*256+PEEK(&H3FFF):PRINT B
270 GOTO 250
```

Fig. 21–8. Program for rotational speed sensing.

accurate, given that some counts may be missed due to the BASIC overhead of about 38 ms per call to the machine-language code, as shown.

HALL-EFFECT SWITCHES

Another magnetic-field-operated switch is the *Hall-effect switch.* Hall-effect switches are used in keyboard switches and similar applications.

These are physically small electronic devices, similar in appearance to and about the size of a transistor.

The schematic diagram of a Hall-effect circuit is shown in Fig. 21–9. The switch operates with a 5- to 16-volt power supply and is normally off. The device turns on (output goes to ground) when a magnetic field of 300 gauss is present. Hard to relate to the real world? Five stacked Radio Shack ceramic magnets operated the Hall-effect switch at a distance of about ¼ inch. The conclusion to be drawn is that these switches should be used with more powerful magnets, unless you're prepared to live with closer sensing distances than the reed switches.

AIR PRESSURE SWITCH

Another switch that should be mentioned here is a *sensitive air pressure switch* (part No. 41,623) from Edmund Scientific in Barrington, NJ 08007. This is an extremely sensitive switch that operates from the pressure difference between two inlet ports. It can be used as a high-wind

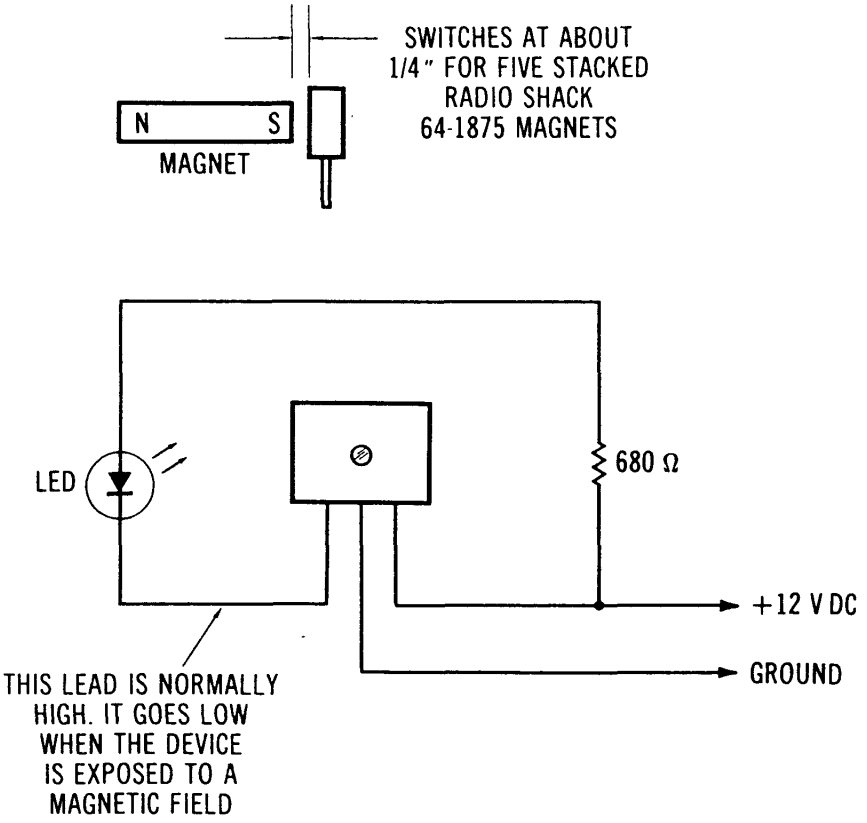


Fig. 21–9. Hall-effect switch circuit.

alarm, flow-rate switch, fan-failure switch, or the like. To give you an idea how sensitive it is: blowing at one of the ports from a few inches away will activate the switch. This is a single-pole, normally open switch that will handle only 10 mA of a resistive (not inductive) load, but it makes an excellent computer system switch for monitoring real-world conditions. The 10 mA limit is no restriction in the type of interfaces we're talking about here. In case of doubt: Keep resistance greater than 500 ohms when working with 5 volts or greater than 600 ohms for 6 volts.

In the next two chapters of this section, we conclude this section by looking at some very interesting devices, including thermistors, an LM334 temperature sensor, a tachometer wand, a dc motor generator, a solar cell, and an accurate pressure transducer, as well as ways to amplify input signals from these devices.

chapter **22**

Amplification of A/D Inputs

The last chapter describes some simple switches for TRS-80 Models I and III and Color Computer interfacing, and general approaches to communicating with the outside world from the three computer systems. In this chapter we continue the general subject of "cheap" transducers, devices that will enable us to monitor real-world physical quantities such as windspeed, temperature, and air pressure by looking at ways to amplify low-level signals to make them compatible with the a/d circuits of the three computer systems.

A/D INPUT VOLTAGES

The Color Computer analog-to-digital (joystick) channels operate with voltage inputs in the range of 0 to +5 volts dc. The various analog-to-digital converters we've described for the Models I and III also operate in this approximate voltage range. In general, the devices that we talk about in the next chapter operate at least an order of magnitude lower, in the range of hundreds of millivolts. For that reason, the first thing we should do is look at a general way to amplify the transducer outputs to a range more compatible with these adc's.

USING OP AMPS TO AMPLIFY ADC SIGNALS

Three basic amplifier circuits using operational amplifiers are described here. Operational amplifiers are *linear* integrated circuits that are commonly used as low-frequency amplifiers. They are characterized by high input impedance, low output impedance, and the capability of dealing with input voltages that track each other. There are always two

inputs with op amps, negative and positive inputs. The negative input is called the *inverting* input because a voltage increase on this input will result in a decrease in the output voltage. The positive input is the noninverting input because a voltage increase here will result in an increase in the output voltage.

Earlier op amps used dual power supplies; however, there are some newer versions that use a single supply. We use the single supply type in these circuits.

A typical configuration for an op amp amplifier is the inverting amplifier shown in Fig. 22–1. The voltage gain of this amplifier is determined by the feedback resistor that is connected from the output to the negative input (R2) and the input resistor for the negative input (R1). The voltage gain will be:

$$V_{out} = -V_{in} \times (R2/R1)$$

Typical values for R2 and R1 are 1 MΩ and 100 kΩ and these values will produce a X10 (times 10) op amp that will multiply the input voltage by 10. Note that in this case the input voltage must be negative. Inputting −0.1 volt will produce a +1-volt output, inputting −0.2 will produce a +2-volt output, and so forth. Inputting a positive voltage, such as +0.2, will produce a 0-volt output. The output voltage will increase in linear fashion to about +3.5 volts, as shown in Fig. 22–2, so this particular op amp will only multiply 0 to about −0.35 volt, for outputs of 0 through +3.5 volts.

A X5.6 op amp amplifier can be made by substituting a 560 kΩ resistor in place of the 1 MΩ feedback resistor. Similar substitutions can be made for other voltage gains. We'll be using the X10 and X5.6 inverting op amps in the applications discussed later.

The parts layout for a X10 or X5.6 op amp amplifier is shown in Fig. 22–3. It is built on a Radio Shack 276-175 prototype board, which allows easy connection of integrated circuits and components.

A noninverting op amp is shown in Fig. 22–4. This is another form of op amp that will amplify a positive voltage applied to the positive terminal of the op amp and produce an amplified positive output. The voltage gain in this amplifier is slightly different, but again dependent upon the values of the two resistors, R2 and R1. The voltage gain here is:

$$V_{out} = V_{in} \times ([R2/R1]+1)$$

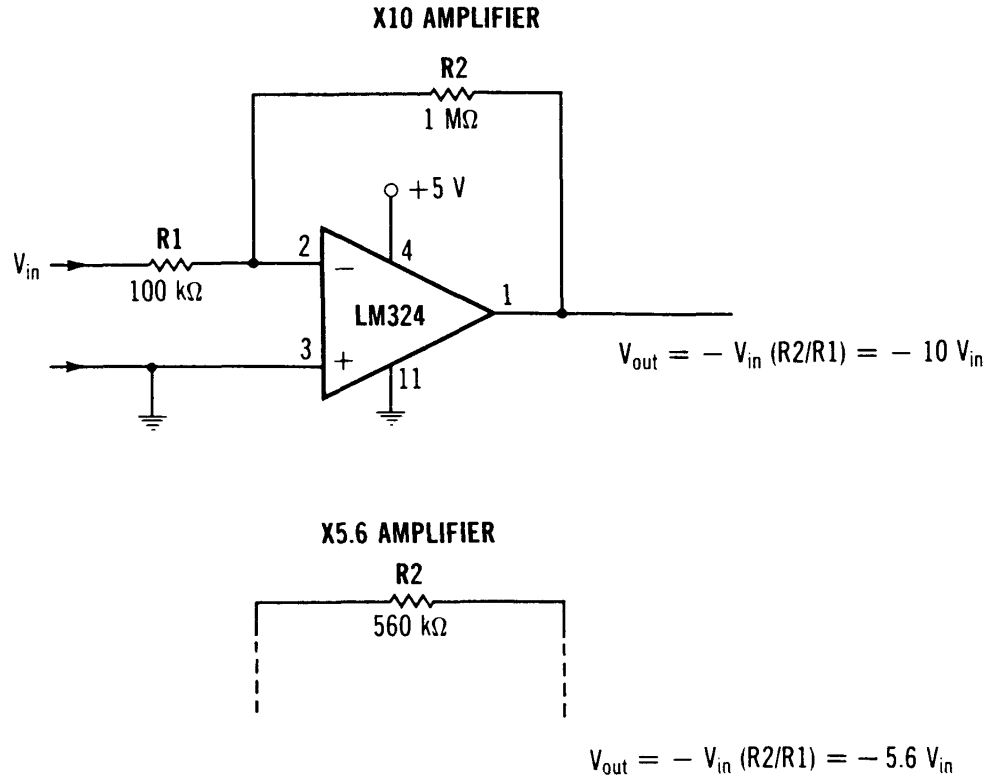


Fig. 22–1. An inverting operational amplifier with a gain of 10.

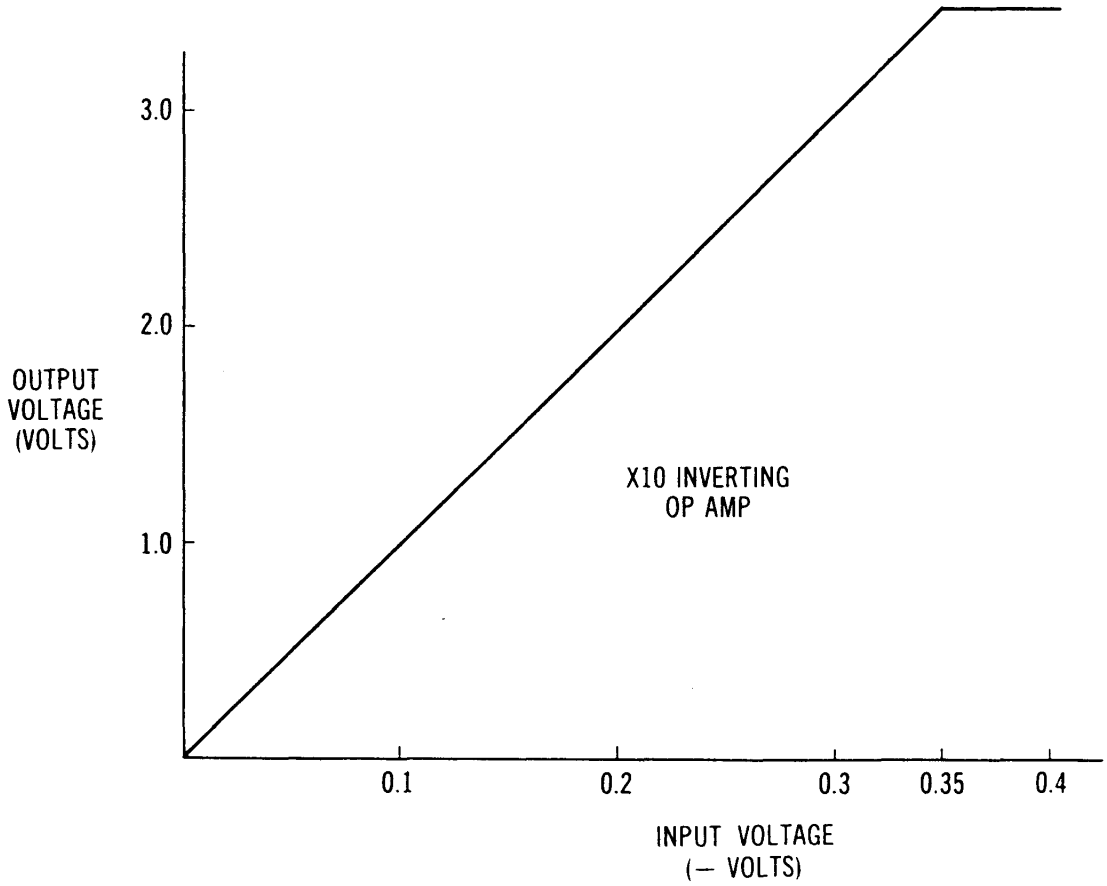


Fig. 22–2. Op amp input vs. output curve.

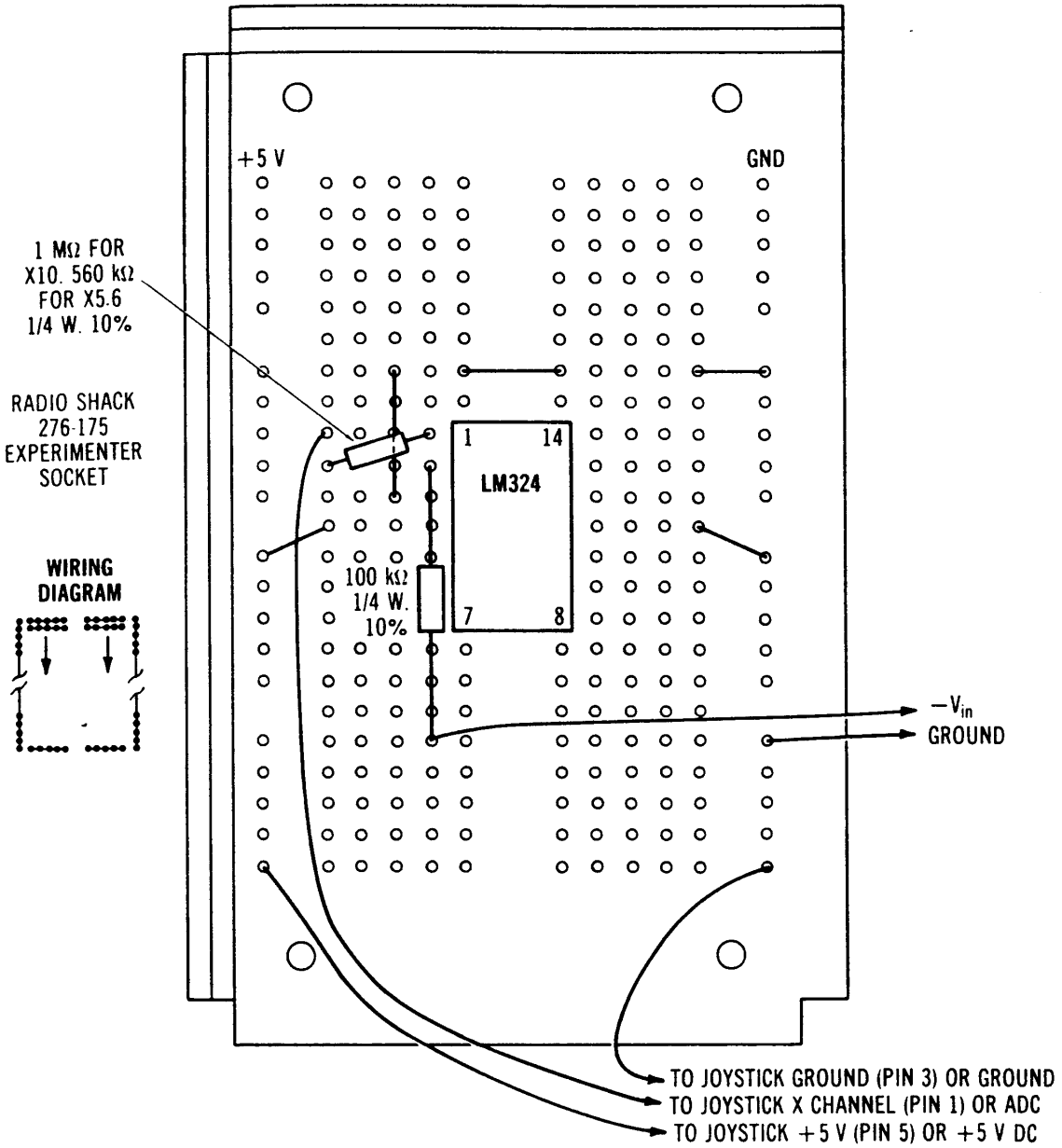


Fig. 22–3. Physical layout of the X10 inverting op amp.

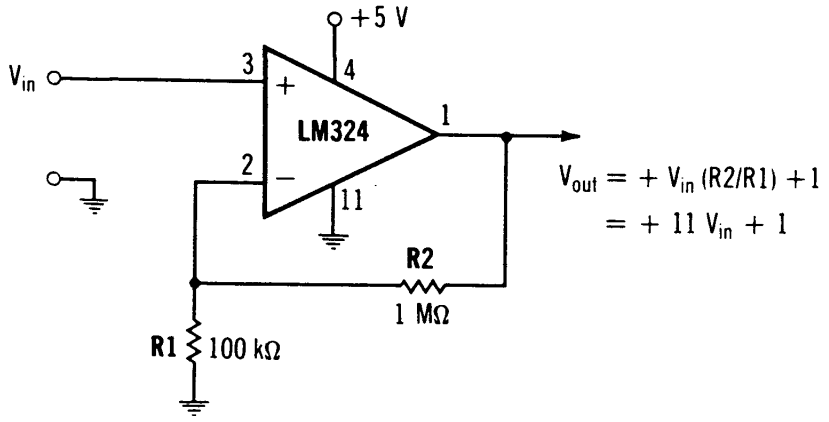


Fig. 22–4. A noninverting op amp circuit with a gain of 10.

The op amp configuration shown in Fig. 22–4 is the one we use with the transducers discussed in the next chapter, so we'll be getting a gain of about 11. The parts layout for the noninverting op amp is shown in Fig. 22–5, again on a prototype board.

A third type of op amp amplifier multiplies the difference between two input signals applied to the positive and negative inputs; the circuit is

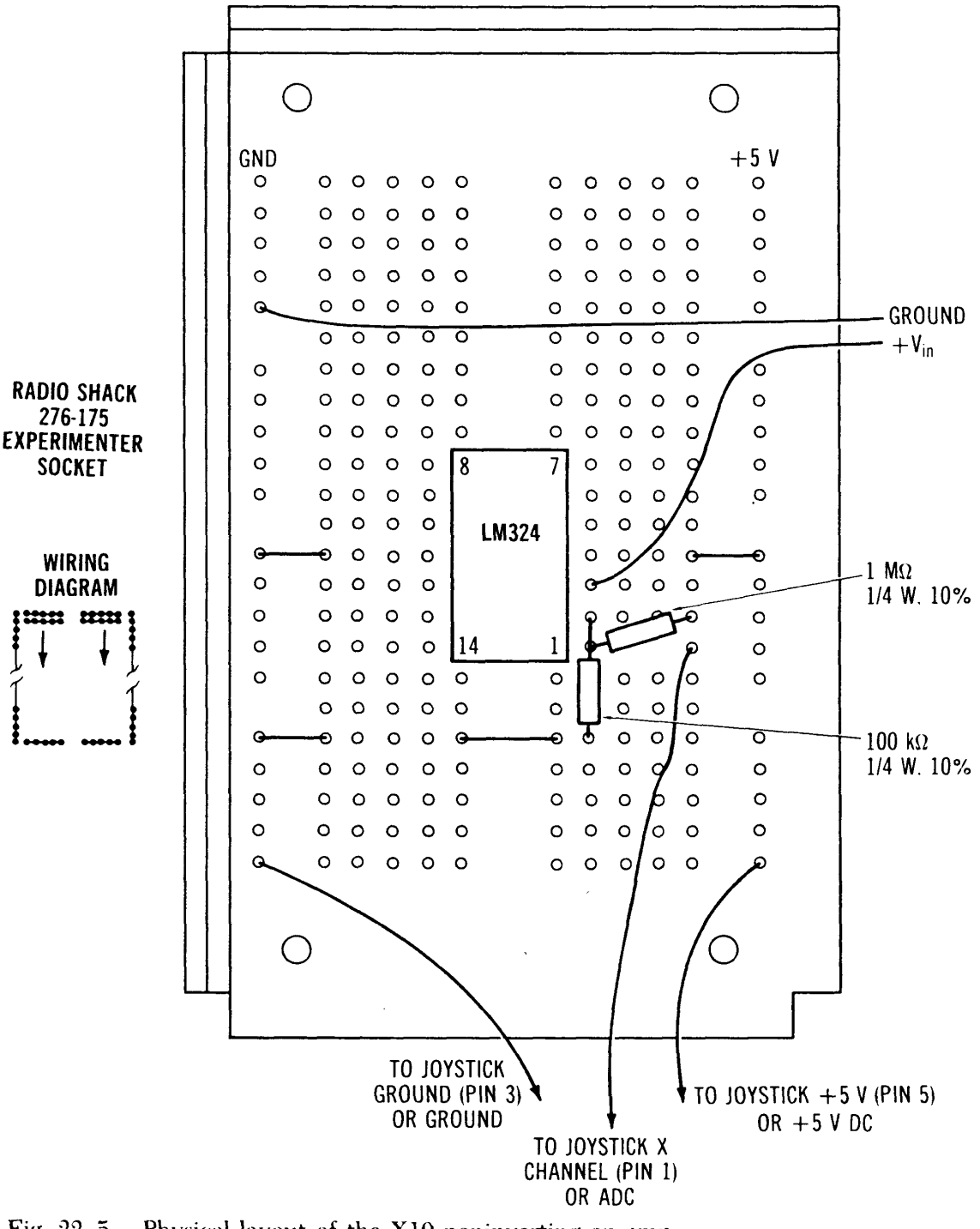


Fig. 22–5. Physical layout of the X10 noninverting op amp.

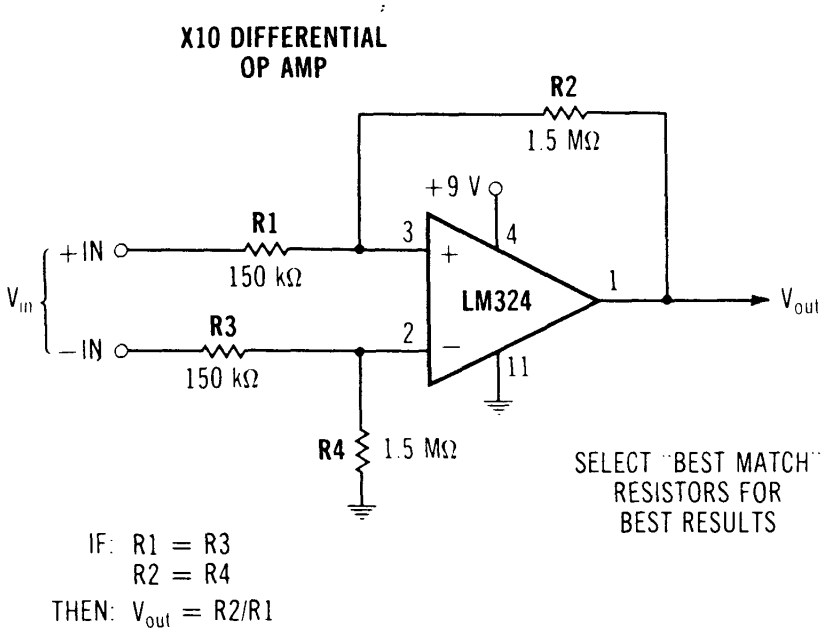


Fig. 22–6. Differential op amp amplifier circuit.

shown in Fig. 22–6. This type of amplifier is handy for amplifying bridge type outputs, where one output increases and the other decreases, as is the case with the National LX0503A pressure transducer. The configuration shown is a X10 amplifier.

In the next chapter we look at transducers which can be connected to provide real-world inputs, many of them via the op amps described in this chapter.

chapter **23**

Transducer Projects

The devices in this chapter are more than switches. They are actual *transducers*, devices that transform one form of energy into another. An example of this is the National LX0503A pressure transducer, which transforms energy in the form of pressure to a voltage via a piezoelectric bridge.

The two most common electrical analogs are a changing voltage, produced by devices such as the LX0503A, and a changing resistance, produced by devices such as a thermistor, which has a resistance that varies with temperature. When the electrical analog sensor involved is a resistance, it's convenient to monitor the changing voltage drop across the varying resistance, which can be more easily manipulated in these types of operations. Therefore, we're really only measuring or monitoring voltage, by means of the Color Computer analog-to-digital converter or an equivalent Model I or III circuit. We use the Color Computer for sample programs in this chapter, but all the concepts apply to the Models I and III as well.

A SOLAR CELL LIGHT DETECTOR

Solar cells are designed to convert sun or incandescent light to electricity. In recent years solar cells have been improved both in price and efficiency. The cell we tested was a Radio Shack 276-124 cell; similar cells are available from Edmund Scientific. Normally one thinks of solar cells as energy converters and not in the same sense as a light detector. However, the solar cell tested turned out to be a good light detector as well.

On a clear day at 25°C (77°F) at noon in direct sunlight, the cell produced about 0.535 volt and 0.18 A, or about 1/10 W. When taken out of direct sunlight, output falls rapidly. Table 23–1 lists voltages devel-

Table 23–1. No-Load Solar Cell Output

Condition	Output (volts)
Direct sunlight at 0900	0.535
Turned 90 degrees from sun	0.490
To sun, overcast	0.482
Turned 180 degrees from sun	0.478
Outdoors, shade gradations	0.39–0.445
Inside house, day, to window	0.225
Inside house, day, to inside	0.065
Inside house, dark hallway	0.003
From 75-watt lamp (inches):	
6	0.426
12	0.359
18	0.228
24	0.163
30	0.116
36	0.085

oped under different conditions, including a 75-watt incandescent lamp. Some other specs that might prove interesting: The solar cell was not responsive to any degree to infrared light. Also, voltage dropped considerably when the cell output was loaded down as shown in Table 23–2.

The solar cell can be used for a light detector that can be activated by a flashlight. Since it has a much larger surface area than a photocell or other photosensitive device, your aim doesn't have to be precise. Using the X5.6 op amp and the Color Computer program in Fig. 23–1, the flashlight could be detected from 8 feet away. With the X10 op amp, the output goes to full scale, but the flashlight beam is detected from 16 or 20 feet. A general BASIC program for use with the 5.6X op amp and solar cell is given in Fig. 23–2.

Table 23–2. Loaded Solar Cell Output

V (no load)	Load (ohms)	V (load)	I (load, mA)
0.482	8.6	0.30	37.00
0.4	46.8	0.35	6.29
0.4	36.5	0.33	7.52
0.4	23.4	0.27	9.45
0.4	8.6	0.13	13.88

Readers may want to compare the solar cell circuit with an earlier photosensitive device, the Radio Shack cadmium sulfide photocell, discussed in Chapter 2. This device decreases in resistance as light intensity increases, swinging from about 20 ohms in sunlight to 5 MΩ in the dark. It can be used in a voltage-divider circuit as shown in Fig. 23–3.

A THERMISTOR TEMPERATURE-SENSING CIRCUIT

Also in Chapter 2 is a discussion on thermistors, another component that changes resistance, in this case with ambient temperature. However, the thermistor discussed in that chapter is ill-suited for a computer system input, as it's a large television-set-type thermistor used to detect overcurrent conditions.

An infinite number of smaller thermistors is available, devices that are very sensitive to small changes in temperature and respond in a second or less. One of the chief suppliers of thermistors is Fenwal Electronics, Framingham, MA. They carry a complete line of every available thermis-

```
100 'X10 FOR LIGHT DETECTOR
110 A=JOYSTK(0)
120 IF A>0 THEN SOUND 100,1
130 GOTO 110
```

Fig. 23–1. Solar cell light detector program.

```
100 'X5.6 OP AMP FOR SOLAR CELL
110 CLS
120 A=JOYSTK(0)
130 PRINT @  256+5,"JOYSTICK VALUE=";A
140 PRINT @ 288+5,"A/D V=";INT((A/64)*4.9*100)/100
150 PRINT @ 320+5,"CELL V=";INT((A/64)*4.9/5.6*100)/100
160 GOTO 120
```

Fig. 23–2. Program to operate solar cell and op amp.

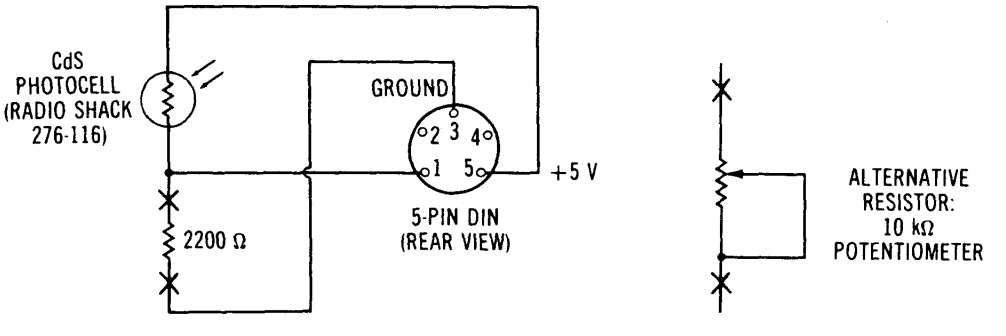


Fig. 23–3. Cadmium sulfide photocell circuit.

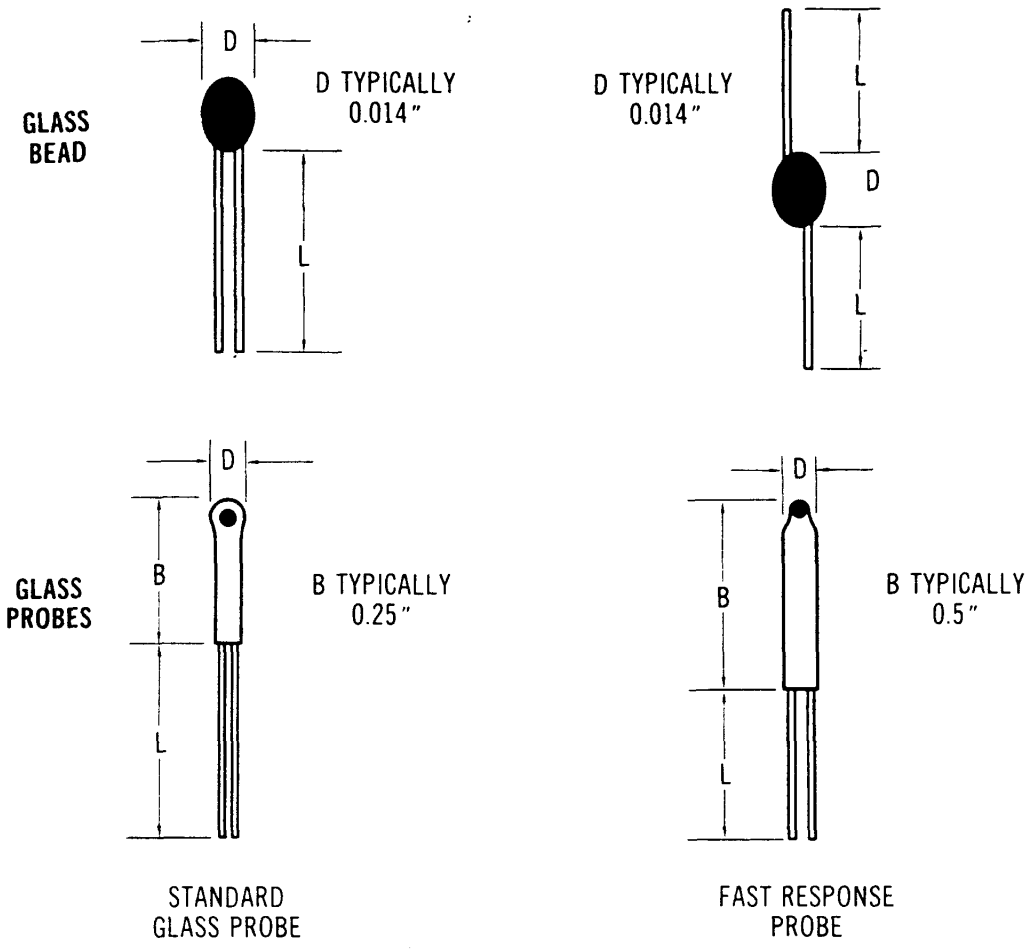


Fig. 23–4. Representative types of thermistors.

tor and have local distributors. Fenwal and other brands of thermistors come in many different types, some of which are shown in Fig. 23–4. They differ primarily in the size and type of casing, which is usually glass. Generally the smaller the package, the more sensitive the thermistor. The probe package is more suitable for immersion in liquid. The nominal resistance (25°C resistance) of Fenwal thermistors ranges from about 1000 ohms to 10 MΩ. Prices are on the order of a few dollars, depending upon the package type.

In our test setup we used a Fenwal GA45P1 thermistor. This is a 50 kΩ thermistor in a standard glass probe configuration that responds (changes resistance) in a few seconds. The input to the adc is shown in Fig. 23–5. The resistance of the GA45P1 for various temperatures is shown in Table 23–3, along with the expected voltage at the junction of the voltage divider. The R_{COEFF} is the coefficient to be multiplied times the 25°C resistance to find the resistance at various temperatures. You can see that the resistance/temperature curve is far from linear. We've got to deal with wide extremes in resistance.

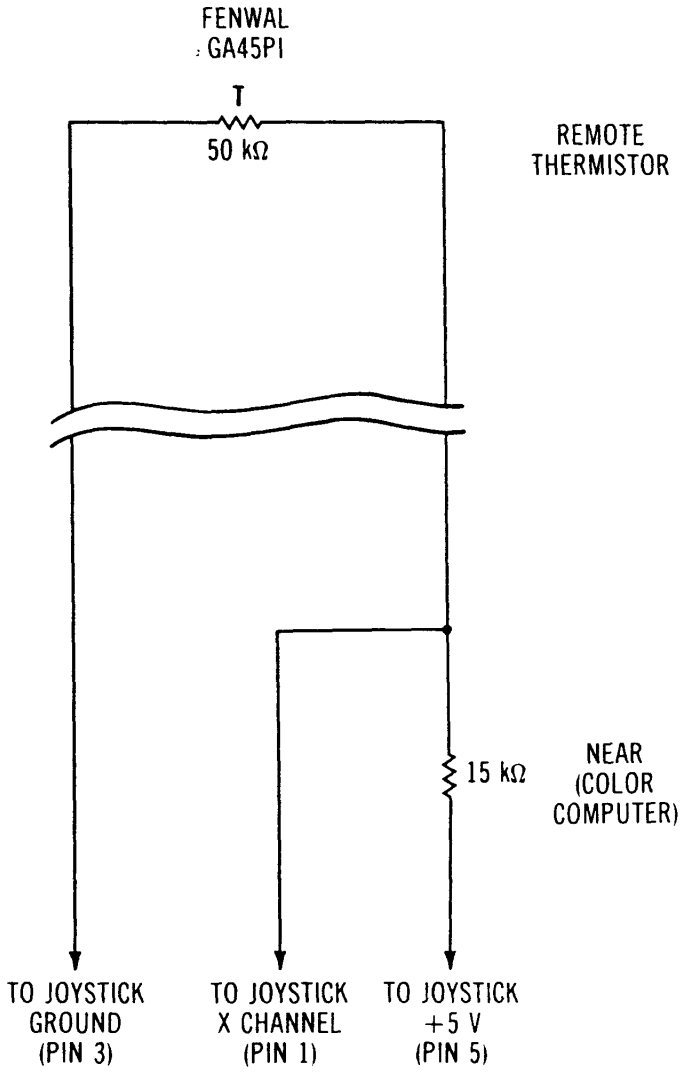


Fig. 23–5. Thermistor input to an adc.

We used a Color Computer BASIC program to read in the voltage divider input, and came up with the readings in Table 23–4. These values represent the voltage normalized to JOYSTK values of 0 through 63; each count represents about 78 mV (5 volts/64).

A more elaborate BASIC program that interpolates values based upon resistance values from Table 23–3 is shown in Fig. 23–6.

Plotting the expected voltages at various temperatures versus measured values produced the plot shown in Fig. 23–7. Rather than fake the readings, as we used to do in university physics lab, we left in the anomaly. Temperature was varied by immersing the probe in a container of water to which ice or hot water was added.

Although the thermistor loses sensitivity (in this setup) at temperature extremes, it is still a very useful device, as it is small, uncomplicated, and

Table 23–3. GA45P1 Resistance vs. Temperature

Temperature		$R_{(COEFF)}$	R_{THERM}	Voltage Divider
−10°C	14°F	6.12	305,000	4.71
0	32	3.51	175,500	4.55
10	50	2.08	104,000	4.32
20	68	1.27	63,500	4.04
25	77	1.00	50,500	3.81
30	86	0.794	39,700	3.59
40	104	0.510	25,500	3.11
50	122	0.336	16,800	2.61
60	140	0.226	11,300	2.12
70	158	0.155	7,750	1.68

$$V = \left(\frac{R_T}{15{,}000 + R_T} \right) \times 4.95$$

inexpensive. Large thermistor values will be unaffected by long runs of wire, and the device can be located any distance away from the computer. Resolution from freezing to 125°F is fine, allowing us to detect changes in temperature of 4° or 5°F.

We haven't discussed an important aspect of thermistors, which you may care to experiment with—*self-heating mode*. If current through a thermistor increases without limit, the thermistor heats up, lowers its resistance, increasing the current, and so forth, until *thermal runaway* occurs, burning up the thermistor. This self-heating mode can be initiated and held in check by a suitable series resistance. The thermistor will

Table 23–4. GA45P1 ADC Circuit Values

Temperature (°F)	VOM	Color Computer Joystk (0)
116	2.83	36
110	2.92	37
104	3.07	39
100	3.23	41
96	3.41	44
90	3.49	45
85	3.64	48
80	3.73	49
57	3.94	52
41	4.33	57

```
100 ' GA45P1 THERMISTOR I/O
101 CLS
102 DATA 14,4.71,4.55,32,4.55,4.32,50,4.32,4.04,68,4.04,3.59
103 DATA 86,3.59,3.11,104,3.11,2.61,122,2.61,2.12
104 DATA 140,2.12,1.68,-1,-1,-1
110 A=JOYSTK(0)
120 V=(A/63)*(4.95)
130 RESTORE
140 READ T,V1,V2
150 IF (T=-1 OR V>4.71) THEN PRINT "OUT OF RANGE":STOP
160 IF ((V>=V2) AND (V<=V1)) THEN GOTO 170 ELSE GOTO 140
170 V=((V1-V)/(V1-V2))*18+T
180 PRINT @ 256+10,INT(V*10)/10,"      "
190 GOTO 110
```

Fig. 23–6. Thermistor measurement program.

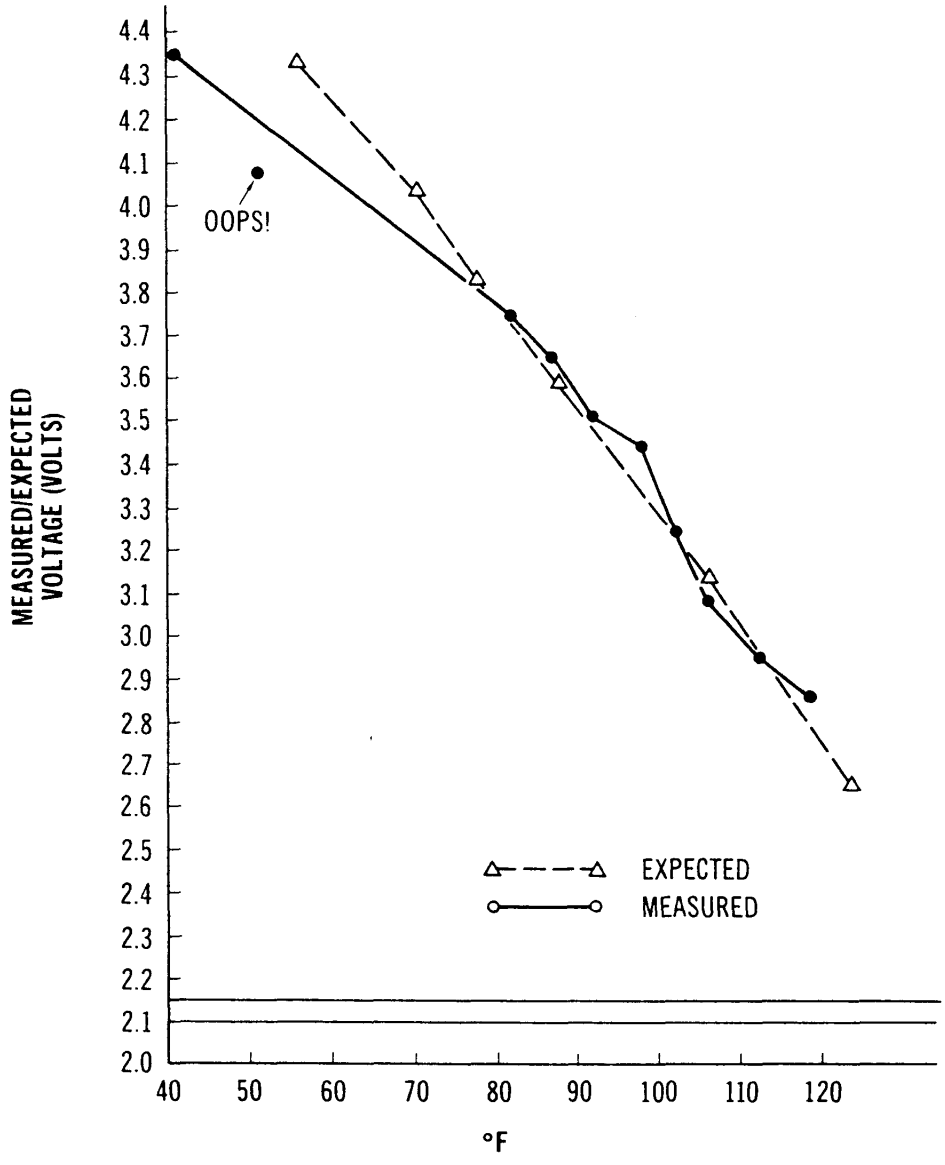


Fig. 23–7. Thermistor operating characteristics plot.

then heat up to 100°C or so. Any change in ambient conditions now affects the thermistor temperature and also the current flow through the thermistor. Blowing on the thermistor, for example, will take heat away from the thermistor by convection, as will flowing fluids. Excellent flowmeters (including anemometers), vacuum pressure gauges, and similar types of instruments may be created by thermistors in the self-heated mode. See Fenwal specifications if you'd care to experiment.

AN LM334 TEMPERATURE-SENSING CIRCUIT

An alternative approach to computerized temperature-sensing is the use of an LM334 (Radio Shack 276-1734). This device is a *temperature sensor and adjustable current source* and is about the size and appearance of a transistor.

Among such applications as current limiting, the LM334 also offers temperature sensing. Its output voltage will change approximately 10 mV per degree Kelvin. Degree Kelvin? Since we rarely deal with absolute zero in our three systems, think in terms of degrees Celsius or Fahrenheit. For every degree Celsius change (1.8°F), the output will change by 10 mV. An 18°F change will result in a change of 0.1 volt, and a 72°F change will result in a change of 0.4 volt. These changes are too small by a factor of 10 for our adc purposes, but we have our X10 amplifier! Note that the temperature changes are linear, unlike the thermistor. See Fig. 23–7. A 1° change in temperature always results in a 10 mV change in output!

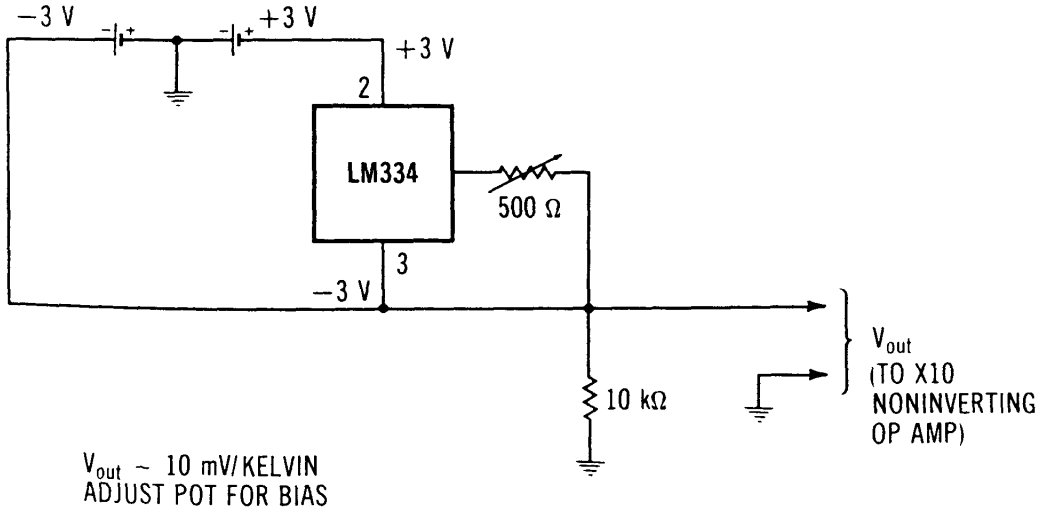


Fig. 23–8. LM334 adc circuit.

The circuit we used in this application is shown in Fig. 23–8. This circuit "floats" the LM334 between +3 and −3 volts. The 500-ohm pot is adjusted until the output is about midscale (2.5 volts) for the center temperature in the range. The output of the LM334 goes to a noninverting X10 op amp, which then connects to the adc channel.

For this test setup, we used an LM334 on a long wire, as shown in Figs. 23–9 and 23–10. An advantage of this circuit, by the way, is that it is a current-type device which will allow long runs to the sensor. The assembly was dipped in PVC cement for waterproofing. Test results are shown in the graph of Fig. 23–11. The slope of the line shows about 10 mV per 0.972°C, which compares favorably with the expected results. Here, the range of input temperatures was about 37°F through 91°F; adjust

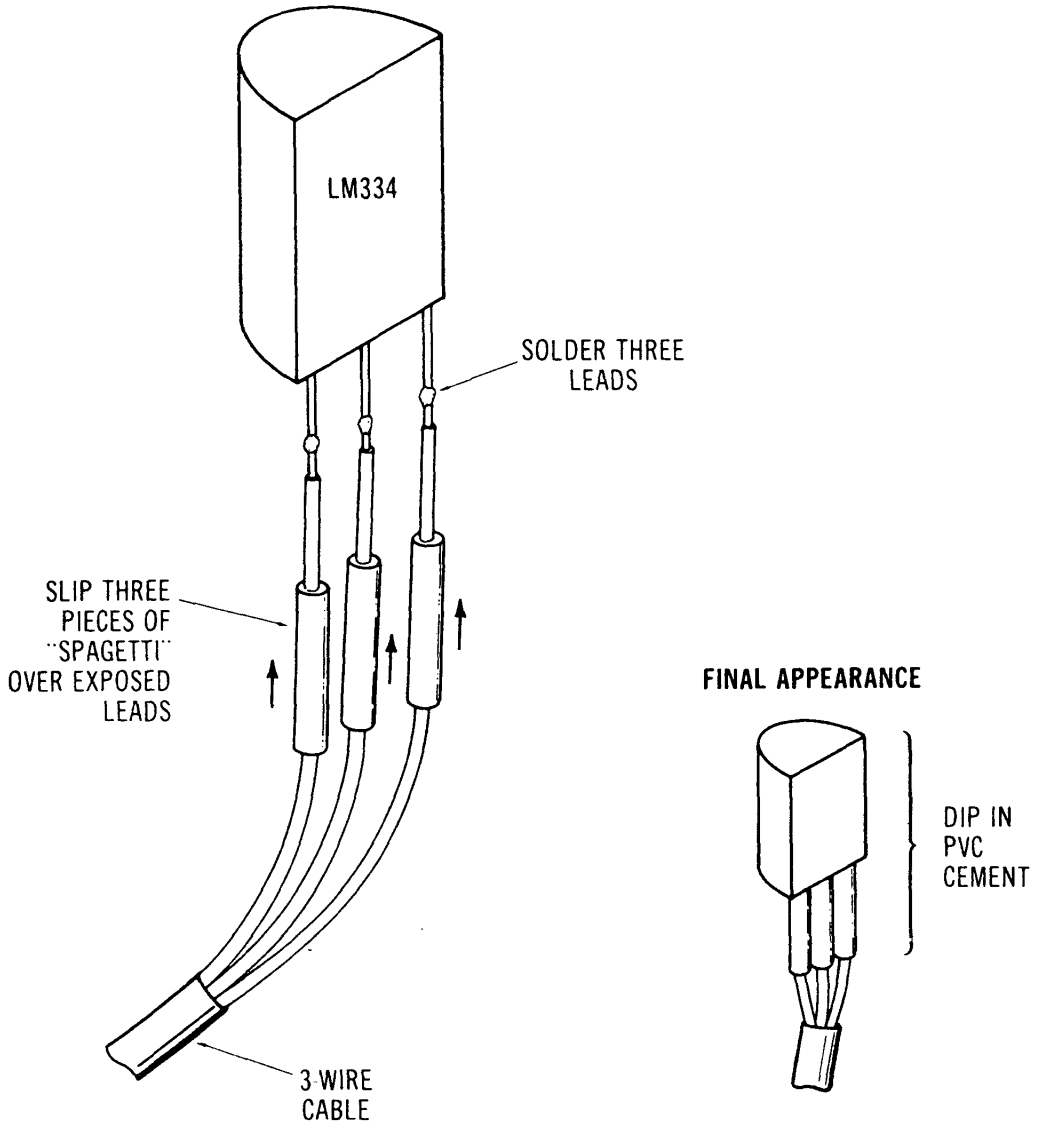


Fig. 23–9. LM334 remote sensing.

the potentiometer for the range you require or use less amplification in the op amp.

The LM334 makes an excellent temperature sensor, and I would tend to use it over the thermistor for precise temperature readings. Changes in temperature of about 1.2°F can be detected with the circuit above.

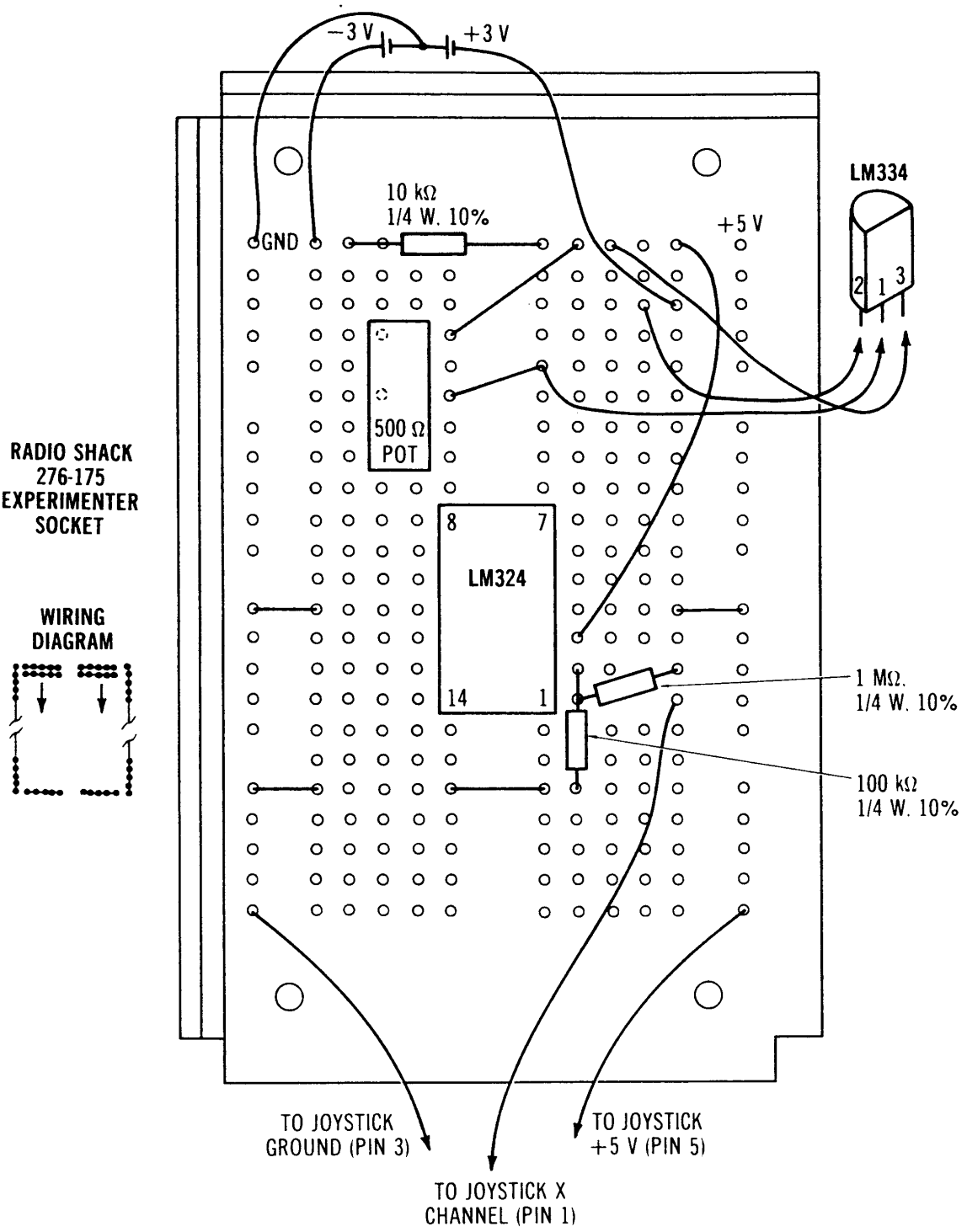
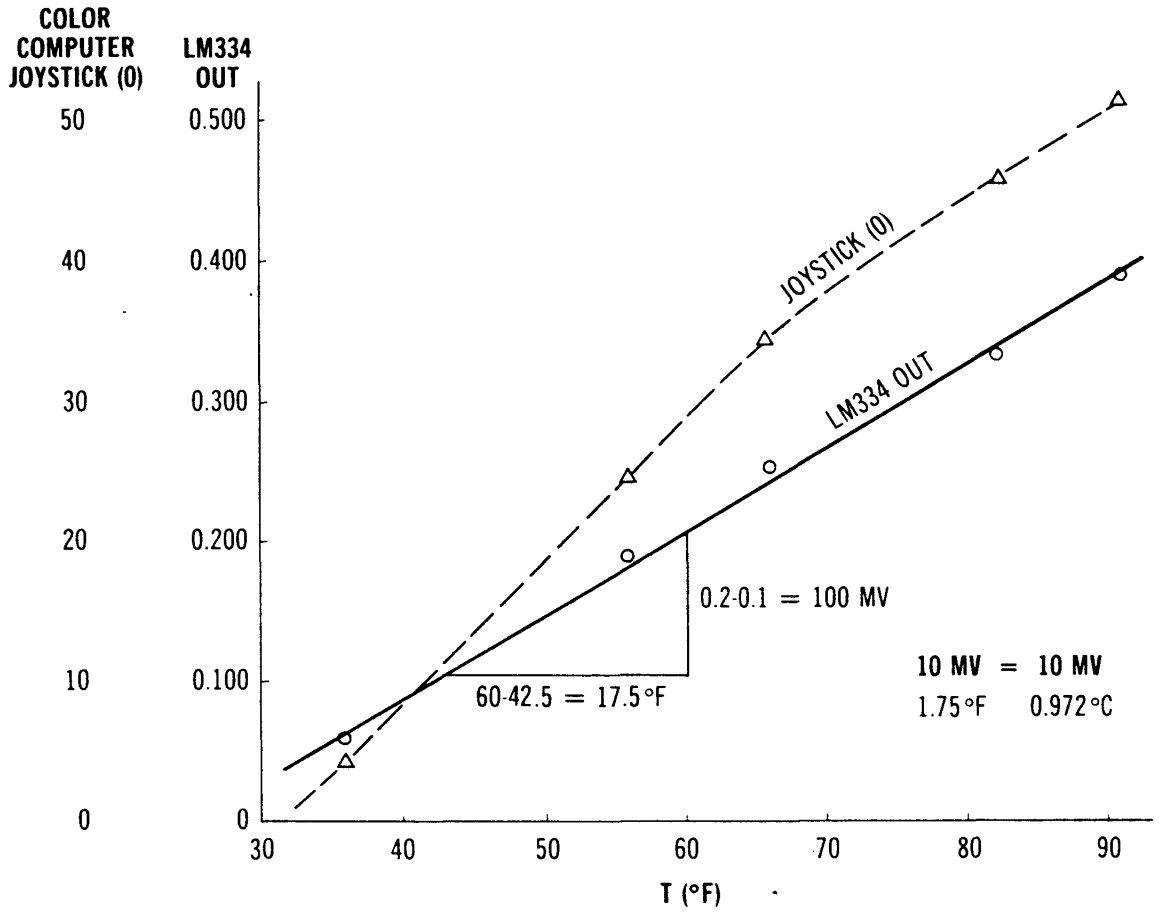


Fig. 23–10. Physical layout of the LM334 adc circuit.



T	LM344 OUT (V)	LM324 OUT VIA COLOR COMPUTER JOYSTICK (0)
37°	0.064	4
57°	0.186	24
67°	0.245	33
82°	0.323	44
91°	0.376	48 (LIMIT)

Fig. 23–11. LM334 operating characteristics plot.

DC MOTORS USED AS GENERATORS

Radio Shack and many other suppliers sell small dc motors that will operate from 1.5 through 6 volts dc and rotate at up to 8000 or 10,000 rpm. A small permanent-magnet dc motor can also be used as a generator if an external force turns the shaft and the motor leads are monitored. Is it feasible to use a dc motor as a generator for measuring rotational speed? Here's what we found out.

The motors used were Radio Shack 273-208. These motors are rated at 1.5 to 6 volts dc and 3550 rpm with no load. The test setup is shown in

Fig. 23–12. The motor on the left was driven by a variable power supply. The motor on the right was driven by the first motor, via a piece of plastic tubing. Output of the motor generator was monitored by a scope. The motor on the left had a disc with a small circular cutout so that the tachometer wand described in the next project could be used. The tachometer wand was used to measure the rotation speed of the shaft. The motor on the left produced about 2200 rpm with a supply voltage of about 1.2 volt. We got up to 4000 rpm, but at this speed the motor had the characteristic ozone smell that anyone who has ever pushed a model electric train to its limit will recognize!

A typical output from the motor-generator appears as shown in Fig. 23–13. There is an ac component on top of a dc level. This ac component has about the same proportion of the output regardless of speed. The period between *breaks* (low points) on the ac component is 1/6th of the actual period for the speed of the motor. In other words, the motor *commutates* (reverses current direction) six times per revolution and the true rotation speed is given by:

$$\text{Revolutions per second} = 1/(P \times 6)$$

$$\text{RPM} = 10/P$$

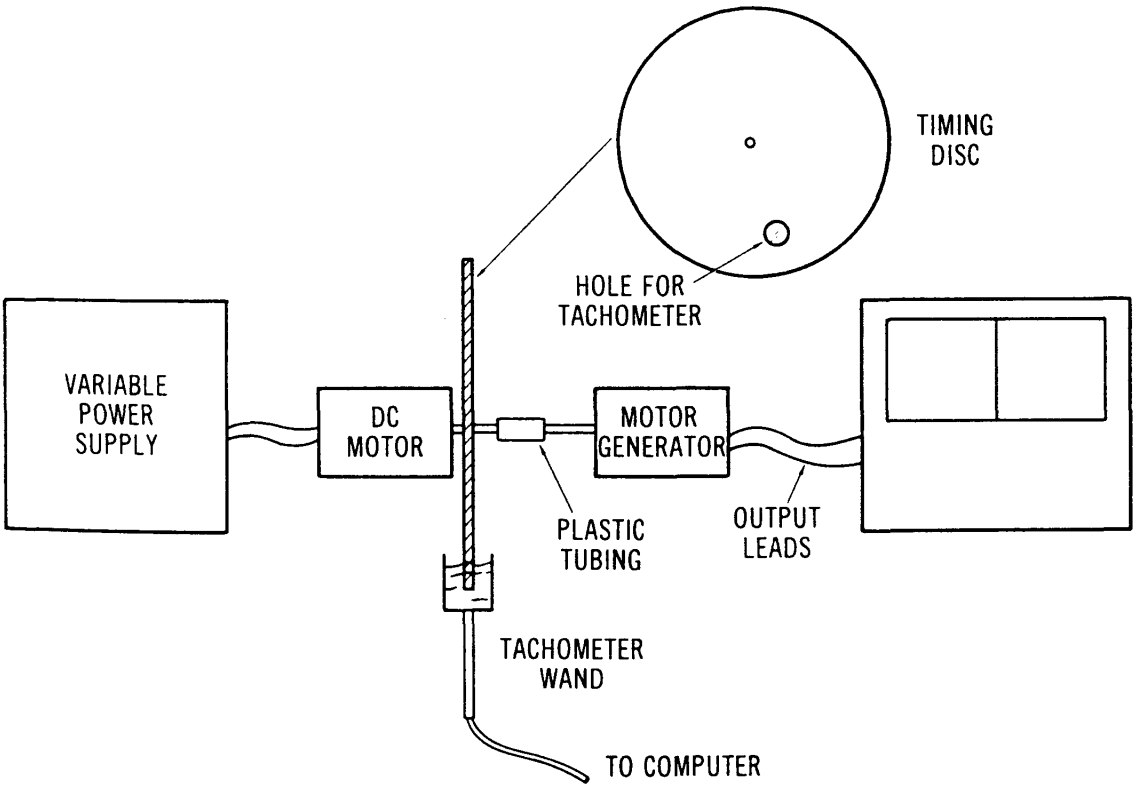


Fig. 23–12. Dc motor-generator testing setup.

This suggests that an adc circuit that was fast enough could derive the rotation speed of the motor-generator directly from the output waveform by measuring the period between breaks. The adc software would have to be in assembly language, of course. The voltages produced for various rotation speeds are shown in Fig. 23–14. The output is linear and ranges from 0 through 0.68 volt for speeds from 0 through 2790 rpm.

We did not measure the output of the motor-generator with an adc. Before you do, you should filter the output to smooth ripple and get rid of noise spikes generated by the motor's mechanical actions. If you want to retain the ac component, bypass noise spikes by putting a 0.1-μF capacitor to ground from the motor output. It might also be a good idea to use a zener diode to limit the input and prevent excessive voltage. These schemes are shown in Fig. 23–15.

A TACHOMETER WAND

The device used to measure rotation speed is a tachometer wand. The circuit for this device is described in Chapter 15 and is shown in Fig.

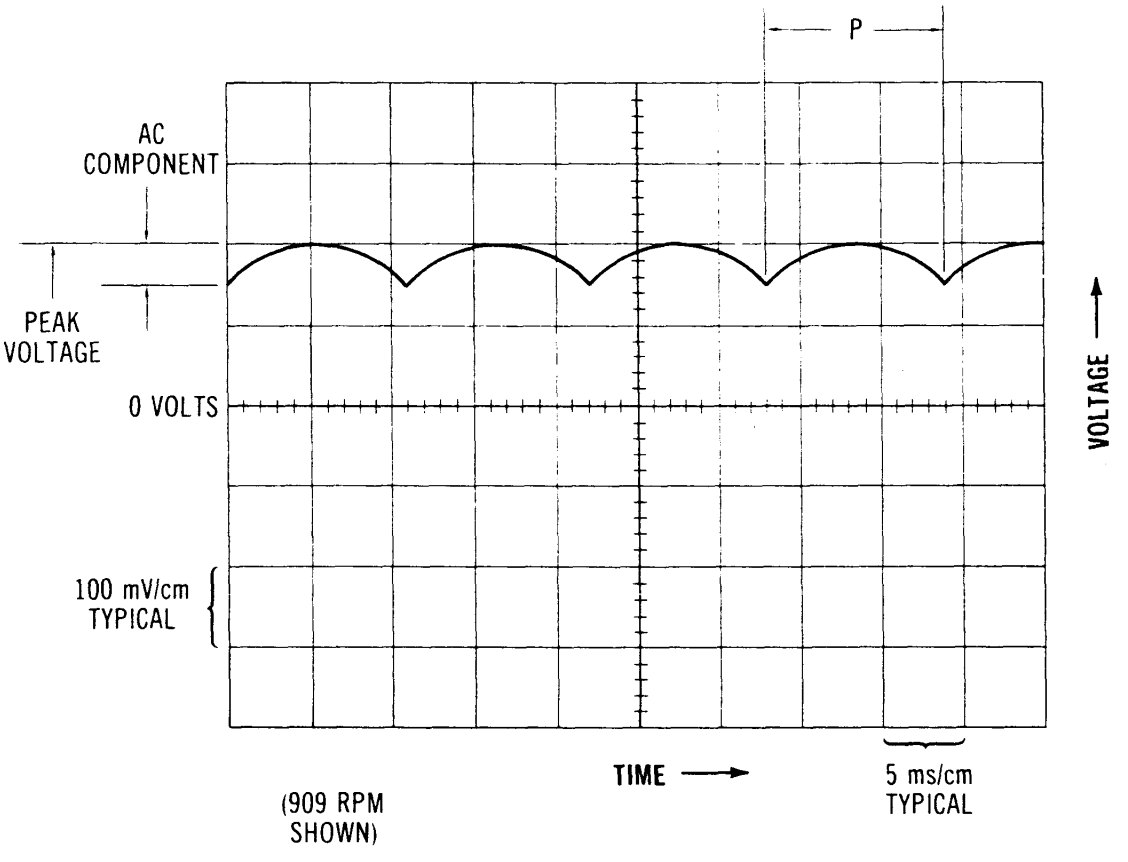
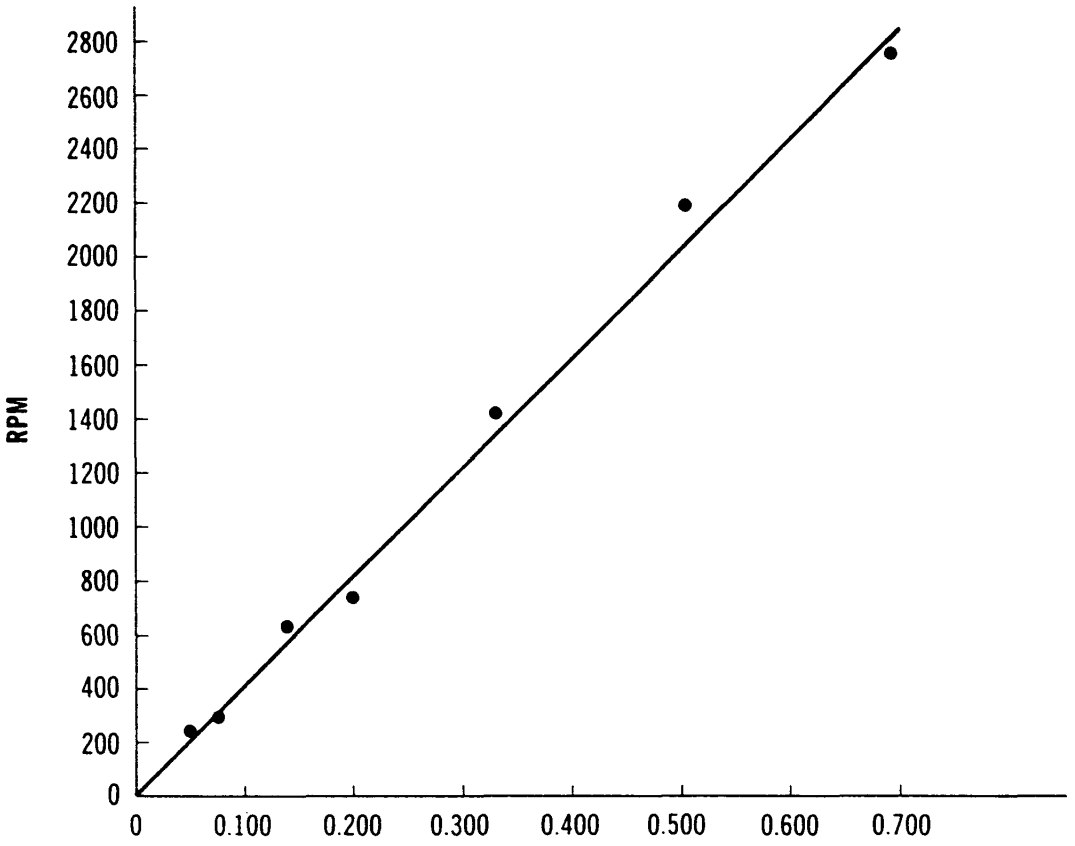


Fig. 23–13. Typical output from a dc motor-generator.

23–16. It uses a high-output infrared LED (RS 276-143) and an infrared phototransistor (RS 276-145). When the infrared light is blocked, the phototransistor output goes to about 4.5 volts. One word of warning: Use the wand away from a strong incandescent light source; there is enough IR component to trigger the phototransistor.

The wand was mounted in a Vector Slit 'N Wrap wiring device (a somewhat expensive way to fabricate it!). The Slit 'N Wrap tool needs no modification except for a hacksaw cutoff of the wrapping end. The two resistors are mounted within the barrel of the tool, and the two IR



RPS	RPM	PEAK V	AC
4.0	240	0.05	0.016
5.0	300	0.075	0.025
11.1	666	0.140	0.040
13.0	778	0.20	0.070
24.4	1464	0.33	0.090
37.0	2220	0.50	0.16
46.5	2790	0.68	0.24

Fig. 23–14. Dc motor-generator operating data plot.

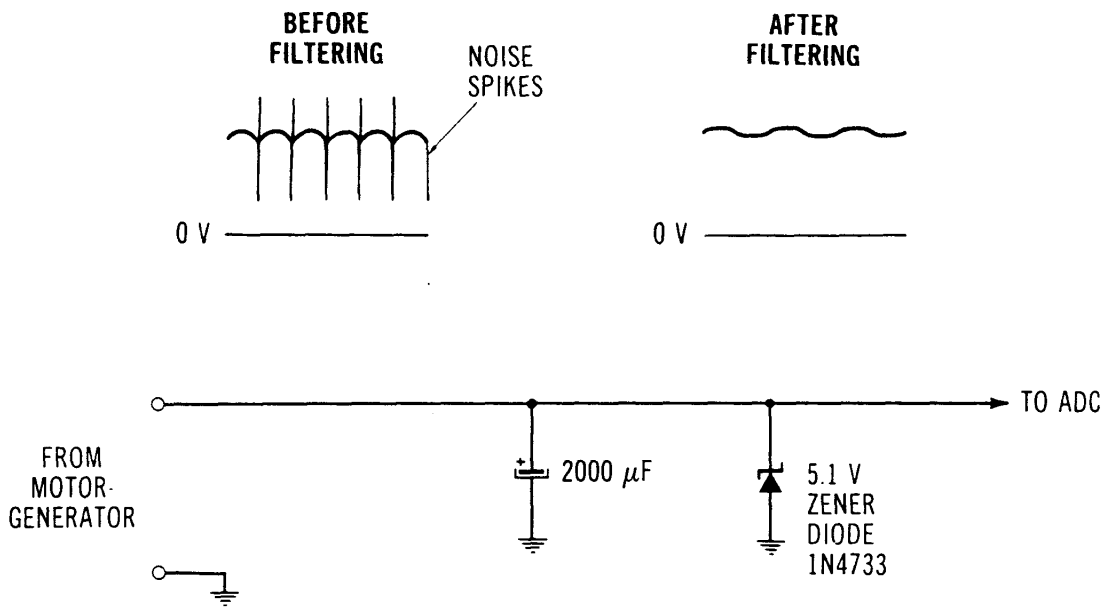


Fig. 23–15. Dc motor-generator signal-conditioning circuit.

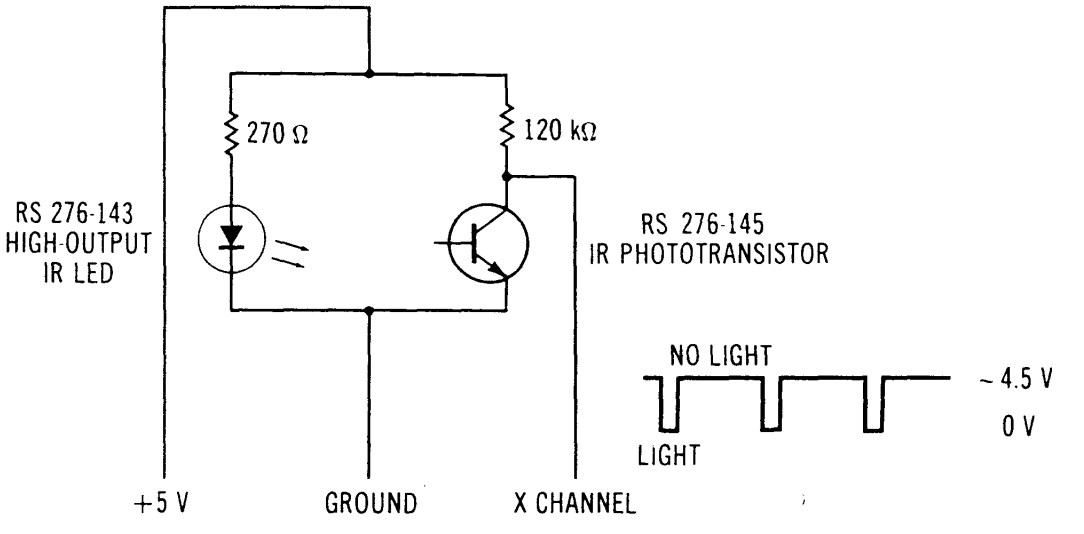


Fig. 23–16. Tachometer wand circuit.

devices fit perfectly into the holes of the U-shaped section of the tool. See Fig. 23–17. Refer to Chapter 15 to get some ideas on assembly language programs to read rotation speed directly.

A PRESSURE TRANSDUCER

The last device considered here is a National Semiconductor LX0503A pressure transducer. This device is the most expensive of all that we

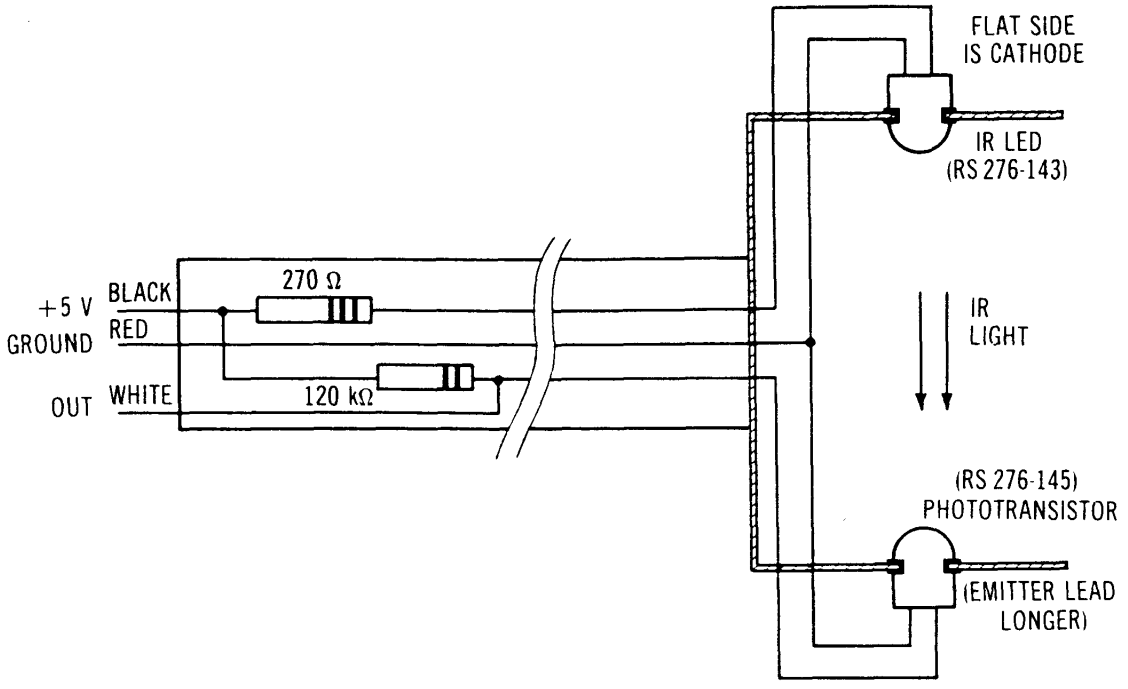


Fig. 23–17. Physical layout of the tachometer wand components.

considered, but it is still inexpensive at less than $20.00. The LX0503A is a device that changes pressure into voltage. This version operates in the range of 0 to 30 pounds per square inch (psi). Normal atmospheric pressure is about 14.7 psi.

The physical appearance of the device is shown in Fig. 23–18. It is mounted in a TO–5 size can (the size of a typical metal can transistor), with an inlet port on the top of the can. Eight leads come out of the device, five of which are used.

The circuit for the LX0503A is shown in Fig. 23–19. A piezoelectric (crystal) element forms one leg of the bridge. Output is taken between V2 and V1. This is a *differential* type output, where V2 goes more negative as the pressure increases, and V1 goes more positive. Output changes approximately 2 to 8 mV with 1 psi change in pressure. You can see that over the range of 30 psi, there will be a change of 60 to 240 mV; therefore, some amplification is going to be required. Power supply voltage is from about 5 to 12 volts.

The National Pressure Transducer Handbook, 1981 Edition, contains recommended interface circuits for the transducer. It places a strong emphasis on temperature compensation. For environments in which there will be no radical changes in temperature, however, we can dispense with the temperature compensation circuits and greatly simplify the circuit. Furthermore, supplying the excitation voltage directly to the

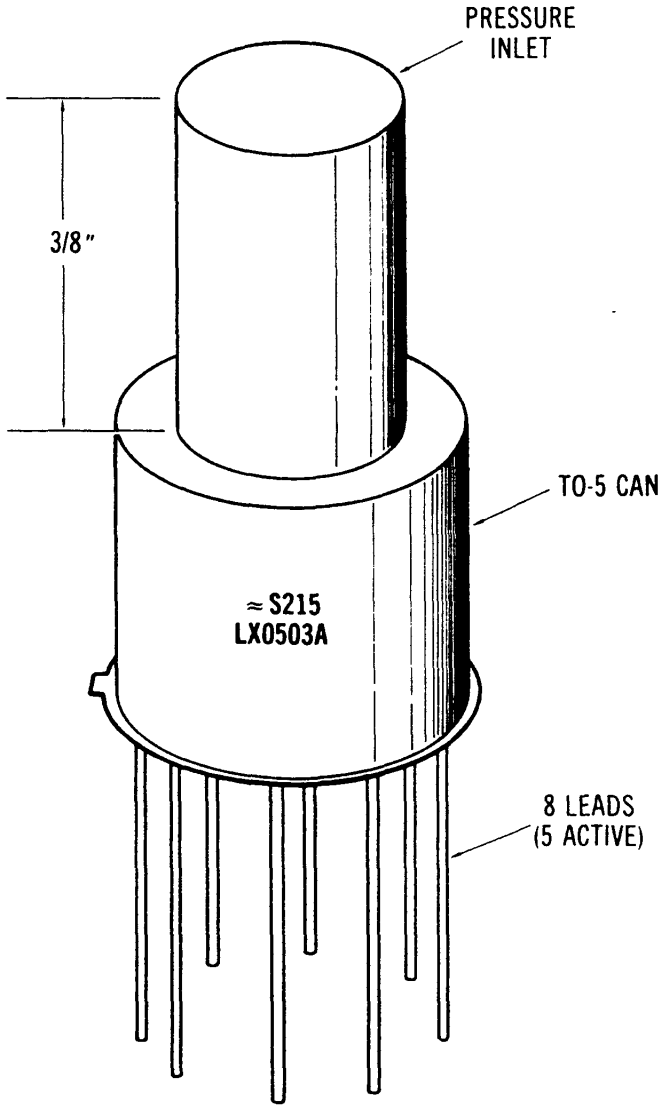
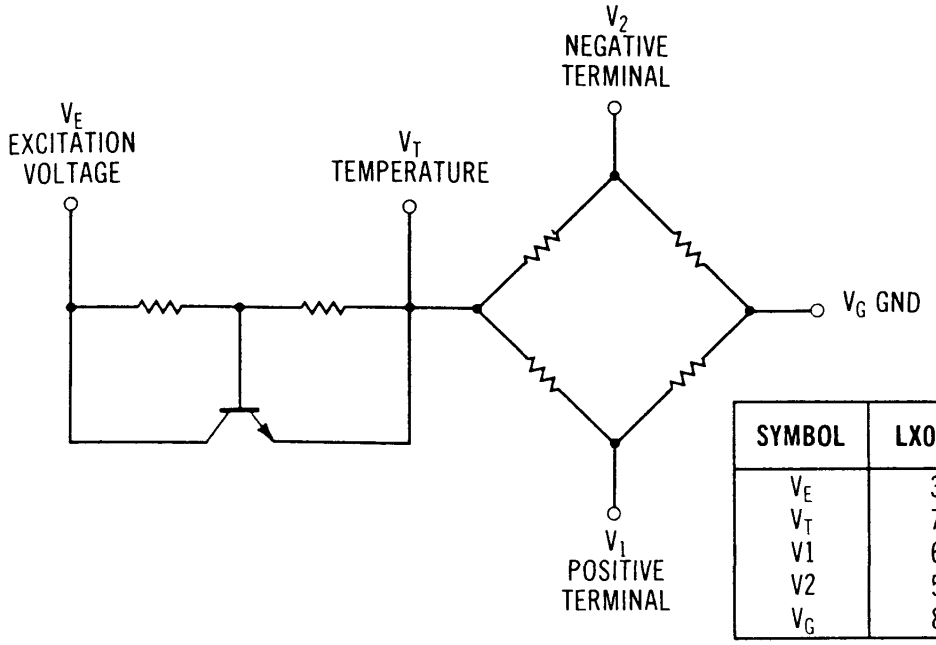


Fig. 23–18. LX0503A transducer.

VT terminal (instead of the VE terminal) increases the sensitivity of the device. I found about 10 mV per psi with a 9-volt supply voltage when the circuit shown in Fig. 23–20 was used.

At normal ambient pressure, output of V1 referenced to ground is about +4.71 volts, output of V2 is +4.55 volts, and the differential, of course, is 0.16 volt. The output of the LX0503A in this case went to a X10 noninverting differential op amp amplifier. The static output was about 1.6 volts. Testing was far from ideal. We used a rubber bulb to increase the pressure via a piece of plastic tubing slipped over the inlet port. Maximum reading obtained was 2.4 volts, indicating a pressure of about 22 psi. The output of the device is linear, and no doubt further testing would reveal it to be an accurate pressure transducer.



SYMBOL	LX05XX	LX06XX
V_E	3	5
V_T	7	1
V1	6	3
V2	5	4
V_G	8	2

Fig. 23–19. LX0503A internal circuit.

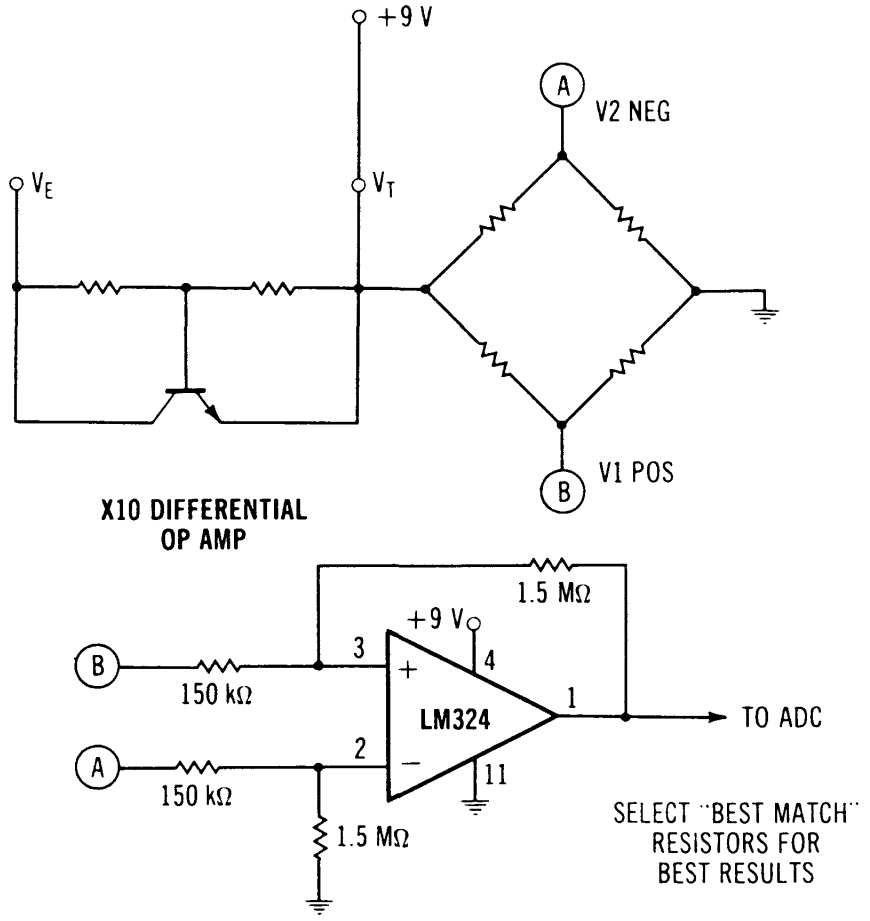


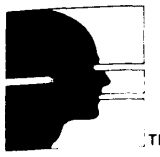
Fig. 23–20. LX0503A adc circuit.

The LX series pressure transducers come in a number of different ranges and two versions, absolute (such as the LX0503A) and differential. The differential type measures the pressure difference between two inlet ports; the absolute type is referenced to a vacuum. Pressure ranges for either type are 0–30, 0–100, 0–1000, and 0–3000 psi. You might consider designing a barometer driven by the LX0503A. With suitable biasing and another stage or two of amplification, a sensitive working barometer should be possible.

IN CONCLUSION

That's just a small sampling of some of the inexpensive switches, transducers, and other devices that you can use to interface your Model I, III, or Color Computer to the real world. There's no reason that your small computer shouldn't be able to monitor temperature, pressure, ambient light, and other physical quantities and effectively control your home without an outlay of thousands of dollars. All three systems offer unlimited opportunities for control and monitoring of their surroundings. It's up to you to put some of these ideas in practice!

Index



SAMS TRS-80 BOOKS

Many thanks for your interest in this Sams Book about TRS-80® microcomputing. Here are a few more TRS-80-oriented Sams products we think you'll like:

TRS-80® COLOR COMPUTER INTERFACING

Teaches you interfacing techniques, inner workings, and operation of the TRS-80 Color Computer, its 6809 microprocessor, and the expansion connectors. Excellent info for budding techs and engineers at all levels. By Andrew D. Staugaard, Jr. Approximately 192 pages, 5½ x 8½, soft. ISBN 0-672-21893-3. © 1983.

Ask for No. 21893 ..... **$14.95 Tentative**

TRS-80® MODEL I, III, AND COLOR COMPUTER INTERFACING PROJECTS

Fourteen simple, useful, and easy-to-build construction projects help you make use of your TRS-80 computer in the real world. An easily understood tutorial with fully documented and debugged software. By William Barden, Jr. Approximately 296 pages, 5½ x 8½, soft. ISBN 0-672-22009-1. © 1983.

Ask for No. 22009 ..... **$14.95 Tentative**

USING THE Z-80 IN THE TRS-80®

Shows you how to access the powerful Z-80 in the TRS-80 models I and III. Learn the Z-80 instruction set, its TRS-80 implementation, hardware, software, and more. You'll need ready access to a TRS-80 model I or III. By Elmer Poe. 258 pages, 5½ x 8½, soft. ISBN 0-672-21839-9. © 1982.

Ask for No. 21839 ..... **$13.95**

REAL-TIME CONTROL WITH THE TRS-80®

Shows how to use your TRS-80 for sophisticated, low-cost control of mechanical or electrical devices. Only an ordinary knowledge of Level II BASIC is needed, and no computer modification is necessary. Provides everything you need, including where to find the sensors and actuators. By Russell M. Genet. 168 pages, 5½ x 8½, soft. ISBN 0-672-21831-3. © 1982.

Ask for No. 21831 ..... **$14.95**

TRS-80® ASSEMBLY LANGUAGE MADE SIMPLE

Learn how to plan, write, and hand-assemble your own assembly language programs in memory, using the T-BUG and Level II BASIC ROM subroutines. Provides immediate, short-cut results for the user who can simply use existing routines. By Earles McCaul. 192 pages, 5½ x 8½, soft. ISBN 0-672-21851-8. © 1981.

Ask for No. 21851 ..... **$12.95**

TRS-80® — MORE THAN BASIC

Learn to program in Z-80 mnemonics, using more than 26 available (and changeable) commands. Interactive monitor program automatically flags incorrect instructions or commands and turns your TRS-80 into a cost-effective development system! By John Paul Froehlich. 224 pages, 5½ x 8½, soft. ISBN 0-672-21813-5. © 1981.

Ask for No. 21813 ..... **$10.95**

INTERMEDIATE PROGRAMMING FOR THE TRS-80® Model I

Shows you first how to be more effective with Level II BASIC. Then, leads you gradually into assembly and machine-language programming on the TRS-80 Model 1. Many operating details and programming tips neglected elsewhere. By David L. Heiserman. 240 pages, 5½ x 8½, soft. ISBN 0-672-21809-7. © 1982.

Ask for No. 21809 ..... **$9.95**

MOSTLY BASIC: APPLICATIONS FOR YOUR TRS-80®, BOOK 1

An assortment of 28 fun-and-serious, debugged BASIC programs for your TRS-80, all of which are ready to run. Complete with explanation, sample run, and listing for each program. By Howard Berenbon. 168 pages, 8½ x 11, comb. ISBN 0-672-21788-0. © 1980.

Ask for No. 21788 ..... **$12.95**

MOSTLY BASIC: APPLICATIONS FOR YOUR TRS-80®, BOOK 2

Second gold mine of 32 all-new BASIC programs! Includes 3 dungeons, 11 household programs, 7 or money and investment (including 3 on the stock market), 2 that test your ESP level, and more. Com plete with explanations, sample runs, and listings. By Howard Berenbon. 224 pages, 8½ x 11, comb ISBN 0-672-21865-8. © 1981.

Ask for No. 21865 ..... **$12.9**

CIRCUIT DESIGN PROGRAMS FOR THE TRS-80®

Provides you with a number of Level II BASIC programs to use during design and analysis of elec tronic circuits. Each one can also be used as a subroutine inside of a larger program if you wish, anc all are ready to run. By Howard M. Berlin. 144 pages, 8½ x 11, comb. ISBN 0-672-21741-4. © 1980.

Ask for No. 21741 ..... **$14.5**

TRS-80® INTERFACING, BOOK 1

Introduces you to the various I/O signals of the TRS-80 and suggests their use in a number of practi cal circuits. Many interesting experiments for those with a fairly good understanding of Level I BASIC. By Jonathan A. Titus. 192 pages, 5½ x 8½, soft. ISBN 0-672-21633-7. © 1979.

Ask for No. 21633 ..... **$10.9**

TRS-80® INTERFACING, BOOK 2

Gives you a number of advanced ways to use the knowledge from Book 1, including generation o control voltages and currents, driving high-voltage and high-current loads, and many more. Complete software furnished. By Jonathan A. Titus, Christopher A. Titus, and David G. Larsen. 256 pages, 5½ x 8½, soft. ISBN 0-672-21739-2. © 1980.

Ask for No. 21739 ..... **$11.9**

For both books, ask for No. 21765 ..... **$20.9**

You can usually find these Sams products at better computer stores, bookstores, and electroni distributors nationwide.

If you can't find what you need, call Sams at 800-428-3696 toll-free or 317-298-5566, and charge it t your MasterCard or Visa account. Prices subject to change without notice.

For a free catalog of all Sams Books available, write P.O. Box 7092, Indianapolis IN 46206.

SAMS BRINGS YOU MIND TOOLS™ FOR FINANCIAL PLANNING IN BUSINESS

Special, ready-to-use software that temporarily interlocks with the spreadsheet in your regular ve sion of VisiCalc® so you can immediately perform 17 common financial planning calculations with out wasting time manually setting up the sheet. All you do is enter the data — the proper formula and column headings are there automatically!

Mind Tools allow you to instantly calculate present, net present, and future values, yields, interna and financial management rates of return, and basic statistics.

Also lets you do break-even analyses, depreciation schedules, and amortization tables, as well a compute variable- and graduated-rate mortgages, wraparound mortgages, and more!

Allows you to use your regular spreadsheet as you always have, at any time. Ideal for any busines man with financial planning responsibilities, as well as for business students and instructors.

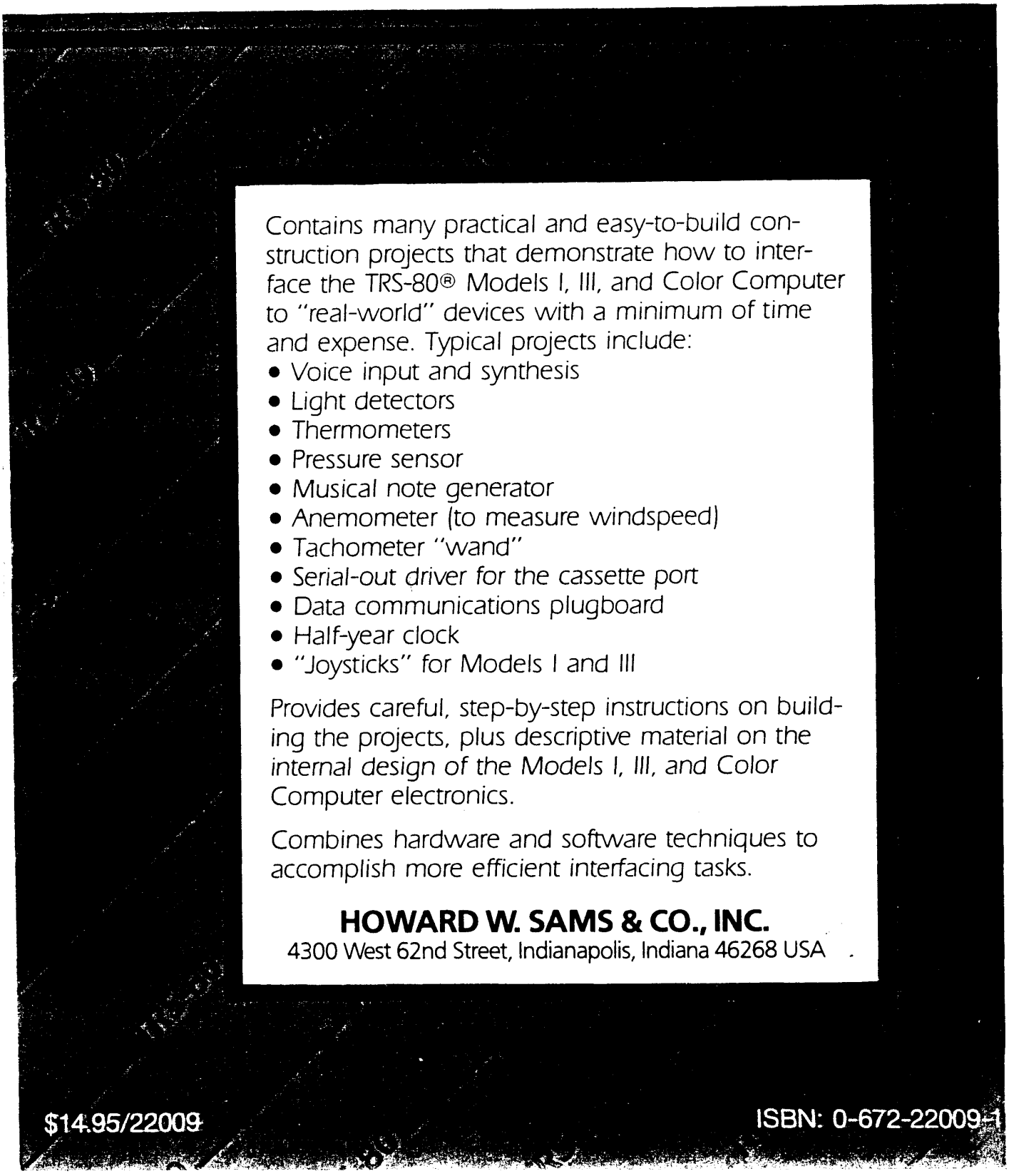
Supplied with disk and complete documentation, including 136-page text and 68-page quick reference guide, all in a looseleaf binder.

EXECUTIVE PLANNING WITH VISICALC

TRS-80® Model II Version, ISBN 0-672-22062-8.

Ask for No. 22062 ..... **$59.9**

TRS-80® Models I, III, & Color Computer Interfacing Projects



Contains many practical and easy-to-build construction projects that demonstrate how to interface the TRS-80® Models I, III, and Color Computer to "real-world" devices with a minimum of time and expense. Typical projects include:

- Voice input and synthesis
- Light detectors
- Thermometers
- Pressure sensor
- Musical note generator
- Anemometer (to measure windspeed)
- Tachometer "wand"
- Serial-out driver for the cassette port
- Data communications plugboard
- Half-year clock
- "Joysticks" for Models I and III

Provides careful, step-by-step instructions on building the projects, plus descriptive material on the internal design of the Models I, III, and Color Computer electronics.

Combines hardware and software techniques to accomplish more efficient interfacing tasks.

HOWARD W. SAMS & CO., INC.
4300 West 62nd Street, Indianapolis, Indiana 46268 USA

$14.95/22009

ISBN: 0-672-22009-1